Optimizing Constrained Concurrent Applications at Run-time

by

Daniel R. Fay

B.S., University of Illinois, 2004

M.S., University of Colorado, 2007

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

Department of Electrical, Computer, and Energy Engineering

2011
This thesis entitled:
Optimizing Constrainted Concurrent Applications at Run-time
written by Daniel R. Fay
has been approved for the Department of Electrical, Computer, and Energy Engineering

______________________________
Professor Dirk Grunwald

______________________________
Professor Li Shang

______________________________
Professor Jeremy Siek

______________________________
Professor Jason Marden

______________________________
Professor Nicolaus Correll

Date ____________________

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.
Computer systems are resource constrained. Application adaptation is a useful way to optimize system resource usage while satisfying an application’s performance requirements. Current multicore computer systems supporting these applications, however, are not designed to reliably meet these requirements. Meanwhile, these computer systems are resource-limited, e.g., have power-induced energy and thermal constraints. Compounding the application’s performance requirements are increasingly-stringent microprocessor thermal constraints. Previous application adaptation efforts, however, were ad-hoc, time-consuming, and highly application-specific, with limited portability between computer systems.

This thesis presents OCCAM, a software platform for developing multicore adaptable applications. OCCAM’s design-time platform consists of design patterns, APIs, and data structures that allow application developers to specify the performance constraints and application-specific optimization techniques. OCCAM generates a run-time controller offline, using profiling data. It then uses this profiling data to generate an internal model that it subsequently employs to generate a robust Markov Decision Process-based Model Predictive Controller. Using a set of Recognition, Mining, and Synthesis benchmarks, the experimental study demonstrates that OCCAM can successfully optimize the system while meeting the systems performance requirements across a wide range of computer platforms, ranging from an energy-constrained single-core system to a high-performance 16-core system. Finally, OCCAM presents a simulation-based, stochastic model checking-based framework for quantifying the robustness of the controller.
Dedication

To my family, and especially my wife Carol, who kept me sane and with my nose pointed to the grindstone even in the roughest of times.
Acknowledgements

This thesis would not have been possible without my family’s support and encouragement to further my education. Hopefully this document will allay their suspicions that I have not done anything for the past seven years.

As the old saying goes, “it takes a village to raise a child”. I would like to thank everyone who was either directly or indirectly involved with my graduate studies, internships, and research for the large collection of varied experiences which made me into the person that I am today, as my graduate experience was highly formative for me.

Finally, I would also like to thank Professor Sriram Sankaranarayanan for his generous help in developing a formal framework in which to describe my system.
Contents

Chapter

1 Introduction 1
   1.1 Thermal Control Example: Driverless Car 1
   1.2 Problem Statement 4
      1.2.1 Key Related Work 6
      1.2.2 Thesis Statement 7
   1.3 Contributions 8
      1.3.1 Cyber-Physical Systems 9
      1.3.2 Recognition, Mining, and Synthesis Applications 10
      1.3.3 Application Adaptation 10
   1.4 OCCAM System Overview 11
   1.5 Design Time Programming Framework 12
   1.6 Run-Time System Control 15
      1.6.1 MPC Controller 17
      1.6.2 Scalable Thermal Control 19
   1.7 Structure 22

2 The OCCAM Platform 23
   2.1 Overview 25
   2.2 Computational Model 26
2.3 OCCAM API and the Data Pyramid ........................................ 28
  2.3.1 OCCAM Run-Time System ............................................. 32
  2.3.2 Writing and Executing an OCCAM Application ...................... 36
  2.3.3 Using OpenCL instead of Intel’s Threading Building Blocks ....... 36

3 Benchmarks ................................................................. 39
  3.1 Overview ................................................................. 39
  3.2 stereo_vision ............................................................. 43
  3.3 wordcount ............................................................... 47
  3.4 tachyon ................................................................. 49
  3.5 seismic ................................................................. 50
  3.6 hearing_aid ............................................................ 52
  3.7 histogram .............................................................. 53
  3.8 lr ................................................................... 55

4 Run-Time System .......................................................... 57
  4.1 Controller Overview ..................................................... 58
  4.2 Offline Profiling ......................................................... 64
  4.3 System Identification .................................................. 66
  4.4 Making a Markov Decision Process ................................... 69
  4.5 Determining the Optimum Control Policy .............................. 70
  4.6 Run-Time Scheduler ................................................... 75

5 Case Study of the Benchmarks’ Homogeneity ........................ 78
  5.1 Overview ................................................................. 78
  5.2 IPC/LLC Cache Miss Homogeneity ..................................... 79
  5.3 Test Setup .............................................................. 87
  5.4 Analysis ................................................................. 87
6 Non-Thermal Control Results
   6.1 Overview ......................................................... 91
   6.2 Test Platform .................................................... 93
   6.3 OCCAM-STOCHASTIC vs. powernowd .......................... 95
   6.4 OCCAM-MPC vs. OCCAM-STOCHASTIC Results .................. 99

7 Thermal Control ..................................................... 112
   7.1 Overview ......................................................... 112
   7.2 Infinite Horizon Optimization .................................... 117
   7.3 Thermal Modeling ................................................ 119
       7.3.1 Offline Profiling .......................................... 120
       7.3.2 Thermal Model .............................................. 121
   7.4 Tractable Control with Heterogeneity .......................... 123

8 Safety, Optimality, and Setting Time of OCCAM’s MPC Controller .... 127
   8.1 Optimality ......................................................... 127
   8.2 Safety ............................................................ 127
       8.2.1 Overview of OCCAM’s MPC-based controller ............... 128
       8.2.2 Safety of the Computer System ............................ 129
       8.2.3 Safety of OCCAM Applications ............................ 129
   8.3 Settling Time ..................................................... 130
   8.4 Effect of Tractability Optimizations on Optimality ............... 130

9 Thermal Control Results ............................................ 133
   9.1 Overview ......................................................... 133
   9.2 Test Setup ...................................................... 134
   9.3 Thermal Control Results ......................................... 135
       9.3.1 HI’s Results ................................................. 135
9.3.2 *MULTI*’s Results .................................................. 137
9.4 Comparison of Different Horizon Lengths .......................... 146
9.5 Action Pruning Results ..................................................... 150
9.6 Runtime Overhead ............................................................. 154

10 Model Checking ................................................................. 160
   10.1 Overview ................................................................. 160
   10.2 Background: Probabilistic Model Checking ......................... 161
   10.3 Applying Model Checking and OCCAM ............................. 162

11 Model Checking Robustness Results ..................................... 163
   11.1 Markov Probability Robustness ..................................... 164
   11.2 Input Measurement Robustness ..................................... 164

12 Related Work ................................................................. 177
   12.1 Application Adaptation .............................................. 178
   12.2 Data Resolution ........................................................ 179
   12.3 Thermal Control Techniques ...................................... 182
   12.4 Phase Prediction ..................................................... 184
   12.5 DVFS-Based Techniques ........................................... 185
   12.6 Real-Time Power-Aware Scheduling ............................... 187
   12.7 Hardware/Software Trends ......................................... 189

13 Summary And Conclusions .................................................. 191

Appendix

A Benchmark Constraints .................................................. 193

B Thermal Logmodel Graphs .................................................. 197
C Setpoints for Cascaded PID Controllers 203

Bibliography 204
Tables

Table

3.1 List of the benchmarks, what they do, and how they implement the Data Resolution. 40
3.2 Available Data Resolution settings for the benchmarks along with the downsampling levels for each. 41
3.3 How the benchmarks measure Quality. 42
3.4 The benchmarks and what constitutes a Throughput Frame in each of them. 43

5.1 Hardware Configuration tested. 80

6.1 Hardware Configurations tested. 93

12.1 Comparison of OCCAM to Code Perforation. 180

A.1 Constraints for hearing_aid. 193
A.2 Constraints for histogram. 194
A.3 Constraints for lr. 194
A.4 Constraints for stereo_vision. 195
A.5 Constraints for tachyon. 195
A.6 Constraints for seismic. 196
A.7 Constraints for tachyon. 196

C.1 P, I, and D values for the different PID controllers. 203
Figures

Figure

1.1 Driverless car example. Since the car’s controller is not aware of the temperature of the processors, it attempts to pass the car in front of it which subsequently overheats the processors, thus requiring the car to bail out of the passing maneuver. 3
1.2 Data flow-based overview of the OCCAM System. 11
1.3 Overview of OCCAM’s design time framework and how it functions at run time. 13
1.4 Overview of OCCAM’s run-time framework. 16
1.5 Peak temperature comparison on MULTI, a 16-core system. 18

2.1 Structural overview of the OCCAM System. 24
2.2 Diagram of OCCAM’s computational model. 26
2.3 The Data Pyramid. 28
2.4 Tiered Performance Requirements. 29
2.5 The Data Resolution concept. 30
2.6 Diagram of the OCCAM-STOCHASTIC and OCCAM-MPC controllers. 34

3.1 stereo_vision left-hand (input), right-hand (input), and difference (output) image. 45
3.2 tachyon output from the train and run data, respectively. 48
3.3 Output of the seismic benchmark. 51
3.4 histogram input image. 53
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Average Instruction Count with different Data Resolution settings.</td>
<td>59</td>
</tr>
<tr>
<td>4.2 Average Instruction Throughput with different Data Resolution settings.</td>
<td>61</td>
</tr>
<tr>
<td>4.3 Average Instruction Throughput with different DVFS settings.</td>
<td>63</td>
</tr>
<tr>
<td>4.4 Benefits of using phase analysis to convert residuals into normally-distributed white noise (Instruction Throughput).</td>
<td>67</td>
</tr>
<tr>
<td>4.5 Benefits of using phase analysis to convert residuals into normally-distributed white noise (Instruction Count).</td>
<td>68</td>
</tr>
<tr>
<td>5.1 IPC and Last-Level Cache miss data for the histogram benchmark.</td>
<td>80</td>
</tr>
<tr>
<td>5.2 IPC and Last-Level Cache miss data for the hearing aid benchmark.</td>
<td>81</td>
</tr>
<tr>
<td>5.3 IPC and Last-Level Cache miss data for the stereo vision benchmark.</td>
<td>82</td>
</tr>
<tr>
<td>5.4 IPC and Last-Level Cache miss data for the lr benchmark.</td>
<td>83</td>
</tr>
<tr>
<td>5.5 IPC and Last-Level Cache miss data for the tachyon benchmark.</td>
<td>84</td>
</tr>
<tr>
<td>5.6 IPC and Last-Level Cache miss data for the seismic benchmark.</td>
<td>85</td>
</tr>
<tr>
<td>5.7 IPC and Last-Level Cache miss data for the wordcount benchmark.</td>
<td>86</td>
</tr>
<tr>
<td>6.1 Structural diagram of the OCCAM-STOCHASTIC controller.</td>
<td>92</td>
</tr>
<tr>
<td>6.2 Structural Diagram of the OCCAM System.</td>
<td>94</td>
</tr>
<tr>
<td>6.3 Energy consumption comparison.</td>
<td>95</td>
</tr>
<tr>
<td>6.4 OCCAM vs. powernowd: HI’s power consumption and CPU frequency over time.</td>
<td>96</td>
</tr>
<tr>
<td>6.5 OCCAM vs. powernowd: success at controlling the application over time.</td>
<td>97</td>
</tr>
<tr>
<td>6.6 OCCAM vs. powernowd: success at controlling stereo vision over time on MULTI.</td>
<td>98</td>
</tr>
<tr>
<td>6.7 Comparison of the percentage improvement in total cost function’s value of OCCAM-MPC controller versus the baseline OCCAM-STOCHASTIC controller</td>
<td>100</td>
</tr>
<tr>
<td>6.8 Comparison of the PID vs. MPC controllers on HI’s train data.</td>
<td>101</td>
</tr>
<tr>
<td>6.9 Comparison of the PID vs. MPC controllers on HI’s run data.</td>
<td>102</td>
</tr>
<tr>
<td>6.10 Comparison of the PID vs. MPC controllers on LO’s train data.</td>
<td>103</td>
</tr>
</tbody>
</table>
6.11 Comparison of the PID vs. MPC controllers on *LO*'s run data.
6.12 Comparison of the PID vs. MPC controllers on *MULTI*'s train data.
6.13 Comparison of the PID vs. MPC controllers on *MULTI*'s run data.

7.1 Peak temperature comparison on *HI*.
7.2 Peak temperature comparison on *MULTI*.

9.1 Summary cost comparison of the benchmarks. The first six Throughput Frames were not counted, in order to give the system time to settle. Overall, OCCAM-THERMAL provides comparable, if not superior, control over either FIXED or Linux. The advantage of OCCAM-THERMAL widens for the run data. Such behavior is likely due to the flexibility provided by the MPC controller over the manually-tuned PID controller.

9.2 Cost, temperature, quality, and timing plot on *HI* across time for histogram. OCCAM-THERMAL does not provide an appreciable improvement over the FIXED policy because histogram is a highly predictable workload with virtually no change in its characteristics across time or between data sets. Differences in the executions of histogram are entirely due to different initial conditions in the system.

9.3 Temperature and cost plot on *HI* across time for stereo_vision. In this benchmark, OCCAM-THERMAL has a minor advantage over FIXED. In the train data, Linux keeps the processors’ temperature too high, thus not meeting the temperature requirement. For the the run data, on the other hand, Linux keeps the processors’ frequencies too low, leading to a worse Timing outcome.

9.4 Temperature and cost plot on *HI* across time for lr. lr demonstrates how OCCAM-THERMAL’s superior knowledge of the system’s requirements and behavior provides better results, both with thermal control as well as by better meeting the timing and quality performance requirements.
9.5 Temperature and cost plot on HI across time for wordcount. Save for a few instances where OCCAM-THERMAL briefly exceeds (and by a small amount) the thermal threshold, OCCAM-THERMAL better meets the system’s performance requirements. The brief instances in the train dataset where the temperature exceeds the thermal threshold is likely due to small thermal modeling errors. In the run data, the MPC-based controllers make a different trade-off between Quality and Timing than the PID controller.

9.6 Temperature and cost plot on HI across time for hearing aid. OCCAM-THERMAL, as it is based on a Model Predictive Controller, can more quickly react to the system’s parameters and adjust the decisions than can the PID controller used in FIXED.

9.7 Temperature and cost plot on HI across time for tachyon. Overall, due to OCCAM-THERMAL’s better model-based knowledge of the application and its requirements, it can make better decisions adapting the application and computer system. This benchmark also provides a good test of the respective controllers’ abilities to adapt to run data that is radically different from the train data.

9.8 Temperature and cost plot on HI across time for seismic. Note that the MPC-based controllers, particularly OCCAM-THERMAL, have a difficult time controlling this application well due to the rapidly-varying stochastic behavior of the seismic application. The Quality values are identical for seismic because seismic does not measure Quality, as seismic does not use the Data Resolution.

9.9 Cost plot on MULTI across time for histogram with the train data.

9.10 Cost plot on MULTI across time for stereo vision with the train data.

9.11 Cost plot on MULTI across time for lr with the train data.

9.12 Cost plot on MULTI across time for wordcount with the train data.

9.13 Cost plot on MULTI across time for hearing aid with the train data.

9.14 Cost plot on MULTI across time for tachyon with the train data.

9.15 Cost plot on MULTI across time for seismic with the train data.
9.16 Horizon length comparison for hearing aid. ........................... 150
9.17 Horizon length comparison for histogram. ............................ 151
9.18 Horizon length comparison for lr. ....................................... 151
9.19 Horizon length comparison for stereo vision. ..................... 152
9.20 Horizon length comparison for seismic. .............................. 152
9.21 Horizon length comparison for tachyon. ............................ 153
9.22 Horizon length comparison for wordcount. ......................... 153
9.23 Decision pruning comparison for stereo vision. ................... 155
9.24 Decision pruning comparison for histogram. ....................... 155
9.25 Decision pruning comparison for hearing aid. .................... 156
9.26 Decision pruning comparison for lr. .................................. 156
9.27 Decision pruning comparison for seismic. ........................... 157
9.28 Decision pruning comparison for tachyon. ........................ 157
9.29 Decision pruning comparison for wordcount. ...................... 158

11.1 Markov Robustness comparison for the histogram benchmark. .. 165
11.2 Markov Robustness comparison for the lr benchmark. ............ 166
11.3 Markov Robustness comparison for the stereo vision benchmark. 167
11.4 Markov Robustness comparison for the wordcount benchmark. 168
11.5 Markov Robustness comparison for the tachyon benchmark. .......... 169
11.6 Markov Robustness comparison for the seismic benchmark. .... 170
11.7 Input Robustness comparison for the histogram benchmark. .... 171
11.8 Input Robustness comparison for the lr benchmark. .............. 172
11.9 Input Robustness comparison for the stereo vision benchmark. .. 173
11.10 Input Robustness comparison for the wordcount benchmark. .... 174
11.11 Input Robustness comparison for the tachyon benchmark. ....... 175
11.12 Input Robustness comparison for the seismic benchmark. ....... 176
B.1 Thermal model comparison for histogram on CPU1 (cool) . . . . . . . . . . . . . . . . . . 198
B.2 Thermal model comparison for histogram on CPU1 (heat) . . . . . . . . . . . . . . . . . . 199
Chapter 1

Introduction

Cyber-Physical Systems (CPS) are highly resource constrained, yet have stringent performance requirements. Compounding the problem is that dynamics also exist that make the system behave in a stochastic manner. Another problem lies with thermal control, where the processor cores’ temperatures are kept below a pre-specified, safe threshold. Maintaining the scalability of this thermal control for larger-scale multicores is a highly challenging problem.

To handle these problems, this thesis presents a software system that combines a design time programming framework with a runtime optimization system. This system, called OCCAM (Optimizing Constrained Concurrent Applications at runtiMe), allows the developer to specify the application’s performance requirements and how to adapt the application while the runtime optimizes the system’s resource usage while the application runs. Control for OCCAM comes from a Model Predictive Controller (MPC) based on a Markov Decision Process (MDP) that helps provide intelligent control of the system based on a model developed offline using data gathered during profiling runs. Thermal scalability to multiple CPU cores is provided by using several different heuristics to reduce the search space needed to find a control policy.

1.1 Thermal Control Example: Driverless Car

Having a holistic control system that optimizes together multiple parameters provides several key advantages. Overall, it is superior to optimize the entire system, and all of the performance requirements and control parameters, as a single unit. Such optimization allows for an improved
decision to be made that successfully takes into account all of the different trade-offs. To illustrate the advantage of a holistic controller that takes into account all of the different performance requirements and control parameters, this section provides a real-world example of the benefits of such a controller.

The example shown here involves a driver-less car, similar to the DARPA Urban Challenge [18], and discusses this driver-less car in the context of the stereo_vision benchmark. As discussed previously, stereo_vision is a benchmark uses two camera inputs to determine the distance to objects in the visual field. stereo_vision trades off the ability to resolve far away objects in exchange for reduced computational complexity. In this example, the speed of the car is limited by how far away the computer vision system can resolve far away objects so that it can maintain a safe reaction time. As a result, it requires more computational power to drive quickly than it does to drive at a slower speed.

Illustrated in Figure 1.1, the driver-less car is driving on a two-lane road when it comes up behind another car in its lane moving relatively slowly. The driver-less car decides that it can safely pass the slow-moving car, so it accelerates and moves into the other lane in order to overtake it. The increased speed of the car requires that the computer speed up, so the processors’ DVFS settings are raised to provide sufficient compute power to the computer vision system. As a result of the higher DVFS settings, the temperature of the processors increases to the point where it exceeds its thermal limits, and the processor speed is accordingly throttled. Because of this new dearth of compute power, the computer vision system cannot provide the needed distance resolution to move at the desired speed. As a result, a fail-safe mechanism trips that causes the car to bail out of the passing maneuver by slamming on the brakes and returning to the right-hand lane.

Had the system been able to take into account the thermal characteristics of the processors when deciding whether to pass, it would have not attempted to pass the car in front of it. As a result, fuel would not have been wasted, wear put on the drivetrain and brake systems, and the passengers not jilted by a sudden braking maneuver. Instead, the driver-less car would have waited until another opportunity arose to pass the car when it would not have tripped the processors’
Figure 1.1: Driverless car example. Since the car’s controller is not aware of the temperature of the processors, it attempts to pass the car in front of it which subsequently overheats the processors, thus requiring the car to bail out of the passing maneuver.
1.2 Problem Statement

Computer systems and the applications that run on them are heavily constrained. Applications have performance requirements such as result quality and real-time throughput constraints. The computer system itself is constrained as well. The computer system suffers from limited resources such as computation time, memory, and interprocessor communication. Moreover, newer computer systems have to contend with additional constraints such as temperature constraints and/or power and energy constraints. In addition to these constraints, the application and computer system must contend with variability caused by dynamics. Ranging from application-level dynamics caused by changing input data to changing environmental conditions, the system must meet all of the aforementioned constraints while contending with variability that can change rapidly, and suffer from large-magnitude, potentially nonlinear changes. Variability in a computer system makes consistently meeting these constraints and requirements difficult. As a result, system designers must over-design systems with large margins of error to ensure that the system can meet its constraints and requirements.

Application adaptation is a useful way to optimize the application’s resource usage so that the aforementioned performance requirements can be met. Adaptation allows for making trade-offs between computational complexity and the quality of the results provided by the application. While application adaptation has been widely studied in the past, previous efforts involving application adaptation were ad-hoc, time-consuming, and highly application-specific with limited portability between computer systems. Portability is going to become increasingly important in the future as many applications will be written once with the expectation that they can run on a large variety of systems, such as smartphones, netbooks, laptop computers, and various embedded computer systems. Another issue not widely studied in the past is application adaptation for parallel applications. Parallel application adaptation should allow applications to run on a variable number of processing elements/contexts, up to and including running on many-core (> 100 cores)
systems. Moreover, the application adaptation used should not harm the parallel scalability of the application. Likewise, application adaptation should scale the memory bandwidth along with the computational requirements to ensure performance scalability on systems that are increasingly constrained by memory bandwidth.

Application and computer system adaptation requires a run-time system that can leverage the application adaptation capabilities of the application and computer system while optimizing the resource usage. This run-time must handle both the application’s execution dynamics as well as the computer system’s stochastic behavior. This run-time should be tractable to generate, and enjoy low run-time overhead. At the same time, keeping processors’ temperatures below a certain threshold is becoming increasingly critical. This is in large part due to the increasing power density of microprocessors, thanks to transistors that are scaling in size much faster than their power consumption is. More-sophisticated cooling solutions are not a complete solution, either: conventional air cooling is limited to about 150 W. Moreover, there are numerous instances, such as in portable and embedded devices, where sophisticated cooling techniques are impractical or impossible.

Controlling the processors’ temperatures so that they stay within their operating limits is a highly challenging control problem in large part due to the large number of cores whose temperatures need to be tracked and controlled. Moreover, it is necessary to develop an accurate per-core thermal model as well as an infinite-horizon control policy because it is necessary for the control policy to take into account the fact that thermal decisions made now affect the system well into the future. Compounding the control problem is that thermal heterogeneity exists between cores in key emerging systems; that is, different processor cores will experience different temperatures even while executing the same code at the same DVFS setting.

Finally, an adaptable application development framework should provide tools for measuring the generated controller’s robustness as well as how well the system can meet the application’s and computer system’s performance requirements. Robustness quantifies how well the control policy can cope with inaccuracies in the controller’s inputs. There should also be a way to measure how well the control policy is anticipated to meet the different performance requirements. Such a tool
allows developers to anticipate how well the system will perform without actually running it on the system.

### 1.2.1 Key Related Work

The work presented in this thesis is not the first work to deal with controlling critical stochastic system dynamics in computer systems. Other, closely-related work fits into two broad categories: projects targeting large datacenters, and projects targeting embedded systems. Datacenters and embedded systems have several key characteristics that make solutions targeting them significantly different. Datacenters are characterized by large numbers of networked computers. They have Quality of Service (QoS) requirements that require that the average performance be within a certain bounds. Moreover, datacenters can have a large variety of different applications running together on them. As a result, a relatively simple reactive controller suffices for adequately controlling the system. Embedded systems, on the other hand, have more stringent real-time performance requirements, as they often have to directly interact with the real world. Moreover, embedded systems are usually much more resource-constrained, having considerably more limited processing power, power/energy budget, and more limited cooling. Embedded systems, while heavily constrained, are considerably easier to statically characterize, as their range of operating environments are known in advance, as well as the applications running on them. As a result, is necessary and desirable to provide a low-overhead, fast-acting, predictive controller. Simultaneously, such a controller is more feasible to produce, as it is possible to characterize the system’s range of performance offline.

Key datacenter work includes the Green [12] and Harmony [90] projects. These works investigate various ways to adapt systems by providing them with different library functions that trade off result quality for CPU utilization. Green also adapts applications by terminating loops early. The primary goal of both projects is to adapt the applications running so that the QoS requirements of all of the applications are met; moreover, Green adds the goal of optimizing power consumption in the datacenter. Finally, the Code Perforation project [6] investigated using automated techniques
to modify loops (and later, command line parameters) to trade off computational complexity for result quality in datacenter-type applications. They use a simple, reactive controller to control the modified applications.

Key embedded computer system work includes the Illinois GRACE Project [5] and the CMU Odyssey project [76]. Both of these projects focuses on mobile devices, were they traded off processing power for network bandwidth. The GRACE project implemented this control using a formal controller that traded off how the aggressiveness of video compression in a video conferencing application with the amount of network bandwidth it used. Finally, the CMU Odyssey project traded off power-hungry local processing for lower-power remote processing that is streamed to the mobile device over a wireless network. These two types of processing are traded off so as to ensure an adequate user experience throughout a variety of different mobile network conditions.

1.2.2 Thesis Statement

A design time adaptation-based programming framework paired with a sophisticated control-based runtime will allow computer systems to cope with application and computer system variability while still meeting the system’s performance requirements.

To forward this thesis statement, this dissertation makes the following contributions:

(1) Developed a design pattern that leverages parallel stream programming techniques to provide a method for writing adaptable applications, called the Data Resolution, that provides a uniform way to develop adaptable applications while not interfering with their parallel scalability.

(2) Developed a design pattern for temporally partitioning real-time work (the Throughput Frame) in a way that functions well with a wide variety of CPS/RMS applications and provides coarse-grained points for specifying when to perform application and computer system adaptation.
(3) Developed a parallel programming framework that implements the Data Resolution and Throughput Frame design patterns.

(4) Ported seven CPS/RMS benchmarks to the OCCAM framework in order to test and demonstrate its efficacy.

(5) Developed a control-based software infrastructure that allows for developing controllers that co-adapt together the application and the computer system.

(6) Developed a Model Predictive Controller (MPC) (called OCCAM-MPC), to control and adapt the entire OCCAM system. OCCAM-MPC is based on a Markov Decision Process (MDP) that uses clustering-based techniques (called phase detection in this thesis), to determine the MDP’s states. Optimal control is assured through the optimal infinite horizon policy search algorithm called Policy Iteration. The resulting policy comprises a robust, safe, table-based controller that enjoys low run-time overhead.

(7) Expanded the MPC to support multicore thermal management. In order to ensure that this revised MPC-based controller (called OCCAM-THERMAL) can scale to large numbers of cores, several different techniques were studied and subsequently employed to improve tractability by reducing the size of the MDP.

1.3 Contributions

This thesis forwards the OCCAM (Optimizing Concurrent Constrained Applications at runtMe) platform as a solution to these problems. OCCAM is a software platform for developing multicore adaptive applications along with a run-time system for controlling them and the computer system as the application executes. OCCAM’s design-time platform consists of Application Programming Interfaces (APIs) and data structures that allow application developers to specify the performance constraints and application-specific optimization techniques. OCCAM’s run-time system is a lightweight userspace library that dynamically manages the application behavior and
optimizes system resource usage for energy and/or power consumption. OCCAM targets a key subset of Cyber-Physical Systems (CPS) applications: Recognition, Mining, and Synthesis (RMS) Applications [21]. Control of the application and computer system is provided using a stochastic Model Predictive Controller (MPC) that employs offline profiling data and control policy generation to provide low-overhead control of the adaptation mechanisms of the system at run time. This controller has been integrated into the run-time system and demonstrates that the OCCAM system improves the system’s success in meeting the application’s and computer system’s constraints while minimizing resource usage.

1.3.1 Cyber-Physical Systems

Lee, et al. [63] asserted that, for future Cyber-Physical Systems (CPS), “Components at any level of abstraction should be made predictable and reliable if this is technologically feasible. If it is not technically feasible, then the next level of abstraction above these components must compensate with robustness.” This quote primarily discusses the need for performance predictability requirements in CPS applications. Performance requirements, e.g., timing and result quality, unfortunately, are not the only requirements faced by CPS. Computer systems, the cyber components of CPS, are often highly resource-limited, with these limitations varying from memory and CPU limitations to power-related energy and thermal requirements. Moreover, many important emerging CPS applications will be parallel, in large part due to their need for high performance implementations.

CPS applications are also highly dynamic in nature. Their run-time resource usage may significantly vary due to changes in the system inputs and in the ambient environment. For instance, in a stereo vision application, higher resolution images are needed when the robot is outside to determine the distance to faraway objects than when the robot is in an enclosed environment such as the inside of a building. To summarize, the stringent system performance requirements, need for parallel processing support, and significant application dynamics make CPS system design and run-time management challenging.
1.3.2 Recognition, Mining, and Synthesis Applications

A key class of CPS application will become increasingly important in the future: Recognition, Mining, and Synthesis (RMS) applications [21]. These applications, which involve processing large quantities of data from the environment, require large amounts of compute power. As a result, they will need to be made parallel so that they can fully utilize future computer systems with multiple-to-many, possibly heterogeneous, processing elements. CPS/RMS applications also have highly stochastic behavior. Their run-time resource usage may vary significantly due to changes in the system inputs and in the ambient environment. For instance, in a stereo vision application in a mobile robot or driver-less car, higher resolution images are needed when the robot is outside (in order to determine the distance to faraway objects), than inside a building. CPS applications are typically interactive, as opposed to batch, applications. Such stringent system performance requirements, need for parallel processing support, and significant application run-time dynamics make CPS system design and effective run-time management highly challenging. In order to study the OCCAM system’s performance, this thesis work implemented and/or modified seven CPS/RMS benchmark applications using OCCAM: three recognition benchmarks (histogram, lr, and stereo_vision); one mining benchmark (wordcount); and three synthesis benchmarks (tachyon, seismic, and hearing_aid).

1.3.3 Application Adaptation

Past work, such as in the Illinois Grace Project [5] Kim, et. al., [57], Kumar, et. al., [59] the CMU Odyssey Project [76], and Shafique, et. al. [82] has attempted to develop techniques to optimize applications in a highly dynamic environment using application adaptation. These past implementations of application adaptation are not well-suited for future compute-intensive, parallel CPS applications for several reasons. First, they lack the systematic design-time and run-time support needed to provide sufficient performance robustness to CPS. In these systems, application optimization was mostly done manually using domain-specific knowledge, which is ad hoc and
time-consuming. Each adaptable application was a one-off design that lacked a common run-time system to systematically direct the application’s online adaptation. CPS need a common runtime in order to provide application portability between different computer systems with different characteristics and performance requirements. Moreover, past work did not provide design-time and run-time support for scalable parallel processing. Parallel processing is becoming increasingly necessary in key emerging CPS applications in order to leverage future high performance computing platforms.

1.4 OCCAM System Overview

The OCCAM platform, shown in Figure 1.2, provides a comprehensive system for implementing and running adaptable applications on parallel systems that comprises design time APIs for developing adaptable parallel applications using a stream-based programming model along with a control-based runtime for adapting and optimizing the application and computer system. Developers use OCCAM’s APIs and data structures to specify how to adapt the parallel application via the Data Resolution as well as how to measure the application’s performance. OCCAM’s runtime then utilizes this information to measure the application’s run-time performance and to
adapt both the application as well as to optimize the computer system (through such features as processor voltage and frequency scaling) to meet the application and computer system’s performance requirements while optimizing resource usage (e.g. power and/or energy consumption).

OCCAM’s run-time framework consists of several key components. Data-parallel adaptation is currently provided using the task-based, multithreaded scheduling components of the Intel Threading Building Blocks (TBB) library. The CPU Aggregate simplifies control of multicore systems by abstracting away control of individual CPU cores into a single, virtual compute unit. Finally, OCCAM provides run-time optimization with a stochastic Model Predictive Controller that uses offline profiling to develop an internal model of the system as a way to adapt the application as well as to optimize the computer system.

1.5 Design Time Programming Framework

Shown in Figure 1.3, OCCAM provides a series of APIs and data structures that allow parallel CPS applications to specify their performance requirements and adaptation techniques so that an underlying run-time system can rigorously adapt both the CPS application and the computer system to meet both the application and computer system requirements. Providing such an abstraction holds the promise of efficiently providing the performance predictability needed by CPS applications while still being able to use existing, low cost, high performance hardware and software systems.

This design time framework allows the run-time system to automatically acquire and comprehend application-specific knowledge, such as application performance requirements and domain-specific control and optimization techniques. Without such knowledge, a run-time system can only adapt applications in limited ways. Developers, on the other hand, can identify and provide application-specific knowledge, such as application throughput and result quality requirements, as well as specific adaptation techniques to trade off computation–communication–memory resource usage and result quality in their applications.

OCCAM is novel in that previous run-time systems did not have the capability to system-
Figure 1.3: Overview of OCCAM’s design time framework and how it functions at run time.
atically interact with developers and leverage application-specific knowledge. Like the design-time-based related work mentioned previously, these run-time platforms do not provide intrinsic support for parallel applications. OCCAM’s design-time framework also encourages developers to specify tiered levels of performance requirements. These differing performance levels allow for gracefully relaxing the requirements in a structured manner that allows the CPS application to operate correctly at a reduced level of performance.

Applications written for OCCAM use a data-parallel, stream-based programming model. Stream programming consists of two key components: a stream and a kernel. A stream is a task comprising a collection of elements that can be processed in parallel, while the kernel comprises the concurrent operations to be performed on each of the stream’s elements. Such a model allows for automatically partitioning and scheduling heterogeneous parallel work for varying numbers of processors at run time.

OCCAM also leverages this programming model to implement the Data Resolution design pattern. The Data Resolution allows OCCAM to trade application quality for reduced computational complexity by processing fewer data elements in each stream. Image and video processing applications, for example, can process fewer pixels for a lower-quality, lower-resolution image. Mining applications, such as search applications, can search fewer data elements in each stream to get a faster but lower quality search. Finally, synthesis applications, such as raytracing applications, can process fewer output pixels to more quickly produce a lower-fidelity image. Implementing the Data Resolution is a special data structure called the Data Pyramid. The Data Pyramid serves to bridge the design time and run-time components of OCCAM. It does this by transparently taking care of scaling up and down the Data Resolution as needed during run time, as well as by measuring the quality of the result data.

OCCAM augments this stream programming model with real-time throughput semantics via an abstraction called the Throughput Frame. The Throughput Frame is a periodically-executing unit of work comprising a collection of streams to be executed together within a unit of time specified by the application’s real-time performance requirements. The Throughput Frame is an
atomic unit of application adaptation: adaptation settings are only set once at the beginning of a Throughput Frame’s execution, and are not changed until the completion of the Throughput Frame. Moreover, a new Throughput Frame is not started until the previous one is complete.

### 1.6 Run-Time System Control

Shown in Figure 1.4, the OCCAM Run-Time System is a control-based system that simultaneously optimizes the application and computer system by predicting future behavior based on past behavior. It treats the system’s behavior as Markovian; that is, future behavior can be predicted using only the most recent state. For the CPS/RMS applications studied, this is a reasonable assumption, since CPS applications involve processing real-world data. Between Throughput Tasks, it is assumed that the input data change will be small enough and/or predictable enough that a controller that only knows about the past data can make a good control decision.

Control of the OCCAM system is provided via a robust (as opposed to adaptive) Model Predictive Controller (MPC), called *OCCAM-MPC*. MPCs use an internal model of the system to generate the control policy. This internal model is based on a Markov Decision Process (MDP), with the MDP’s states being determined using a clustering-based technique called *phase detection*. Using this MDP, along with a cost function based on the system’s performance requirements and resource usage, an infinite horizon optimal control policy is determined. Next, thermal control of the processor cores requires special consideration in order to ensure scalability due to the heterogeneous thermal properties of the processor cores as well as the large number of possible thermal states. In order to determine a control policy, it is therefore necessary to reduce the search space through several different techniques that leverage the unique properties of the processors’ thermal behavior in order to reduce the number of thermal states that need to be considered.

The final result is a robust, stable controller that enjoys low run-time overhead. This low overhead is provided by the fast table lookups of the two-level hierarchy of tables used to determine the control policy. Stability of this controller is ensured by the fact that every possible input maps to a finite set of control decisions within the control tables.
Figure 1.4: Overview of OCCAM’s run-time framework.
1.6.1 MPC Controller

A major impediment to effective control of the system is the stochastic behavior in the system comprising random, white noise variation along with discrete, Markovian state-like behavior. OCCAM uses a Model Predictive Controller (MPC) (referred to as OCCAM-MPC) to efficiently control the system in the midst of this stochastic behavior. Using profiling data, OCCAM-MPC models the system as a Markov Decision Process (MDP) augmented with white noise, where the Markov states represent the large-scale stochastic behavior and the small-scale random behavior is modeled as normally-distributed white noise in the values of each Markov state. Finally, OCCAM uses well-known Markov Decision Process (MDP) techniques to obtain a control policy. OCCAM performs the aforementioned system identification and policy search offline, which allows for using sophisticated optimization techniques without having to be concerned about their effect on run time overhead.

Dynamics seriously impede effective control of the system, as shown in Figures 4.4 and 4.5, and by extension, confound effective identification of the system into a series of discrete states. Dynamics stem from three major causes. The first cause is from system-level interference, such as other processes stealing CPU time; and from non-determinism caused by parallel load balancing. Second, the CPS applications themselves are subject to dynamics caused by changing input data. Finally, measurement error contributes to system-level dynamics. These dynamics make the system difficult to model and control because they make the system’s behavior stochastic and non-linear. As a result, simple feedback controllers, such as Proportional-Integral-Derivative (PID) controllers, are not well-suited for controlling OCCAM applications.

OCCAM handles these dynamics using a technique called phase detection. Phase detection originated in the program analysis community, where tools such as SimPoint [83] employed it to provide representative analyses of program behavior in order to reduce simulation time. OCCAM performs phase detection on two of the three different measurements made by the Data Pyramid: the instruction count and the instruction throughput. quality phases
Figure 1.5: Peak temperature comparison on \textit{MULTI}, a 16-core system.
are handled differently, with each phase comprising an interval between two quality constraints. *OCCAM-MPC* determines the instruction count and instruction throughput phases using *k*-means clustering. *k* is determined by iteratively trying values of *k* in the range of [2, 31] and using a modified version of the Bayesian Information Criterion (BIC) to determine the best value of *k*.

*OCCAM-MPC* searches the decision space based on the probabilistic cost using infinite horizon MDP search techniques such as Policy Iteration and Value Iteration. This cost function can be readily scaled to support optimizing more than one application running together at the same time. *OCCAM-MPC*’s goal is to minimize the total cost of the applications. Application priorities can likewise be implemented by multiplying the applications’ costs by a factor that emphasizes or de-emphasizes each application’s relative priority. The resulting control policy enjoys low overhead at run time due to its implementation as a two-level hierarchy of fast table lookups. Using the current values for the quality, instruction count, instruction throughput, and the past Data Resolution decision, *OCCAM-MPC* performs a two-level hierarchy of fast, low overhead table lookups at run time. The first layer of these table lookups involve converting the continuous quality, instruction count, instruction throughput, and temperature values into their respective phases. The second-layer table lookup involves using the phases and the past Data Resolution decision to determine the set of decisions to implement for the next Throughput Task.

### 1.6.2 Scalable Thermal Control

Scalable thermal control in *OCCAM-MPC* requires two components: an accurate thermal model of the processor cores, and a means to reduce the search space of the thermal model. The thermal model should allow *OCCAM-MPC* to predict the future temperature of a microprocessor core given its current temperature, the DVFS setting chosen, and the time duration of the Throughput Task. This thermal control framework, called *OCCAM-THERMAL*, uses a log-based model, generated using offline profiling, to model the processors’ thermal behavior.

*OCCAM-THERMAL* builds its thermal model using two stages of thermal profiling: one to
determine the thermal phases (these are similar to the instruction count and instruction throughput phases discussed previously; and one stage to determine the heating/cooling model for the system. OCCAM determines the thermal phases for each of the processor cores by randomly varying the Data Resolution and CPU Aggregate settings. Next, it uses this profile data to obtain a heating/cooling model for each core that allows for predicting the future temperature of a core given its current temperature, the DVFS setting used, and the anticipated time duration of the Throughput Task. It produces this model from the profile data by fitting the observed temperature profiling data to a logarithmic model using least squares fitting.

One of the major issues with thermal control is that objects hold heat over a relatively long period of time (i.e., longer than the typical duration of a Throughput Task). As a result, a control decision made now will affect the processor cores’ temperatures well into the future. Speeding up the processors in order to meet a timing deadline can heat up the processors to the point that the next Throughput Task(s) must use the CPU core(s) at a less-than-optimal frequency lest they overheat the CPU(s) at the end of their Throughput Task(s). As a result, it is necessary to generate control policies using infinite horizon optimization techniques for MDPs such as Policy Iteration and Value Iteration. Infinite horizon optimization involves calculating the cost for a given control policy not just for the current state, but also for future states. Thermal variations also exist between cores in current and future systems; that is, different processor cores will experience different temperatures even while executing the same code at the same DVFS settings, as shown in Figure 1.5. Such thermal heterogeneity exists for several reasons. First, different processor cores can experience different cooling rates due to their different placement in the computer’s chassis and from manufacturing variations in cooling components like the heatsink and thermal compound. Another critical emerging cause of thermal heterogeneity is due to process variation [106]. As feature size scaling approaches atomic scale dimensions, within-die variation becomes a large problem, causing different cores in the same die to exhibit different thermal characteristics.

Tractability is the next challenging problem for successful scalable, multicore thermal con-
Tractability is essential for multicore thermal control because each processor core has its own set of thermal states. As a result, the number of combined states, where these states are a vector of all of the processor cores’ thermal states, grows exponentially with the number of cores if the combined states are the Cartesian product of the cores’ possible thermal states. As a result, successfully controlling such a system rapidly becomes intractable with high core counts due to the large number of states that have to be used in order to optimize the whole system. Similarly, non-global DVFS (i.e. where individual cores or groups of cores can have their own separate DVFS settings) further confounds the tractability issue.

Fortunately for thermal control, there are a number of opportunities to eliminate many of these states. The first key opportunity for state-space pruning is to use reachability analysis to eliminate states that cannot be reached during the application’s execution. Another opportunity for decision pruning involves eliminating a large number of the possible DVFS decisions, which not only reduces the number of possible actions, but also the number of possible, reachable next states.

Reachability analysis is a powerful technique for reducing the search space because there are a great many states that can be generated but will never be reached in practice. On many, if not most, systems that support non-global DVFS, full per-core DVFS is not supported. For example, on the 16-core Multi system, DVFS can only be performed at a per-socket level. As a result, the cores within a given DVFS scaling group have thermal phases that correspond to all of the cores running at the same DVFS setting at the same time. This means that if the DVFS scaling group has been running at the highest DVFS setting, the cores will all have similar, high temperatures: there will not be a core that is at its lowest temperature phase while another core is at its highest temperature phase.

Pruning non-optimal control decisions also vastly reduces the search space. Occam-Thermal prunes away non-optimal thermal control decisions in three ways: through thermal pre-scheduling, by eliminating certain useless DVFS configurations, and by only considering the best \( n \) possible decisions. The OCCAM system abstracts away the computer system’s parallel compute resources into a single, virtual compute unit, called the CPU Aggregate. The CPU
Aggregate measures the combined, additive CPU frequencies as a proxy for system performance. By assuming that the CPU cores are at least somewhat homogeneous in their power/thermal characteristics and somewhat homogeneous in their performance characteristics, the best system DVFS configuration is one where the CPU cores are no more than one DVFS step away from each other. Doing so significantly reduces the number of possible DVFS settings that need to be investigated. Another heuristic used by OCCAM is to “thermally pre-schedule” the DVFS settings. Such thermal pre-scheduling involves assigning the highest core DVFS settings to the coolest cores, all the way down the line to the lowest DVFS settings to the hottest cores.

1.7 Structure

The rest of this thesis is structured as follows. Chapter 2 describes the basic OCCAM platform in detail. Chapter 3 describes in detail the seven CPS/RMS benchmarks that were ported to the OCCAM platform. Next, Chapter 4 discusses the MPC controller, and Chapter 6 provides and discusses test results for the controller. Next, Chapter 7 discusses OCCAM-thermal, which is OCCAM-MPC augmented with thermal control, and Chapter 9 presents and discusses the success of this controller. Next, Chapter 8 discusses the safety, optimality, and the settling time of the MPC-based controllers. After that, Chapters 10 and 11 discuss using Monte Carlo-based probabilistic model checking to quantify the robustness of the controllers. Chapter 12 discusses the related work. Finally, Chapter 13 summarizes and concludes this thesis.
Chapter 2

The OCCAM Platform

This chapter provides an overview of the software system that form the basis for the OCCAM system. It starts by providing an overview of the programming model used by OCCAM. This programming model contains two key components: the Data Resolution, which leverages stream programming-based methods to provide a way to trade off computational complexity for result quality; and the Throughput Frame, which is an atomically-executing group of streams that is used to measure the real-time throughput performance of the application. The next part of the OCCAM system is the Data Pyramid. The Data Pyramid is a special data structure that is used to implement the functionality of the Data Resolution within the OCCAM system at run-time. Also discussed are the various APIs OCCAM provides to the application designer to allow him/her to specify the application’s performance requirements, how to measure the application’s output quality, and how to scale up and down the application’s Data Resolution. After describing the aforementioned software components, this chapter describes the control-based run-time system for OCCAM, which adapts the computer system and application in order to meet the system’s performance requirements while minimizing resource usage. Finally, this chapter provides a discussion of how applications are written for OCCAM and provides a step-by-step description of how OCCAM applications are executed at run time.
Figure 2.1: Structural overview of the OCCAM System.
2.1 Overview

As shown in Figure 2.1, the OCCAM platform consists of three major components: the OCCAM APIs, the Data Pyramid, and the OCCAM Run-Time System. The OCCAM APIs provide the interface used by application developers to systematically specify application performance requirements and application-specific control and optimization techniques. The Data Pyramid is a data structure that provides input data to the application and accepts computed result data from the application. The OCCAM Run-Time System optimizes computer system resource usage, i.e., power consumption and multicore load balancing, while satisfying application performance requirements.

Using the OCCAM API, an application designer specifies two types of performance constraints: result quality constraints and real-time throughput constraints. OCCAM requires the designer to provide a function to evaluate whether the result meets the performance constraints, but only requires a time value to be provided to evaluate the system’s real-time throughput. The OCCAM API also provides a structured way to relax the application’s performance constraints by allowing the application designer to specify a series of progressively less-desirable constraints along with providing a cost that progressively penalizes less-strict requirements. Providing structured requirement relaxation allows OCCAM to optimize an application on computer systems where limited computation resources make it impossible to meet the application’s baseline requirements.

The OCCAM Run-Time System is a control-based system. At run time, it rigorously adapts application behavior and optimizes system resource usage to conform to application performance constraints. Besides application-specific knowledge, OCCAM also supports general-purpose runtime adaptation and optimization techniques. General-purpose techniques include both software-based methods, e.g., multicore scheduling; and hardware-based mechanisms, e.g., dynamic voltage and frequency scaling (DVFS), and power- and thermal-aware throttling. Application-specific and general-purpose techniques are leveraged in unison for run-time application and system adaptation, yet managed separately to maximize the reusability of the runtime. Only application-specific
techniques need to be specified and incorporated in order for a Run-Time System to support new applications.

2.2 Computational Model

As shown in Figure 2.2, OCCAM uses a model of computation based on the stream programming model, as shown in Figure 2.2. Stream programming is based on two components: a data stream, and a program kernel. A stream is an array of elements that can be operated on in parallel, while the kernel performs work on each of the elements in the stream. The parallelism provided by this programming model provides an efficient way to partition and schedule an application for parallel execution on multiple processing cores. This programming model has been used successfully in the past by stream computing languages such as Brook [62] and StreamIT [34] to describe a wide variety of algorithms for execution on parallel systems with varying numbers of compute resources. Moreover, stream-based computing has been successfully used to provide scalable parallelism on a variety of other hardware systems besides multicore microprocessors, such as GPUs [17] and the Cell microprocessor [105]. By leveraging the streaming programming model, OCCAM can potentially support these important future architectures in addition to the multicore systems studied in this paper.

OCCAM also modifies the stream programming model by adding in real-time throughput
semantics via an abstraction called the Throughput Frame (frame was chosen due to the definition provided in Mok and Chen [73]). The Throughput Frame is a periodically-executing unit of work containing a collection of streams to be executed together within a unit of time specified by the application’s real-time performance requirements. The Throughput Frame is an atomic unit of application adaptation: adaptation settings are only set once at the beginning of a Throughput Frames execution, and are not changed until the completion of the Throughput Frame; and a new Throughput Frame is not started until the previous one is complete. The Throughput Frame possesses several key characteristics:

1. Streams are grouped into units called Throughput Frames.
2. A new Throughput Frame cannot be started until after the previous Throughput Frame has completed.
3. How well the system is meeting its performance constraints is assessed after each Throughput Frame.
4. The application can only be adapted between Throughput Frames.

Dividing an OCCAM application into a series of parallel, coarse-grained, throughput-oriented frames provides several advantages. First, many CPS/RMS applications map naturally to this task model. For example, in a sequence-of-frames application such as stereo vision, each frame can be mapped to a separate Throughput Frame. Similarly, for a real-time data mining application, collections of data can be mapped to separate tasks. Second, this sequential task model facilitates effective, low-overhead control of OCCAM applications by the run-time. It provides a non-application-specific way to measure the application’s quality and throughput. Moreover, the boundaries between tasks provide coarse-grained points to evaluate and apply control decisions. Finally, this model maps well to execution on auxiliary computational units such as DSPs, GPUs, and reconfigurable logic blocks. A task can be sent to an auxiliary processing unit, executed, and sent back without having to deal with large amounts of complicated synchronization.
2.3 OCCAM API and the Data Pyramid

Applications interface with OCCAM via the OCCAM APIs and the Data Pyramid. The OCCAM APIs provide an interface for specifying an application’s performance constraints, and for specifying how to adapt the application. The Data Pyramid, shown in Figure 2.3, provides an interface between the application and the OCCAM Run-Time System for the input data and the output data.

The designer specifies two performance requirements via the OCCAM API: a result quality constraint and a real-time throughput requirement. OCCAM requires the designer to provide a function to evaluate whether the result produced at the end of a Throughput Frame meets the performance constraints. While general techniques exist, such as interval arithmetic, to quantify the numeric error of a result, they cannot determine what the application deems correct. As a result, the designer must provide a way for OCCAM to determine whether the output meets the performance constraints. Due to OCCAM’s computational model, it is quite simple to specify the real-time throughput requirements: all that needs to be provided is the maximum amount of time allowed for the execution of a single Throughput Frame. The OCCAM API also provides a structured way to relax an application’s performance constraints by allowing the application designer to specify a series of progressively less-desirable constraints, as shown in Figure 2.4. Providing structured constraint relaxation allows OCCAM to optimize an application on computer
Figure 2.4: Tiered Performance Requirements.
systems where limited computation resources make it impossible to meet the application’s baseline constraints.

OCCAM leverages the stream programming model to allow an application to run on a variable number of processor cores as well as to trade off result quality for computation complexity. Each data point within a stream can be processed independently of the other data points, exposing large amounts of parallelism for the OCCAM Run-Time System. Using this programming model, trading off result quality for computation complexity involves processing fewer data elements per Throughput Frame by reducing the Data Resolution. Figure 2.5 depicts the Data Resolution. Reducing the Data Resolution entails representing the information contained in the Throughput Frame’s streams with fewer data points that provide a lower-quality representation of the information contained within the Throughput Frame.

The Data Resolution concept is useful for several reasons. First, many important classes of algorithms can be easily implemented using this design pattern. Many Recognition, Mining, and Synthesis applications, which are an important class of emerging CPS applications, map well to this design pattern. Moreover, many multimedia, Digital Signal Processing (DSP), and image processing algorithms can utilize this concept with little modification: computer vision and image processing algorithms can simply change the resolution of the images being processed, while many DSP algorithms can lower their sampling rates. Similarly, approximate matrix multiplication, as
described in Yale [97], can scale down its Data Resolution by reducing the number of matrix rows and columns computed. Search algorithms can likewise lower their Data Resolutions by processing fewer records. Finally, Chu, et. al. [22] showed that many machine learning algorithms can be expressed as a sum over points. Scaling down the Data Resolution with these algorithms should be possible by reducing the number of data points processed and correspondingly scaling up the result.

OCCAM’s Data Resolution adaptation is important for multicore applications because it is orthogonal to the parallel adaptability of an application. Scaling up or down the Data Resolution only changes the number of data points that need to be processed: the remaining data points can still be partitioned and scheduled to run on multiple cores. Scaling down the Data Resolution also scales down the working set size and the amount of memory bandwidth used by the application, reducing communication overhead and cache misses. Scaling down memory bandwidth is important because memory bandwidth is becoming a major limiting factor within multicore microprocessors for many applications, as discussed in Murphy [74].

Applications interface with OCCAM’s parallel application adaptation framework via a special data structure called the Data Pyramid. The Data Pyramid is conceptually similar to an image pyramid in image processing; like an image pyramid, it can provide a Throughput Frame’s stream data at various resolutions. In all of applications studied by this thesis, the Data Pyramid functions as a lazy data structure that provides the data at the proper Data Resolution dynamically. The Data Pyramid is implemented as an inheritable C++ class. Implementing the Data Pyramid as a C++ class allows a designer to customize the Data Pyramid for an application. To facilitate moving data between the application and the Data Pyramid, OCCAM provides a dataObject class as a generic container. By inheriting from this class, an application can specify what kind of data should be carried within the dataObject. The Data Pyramid provides the following interface to the application:

1. getData(). An application uses the getData() method to obtain the input data
for the current Throughput Frame. This method must be provided by the application
designer, since Data Resolution scaling is highly application-specific. 
\texttt{getData()} takes no arguments, and returns a pointer to a 
\texttt{dataObject}.

(2) \texttt{putData()}. An application uses \texttt{putData()} to output the finished data for a Throughput Frame. The purpose of \texttt{putData()} is to perform any application-specific post-processing of the output data, such as upscaling or filtering. Like \texttt{getData()}, 
\texttt{putData()} must be provided by the application designer because processing the output data is highly application-specific. Calling this method also signals to the OCCAM Run-Time System that the application has finished processing a Throughput Frame. 
\texttt{putData()} takes a \texttt{dataObject} reference as an argument, and returns nothing.

\subsection{OCCAM Run-Time System}

The OCCAM Run-Time System is a control-based system that optimizes the application and computer system based on their past behavior. It optimizes the system’s resource usage by adapting the application and computer system while ensuring that the performance requirements of the system are met. The OCCAM Run-Time System bases its control on the idea that past behavior serves as a good predictor of future behavior. For the CPS applications studied, this is a reasonable assumption, since CPS applications involve processing real-world data. As a result, over the time span of a Throughput Frame, the input data change will be small enough and/or predictable enough that a controller that only knows about the past data can make a good control decision.

OCCAM optimizes the system within the constraints of the application and the computer system by concurrently adapting both the application and the computer system. Partitioning the performance constraints and adaptation techniques into application and computer system groups allows the application to be portable among different computer systems. Optimizing the application on a computer system with new constraints and/or new adaptation parameters only involves
modifying the Run-Time System to support them. Examples of computer system constraints include limited processor resources, energy consumption constraints, and thermal constraints. Examples of ways to adapt the computer system include scheduling work across different numbers of available CPU cores, using the CPUs’ DVFS to trade off lower performance for reduced power consumption, and throttling overheating CPU cores.

The OCCAM Run-Time System leverages efficient control techniques to provide stability, fast response, and a quick settling time. Error is minimized by meeting the performance requirements with minimal resource usage. With OCCAM, a positive error indicates that one or more of the system’s constraints are not being met. Positive errors occur when, for example, the result quality is not high enough or the system is not meeting its real-time throughput constraints because the CPU cores are running at too low of a frequency. A negative error, on the other hand, indicates that the system is not performing at its maximum efficiency due to slack in the system. A negative error, for example, can occur when the result quality is higher than needed or the system meets its real-time throughput requirements faster than needed because the CPU cores are running at too high of a frequency. Such data redundancy and system slack should be minimized in order to optimize the system’s resources. The OCCAM Run-Time System adapts the system in discrete steps, rather than continuously. This design pattern better resembles the way real applications work. For example, application Data Resolution options such as image resolutions or DSP sampling rates map to a set of discrete options, rather than to a continuous spectrum of choices. Likewise, computer systems have discrete numbers of processing cores and DVFS options.

The OCCAM Run-Time System consists of two major parts: the System Controller and the CPU Aggregate. The System Controller is a control-based component that is responsible for controlling the system based on the execution of the previous Throughput Frame. Two types of controllers were studied, as shown in Figure 2.6: the first controller, OCCAM-STOCHASTIC, consists of a cascaded series of Proportional-Integral-Derivative (PID) controllers. The second controller, OCCAM-MPC, is a stochastic Model Predictive Controller (MPC) that uses offline profile data to develop a stochastic, Markov model of the system and uses that model to develop a control policy.
Figure 2.6: Diagram of the **OCCAM-STOCHASTIC** and **OCCAM-MPC** controllers.
The OCCAM Run-Time System adapts the computer system through two major interfaces. Application adaptation occurs through the Data Pyramid’s Data Resolution control interfaces. System adaptation occurs through an abstraction called the CPU Aggregate. Made possible by the multicore application adaptation in OCCAM, the CPU Aggregate abstracts away the multiple cores in the system into a single, virtual compute unit. The execution performance of each Throughput Frame is measured for the entire CPU Aggregate, as opposed to individual CPU cores. Moreover, the controllers control the DVFS settings of the entire CPU Aggregate, as opposed to directly controlling individual CPU cores.

The CPU Aggregate optimizes the application’s multicore characteristics using Intel’s Threading Building Blocks’ (TBB) [25] runtime to partition and schedule the workload across the computer system’s multiple cores. TBB adapts to application heterogeneity by load balancing using a work-stealing algorithm. The Computer System Controller also adapts the computer system using the processor’s DVFS features. The CPU Aggregate implements DVFS control by providing to the System Controller an “aggregate frequency” that is the sum of the frequencies of all of the CPU cores. The CPU Aggregate determines these aggregate frequencies using the procedure outlined in Algorithm 1. In short, the CPU Aggregate determines aggregate frequencies by going across the CPU cores and raising each by one DVFS step. Once all of the CPU cores have been incremented by one DVFS step, then the process starts over with the next DVFS step, until all of the CPU cores are at their maximum DVFS settings.

\[
\begin{align*}
\text{Set all CPU cores to their lowest DVFS setting} \\
\text{CPU Aggregate Frequency} &:= \text{Sum}(\text{CPU core frequencies}) \\
\text{CPU Core Frequencies}[\text{CPU Aggregate Frequency}] &:= \text{CPU core frequencies} \\
\text{Append CPU Aggregate Frequency to CPU Aggregate frequencies} \\
\text{while} \quad \text{Frequency(All CPU Cores)} \neq \text{Max. CPU frequency} \text{ do} \\
\text{Find CPU core with lowest frequency} \\
\text{Increment DVFS Setting of CPU core} \\
\text{Aggregate Frequency} &:= \text{Sum}(\text{CPU core frequencies}) \\
\text{CPU Core Frequencies}[\text{CPU Aggregate Frequency}] &:= \text{CPU core frequencies} \\
\text{Append CPU Aggregate Frequency to CPU Aggregate frequencies} \\
\text{end while}
\end{align*}
\]

\textbf{Algorithm 1: Determining CPU Aggregate Frequencies}
2.3.2 Writing and Executing an OCCAM Application

Writing an application for OCCAM consists of the following steps. First, the developer writes the application in a data-parallel form. Second, the developer provides a `getQuality()` function that describes the quality of the output data. Third, the developer provides `getData()` and `putData()` functions that respectively handle scaling down and scaling up the Data Resolution of the input and output data. Finally, the developer provides a series of application-level constraints with respective constraint costs that allow OCCAM to relax and tighten the application’s constraints as needed.

The execution of OCCAM applications consists of sequentially executing a series of parallel, throughput-oriented tasks. During each task, OCCAM and the application perform the following steps, as shown in Algorithm 3. First, the application requests input data and parameters from the OCCAM Run-Time System. Next, the application prepares one or more streams to be sent to the OCCAM Run-Time System for execution. Next, the parallel run-time partitions and schedules the stream(s) into subtasks for parallel execution on a multicore computer system. Once the execution is complete, the OCCAM Run-Time System evaluates whether the performance constraints of the application and computer system were met. Finally, the OCCAM Run-Time System adapts the application and computer system based on how well the performance constraints were met.

2.3.3 Using OpenCL instead of Intel’s Threading Building Blocks

While the current version of OCCAM employs Intel’s Threading Building Blocks (TBB) as its parallel scheduler and runtime, it should be possible to use other parallel runtimes with OCCAM, in particular OpenCL. OpenCL [38] is an open language maintained by the Khronos Group, the same organization responsible for OpenGL. It is a hardware-independent streaming language designed to allow applications written for it to be run on a variety of different architectures, such as multicore CPUs, GPUs, DSPs, and even the CELL microprocessor.

Replacing TBB with an OpenCL runtime in OCCAM should be fairly straightforward. OC-
$T := \text{Whether another task exists}$

while $T == \text{true}$ do
  $ts := \text{getTime()}$
  $is := \text{getInstructionCount()}$
  $data := \text{getData()}$
  $subtasks := \text{split(data)}$
  for all $subtasks$ do
    Schedule $subtask$ for execution
  end for
  for all $subtasks$ do
    $result := \text{fold}(subtask, result)$
  end for
  $putData(result)$
  $tf := \text{getTime()}$
  $if := \text{getInstructionCount()}$
  $time := tf - ts$
  $instructionCount := if - is$
  $q := \text{getquality()}$
  $\text{controlApplication}(q)$
  $\text{controlSystem}(instructionCount, time)$
end while

Algorithm 2: Execution Process of an OCCAM Application
CAM uses the stream programming model for developing its applications. As a result, its programming model should also be compatible with OpenCL-based applications. Moreover, the coarse-grained nature of the Throughput Frame facilitates the coarse-grained execution approach used by OpenCL, where a unit of work is prepared, sent to be processed, retrieved, and postprocessed.
Chapter 3

Benchmarks

This chapter presents and discusses the benchmarks used to explore OCCAM’s programming framework as well as to test the efficacy of the OCCAM platform. This thesis implemented and/or modified seven RMS benchmark applications using OCCAM: three recognition benchmarks (histogram, lr, and stereo_vision); one mining benchmark (wordcount), and three synthesis benchmarks (tachyon, seismic; and hearing_aid). The rest of this chapter first provides an overview of the seven benchmarks written for OCCAM, and then delves into a detailed description/discussion of each of the benchmarks.

3.1 Overview

OCCAM is designed to facilitate making emerging CPS applications adaptable. This thesis focuses on a subset of CPS applications, the Recognition-Mining-Synthesis (RMS) [21] applications. It focuses on the RMS subset of CPS applications for several reasons. First, RMS applications are highly resource-constrained. RMS applications share in common the need to make sense of large amounts of data, a task that will only become more important as more- and higher-resolution sensors provide Cyber-Physical Systems with increasing amounts of data that need to be processed. Besides being compute power-limited, many of the RMS benchmarks studied are also designed for portable, energy-constrained systems such as mobile robots and driver-less cars. In addition to their increasing importance in future Cyber-Physical Systems, RMS applications have other characteristics which make optimizing their performance important. First, RMS appli-
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>What It Does</th>
<th>Data Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>image histogram</td>
<td>skips pixels</td>
</tr>
<tr>
<td>lr</td>
<td>linear regression</td>
<td>skips data values</td>
</tr>
<tr>
<td>stereo_vision</td>
<td>stereo distance between two images</td>
<td>lowers horizontal resolution</td>
</tr>
<tr>
<td>wordcount</td>
<td>word count</td>
<td>skips blocks of text</td>
</tr>
<tr>
<td>tachyon</td>
<td>3D raytracing</td>
<td>lowers output resolution</td>
</tr>
<tr>
<td>hearing_aid</td>
<td>digital hearing aid</td>
<td>lowers sampling rate</td>
</tr>
<tr>
<td>seismic</td>
<td>seismic simulation</td>
<td>N/A†</td>
</tr>
</tbody>
</table>

†_seismic does not use the Data Resolution._

Table 3.1: List of the benchmarks, what they do, and how they implement the Data Resolution.

Applications are highly computation–communication–storage intensive, leaving them heavily resource-constrained on existing and future computer systems. Moreover, RMS applications are highly parallel in nature, which means that they will take full advantage of current and future high-performance embedded computation platforms. These platforms will include multi- and many-core microprocessors as well as heterogeneous microprocessors with auxiliary computational units such as DSPs, GPGPUs, and/or reconfigurable logic. Finally, RMS applications are an excellent candidate for application adaptation, as they are inherently error-tolerant, which makes it feasible for them to trade off result quality for computational complexity.

Tables 3.1, 3.2, 3.3, and 3.4 describe the key parameters of the seven benchmark applications. The first table, Table 3.1, provides a summary of what the benchmarks do as programs and how they implement the Data Resolution. Next, Table 3.2 lists the number of possible Data Resolution settings for each benchmark as well as the downsampling rate of the input data for each possible Data Resolution setting. Table 3.3 summarizes how the quality of the benchmarks’ results are measured at a conceptual level. Finally, Table 3.4 describes the data/work units processed by each of the applications during a single Throughput Frame.

_histogram_ is derived from the like-named benchmark used in the Phoenix [79] system, an implementation of Google’s MapReduce [26] for multi-core systems. _histogram_ represents a basic machine learning classifier that provides a statistical distribution of pixel colors in an image.
Table 3.2: Available Data Resolution settings for the benchmarks along with the downsampling levels for each.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Num. Data Res. Settings</th>
<th>Data Resolution Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>hearing_aid</td>
<td>5</td>
<td>64, 128, 256, 512, 1024</td>
</tr>
<tr>
<td>histogram</td>
<td>12</td>
<td>2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1</td>
</tr>
<tr>
<td>stereo_vision</td>
<td>6</td>
<td>32, 16, 8, 4, 2, 1</td>
</tr>
<tr>
<td>lr</td>
<td>12</td>
<td>2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1</td>
</tr>
<tr>
<td>seismic†</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>tachyon</td>
<td>7</td>
<td>(8, 8), (8, 4), (4, 4), (4, 2), (2, 2), (2, 1), (1, 1)</td>
</tr>
<tr>
<td>wordcount</td>
<td>12</td>
<td>2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1</td>
</tr>
</tbody>
</table>

† seismic does not use the Data Resolution.
Table 3.3: How the benchmarks measure Quality.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>How Quality is Measured</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>sampling theorem</td>
</tr>
<tr>
<td>lr</td>
<td>sampling theorem</td>
</tr>
<tr>
<td>stereo_vision</td>
<td>% of unresolvable blocks</td>
</tr>
<tr>
<td>wordcount</td>
<td>sampling theorem</td>
</tr>
<tr>
<td>tachyon</td>
<td>output resolution</td>
</tr>
<tr>
<td>hearing_aid</td>
<td>high frequencies missed</td>
</tr>
<tr>
<td>seismic</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**histogram** trades result quality for computational complexity by evaluating fewer data points and then scaling up the output to provide an approximate result. **histogram** provides to OCCAM a result quality function that estimates the result quality using the sampling theorem.

**lr** is a recognition benchmark which also comes from the Phoenix benchmark suite. **lr** represents a widely-used machine learning classifier: linear regression analysis. Like **histogram**, the OCCAM version of **lr** trades result quality for computational complexity by processing fewer data points. **lr** estimates result quality using the sampling theorem.

**stereo_vision** is a recognition benchmark derived from an NVIDIA CUDA application [4]. **stereo_vision** computes the stereo vision distance for a pair of images using a parallel block matching algorithm. It trades off the ability to see far away objects for reduced computational complexity by reducing the horizontal resolution of the image. **stereo_vision** measures the result quality by counting the number of pixels in the result image whose distance cannot be resolved. The input stereo images come from [27].

**wordcount** is a mining benchmark derived from the like-named benchmark in Phoenix. **wordcount** parses a text file and tracks the number of occurrences of words in the text file. It is parallelized by dividing up the text file into smaller pieces that are then independently processed. **wordcount** trades off computational complexity for result quality by only processing parts of the text file and scaling up the resulting count. It estimates the result quality using the sampling theorem.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Composition of Throughput Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>one image</td>
</tr>
<tr>
<td>lr</td>
<td>one text file</td>
</tr>
<tr>
<td>stereo_vision</td>
<td>one set (stereo pair) of frames</td>
</tr>
<tr>
<td>wordcount</td>
<td>one text file</td>
</tr>
<tr>
<td>tachyon</td>
<td>one frame</td>
</tr>
<tr>
<td>hearing_aid</td>
<td>100 sound sample groups</td>
</tr>
<tr>
<td>seismic</td>
<td>one frame</td>
</tr>
</tbody>
</table>

Table 3.4: The benchmarks and what constitutes a Throughput Frame in each of them.

**tachyon** is a synthesis benchmark based on a parallel raytracer application shipped with TBB, which is based off of the **tachyon** benchmark in ALPbench [64]. **tachyon** is parallelized by allowing each horizontal line in the output image to be rendered in parallel. Trading result quality for computational complexity occurs by changing the resolution of the outputted image. **tachyon’s** result quality is a direct function of the current Data Resolution.

**seismic** is a synthesis benchmark which visually simulates seismic waves as they propagate through water and different types of bedrock. **seismic** is parallelized by dividing up the image to be rendered into tiles which can then be concurrently executed. Versus the original TBB example, the OCCAM version of **seismic** renders only tiles with seismic waves above a certain intensity threshold. **seismic** is an example of an OCCAM application that does not use the Data Resolution.

**hearing_aid** is a synthesis benchmark based on the sampling rate-adaptive hearing aid algorithm proposed by William Dieter in [28]. **hearing_aid** simulates a hearing aid application that can reduce its computational complexity by reducing the sampling rate of the sound signal.

### 3.2 stereo_vision

Stereo vision is used to measure the distance between the viewer and an object in the visual field by processing images from two cameras that are a fixed distance apart. It performs this job by measuring the difference in the horizontal position of the object in the two images, outputting
a difference image of the distances. Examples of input images and the resulting difference image are shown in Figure 3.1. With stereo vision, objects with a large positional difference are closer to the viewer than objects with a smaller difference.

For the **stereo_vision** benchmark, a block matching algorithm with a $10 \times 10$ block size is used. To start, the algorithm divides up the images into a series of rows, with each row being the same height as the block (ten pixels for this particular implementation). Then, each unique $10 \times 10$ block in the left image is compared with every unique $10 \times 10$ block in the corresponding row in the right image. The comparisons use a Sum of Squared Differences (SSD) metric, where

$$SSD = \sum_{i=0}^{10 \times 10} (\text{pixel}_{\text{left},i} - \text{pixel}_{\text{right},i})^2.$$  

The block in the right image with the lowest SSD value is chosen to be the “matching” block in the image. This procedure is repeated for all of the other possible blocks in all of the rows of the left image, and the result is a “distance map” of how far away the different objects are from the viewer.

An important step in writing the application for OCCAM is to decompose the algorithm into a series of Throughput Frames and parallel stream sub-tasks. For **stereo_vision**, the Throughput Frames consist of each pair of images. A straightforward way to provide the parallel tasks within each Throughput Frame is to apportion every ten pixel-high row as a separate, parallel unit of computation. Each of these parallel tasks contains one iteration of the **parallel_for()** method. If additional parallelism is desired, these parallel tasks can be further decomposed into parallel subtasks that involve evaluating individual $10 \times 10$ blocks and implementing each sub-task as an iteration of a **parallel_reduce()** loop, and combining the SSD values to find the lowest SSD value via the **join()** method.

After successfully decomposing the stereo vision algorithm into series of Throughput Frames and parallel subtasks, it is necessary to figure out how the algorithm can implement the Data Resolution concept in order to trade off computation for result quality. A useful way to make such a trade-off is to trade off computation for the ability to determine the distance to far away objects. Such a trade-off can be implemented by changing the horizontal resolution of the input image. By reducing the horizontal resolution, fewer $10 \times 10$ blocks need to be compared at the expense of
being unable to resolve the smaller stereo distances between far away objects.

Next, the developer needs to specify the performance requirements as a series of constraints. A good way to think about what the performance requirements should be for stereo_vision is to envision a likely use for computer vision – a mobile robot. No matter how low the Data Resolution is, the stereo_vision algorithm must provide for the safety of the robot by being able to spot nearby objects in a timely manner before the robot, moving at a certain speed, collides with them. If only a low level of application performance is possible, the robot can slow down to allow the computer vision subsystem more time to locate objects. Similarly, slower movement gives the robot more time before it will collide with far away objects. Finally, it is necessary to develop a way to measure the result quality. While making the quality a direct function of the horizontal resolution is a straightforward way to measure the quality, a more flexible way is to measure the percentage of blocks that are too far away to be resolved. Such a measurement accounts for whether the robot is operating indoors, where everything is close by (and the robot has to move relatively slowly), versus in an open area where there are many potentially far away objects (and the robot can move relatively quickly).

The Data Resolution is now ready for implementation via the getData(), putData(), and getQuality() methods. The getData() method will implement scaling down the horizontal resolution with the Data Resolution. Next, the putData() method scales up the horizontal resolution of the output data so that the final output is of a consistent resolution. Finally, the getQuality() method counts the number of blocks that could not be resolved.
\( T := \) Whether another frame exists

While \( T == \) true do

\( ts := \) getTime()
\( is := \) getInstructionCount()
\( data := \) getData()
\( subtasks := \) split(\( data \))

For all subtasks do

Schedule subtask for execution

End for

For all subtasks do

result := fold(subtask, result)

End for

putData(result)

\( tf := \) getTime()
\( if := \) getInstructionCount()
\( time := tf - ts \)
\( instructionCount := if - is \)
\( q := \) getquality()
controlApplication(\( q \))
controlSystem(\( instructionCount, time \))

End while

Algorithm 3: Execution Process of an OCCAM Application
The coupling between the real-time throughput (how quickly the computer vision system processes frames) and the result quality (from how far away the computer vision system can spot objects) suggests that the real-time throughput and quality performance requirements should be relaxed together. It is of little use to be able to spot far away objects if the framerate is so low that the robot reaches the objects before the computer vision system can recognize them. Likewise, it is not very helpful to be able to quickly spot only nearby objects, since the robot has to move slowly to avoid the objects anyway.

Execution of stereo_vision consists of sequentially executing each Throughput Frame while concurrently executing the parallel stream tasks within each Throughput Frame. During each frame, OCCAM and the application perform the steps shown in Algorithm 3. First, the application requests input data and parameters from the OCCAM Run-Time System. Next, the application prepares one or more streams to be sent to the OCCAM Run-Time System for execution. Next, the OCCAM Run-Time System partitions and schedules the stream(s) into subtasks for parallel execution on a multicore computer system. Once the execution completes, the OCCAM Run-Time System evaluates whether the performance requirements of the application and computer system were met and adapts the application and computer system accordingly.

### 3.3 wordcount

wordcount is a mining benchmark derived from the like-named benchmark in Phoenix’s benchmark suite. wordcount parses a text file and tracks the number of occurrences of different words in the text file. wordcount is parallelized by dividing up the input text into progressively smaller components. These subdivided components are split along lexical word boundaries, which ensures that words otherwise straddling boundaries between units of parallel work are counted.

wordcount’s implementation of the Data Resolution entails further subdividing these units of parallel work by \( n \), with \( n \) being the Data Resolution. Therefore, the amount of work done by the benchmark is equal to \( \frac{\text{work}}{n} \). The principle behind this implementation of the Data Resolution is that, in a large text document (since this is an RMS application, it is assumed that wordcount...
is processing a large amount of data) there are a large number of recurrences of each word in the document. Therefore, it is possible to approximate the actual wordcount by processing a fraction $\frac{1}{n}$ of the total data and scaling the resulting wordcount by $n$. As a result, higher values of the Data Resolution entail examining a progressively smaller fraction of the input data.

This benchmark is parallelized by dividing up the text file into smaller pieces that are then independently processed. wordcount trades off computational complexity for result quality by only processing parts of the text file and scaling up the resulting count. Equations 3.1, 3.2, and 3.3 show how the quality is calculated. Each word is hashed into a numeric value before using the SEM calculation to determine the quality.

$$ \text{quality} = \frac{\text{SEM}(\text{values})}{0.5 \times \text{TotalWords}} $$ (3.1)

$$ \text{SEM}(\text{values}) = \sqrt{\frac{\sum_{i=0}^{\text{TotalWords}} \text{values}_i \times [(\text{values}_i - \text{mean(values))}]^2}{\text{TotalWords}}} $$ (3.2)

$$ \text{mean(values)} = \frac{\sum_{i=0}^{\text{TotalWords}} \text{values}_i}{\text{TotalWords}} $$ (3.3)
3.4 tachyon

_tachyon_ is a synthesis benchmark based on a parallel ray tracer application shipped with TBB, which is based off of the _tachyon_ benchmark in ALPBench [64]. _tachyon_ renders a series of three-dimensional raytraced images using an input file that specifies the location, shape, and size of various polygons. Ray tracing is a highly realistic, albeit computationally-intensive, rendering technique that works by casting rays from the viewer’s perspective for each rendered pixel. These rays then bounce off of objects, and if they ultimately reach a light source in the scene, they are made part of the rendering. A ray tracing application was chosen to be part of the benchmark suite because ray tracing is an example of a compute-intensive image rendering application. Photorealistic three dimensional rendering is an important future application that needs large amounts of computation, as virtual worlds, games, and some forms of augmented reality will require complex scenes to be rendered in real time.

Ray tracing provides ample opportunities for parallelism, as each pixel in the result image is independent of the other pixels. The implementation of _tachyon_ used for this study parallelizes the image by giving every \( n^{th} \) thread \( \frac{1}{n} \) lines to render. This division of work is done so that, e.g. Thread 0 gets Line 0, 0 + n, etc., Thread 1 gets Line 1, 1 + n, and so forth. A per-line parallelization scheme was chosen for several different reasons. First, such a scheme mimics many progressive JPEG/GIF image formats. Second, this progressive line rendering allows the renderer to function in an online algorithm-like manner, rendering at least a usable sample of the final image. Finally, parallelizing the work at the line granularity helps to facilitate spatial data locality as well as providing a straightforward progression of data locations that can be prefetched by a simple stream or stride prefetcher. _tachyon_ implements the Data Resolution by reducing the resolution of the image. For image processing, the image resolution is a natural choice for trading off quality for computational complexity. As shown in Table 3.2, there are two “components” \((m, n)\): the downsampling in the horizontal resolution; and the downsampling in the vertical resolution. The downsampling in each resolution is the fraction \( \frac{1}{m} \) of the horizontal lines and \( \frac{1}{n} \) fraction of the
vertical lines. Similarly, the quality is determined as a direct function of the resolution chosen, with \( \text{quality} = \frac{\text{resolution index}}{\text{number resolutions}} \).

Since \textit{tachyon} is intended as a benchmark for rendering of scenes designed for real-time viewing by a person, the constraints were chosen accordingly. The quality constraints are held constant, while the timing values vary from roughly 0.5 to 4 frames/second. The low frame rates were chosen to make the benchmark practical to study on current parallel computer hardware.

There are two distinct input data sets for \textit{tachyon}: the training data set, and the running data set, which are both shown in Figure 3.2. The training data set is designed to provide profiling data to allow OCCAM’s runtime to generate a control policy, while the running data is designed to test the efficacy of the run-time system. The training data for \textit{tachyon} was chosen to be a very simple grayscale collection of stuck-together balls with a simple, uniform lighting scheme. In stark contrast is the running data, which renders a much more complicated set of stuck-together balls. Both of these input data sets change the rendering in that, as time progresses, the camera “zooms” in on the object. This zooming in is designed to simulate a changing scene over time and the according change in rendered scene complexity. These two distinct data sets were deliberately chosen to vary greatly in complexity, and to represent the different scene complexities that may, for example, be seen in a virtual world. Having greatly contrasting scene complexities also allows for testing OCCAM’s ability to control for novel input datasets. Testing OCCAM’s ability to control for novel datasets is important because OCCAM’s control policy is not updated at run time, limiting its theoretical ability to adapt to novel conditions.

3.5 \textit{seismic}

\textit{seismic} is a synthesis benchmark which visually simulates seismic waves as they propagate through water and different types of bedrock. Seismic analysis is widely used in geological studies as well as with oil and gas exploration. The goal of this application is to provide a fast enough rendering of the earthquake wave propagation so that a user can view the rendering in real-time. The left side of Figure 3.3 shows an example frame early in the simulation, where an
underwater explosion has just occurred.

`seismic` is parallelized by dividing up each frame to be rendered into a series of tiles. These tiles can then subsequently be rendered in parallel. The original `seismic` benchmark comes from an example application that is part of the TBB distribution; however, `seismic` for OCCAM is modified in a key way. The OCCAM version of `seismic` adds in selective rendering of the image tiles so that, between frames, `seismic` only re-renders tiles that have seismic waves over a certain magnitude propagating through them. The right side of Figure 3.3 shows the selective tile rendering in action, by displaying only the re-rendered tiles.

This selective optimization has several important implications for the execution behavior of `seismic`. First, it reduces the execution predictability of the application. Before the selective rendering, `seismic` enjoyed consistent execution performance regardless of what the seismic wave looked like, as for every pixel, the same computation was performed. Now, with the selective tile rendering, the execution of `seismic`’s frames is essentially determined by the size of the seismic wave. When the seismic wave is small right after the explosion, relatively fewer tiles are rendered than when the wave has a chance to propagate to structures filling more of the screen. Another important control challenge provided by `seismic` to the OCCAM runtime is that `seismic` does not use the Data Resolution. Instead, OCCAM is limited to only adapting the computer system through the DVFS settings.

There are no quality constraints used by `seismic`, since there is no possible Data Resolution adaptation available for OCCAM to use. The timing constraints are designed to encourage the
OCCAM runtime to provide framerate that gives a relatively fluid rendering of the seismic waves across time.

3.6 hearing_aid

hearing_aid simulates the software component of a digital hearing aid. Digital hearing aids provide a key advantage over conventional hearing aids in that they can be software-configured by an audiologist to precisely compensate for a given person’s unique hearing loss profile. Digital hearing aids can be programmed to selectively amplify specific spectral components of the audio spectrum, or can compress the dynamic range of the audio for people with extensive high frequency hearing loss.

At the heart of most digital hearing aids is a series of digital filters. These filters can be implemented in the time domain as a series of Finite Impulse Response (FIR) filters or in the frequency domain using a Fast Fourier Transform (FFT). Since, in real life, a person’s sound environment is constantly changing, an optimization technique becomes evident: leverage the Nyquist Sampling Theorem to adaptively reduce the sampling rate of the input sound. Since the Nyquist Sampling Theorem states that an analog signal can be completely represented without aliasing by using $2 \times$ highest frequency, the sampling rate used need only be as high as the highest frequency in the sound environment. Such a technique was successfully utilized as a technique to save energy by William Dieter in [28]. Since neither the code (which was written in assembly language for a TI DSP) nor the input data was available, hearing_aid is a clean-room re-engineering of his code.

hearing_aid provides a series of possible sampling rates that can be used: $48KHz$, $24KHz$, $12KHz$, $6KHz$, and $3KHz$. Each collection of input sound samples is split up into a series of frames, each of which contains, for the $48KHz$ sampling rate, 1024 samples. For each of the lower sampling rates, there are correspondingly fewer samples available. In order to determine the appropriate sampling rate, it is necessary to perform spectral analysis of a frame’s frequency content. Such analysis is performed using the xtract_spectrum() function that is
part of $\text{libxtract}$ [1]. The resulting frequency spectrum is then analyzed to find the highest spectral value that is above a threshold of $20\text{db}$, which is roughly the hearing threshold for people with normal hearing. The highest spectral value above the threshold represents the cutoff point where it is possible to safely ignore the upper frequencies and thus reduce the sampling rate.

$\text{hearing\_aid}$ has a set of different FIR filter banks for each of the different sampling rates, since FIR filter coefficients depend on the sampling rate of the input signal. Since the 1024-sample audio frames are processed very quickly, each Throughput Task consists of 100 frames. $\text{hearing\_aid}$ is parallelized by having each thread handle one 1024-sample frame.

### 3.7 histogram

The $\text{histogram}$ benchmark computes an image histogram. An image histogram provides a tonal representation of the image by quantifying the distribution of colors into a histogram-like quantity. Image histograms are frequently computed online, for example, in digital cameras as an aid to the photographer. As part of the OCCAM benchmark suite, $\text{histogram}$ represents a basic machine learning classifier that is used to study the behavior of recognition-type applications. It
is derived from the like-named benchmark used in the Phoenix [79] system, an implementation of Google’s MapReduce [26] for multi-core systems.

The execution for a given input image consists of three stages: setup, execution, and teardown. The setup stage of histogram consists of dividing up the input image into tasks that can be scheduled for TBB’s threads. The execution phase consists of computing the pixel value histograms for the Red, Green, and Blue channels. Each subtask tracks the histogram for its subsection of the image. During the teardown stage, the histogram values are added together to get the final histogram values.

The Data Resolution is implemented in histogram by “skipping” some of the pixels in the image. Lower values of the Data Resolution “skip” progressively more of the pixels, with each of the lower Data Resolutions skipping twice as many pixels. This power-of-two increase in the number of pixels skipped serves two purposes. First, it provides a way to stress OCCAM’s ability to control for nonlinear systems by providing an approximate quadratic relationship between the Instruction Count and the Data Resolution. Second, this Data Resolution specification also provides a wide range of different Instruction Counts, allowing the application to perform well on a wide variety of systems with different processing capabilities. histogram measures the result quality at the end of the execution stage.

$$quality_{final} = \max(quality_{red}, quality_{green}, quality_{blue})$$ (3.4)

$$quality_{channel} = \frac{SEM(values)}{128.0}$$ (3.5)

$$num_{color}(values) = \sum_{i=0}^{255} values_i$$ (3.6)

$$SEM(values) = \sqrt{\frac{\sum_{i=0}^{255} values_i \times [(values_i - mean(values))]^2}{num_{color}(values)}}$$ (3.7)

$$mean(values) = \frac{\sum_{i=0}^{255} values_i}{256.0}$$ (3.8)

The result quality is measured by obtaining the sampling error, which is shown in Equa-
tions 3.4, 3.5, 3.6, 3.7, and 3.8. histogram modifies this sampling error by making the final Result Quality error be the highest sampling error of the three Red, Green, and Blue channels. The sampling error starts with determining the Standard Error of the Mean (SEM). The SEM assumes that the data being sampled is uncorrelated; while the data points are not randomly sampled, the SEM should be sufficient for estimating the sample error of the image histogram. The SEM is determined by the sample standard deviation divided by the square root of the number of samples. The sample standard deviation is the Root-Mean-Square (RMS) value of the residuals, i.e. the difference between a given value and the overall population’s mean. Finally, histogram uses the same training and testing input data. Using the same input data provides a satisfactory way to test histogram because histogram’s execution characteristics do not change with different input images.

3.8 lr

lr is a linear regression benchmark that forms one of the mining benchmarks in OCCAM’s benchmark suite. Linear regression is a form of regression analysis where a predictive linear function is determined for a set of data points. Linear regression is a straightforward, useful machine learning algorithm for characterizing sets of data. lr provides the values shown in Equations 3.9, 3.10, 3.11, 3.12, and 3.13.
The linear regression algorithm used by lr computes the linear regression for a series of data points that are Cartesian coordinates. The input data used for lr is based on the numerical equivalent of the input data used for wordcount; that is, it uses text from a work obtained from Project Gutenberg. Similar to wordcount and histogram, lr uses the Sampling Error to determine the Quality measurement, as shown in Equation 3.14. Finally, lr is partitioned into a series of Throughput Tasks by having each Throughput Task compute a set of input data, i.e. one of the Project Gutenberg text files.

\[
\text{quality} = -\sqrt{\text{abs}\left(\frac{(N \times SXY - SX \times SY) \times (N \times SXY - SX \times SY)}{(N \times SXX - SX^2) \times (N \times SYY - SY^2)}\right)} \quad (3.14)
\]
Chapter 4

Run-Time System

This chapter discusses the Model Predictive Controller used by the OCCAM Run-Time System to control the application and computer system. This Model Predictive Controller uses a model based on a Markov Decision Process (MDP) to generate a controller for the system. OCCAM uses offline profiling and model generation to produce the model. Using a cost function that accounts for the system’s performance requirements and the resource usage, OCCAM generates the final control policy. This control policy uses an optimal infinite-horizon MDP solver, policy iteration, to obtain a final control policy. The resulting controller consists of a two-layer hierarchy of tables: the first layer of tables converts the observations into states; and then the second table uses these state values to determine the next best control decision to make. Since these control decisions are based on a series of fast table lookups, the overhead for the controller is quite low.

This chapter is broken up into several components. Section 4.1 provides an overview of the controller used for OCCAM-MPC. Next, Section 4.2 describes and motivates the offline profiling technique used to characterize the application and computer system. Subsequently, Section 4.3 discusses the system identification process used to turn the profiling information into a model, and discusses in particular the phase detection technique used to convert the system’s behavior into a series of finite states. Next, Section 4.4 describes the actual process of converting the model into a Markov Decision Process (MDP). After that, Section 4.5 discusses how an optimal control policy is determined using a cost function along with a greedy search policy. Finally, Section 4.6 discusses the multithreaded scheduler and runtime.
4.1 Controller Overview

For this thesis, a **model** is defined as per Definition 2, and a **controller** is defined as per Definition 1. OCCAM-MPC’s modeling of the application and computer system consists of two key components. Both components of the model produce a model for the Quality, the Instruction Count, and the Instruction Throughput (these three parameters are defined in Definition 3). The first component is a discrete time Markov chain component, which models the various stochastic parts of the system’s behavior. The discrete time Markov chain models the behavior of the Instruction Count as well as the Instruction Throughput. The next component of OCCAM-MPC’s model is a decision-based model that relates the decisions to how they affect the system’s behavior.

**Definition 1.** The **controller** is defined as a function that, given the current state, provides the next decision: \( f_{ctrl} = f(E_{curr}), f_{ctrl} \in \{\text{Decisions}\}, E_{curr} \in \{\text{States}\} \).

**Definition 2.** A **model** is a function that, given a current state, the current Data Resolution decision, and a next decision, provides a probability distribution of the next states: \( f_{model} = f(E_{curr}, D_{datares_{curr}}, D_{next}), f_{model} = \{(E_{next_0}, P_{E_{next_0}}), \ldots, (E_{next_{n-1}}, P_{E_{next_{n-1}}})\}, n = |\{\text{States}\}|, E_{next_i} \in \{\text{States}\}, \sum_{i=0}^{n-1} P_{E_{next_i}} = 1 \).

**Definition 3.** \( S_{inscnt} \) is the set of all possible Instruction Count states: \( S_{inscnt} = \{\text{Set of Instruction Count States}\} \). \( S_{insthr} \) is the set of all possible Instruction Throughput states: \( S_{insthr} = \{\text{Set of Instruction Throughput States}\} \). \( S_{Qual} \) is the set of all possible Quality states. A Quality state consists of a Quality phase as well as the past Data Resolution decision made: \( S_{Qual} = \{(\text{Data Resolution of } E_{ImmPast}, S_{qual})\} \).

The Markov chain models model the system as if the control decisions never change. The Markov chain is a Discrete Time Markov Chain (DTMC), where each time unit is the duration of one Throughput Frame. Note that the time value is variable from a wall-clock perspective, and is determined by the amount of time the Throughput Task takes to execute. Since the Markov
Figure 4.1: Average Instruction Count with different Data Resolution settings.
chain models model only behavior not related to the Data Resolution and DVFS decisions, these models provide a pure view of the system’s stochastic behavior without any effect from the control decisions. The stochastic behavior that gets captured in this model includes:

(1) Measurement error

(2) Changes in behavior caused by changing input data

(3) Computer system-related stochastic behavior, such as:

(a) Cache effects

(b) Parallelism-related non-determinism (stochastic behavior related to the work stealing-based load balancing, lock contention, etc.)

(c) Processor-based non-determinism (loop buffer on Intel Core 2-based architectures, out-of-order execution, etc.)

(d) Operating system-based non-determinism (interrupts, other processes executing on the system, etc.)

The end result is a Markov chain model that, for every possible Markov state (phase), provides a probability distribution of what the next state after it will be.

OCCAM-MPC’s next component is a series of models that relate how different control decisions affect the behavior of the application and computer system. This model essentially ignores the stochastic behavior of the system by averaging it out among many different observations of the system under a specific control decision. This component contains two (and later, three) different models:

(1) A model that relates the Data Resolution to the Instruction Count.

(2) A model that relates the DVFS settings and the Data Resolution to the Instruction Throughput.
Figure 4.2: Average Instruction Throughput with different Data Resolution settings.
(3) In Chapter 7, the thermal model will be discussed, which relates the processors’ DVFS settings, and the Data Resolution to each processor core’s future temperature.

These first two models are implemented as a lookup table. A lookup table, as opposed to fitting observed data to a polynomial function or using a neural network, was chosen for the models because the control parameters are discrete functions and do not need the continuous resolution of a fitted function. It also potentially improves the accuracy of the model, as there is no need to try to fit varying behaviors to a single function.

Since the Instruction Count is an invariant program parameter (i.e. it does not change with different DVFS settings) that is solely based on the program’s behavior itself, the model only needs to account for changes in the Data Resolution. Also, as shown in Figure 4.1, there is a fairly straightforward empirical relationship between the Data Resolution and the Instruction Count. Modeling the Instruction Throughput, on the other hand, requires accounting for both the DVFS settings of the processors as well as the Data Resolution. The reason for this is because the Instruction Throughput is a measure of the performance of the underlying hardware when running a specific piece of code. It has the benefit of being a specific measure of the applications performance (this is discussed empirically and in more detail in Chapter 5) while also being a measurement of the systems throughput performance. For the most part, as shown in Figure 4.3, the Instruction Throughput depends on the DVFS settings of the processors. As Figure 4.2 shows, however, there is also a dependence of the Instruction Throughput on the Data Resolution. This Data Resolution dependence stems from the fact that the Data Resolution affects the working set of the data, which in turn affects the hit rate of the processors’ caches.

OCCAM-MPC’s model that predicts the Quality has been left out so far. The reason for this omission is because the Quality model is fundamentally different from the other two sets of models in that it is actually a pure Markov Decision Process that predicts the Quality for the next Throughput Frame based on the past Throughput Frame’s Quality, the Data Resolution for that past Throughput Frame, and the current Data Resolution decision. Quality has to be modeled using its
Figure 4.3: Average Instruction Throughput with different DVFS settings.
own, combined MDP because Quality is inherently trickier to model in that it is difficult, if not impossible, to separate out the Quality effects caused by the input data and the Quality effects caused by the Data Resolution in an general, non-application-specific way.

4.2 Offline Profiling

To minimize the overhead of the run-time system, OCCAM-MPC develops the control policy offline, using profile data obtained from measurements available on most computer systems. OCCAM-MPC also exploits the structure of OCCAM applications to reduce the amount of profiling needed. OCCAM-MPC exploits this structure in two ways. First, OCCAM-MPC separates the different factors that affect system behavior into independently-profiled components. Second, OCCAM-MPC simultaneously reduces the amount of profiling while improving the quality of the model by replacing some of the profiling with static program analysis, via a Worst Case Execution Time (WCET) analysis tool.

OCCAM-MPC measures three different values during profiling: the instruction count of each Throughput Task; the overall instruction throughput of each Throughput Task; and the quality of the result data of the Throughput Task. Instruction count provides a portable (nearly all contemporary microprocessors provide a way to measure instruction count), computer system adaptation-invariant measurement of the application’s computational requirements via the PAPI API [30]. Similarly, instruction throughput provides an application adaptation-invariant measure of computer system performance that only requires a timer and the instruction count measurement.

OCCAM-MPC exploits the inherent structure of OCCAM applications in order to reduce the amount of profiling needed. It does this by separating the system’s behavior into dynamics-related behavior and adaptation/optimization-related behavior. OCCAM-MPC then partitions this behavior into two components: application-level, design time adaptation; and computer system-level, run-time adaptation. OCCAM-MPC leverages the three different factors affecting the system’s overall behavior (dynamics, design time adaptation, and run time adaptation) to reduce the amount
of profiling needed by only varying one of the factors during a single profile run.

**OCCAM-MPC** also exploits the design-time application adaptation characteristics of OCCAM applications in order to replace some of the profiling with faster, more accurate static analysis. This static analysis is performed using a Worst Case Execution Time (WCET) analysis tool. Besides reducing the amount of needed profiling, static analysis can potentially produce a better model by eliminating measurement error in the *instruction count*. Statically analyzing code has been extensively studied in Colin and Puaut [24], Gustafsson [36], and Li, et. al. [65] as a way to determine the Worst Case Execution Time (WCET) behavior of an application. These tools take a given program, analyze its longest-path execution behavior, and use a processor model to determine the worst case execution time and/or instruction count for the given microprocessor model. OCCAM uses the Chronos [65] tool set to perform WCET analysis. Chronos uses SimpleScalar-based [8] microprocessor models for its WCET analysis.

OCCAM exploits Chronos’ WCET static analysis capabilities by first profiling the application using the CacheGrind tool in Valgrind [75] to determine the hot code kernel(s) in the application. OCCAM then compiles and analyzes this hot code in the Chronos framework with a basic (no instruction cache, no pipelining) microprocessor model. Then, the instruction count is measured and recorded as the Data Resolution is changed. Since the SimpleScalar PISA instruction set and the microarchitectural models provided by Chronos differ greatly from the x86 microarchitectures studied, **OCCAM-MPC** normalizes the results with the x86 processors’ *instruction count* obtained using a brief profiling run.

Four benchmarks were studied using the WCET-based Data Resolution model: *hearing aid*, *histogram*, *lr*, and *stereo vision*. For *tachyon*, WCET-based analysis was infeasible due to the large variance in the *instruction count* inherent to the different input data sets. *seismic*, on the other hand, does not use a variable Data Resolution, so it would not benefit from WCET analysis. Finally, *wordcount* could not be used with Chronos because it makes heavy use of the C++ STL *map* container, which is not compatible with Chronos.

**OCCAM-MPC** also treats different control parameters as independent of each other. Such
independence reduces the amount of profiling needed for OCCAM-MPC to obtain a complete picture of the system’s behavior. Regarding the control parameters as independent increases the needed profiling only linearly with the number of control parameters, instead of geometrically. While the number of profiles required for two dependent control parameters is tractable, extending the system to more control parameters (e.g. number of CPU cores used, individually controlling CPU cores, etc.) rapidly makes the amount of profiling required infeasible.

4.3 System Identification

Dynamics seriously impede effective control of the system, and by extension, confound effective identification of the system into a series of discrete states. Dynamics stem from three major causes. The first cause is from system-level interference, such as other processes stealing CPU time; and from non-determinism caused by parallel load balancing. Second, the CPS applications themselves are subject to dynamics caused by changing input data. Finally, measurement error contributes to system-level dynamics. These dynamics make the system difficult to model and control because they make the system’s behavior stochastic and non-linear. As a result, simple feedback controllers, such as Proportional-Integral-Derivative (PID) controllers, are not well-suited for controlling OCCAM applications.

An important insight that enables the development of the MPC is the observation that the OCCAM system’s behavior can be described as a discrete-time Markov chain (DTMC). A major problem with describing the system as a DTMC, however, is determining what the Markov chain’s states actually are. OCCAM solves this problem using a technique called phase detection. Phase detection originated in the program analysis community, where it was used by tools such as SimPoint [83] to provide representative analyses of program behavior in order to reduce simulation time.

Figure 4.4 shows how phase detection distills noisy, stochastic program behavior into discrete states analyzable by widely-used Markov chain analysis techniques. The first line in Figure 4.4 depicts the system’s dynamics by plotting the observed instruction throughput across
Figure 4.4: Benefits of using phase analysis to convert residuals into normally-distributed white noise (Instruction Throughput).
Figure 4.5: Benefits of using phase analysis to convert residuals into normally-distributed white noise (Instruction Count).
Throughput Tasks. On the same graph are two histogram plots. One of them plots the residuals for the Throughput Tasks’ instruction throughput values against the mean of the instruction throughput values. The second plot shows the instruction throughput residuals against the detected phases, which demonstrates how phase detection successfully characterizes the system’s large-deviance stochastic behavior. The remaining small-deviance stochastic behavior becomes a narrow Gaussian-like distribution centered around each phase value; as a result, a single standard deviation value is enough to describe it. Quality values do not necessarily have this elegant white-noise behavior. As a result, OCCAM instead represents the variance within phases as an explicit distribution of the different quality values. The distribution contains the probability that a given quality phase’s quality value will be within each of the application’s quality constraint intervals.

OCCAM performs phase detection on two of the three different measurements made by the Data Pyramid: instruction count and instruction throughput. Quality phases are handled differently, with each phase comprising an interval between two quality constraints. OCCAM-MPC determines the instruction count and instruction throughput phases using k-means clustering. k is determined by iteratively trying values of k in the range of [2, 31] (using the formula \( k = \min(BIC(x) \forall x, x \in (0, 32)) \)) and using a modified version of the Bayesian Information Criterion (BIC) (shown in Equation 4.1) to determine the best value of k.

\[
BIC = \sum_{\text{clust}=1}^{m} \sum_{\text{elem}=1}^{n} \frac{(\text{val}_{\text{clust}} - \text{val}_{\text{pt}})^2}{\sigma_{\text{err}}^2} + \text{num}_{\text{clust}} \times \log(\text{num}_{\text{pt}}) 
\]  
(4.1)

4.4 Making a Markov Decision Process

After using phase detection to distill the program’s dynamics into a DTMC with white noise, OCCAM then subsequently converts this DTMC into a Markov Decision Process (MDP). An MDP is a DTMC augmented with decision points at each of its states that affect what the probability distribution of the next state will be.
The most straightforward way to produce an MDP is to perform vast amounts of profiling so that every possible combination of DTMC states and decisions are profiled. Such an approach, however, rapidly becomes intractable for large numbers of states and/or decisions. As a result, it is essential to develop models that provide a mathematical relationship between a given decision and how it affects the system’s state. With OCCAM, the relationships between the frequency and voltage scaling and instruction throughput, and the Data Resolution and instruction count need to be determined. Both relationships take the form of a number that describes how the phase’s value will change with a given decision. Due to the high level of application-specificity in the Data Resolution-quality relationship, this relationship needs to be determined by profiling every possible combination of quality phases and Data Resolution settings. By modeling the effects of the decisions as independent of each other, OCCAM can provide a better quality model with less profiling. These three relationships are then combined with the DTMC to produce an MDP by augmenting each state with the decision made to get to that state. Finally, OCCAM-MPC determines the cost of each state using the models (described in detail later in this section) that relate the DTMC state to the set of decisions made.

4.5 Determining the Optimum Control Policy

Using the MDP described in the previous section, OCCAM determines the optimum control policy. It uses this MDP model in conjunction with a cost function to determine the optimum control policy. The cost function determines the cost of a given state and of the decisions made. The cost function used consists of two parts: a design time, performance requirements-based cost function; and a run-time, resource usage-based cost function. Outputted are four different tables: three tables for converting the observed instruction count, instruction throughput, and quality values into their respective phases; and a table which provides the best application adaptation and system optimization decisions to use for the next Throughput Task.

The resulting control policy enjoys low overhead at run time due to the fact that determining the next control policy only involves performing a series of fast table lookups. Using the current
values for the quality, instruction count, instruction throughput, and the past Data Resolution decision, OCCAM-MPC performs four fast, low-overhead table lookups at run time. Three of these table lookups involve converting the quality, instruction count, and instruction throughput values into their respective phases. The fourth table lookup involves using the three phases and the past Data Resolution decision to determine the set of decisions to implement for the next Throughput Task.

**Definition 4.** An execution $E$ is the chain of states that occur with a given run of an OCCAM application. $E_n$ is the state at Throughput Frame $n$ in the OCCAM Application’s execution. For shorthand, $E_{\text{curr}}$ is the current state, $E_{\text{curr}−1}$ is the state immediately before the current one, $E_{\text{start}}$ is the first state in $E$, $E_{\text{end}}$ is the final state in $E$, and $E_\infty$ is the state infinitely far into the future (this is used for the infinite horizon calculation). A sub-execution $E_{\text{sub}mn}$ is a portion of the execution in the range of $[E_m, E_n]$.

**Definition 5.** An execution space, $E_s$, is a subset of the set of all possible executions: $E_s \subset \{\text{All Possible Executions}\}$. A sub-execution space, $E_{\text{sub}mn}$, is a subset of the set of all possible subexecutions over the range of $[E_m, E_n]$: $E_{\text{sub}mn} \subset \{\text{All Possible Sub-Executions Over}[E_m, E_n]\}$.

**Definition 6.** A state/decision cost, $\text{Cost}_{E\text{state}}$, is the linear combination of the state cost as well as the decision cost: $\text{Cost} = \text{Cost}_{\text{state}} + \text{Cost}_{\text{decision}}$. $\text{Cost}_{\text{state}}$ is the cost of a given state, which is determined by how well this state resides within the system’s various performance requirements constraints: $\text{Cost}_{\text{state}} = \text{Cost}_{\text{time}} + \text{Cost}_{\text{qual}} + \text{Cost}_{\text{temp}}$. The $\text{Cost}_{\text{decision}}$ is the cost of a given decision, and is composed of the Data Resolution cost and the DVFS cost: $\text{Cost}_{\text{decision}} = \text{Cost}_{\text{datares}} + \text{Cost}_{\text{DVFS}}$.

**Definition 7.** An execution cost is the total cost of a given execution: $\text{Cost}_E = \sum_{i=0}^{n} \text{Cost}_{E_i}$, $n = \text{NumThroughputFrames}$. A sub-execution cost is the total cost of a given sub-execution: $\text{Cost}_{\text{sub}E_{mn}} = \sum_{i=m}^{n} \text{Cost}_{E_i}$.
The goal of the controller is to minimize the cost of a given execution (defined in Definition 4) by minimizing its execution cost (see Definition 7), $\min(Cost_E)$. At execution time, the controller performs this by, at every state, making decisions that do one of two things. If the controller uses a greedy control policy, then it attempts to minimize $Cost_E$ by making decisions at every state that minimize $Cost_{E,\text{next}}$, the cost of the next Throughput Frame. If an infinite horizon policy is being used, then the controller attempts to minimize $Cost_E$ by minimizing the sub-execution cost over the interval $[Cost_{\text{next}}, Cost_{\infty})$.

The performance requirements-based cost function is determined by the costs supplied by the application designer for meeting or not meeting the performance requirements. OCCAM treats the list of performance requirements like a series of conjoined interval-based sets, with each interval corresponding to a programmer-specified cost. The resulting cost value is the sum of the quality constraint-related cost and the real-time constraint-related cost.

$$\begin{align*}
Cost_{\text{total}}(DVFS, Datares) &= \sum_{i=1}^{thr} \sum_{j=1}^{cnt} \sum_{k=1}^{qual} \left[ (P_i(thr_i) \times P_j(cnt_j) \times P_k(qual_k) \times Cost_{\text{state}}(thr_i, cnt_j, qual_k) \right] \\
&+ Cost_{\text{dec}}(DVFS, Datares)
\end{align*}$$

(4.2)

Resource usage is the next component of the cost function. This cost function, shown in Equation 4.9, generates a resource usage cost based on the DVFS setting of the CPU Aggregate and the Data Resolution setting of the application. Both the Data Resolution and DVFS settings are considered because they both factor into the system’s energy usage: DVFS settings affect the energy efficiency of the processors, while the Data Resolution setting affects the amount of work the processors must complete before they can enter a low-power sleep state. The DVFS cost is proportional to $DVFS^3$ because reducing the processor’s frequency provides both a linear reduction in power consumption due to the lower frequency as well as a quadratic reduction in power consumption due to the lower supply voltage enabled by the lower frequency.
\texttt{best\_cost} := \texttt{BIGNUM}
\texttt{for all throughput states do}
  \texttt{for all inscnt states do}
    \texttt{for all quality states do}
      \texttt{best\_cost} := \texttt{BIGNUM}
      \texttt{best\_decisions} := \texttt{None}
      \texttt{for all DVFS decisions do}
        \texttt{for all Datares decisions do}
          \texttt{cost} := \texttt{Cost(DVFS, Datares)}
          \texttt{best\_cost[throughput state, inscnt state, quality state]} := \texttt{BIGNUM}
        \texttt{end for}
      \texttt{end for}
    \texttt{end for}
  \texttt{end for}
\texttt{end for}
\texttt{end for}

\textbf{Algorithm 4:} Control Policy Generation
Definition 8. Define greedy policy $\text{Policy}_{\text{greedy}} = \{ \text{Search}_{\text{greedy}}(S) \forall S \in \{\text{States}\} \}$

OCCAM searches for a finite horizon control policy using a horizon length of unity, also known as a greedy policy (see Definition 8); that is, it searches for the solution that makes the next state’s cost the lowest. When merely optimizing for the current performance requirements, a finite horizon solution is optimal in that it is no worse than an infinite horizon one. Starting in Chapter 7, however, thermal control will be explored, which will require that a longer time horizon than just the next Throughput Frame to be taken into account.

\[
\text{Cdf}(\text{bound}, \mu, \sigma) = \text{Cdf}_{\text{norm}}\left(\frac{\text{bound} - \mu}{\sigma}\right) \tag{4.3}
\]

\[
\text{Cost}_{\text{qual}}(\text{qual}) = \sum_{i=1}^{\text{cons}} \left[ \text{Cost}_i \times P_{\text{dist}_i} \right] \tag{4.4}
\]

\[
\text{Cost}_{\text{time}}(\mu_{\text{time}}, \sigma_{\text{time}}) = \sum_{i=1}^{\text{cons}} \left[ \text{Cost}_i \times (\text{Cdf}(h_{i}, \mu_{\text{time}}, \sigma_{\text{time}}) - \text{Cdf}(l_{i}, \mu_{\text{time}}, \sigma_{\text{time}})) \right] \tag{4.5}
\]

\[
\text{Cost}_{\text{state}}(\text{thr}, \text{cnt}, \text{qual}) = \text{Cost}_{\text{time}}(\mu_{\text{time}}, \sigma_{\text{time}}) + \text{Cost}_{\text{qual}}(\text{qual}) \tag{4.6}
\]

Definition 9. The probabilistic cost for a greedy solution is the probabilistic cost of the next Throughput Frame: $\text{Cost}_{\text{prob}_{\text{next}}} = \sum_{i=0}^{n-1} P_{E_{\text{next}_i}} \text{Cost}_{\text{state}}(E_{\text{next}_i}) + \text{Cost}_{\text{decision}}(D)$.

OCCAM-MPC searches the decision space based on the probabilistic cost. This probabilistic cost, shown in Equation 4.2 and defined in Definition 9, is a function of the current state as well as the Data Resolution and DVFS decisions made. Equations 4.3 and 4.5 show how the white noise is accounted for in the timing values (the key values, the mean value ($\mu$), and standard deviation ($\sigma$), are computed using Equations 4.10 and 4.11) using the Cumulative Distribution Function ($\text{cdf}$) of a normal distribution (readily calculated using the error function $\text{erf}(x)$, the mean value
of the phase ($\mu$), and the standard deviation of the phase’s white noise Gaussian distribution ($\sigma$).

Since the quality state uses a discrete distribution instead of a continuous Gaussian distribution, Equation 4.4 calculates a summation over the quality phase’s noise distribution.

\[
\text{Cost}(DVFS) \propto DVFS^3
\]  
\[
\text{Cost}(Datares) \propto Datares
\]

\[
\text{Cost}_{dec}(DVFS, Datares) = \text{Cost}(DVFS)
\]  
\[
+ \text{Cost}(Datares)
\]

This cost function can be readily scaled to support optimizing more than one application running together at the same time. In this case, the system minimizes the total cost of the applications. Application priorities can likewise be implemented by multiplying the applications’ costs by a factor that emphasizes or de-emphasizes the application’s relative priority and/or importance.

\[
\mu_{time}(\mu_{\text{thr}}, \mu_{\text{cnt}}) = \frac{\mu_{\text{thr}}}{\mu_{\text{cnt}}}
\]

\[
\sigma_{time}(\mu_{\text{thr}}, \sigma_{\text{thr}}, \mu_{\text{cnt}}, \sigma_{\text{cnt}}) = \frac{\mu_{\text{thr}}}{\mu_{\text{cnt}}} \times \sqrt{\frac{\sigma_{\text{cnt}}}{\mu_{\text{cnt}}} + \frac{\sigma_{\text{thr}}}{\mu_{\text{thr}}}}
\]

4.6 Run-Time Scheduler

Intel’s Threading Building Blocks (TBB) is a C++-based template library that provides a portable, high-level way to parallelize programs. Its underlying implementation makes extensive use of templates. Templates enjoy low overhead due to the fact that, unlike other forms of polymorphism in C++, templates are resolved statically. While OCCAM’s goal is to target high performance embedded systems, and embedded systems usually avoid C++ due to its high overhead and lack of performance predictability, a C++-based runtime was chosen for the research phase of OCCAM due to its ease of use and ability to quickly make changes to the parallel scheduler.
Future versions of the OCCAM system can be implemented in C if necessary or desirable. TBB was also chosen as the parallel runtime for the current version of OCCAM because it provides a straightforward way to implement the stream-based programming model that forms the foundation of OCCAM’s design-time programming framework. TBB is implemented entirely in C++, and Intel provides all of the source code to TBB as open source software.

A major benefit of the stream-like programming model provided by TBB is that it allows the programmer to specify unconstrained parallelism. Unconstrained parallelism is a model of parallel programming where the program is not limited to running on a certain number of parallel contexts. TBB’s unconstrained parallelism allows its runtime to decide how to divide up the parallel work at run time. Doing so allows a program to be written once and be able to run on a single-thread machine and still successfully scale up to machines with support for large numbers of hardware threads.

Using the unconstrained parallelism provided by TBB’s programming APIs, the TBB runtime is able to partition, schedule, and load balance the work for execution on the parallel hardware. TBB uses a task-based model for execution, where each software thread has its own work queue. If a thread’s queue becomes empty, then it randomly steals work from another thread’s queue. Such a random work-stealing algorithm scales well to multiple processors because it minimizes the amount of inter-thread communication needed to load-balance.

TBB provides a wide variety of different ways for the programmer to express the parallelism inherent in an application. One way in which parallelism can be implemented as doall-style parallelism, where each iteration of a loop can be executed separately. doall-style parallelism is very similar to the stream model of having one or more streams of data and a kernel to process them: each element in the stream corresponds to one loop iteration’s input (and output) data, while the code within the loop comprises the kernel. TBB also allows for other forms of parallelism, such as pipeline parallelism, and even allows the programmer to directly provide parallel tasks to the runtime. Another useful set of features provided by TBB that facilitate the sort of highly productive, agile software development that is good for research is how TBB provides a variety of easy-to-use
mutexes and concurrent data structures. TBB provides various locks that leverage C++’s scoping as well as various concurrent data structures. These concurrent data structures, which are concurrent versions of C++ Standard Template Library (STL) data structures, utilize fine-grained locking in order to allow them to successfully scale up to large thread counts.

OCCAM uses the doall parallel functions parallel_for() and parallel_reduce(). parallel_for() implements a for loop whose iterations are independent of each other, similar to the map() operation in functional programming. parallel_reduce() is similar to parallel_for(), except that it adds in a reduce component that allows for combining the stream’s output results into a single result, such as by accumulating a series of outputs. parallel_reduce() is similar to the map-reduce techniques used by Google’s MapReduce [26] framework and the similar, open source, Apache Hadoop [92] project.

Writing applications that use TBB’s parallel_for() and parallel_reduce() involve writing several key methods:

1. operator () – overloading this operator involves actually implementing the kernel’s functionality.

2. split() – the split() method implements the process of dividing up the streams into smaller components to be executed in parallel. An invocation of split() should divide up the current stream into two roughly equal-sized pieces.

3. join() (parallel_reduce() only) – used to implement the post-execution combining of the results into a single result.
Chapter 5

Case Study of the Benchmarks’ Homogeneity

This chapter analyzes the seven CPS/RMS benchmarks written to evaluate the OCCAM platform so as to study how homogeneous the benchmarks’ executions are. It is broken down into three main components: first, Section 5.1 provides an overview and motivation of the benchmark study. Next, Section 5.2 describes and motivates the measurements used to conduct this case study. Subsequently, Section 5.3 discusses the test setup for this study, including the hardware platform used for the evaluation as well as the settings for the benchmark and the evaluation methodology. Finally, Section 5.4 presents and analyzes the results of this case study and discusses the results for the seven benchmarks.

5.1 Overview

The primary purpose of this study is to characterize the homogeneity of the benchmarks’ execution behavior. This study purports to show two things. First, the study shows that the benchmarks experience highly regular, if not completely homogeneous execution from a low-level microarchitectural perspective. If the low-level execution behavior of the RMS applications demonstrates that the execution behavior is fairly predictable, that is, it exhibits either slow-changing, unchanging, or a regularly-changing pattern of behavior, then it should be fairly easy and straightforward to predict.

Second, the study demonstrates whether the benchmarks’ execution behavior is fairly homogeneous within each Throughput Frame. Homogeneous intra-Throughput Frame behavior provides
several advantages to the performance predictability of the CPS/RMS applications. First, it allows for characterizing how much the applications’ performance varies due to various stochastic factors such as the dynamic execution of the processors, load balancing between the threads, lock contention, etc. Second, the study also helps to characterize whether it is reasonable to characterize the application’s behavior only at Throughput Frame boundaries.

5.2 IPC/LLC Cache Miss Homogeneity

This study examines two key microarchitectural characteristics of the applications: Instructions Per Clock (IPC), and Last Level Cache (LLC) miss rate (note that, for all three of the platforms studied, the LLC is the Level 2 cache). IPC is a useful measure of the throughput of the application for several reasons. First, it provides a measurement of the throughput of the application on a given microprocessor. Using the instruction count of a given application along with the IPC, it is possible to predict how long, in CPU clock cycles, a given workload will take to execute. It also gives some insight into the efficiency of the workload on the given microprocessor: high IPC frequently implies that the application is working efficiently in the sense that the application’s execution is not being held up in any way.

Last Level Cache miss rate is another useful microarchitectural measurement that provides an indirect measure of the applications’ data and data access characteristics. A relatively low LLC miss rate means that the program’s working set of data (i.e. the collection of data the program is currently working with) fits within the last level of cache. Working set size is particularly important for RMS applications, due to their large input data sets. Moreover, working set size is even more critical for RMS applications ported to OCCAM, because the Data Resolution allows for dynamically varying the working set size of the application by the Run-Time System.
Figure 5.1: IPC and Last-Level Cache miss data for the histogram benchmark.

<table>
<thead>
<tr>
<th>System</th>
<th>Processor</th>
<th>DVFS Settings</th>
<th>L2 Cache</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>HI</td>
<td>Core 2 Duo T7300</td>
<td>{0.80, 1.20, 1.60, 2.00} GHz</td>
<td>4 MB‡</td>
<td>2</td>
</tr>
</tbody>
</table>

‡ Shared between two cores.

Table 5.1: Hardware Configuration tested.
Figure 5.2: IPC and Last-Level Cache miss data for the hearing aid benchmark.
Figure 5.3: IPC and Last-Level Cache miss data for the stereo_vision benchmark.
Figure 5.4: IPC and Last-Level Cache miss data for the lr benchmark.
Figure 5.5: IPC and Last-Level Cache miss data for the tachyon benchmark.
Figure 5.6: IPC and Last-Level Cache miss data for the seismic benchmark.
Figure 5.7: IPC and Last-Level Cache miss data for the wordcount benchmark.
5.3 Test Setup

For this experimental study, **HI**, the high-performance laptop platform, was chosen. Its hardware specifications are shown in Table 5.1. The laptop platform was chosen for a few reasons. First, the laptop platform is easy to use: it is reasonably fast, as well as easy to work with. Second, the Core 2 Duo processor in **HI** is a fairly high performance platform, with two processing cores and a high performance memory subsystem. Likewise, the processor cores themselves are fairly high performance as well: the processor has an advanced multi-level cache, speculative execution, sophisticated branch prediction, out-of-order execution, and the ability to issue and retire up to four instructions per cycle. Such a high performance platform provides ample opportunity to study the IPC and LLC performance of the seven OCCAM benchmark applications.

The IPC and LLC miss rate statistics are collected using the **pfmon** tool. **pfmon** is a command line monitoring tool that allows for periodically sampling various performance monitoring counters on the processor via the Perfmon library [2]. Sampling involves collecting a count of an event or events after a certain number of clock cycles have passed. Execution variation is quantified by using several different sampling rates: $10^6$ cycles, $10^7$ cycles, $10^8$ cycles, $10^9$ cycles, and $10^{10}$ cycles. Since the processor in **HI** is running at 2.0GHz for these experiments, these sample times correspond to $5 \times 10^{-4}s$, $5 \times 10^{-3}s$, $5 \times 10^{-2}s$, 0.5s, and 5.0s, respectively.

5.4 Analysis

All seven of the benchmark results express the similar patterns regarding their low-level execution performance. At a very fine-grained level, all of the benchmarks express either homogeneous or regularly-varying changes in the IPC and LLC miss rate. At longer sampling rates, the benchmarks either experience unchanging or slowly-changing execution behavior. All of these behaviors are highly amenable to using well-known techniques, such as Markov chains, to predict the future behavior of the system based on past behavior. The rest of this section provides a detailed discussion of each of the benchmarks’ behavior characteristics.
Shown in Figure 5.1, histogram experiences fairly homogeneous IPC and LLC behavior at all of the sampling granularities except for $10^7$. This behavior is likely due to alternating between the tasks and the scheduler. In spite of the large value of the input data (over $100 MB$), the miss rate for this application is quite low, probably due to aggressive prefetching by the processors’ stream prefachers. Such aggressive prefetching is possible thanks to the highly linear way in which histogram accesses the image’s data.

Shown in Figure 5.2, hearing.aid has fairly homogeneous IPC characteristics, in large part due to the fact that the core computation of hearing.aid changes little across different Data Resolutions. No matter what Data Resolution is selected, the core computation of hearing.aid consists of applying a series of Finite Impulse Response (FIR) filters over a collection of input data.

Shown in Figure 5.3, stereo.vision has widely-varying, but highly regular behavior in the IPC and LLC miss rate. At the finer sampling granularities, the regular microarchitectural behavior patterns are likely due to the block-level and row-of-blocks-level processing in the application. At the larger sampling granularities, the variance in behavior is likely due to alternating between processing the images and subsequently writing out the difference image to disk. Also noteworthy is the regularly-occurring high miss rate in the LLC. This is likely due to the way the image components are processed: each row is intensely processed by the code before moving on to the next row. As a result, a linear prefetcher such as a stream or stride prefetcher would have a highly difficult time anticipating the next row and prefetching it into the cache before the next row gets processed.

Shown in Figure 5.4, lr, being an application that serves mainly to process large amounts of fairly regular data, has similar execution characteristics to histogram. It exhibits fairly regular patterns of IPC and LLC miss rate, with regular spikes in the values most likely due to one task finishing, the scheduler executing, and the next task starting up.

tachyon, shown in Figure 5.5, shows some interesting results due to the fact that the complexity of the Throughput Frames increase as the benchmark progresses in time. This behavior
occurs because the tachyon benchmark renders a complicated object first from a far away perspective while continuously zooming in; as a result, the rendering task starts out fairly easy and becomes progressively more difficult as more detail is required. The increasing complexity of the scene causes a concomitant increase in the IPC as the increased scene complexity increases the amount of work that is amenable to Instruction Level Parallelism-friendly parallel processing.

Seismic, shown in Figure 5.6, is a benchmark that renders a series of frames by splitting up the rendering into a collection of tiles that can subsequently be processed in parallel. Moreover, as an optimization, only tiles that are currently experiencing a seismic wave traveling through them are rendered. The regularly-changing pattern of IPC and LLC misses at the higher sampling rates show how the tile pattern affects the low-level execution characteristics of seismic. The lower sampling rates, on the other hand, show the slower-changing behavior of the application, where progressively more tiles have to be rendered per frame as the seismic waves spread throughout the image. In short, the LLC miss rate and the IPC decrease over time as rendering more tiles causes the working set of seismic to increase.

Shown in Figure 5.7, wordcount, is similar to histogram and lr in that it processes a large amount of input data in a regular manner. Like the former two benchmarks, it exhibits fairly regular patterns of IPC and LLC miss rate that likely corresponds to one task finishing, the scheduler executing, and the next task starting up.
Chapter 6

Non-Thermal Control Results

This chapter presents and analyzes all of the non-thermal results for OCCAM, i.e. the results that do not include thermal control. Section 6.1 first provides an overview of the results. The next sections provide several sets of results. The first set of results, presented in Section 6.2, compares the original OCCAM controller (called OCCAM-STOCHASTIC), a cascaded series of Proportional-Integral-Derivative (PID) controllers, to powernowd, a Linux DVFS daemon, and demonstrates that the application-specific knowledge provided to OCCAM allows it to make better DVFS decisions than the heuristic-based DVFS control of powernowd, while consuming less energy. Next, Section 6.3 compares the cascaded PID controller to a more sophisticated controller based on a Markov Decision Process, called OCCAM-MPC. This controller further improves OCCAM-STOCHASTIC by providing a highly responsive, predictive, table-based controller that significantly improves the controller’s ability to meet the application and computer system’s performance requirements while minimizing resource usage.

The experimental results presented in this chapter demonstrate that OCCAM can successfully optimize system resource usage under application performance requirements across a wide range of computer platforms. The most significant result presented is that OCCAM-MPC provides better control of the system than does OCCAM-STOCHASTIC, with an average of 12.04% lower cost and up to a 99.42% improvement with the tachyon benchmark. This Markov Decision Process-based, Model Predictive Controller achieves better control because it successfully leverages the application-specific knowledge provided through OCCAM’s programming framework.
6.1 Overview

OCCAM is designed to facilitate adapting an important emerging subset of CPS applications: Recognition, Mining, and Synthesis (RMS) [21] applications. OCCAM can successfully optimize system resource usage under application performance constraints across a wide range of computer platforms. Three test platforms are studied: a low-performance, energy-constrained Intel Atom-based single-core mobile platform (referred to as \textit{LO}); a high-performance, dual-core platform (referred to as \textit{HI}); and a 16-core server system (referred to as \textit{MULTI}). Each benchmark is developed using OCCAM, compiled, and executed on these platforms with the support of OCCAM’s runtime.

As OCCAM is designed primarily to facilitate adapting emerging CPS/RMS applications, seven RMS benchmark applications were developed, studied, and tested on OCCAM: three recognition benchmarks (\textit{histogram}, \textit{lr}, and \textit{stereo\_vision}); one mining benchmark (\textit{wordcount}); and three synthesis benchmarks (\textit{tachyon}, \textit{seismic}, and \textit{hearing\_aid}). Three variants of the controller part of the OCCAM Run-Time System are compared and studied. There are also three key test platforms compared. The controller, called \textit{OCCAM-STOCHASTIC}, uses a cascaded series of Proportional-Integral-Derivative (PID) controllers. Shown in Figure 6.1, the \textit{OCCAM-STOCHASTIC} controller consists of a hierarchy of three cascaded Proportional Integral Derivative (PID) controllers: the \textit{Application Controller}, the \textit{Computer System Controller}, and the \textit{Contingency Controller}.

These three cascaded controllers in \textit{OCCAM-STOCHASTIC} optimize the system sequentially: the Application Controller first adapts the application; then, the Computer System Controller optimizes the computer system; and finally, the Contingency Controller optimizes the performance requirements. The Application Controller adapts the Data Resolution for each Throughput Frame so that the quality and performance requirements of the application are just barely met, which optimizes the system by eliminating unnecessary computation. The Computer System Controller, in turn, is responsible for optimizing the computer system so that both the application’s real-time
throughput performance requirements and the computer system’s requirements are efficiently met. This controller performs DVFS optimization by keeping the CPUs’ frequencies and voltages as low as possible while still meeting the application’s real-time performance requirements. If the system cannot meet its performance requirements, then the Contingency Controller relaxes the performance requirements in a structured manner.

Two sets of experiments are used to study these three controllers. The first set of experiments compare OCCAM-STOCHASTIC against a baseline configuration that uses the popular Linux DVFS daemon powernowd [23] to control the processors’ performance and power efficiency. It controls the system’s DVFS decisions based on the CPU utilization. If the CPU utilization goes above 80%, then powernowd raises the CPU frequency to its maximum. If the CPU utilization falls below 20%, then powernowd lowers the CPU’s frequency by one step. Overall, OCCAM improves the power efficiency of the HI system by an average of 24% and the LO system by an average of 38% versus the baseline single-core, powernowd-controlled configuration. OCCAM provides these power savings while imposing a maximum of a 5.79% overhead for seismic.

The second set of experiments compare OCCAM-MPC to OCCAM-STOCHASTIC. OCCAM-STOCHASTIC and OCCAM-MPC provide better control of the system than the baseline PID/powernowd-controlled system. These controllers achieve better control because they can...
<table>
<thead>
<tr>
<th>System</th>
<th>Processor</th>
<th>DVFS Settings</th>
<th>L2 Cache</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO</td>
<td>Atom N270</td>
<td>{0.80, 1.07, 1.33, 1.60} GHz</td>
<td>0.5 MB</td>
<td>2†</td>
</tr>
<tr>
<td>HI</td>
<td>Core 2 Duo T7300</td>
<td>{0.80, 1.20, 1.60, 2.00} GHz</td>
<td>4 MB‡</td>
<td>2</td>
</tr>
<tr>
<td>MULTI</td>
<td>Xeon X7350</td>
<td>{1.60, 1.87, 2.13, 2.40, 2.67, 2.93} GHz</td>
<td>4 MB‡</td>
<td>16</td>
</tr>
</tbody>
</table>

† Logical contexts (SMT). ‡ Shared between two cores.

Table 6.1: Hardware Configurations tested.

successfully leverage the application-specific knowledge provided by the OCCAM API. These controllers, by better understanding the application, can make smarter decisions about the DVFS settings to use through keeping the DVFS settings just high enough to meet the real-time constraints. Moreover, OCCAM-MPC generally controls the system better than OCCAM-STOCHASTIC, due to OCCAM-MPC’s better ability to understand the application. This better understanding stems from the fact that OCCAM-MPC uses its application-specific knowledge to build an internal model of the system’s behavior.

6.2 Test Platform

The three test platforms evaluated consist of a low-performance, energy-constrained, 1.6 GHz Intel Atom-based ASUS Eee PC netbook (referred to as LO); a high-performance, 2.0 GHz Intel dual-core computer platform (referred to as HI); and a 16-core, 2.93 GHz Intel Xeon X7300 system (referred to as MULTI). The Core 2 Duo platform contains two cores whose frequencies must be scaled together, while the Atom processor is a single core with two logical contexts (i.e., it supports Simultaneous Multithreading (SMT)). The OCCAM system, illustrated in Figure 6.2, is directly run in userspace on these platforms using Ubuntu 9.04, a popular Linux distribution with a custom-built kernel that supports accessing the processors’ performance counters. OCCAM is run within the normal desktop environment, which means that OCCAM applications running on these systems are not protected from interfering processes. As a result, another application or daemon can wake up and interfere with the OCCAM system at any time. This interference susceptibility makes these systems excellent platforms for testing OCCAM’s ability to adapt for the various
random dynamics experienced by the system.

Eight test platforms were created by permuting the following three configurations – (1) the hardware platform tested (the high performance system (HI) versus the low performance system (LO)); (2) the method of DVFS control used (either OCCAM or powernowd); and (3) whether multicore adaptation was used (sp – no multicore adaptation used versus mp – multicore adaptation used).

Detailed studies are made of OCCAM’s ability to control the application and computer system at run time by capturing, at every Throughput Frame, the instantaneous power consumption, CPU frequencies, and real-time throughput information. This system information is obtained using various microarchitectural interfaces. Power consumption is obtained using Linux’s ACPI drivers, the CPU frequencies are obtained using Linux’s CPU frequency scaling interface (provided through Linux’s /sys interface), and the real-time throughput is measured by using system timers to measure the amount of time the tasks take to execute. OCCAM’s and powernowd’s DVFS control are compared over time by plotting the instantaneous power consumption and the CPUs’ frequencies as the benchmarks execute. OCCAM’s overhead is measured using the gprof profiling tool. Overhead is measured by the percentage of execution time spent in OCCAM-specific
functions and methods, such as `getData()` and `getQuality()`.

OCCAM’s ability to optimize system resource usage is measured by plotting the per-Throughput Frame cost function-based cost for each Throughput Frame. This cost function is the same one used by `OCCAM-MPC` to determine its control policy. This cost function heavily penalizes missing either the `quality` or real-time throughput constraints, while penalizing DVFS decisions and Data Resolution decisions much less. In other words, the OCCAM controllers are required to meet their constraints first, and then focus on efficiency, which well-represents the needs of Cyber-Physical Systems.

### 6.3 **OCCAM-STOCHASTIC** vs. `powernowd`

Figure 6.3 shows the normalized overall energy consumption for the seven benchmarks running under the eight different configurations. Overall, compared against `powernowd`, OCCAM improves the overall system energy efficiency by an average of 24% (HI) and 38% (LO) over the seven RMS benchmarks. On the HI system, the energy consumption improvement occurs because OCCAM’s multicore adaptation allows for a lower DVFS setting. Even though LO has only a single core microprocessor, it benefits from multicore adaptation due to its SMT support. For the energy consumption experiments, the Contingency Controller was disabled, and as a result, LO...
Figure 6.4: OCCAM vs. powernowd: HI’s power consumption and CPU frequency over time.
Figure 6.5: OCCAM vs. powernowd: success at controlling the application over time.
Figure 6.6: OCCAM vs. powernowd: success at controlling stereo.vision over time on MULTI.

does not meet the application’s real-time throughput constraints. Its energy consumption improves because the higher performance of the multicore version allows the benchmarks to finish faster. In addition, by leveraging application-specific knowledge, OCCAM provides more power- and energy-efficient DVFS control. Since powernowd bases its DVFS decisions on CPU utilization, when a Throughput Frame starts, powernowd raises the CPU frequency to the maximum. OCCAM, on the other hand, knows how much time the Throughput Frame has to execute and keeps the CPU frequency just high enough to complete the Throughput Frame.

Figure 6.4 shows the HI system’s power consumption and the DVFS settings across time for both OCCAM and powernowd. For all of the benchmarks except stereo.vision, OCCAM keeps the CPU frequency better controlled. While OCCAM keeps the CPU frequency mostly within a relatively tight band of one or two intermediate DVFS steps, powernowd widely fluctuates between the highest (2.0GHz) and lowest (0.8GHz) frequencies available on HI. While not shown in Figure 6.4 due to space constraints, LO also benefits from OCCAM’s better DVFS control because OCCAM keeps the CPUs consistently at the processor’s maximum frequency, allowing the benchmark to finish faster than it does with powernowd. Results with stereo.vision on MULTI, as shown in Figure 6.6, shows similar results: due to OCCAM’s application-specific knowledge, OCCAM can keep the CPUs’ frequencies at a much lower average level than does powernowd.

Figure 6.5 compares OCCAM’s versus powernowd’s ability to control the system to meet
the application’s real-time constraints. For all of the applications except seismic, OCCAM achieves a lower error than powernowd. A lower error means that OCCAM better controls the applications to meet the applications’ performance constraints while minimizing the system’s resource usage. OCCAM controls these applications better because it knows what the applications’ performance requirements are, and can make better CPU resource decisions based on this information. powernowd controls seismic better than OCCAM does because the non-linear adaptation characteristics of seismic make it difficult to control with OCCAM-STOCHASTIC’s PID controllers.

OCCAM also responds to changes in the system faster than does powernowd. OCCAM shows this benefit most in histogram and lr. By 0.5s into histogram’s execution, OCCAM has reduced the error to −0.04, while at 5.03s, powernowd’s error is at 4.89. Likewise, OCCAM gets lr’s error down to −0.012 in 1.14s, while powernowd still has an error of 2.39 at 2.57s.

OCCAM’s ability to relax the applications’ constraints allows for better control of the applications and the computer system on the LO platform in Figure 6.5. The Fidelity Index value indicates at what level OCCAM has relaxed the constraints: the value of 0 indicates the baseline constraint, while higher values indicate progressively-more relaxed constraints. For all of the benchmarks, the LO platform does not have enough computation resources to meet the application’s performance constraints, which leads to a high, positive error. By relaxing these constraints, OCCAM provides a 10.2x error reduction in stereo_vision. The stereo_vision benchmark demonstrates how the Contingency Controller tightens the constraints when the system has enough resources to meet those constraints. Between 178s and 398s, the Contingency Controller tightens the constraints because stereo_vision’s compute requirements are lower.

6.4 **OCCAM-MPC vs. OCCAM-STOCHASTIC Results**

This section compares the efficacy of OCCAM-MPC versus OCCAM-STOCHASTIC. Overall, OCCAM-MPC controls the OCCAM system better than does OCCAM-STOCHASTIC. This is a result of OCCAM-MPC being a controller that can customize itself directly to the behavior
Figure 6.7: Comparison of the percentage improvement in total cost function’s value of OCCAM-MPC controller versus the baseline OCCAM-STOCHASTIC controller.
Figure 6.8: Comparison of the PID vs. MPC controllers on HI’s train data.
Figure 6.9: Comparison of the PID vs. MPC controllers on HI’s run data.
Figure 6.10: Comparison of the PID vs. MPC controllers on LO’s train data.
Figure 6.11: Comparison of the PID vs. MPC controllers on LO’s run data.
Figure 6.12: Comparison of the PID vs. MPC controllers on *MULTI*’s train data.
Figure 6.13: Comparison of the PID vs. MPC controllers on *MULTI*’s run data.
of the system for which it is being used, by creating an internal model of the system. **OCCAM-STOCHASTIC**, on the other hand, is a controller designed to control linear systems in a single way. PID controllers can only have three variables customized: the Proportional, Integral, and Derivative variables.

**OCCAM**’s ability to optimize system resource usage is measured by plotting the cost function-based cost for each Throughput Frame. This cost function is the same one used by **OCCAM-MPC** to determine its control policy. This cost function heavily penalizes missing either the quality or real-time throughput requirements, while penalizing resource usage much less. In other words, the OCCAM controllers are required to meet the performance requirements first, and then focus on efficiency, which well-represents the needs of Cyber-Physical Systems.

For **OCCAM-MPC**, the independent-variable controller is usually better because, for the limited amount of profiling done, it can obtain a more accurate picture of the overall results. **OCCAM-MPC** can obtain a better model of the system using the independent variables because the system can account for state/decision combinations that it did not see during profiling. **Hearing aid** is an important exception because its design leads to a dependence between the instruction count and quality variables. This dependence leads to the dependent version of the **OCCAM-MPC** controller providing better control of the system.

For the benchmarks where WCET analysis was performed in lieu of profiling, the WCET analysis does a comparable, if not better, job of controlling the system than does the non-WCET setup. In particular, for **Stereo vision**, the WCET analysis provides a better upper bound for the instruction count than do the empirical measurements made for the non-WCET version of **OCCAM-MPC**. For the most part, however, WCET analysis mainly provides a way to reduce the amount of time spent profiling by not having to fully profile the Data Resolution’s parameters, instead allowing **OCCAM-MPC** to obtain them statically, via program analysis.

Figures 6.13, 6.10, 6.11, and 6.12 show the controller’s test results. Presented for each of the seven benchmarks is a plot across time of the cost incurred for each Throughput Frame. These tests compare the two main controller types tested: **OCCAM-MPC** and **OCCAM-STOCHASTIC**.
The three different test systems, \textit{HI}, \textit{LO}, and \textit{MULTI}, are also compared. Finally, several different variations on the controllers are also presented to compare different control designs.

Figures 6.10 and 6.13 show \textit{HI} running the \textit{train} and \textit{run} data. For all of the benchmarks except seismic and histogram (only one dataset was available for both; also, histogram’s behavior does not change with differing input data), the \textit{run} and \textit{train} data are substantially different. These two runs demonstrate how the different controllers can handle different types of data. Once again, \textit{OCCAM-MPC} does the best, with \textit{OCCAM-STOCHASTIC} ahead of \textit{PID-powernowd}. Of particular interest are the results for \textit{tachyon}, where \textit{OCCAM-MPC} still does a superior job of controlling the system despite being trained with much simpler profiling data.

On \textit{OCCAM-MPC}, several of the benchmark results show “spiky” behavior, where the realized cost sharply increases for a single Throughput Frame. This behavior, in particular for histogram on \textit{HI}, is probably due to some process on the system waking up and consuming CPU cycles, causing the Throughput Frame to finish late. Such behavior shows the limitation of adapting the application and system only on Throughput Frame boundaries. While the current program model does not support adapting applications within Throughput Frames, some forms of computer system adaptation (e.g. DVFS) are still possible. Future work will investigate system adaptation during the execution of a Throughput Frame.

Besides demonstrating OCCAM’s ability to successfully scale to large numbers of cores, the 16-core \textit{MULTI} platform demonstrates the scaling advantages of treating the parameters as independent variables. On \textit{MULTI}, there are 96 different possible CPU Aggregate settings. Since the dependent setup can only “understand” a given DVFS setting if it encountered it during profiling at the given phase, it cannot make as good of decisions as the independent variable setup.

A major advantage of \textit{OCCAM-MPC} over \textit{OCCAM-STOCHASTIC} is that \textit{OCCAM-MPC} generates its control policy completely automatically. \textit{OCCAM-STOCHASTIC}, on the other hand, requires hand-tuning three different controllers. While automated techniques exist for tuning PID controllers, it is likely that optimally (or at least near-optimally) tuning three cascaded PID controllers would be at least as complicated as producing the control policy used in \textit{OCCAM-MPC}. 
Figure 6.7 provides a summary comparison of the better control provided by OCCAM-MPC versus OCCAM GENERIC. Next, Figure 6.10 plots the cost for each Throughput Frame across time on HI to demonstrate the faster settling time and greater responsiveness of OCCAM-MPC’s MDP-based Model Predictive Controller over OCCAM-STOCHASTIC’s series of cascaded PID controllers.

In all of the benchmarks except for seismic, the non-WCET version of OCCAM-MPC controls the OCCAM system better than does OCCAM-STOCHASTIC. This is a result of OCCAM-MPC being a controller that can customize itself to the behavior of the system by creating an internal model of the system. OCCAM-STOCHASTIC, on the other hand, is a controller designed to control mainly linear systems and has relatively few customization parameters. PID controllers, such as the ones used in OCCAM-STOCHASTIC, can only have three variables customized: the Proportional, Integral, and Derivative variables that adjust against an error equal to the difference between the measured value and the performance requirement. seismic’s poor performance is explained by its almost-completely-random behavior; as a result, the feedback control of OCCAM-STOCHASTIC can provide better control than OCCAM-MPC, which performs best when the system has stochastic behavior that is at least somewhat predictable. For most of the benchmarks except for hearing aid on LO, the WCET-based controller performs comparably to the non-WCET one, albeit somewhat worse than the non-WCET-based controller. This is most likely because the WCET analysis provides an overly conservative measurement of the expected instruction count for each Throughput Frame.

Figure 6.7 also compares the efficacy of the controllers when using train and run data. The train data are the inputs used by OCCAM-MPC during offline profiling to obtain a control policy. For all of the benchmarks except seismic and histogram (only one dataset was available for both; also, histogram’s behavior does not change with differing input data), the run and train data are not different. These two different datasets show how well OCCAM-MPC can control the system with novel input data that it did not observe during offline profiling. While OCCAM-MPC still controls the system better than OCCAM-STOCHASTIC (except for seismic, as discussed
before), its advantage over *OCCAM-STOCHASTIC* is significantly smaller. While *OCCAM-MPC* does not handle novel data as well as it does with data it saw during profiling, it is important to realize that it can still efficiently control the system when it is handling input data that it has never experienced before. *tachyon*’s results aptly demonstrate this fact, as the training data for *tachyon* is significantly different: the image to be rendered is much simpler than the one rendered by the run data.

The three test platforms, *HI*, *LO*, and *MULTI*, demonstrate the ability of OCCAM to optimize the benchmarks across different platforms with sharply differing performance characteristics. *HI* functions as the baseline platform, as it is a relatively high performance, mainstream personal computer platform. *LO* serves as a performance- and energy-constrained platform representative of many high end embedded platforms such as smartphones and small mobile robots. *LO* allows for testing how well OCCAM can adapt the applications when compute resources are highly limited. Finally, *MULTI* is used to test how well OCCAM can adapt applications to take full advantage of high performance systems as well as to test OCCAM’s ability to scale to systems with large processor counts.

OCCAM’s runtime controller overhead, measured using *gprof*, is also fairly low as a percentage of the OCCAM runtime system plus the application’s execution time. The highest overheads measured are for *wordcount* at 8.57% and *seismic* at 5.79%. The relatively high overhead for *seismic* stems from the technique used to render only the screen tiles that change. For *wordcount*, the high overhead is due to its very short-duration Throughput Frames, which cause the runtime’s relative overhead to be significantly higher (the absolute overhead of the OCCAM run-time is constant regardless of the execution time of a Throughput Frame). The application with the next highest OCCAM-based overhead is for *hearing_aid*, at 0.08%. This overhead is mainly due to the quality measurement function, which uses a Fast Fourier Transform (FFT) to analyze the frequency content of the audio data. For all of the other benchmarks, the overhead imposed by OCCAM was too low to be measured as anything higher than 0.00%. These overhead results are the same for both the *OCCAM-STOCHASTIC* and the *OCCAM-MPC* controllers, except
for wordcount. For wordcount, OCCAM-MPC puts wordcount at a lower Data Resolution than does OCCAM-STOCHASTIC, causing the Throughput Frames to execute faster. The overhead of wordcount for OCCAM-STOCHASTIC is 0.90%.
Chapter 7

Thermal Control

This chapter discusses providing thermal control to the OCCAM system. As the power density of microprocessors increases, it becomes possible that regular software can cause a conventional microprocessor to exceed its thermal limits. Further compounding the problem is thermal heterogeneity, where some of the processor cores in a multicore system heat up to different temperatures than others. This chapter discusses these issues as well as proposing and discussing the thermal modeling and management techniques used in OCCAM.

The first section, Section 7.1, of this chapter provides an overview of the need for scalable multicore thermal management techniques for OCCAM. Next, Section 7.2 discusses the need to extend the OCCAM controller with support for long/infinite horizon control policy; that is, a controller that takes into account a decisions effect not just for the next Throughput Frame, but also for Throughput Frames far into the future. Section 7.3 then discusses the thermal modeling technique used by OCCAM. Finally, Section 7.4 discusses the need to ensure that the control policy generation remains tractable in the face of a large search space, and proposes several heuristics that specifically leverage the architectural characteristics of the underlying computer system to reduce the search space while minimizing the impact on the controller’s optimality.

7.1 Overview

On current and future microprocessors, thermal control is becoming a serious issue. This problem is emerging due to the increasing power density of microprocessors, induced in large
Figure 7.1: Peak temperature comparison on **HI**.

Figure 7.2: Peak temperature comparison on **MULTI**.
part by transistors that are scaling in size much faster than the chip’s supply voltage ($V_{dd}$). More-sophisticated cooling solutions are not a complete solution, either: conventional air cooling is limited to about 150 W, while more expensive solutions, like phase change cooling, are prohibitively expensive for most normal uses. Moreover, there are numerous instances, such as portable and embedded devices, where sophisticated cooling techniques are impractical or impossible.

It is important to note that power consumption control and thermal control, while strongly interrelated, are not identical, as discussed by Liu, et. al. in [66]. Essentially, the temperature of a processor die is a function of how much heat energy it contains. The amount of heat energy in the processor is dependent on the difference between both the rate of energy being put into it minus the amount of heat energy being removed from it. Since heat flow between two different-temperature objects is proportional to the square of the temperature difference, a hotter die will dissipate heat energy at a faster rate. As a result, optimal task scheduling between relatively cool and relatively hot tasks must take into account the die’s temperature.

These aforementioned issues lead to situations where the processor can electrically run at a speed that is not feasible from a thermal perspective. As a result, it is necessary to control the processor cores through some sort of microarchitectural adaptation so that the processor does not exceed its thermal limit. Compounding the thermal control issue is the fact that different applications heat up the processor to different amounts, due to different applications utilizing the microprocessor in different ways. Figure 7.2 shows how the temperatures vary between different applications.

Thermal variance also exists between cores in key emerging systems; that is, different processor cores will experience different temperatures even while executing the same code at the same DVFS setting. Such thermal heterogeneity exists for several reasons. First, different processors can experience different cooling rates due to different placement in the computer’s chassis and from manufacturing variations in cooling components like the heatsink.

Another critical emerging cause of thermal heterogeneity is due to process variation. As feature size scaling leads to increasingly small transistors, it becomes more difficult to control
manufacturing variation. There are two key different types of process variation: die-to-die variation, and within-die variation. For thermal control, die-to-die variation is not a major concern, as that form of manufacturing variation is successfully handled using frequency binning. Within-die variation, on the other hand, is a major concern, as it causes different cores in the same die to exhibit different thermal characteristics.

Thermal control issues have been extensively studied in the past. Past thermal control work can be divided up into several different categories based on the different techniques used. The first category is the hardware control method used to control the processor core’s temperature. One set of techniques involves altering the microarchitecture of the processor core in order to reduce its power consumption, and thus, its temperature. Such techniques include fetch toggling/gating, where the processor reduces and/or stops fetching instructions; cache sub-banking, where only part of a cache line is accessed in order to reduce cache access power at the expense of increased cache latency; and disabling out-of-order execution. These techniques all have the advantage that they can be activated and deactivated very quickly, allowing for very precise, fine-grained control of the thermal properties of the system.

The next category of hardware thermal control involves using clock gating. Clock gating is a power conservation logic design technique where dynamic power is saved by shutting off the clock signal to parts of the chip. Similar to the microarchitectural hardware adaptation techniques discussed previously, clock gating can be toggled very quickly. Clock gating is used for thermal control on contemporary microprocessors such as the Intel Pentium 4.

Dynamic Voltage and Frequency Scaling (DVFS) provides the final major technique for controlling processor temperature. DVFS adjusts the frequency of the CPU, and leverages lower CPU frequencies to reduce the supply voltage $V_{dd}$ as well. This combination allows for a cubic reduction in the power in exchange for a linear reduction in the frequency. As a result, DVFS has the potential to provide very effective, efficient thermal control. DVFS has several disadvantages relative to the aforementioned techniques, however. First, DVFS frequency changes take a long time, ranging from 10$\mu$s to upwards of 10ms. These slow reaction times result from the need for the
clock-generating PLL to lock as well as for the power supply circuitry to adjust $V_{dd}$. Another issue with DVFS is its limited granularity. For most commercial microprocessors, DVFS adaptation is limited to a few different discrete frequency/voltage settings. Similarly, few microarchitectures allow for per-core DVFS; the DVFS settings are frequently limited to either controlling the whole chip or groups of cores.

A similar, complementary technique to DVFS is Adaptive Body Biasing (ABB). In short, ABB is a technique to dynamically vary the threshold voltage $V_t$ by changing the transistor’s body potential. Changing $V_t$ allows for trading transistor performance for reduced subthreshold leakage. This technique requires special hardware support to allow for changing the transistor’s body bias, and can be combined with DVFS to help reduce power in high leakage, advanced deep-submicron microprocessors.

The thermal control studied in this thesis focuses on DVFS for two key reasons. First, DVFS provides the largest opportunity for thermal control due to its cubic power reduction with frequency. Second, DVFS, due to its limited, discrete control options as well as its high transition latency, is particularly challenging to effectively utilize for thermal control. Due to its high transition latency, it is not well-suited for reactively throttling a CPU core when it crosses a certain temperature threshold, unlike the microarchitectural adaptations discussed previously. Another reason for focusing on DVFS is that DVFS-based thermal control is readily available in most current microprocessors, ranging from low-cost embedded microprocessors all the way up to high performance server microprocessors. Such ready availability allows for testing OCCAM’s thermal control on real microprocessors. Note that while the control system studied here only explored DVFS, the offline profiling and MDP-based control techniques discussed here can be trivially extended to these other thermal control techniques as well by making the different microarchitectural adaptation techniques be additional control parameters in the MDP.

Another major advantage of OCCAM’s prediction-based DVFS control is that it allows for using powerful, but high-latency thermal management techniques such as DVFS to help prevent hotspots on the chip. While the entire core’s temperature may change only relatively slowly, certain
small areas of the processor, such as the register file and functional units, can suffer very rapid changes in temperature. Such rapid temperature changes make using high-latency techniques such as DVFS difficult in a reactive thermal control system (past work in reactive thermal hotspot control focuses on fast-acting microarchitectural adaptations due to this very issue). Due to the lack of fine-grained temperature sensing in current microprocessors, OCCAM did not study how well it controls thermal hotspots; such control would be a straightforward extension of the current thermal control and would make an excellent avenue of future work.

Discussed in the next three sections are how OCCAM’s offline profiling-based MDP controller is extended to support multicore thermal control. Such extension involves three key sub-problems. The first problem is how to generate a thermal model for the system. Presented in Section 7.3 will be an accurate, per-core, profiling-based log model of the thermal behavior of the processor cores as a function of the time and the DVFS setting. The next sub-problem involves tractability. Tracking the temperature of a large number of cores along with providing optimal DVFS settings for them creates a large search space that needs to be somehow reduced in order to make the control problem solvable on today’s computer systems. Section 7.4 will discuss several techniques and heuristics for reducing the number of system states and control decisions that need to be explored.

7.2 Infinite Horizon Optimization

One of the major issues with thermal control is that objects hold heat over a relatively long period of time (i.e., longer than the typical duration of a Throughput Task). As a result, a control decision made now will affect the processor cores’ temperatures well into the future. Speeding up the processors in order to meet a timing deadline can heat up the processors to the point that the next Throughput Task(s) must use the CPU cores at a significantly lower frequency, lest they overheat the CPUs. As a result, it is necessary to generate control policies using infinite horizon optimization, as defined in Definition 10. Infinite horizon optimization involves calculating the cost for a given control policy not just for the current state, but also for all future states (note that
to ensure that the infinite horizon policy converges, there is typically some sort of discount factor that reduces the importance of events occurring well into the future).

**Definition 10.** An *infinite horizon* solution is a solution that accounts for all of the Throughput Frames from the next Throughput Frame all the way to infinity.

Long and infinite horizon policy search (defined in Definition 11) has been extensively studied for MDPs, and a large number of efficient solver techniques have been developed, such as Value Iteration, Policy Iteration, Q-Learning, and formulating the MDP as a linear program. Likewise, there are a wide variety of tools and techniques for providing long/infinite horizon solutions for MDPs. All of these techniques involve determining an infinite horizon search policy using the **discounted probabilistic cost** (defined in Definition 12), where states farther out into the future are counted less than ones in the near future. Discounting the future is essential in order for these algorithms to successfully converge.

**Definition 11.** Define **infinite horizon policy** $Policy_\infty = \{Search_\infty(S) \forall S \in \{States\}\}$. The **infinite horizon policy search** is: $Search_\infty(E_{curr}) = \min(Cost_{prob_\infty}(f_{model}(E_{curr}, D_{datares_{curr}}, D)), \forall D \in \{Decisions\})$

For this study, OCCAM uses INRA’s MDP Toolbox [19], which is a mature MDP solver toolkit for MATLAB. INRA’s MDP Toolbox provides a variety of different solver techniques for MDPs, and takes as its input two cell arrays (a cell array is a generic container array in MATLAB) whose indices correspond to different possible decisions. Within each of the first cell array’s container entries are sparse matrices that describe the next state probabilities for the array entry’s corresponding decision. These $n_{states} \times n_{states}$ matrices are structured so that the rows correspond to possible current states, the columns represent the next states, and the values in the matrices represent the probability, given the current state, that it will transition to that particular next state. For both the actions and states, the tuple containing the different state values is mapped to an integer value. For the other cell array, each cell array entry contains a $n_{states} \times n_{states}$ matrix representing the cost of being in the current state and transitioning to a particular next state.
Definition 12. The discounted probabilistic cost for an infinite horizon solution is the probabilistic cost out to infinity multiplied by a discount factor $\gamma$: 

$$Cost_{prob\infty} = \sum_{i=0}^{\infty} \gamma^i Cost_{prob\,i}, \gamma = (0, 1)$$

7.3 Thermal Modeling

In order to predict the temperature of a given microprocessor core, it is necessary to have a model that allows for predicting the temperature of a given microprocessor core. This model should allow one to predict the future temperature of a microprocessor core given its current temperature, the DVFS setting chosen, and the time duration.

An extremely useful simplifying assumption for the thermal model is to ignore the lateral heat transfer between cores on a given processor die. This is a valid assumption for two reasons: first, silicon is a far worse conductor of heat than the copper and/or aluminum heat sink used to cool the processor die; and second, the lateral area of the silicon die is very small, particularly when thinned wafers are used to improve the heat transfer from the die to the heatsink. These assumptions have also been used in a wide variety of previous work on thermal control of microprocessors.

OCCAM’s thermal control treats each microprocessor core in isolation; that is, the temperature of each microprocessor core is modeled separate from the other microprocessor cores. This both a useful assumption in that it vastly simplifies the thermal modeling, and it is a valid model due to the aforementioned limited lateral heat transfer within a processor die. OCCAM’s thermal control uses two stages of thermal profiling: one to determine the thermal phases (these are similar to the instruction count and instruction throughput phases discussed previously; and one stage to determine the heating/cooling model for the system. Next, it obtains a heating/cooling model for each core that allows for predicting the future temperature of a core given its current temperature, the DVFS setting used, and the time duration. It accomplishes this by fitting the observed temperature profiling data to a log model using the least squares technique.
7.3.1 Offline Profiling

OCCAM uses offline profiling to determine a per-core thermal model for the system. This offline profiling can be split up into two parts. In the first part, the system’s Data Resolution and DVFS settings are randomly varied in order to determine what kind of temperature phases the system will see during execution; and a heating/cooling profiling run where the processor cores are heated/cooled to a baseline starting temperature, and then the temperature is periodically measured while the benchmark is run at a single DVFS setting.

The first profiling run is simply a run where the Data Resolution and DVFS settings are randomly varied as the benchmark executes, and the temperature values are recorded. The purpose of this profiling run is to determine what kind of “real-world” core temperatures the system is likely to encounter when the benchmark is running. This “real world” data is useful for two reasons: first, it allows the phase detection algorithm to determine realistic temperature phases for the benchmark and system; and second, it paints a realistic picture of the combinations of thermal phases the system is likely to encounter. This second feature’s utility will be discussed later in Section 7.4, which discusses the techniques used to reduce the complexity of the thermal model without significantly compromising its accuracy.

The second profiling run is used to determine the heating and cooling rate of the processor cores for a given current temperature and DVFS setting. There are two sets of profiling runs for these values. The first set of runs is a “cool” run where the processor die is first heated up to its maximum possible temperature by doing a benchmark run at the cores’ maximum DVFS setting, and then another benchmark run is conducted at one of the DVFS settings. This test is repeated for all of the possible DVFS settings. The second set of runs, the “heat” runs, do the opposite of the “cool” runs: the processor cores are “cooled” to a baseline thermal value by doing a benchmark run at the cores’ lowest DVFS setting, and then a benchmark run is conducted at the DVFS setting being tested. Likewise, the second set of test runs is conducted for all of the cores’ possible DVFS settings. Note that, to minimize the number of profiling runs, all of the cores are profiled together.
at the same DVFS setting, even if per-core DVFS is allowed. For thermal measurement, doing so is acceptable because low lateral heat transfer within the die means that adjacent cores’ temperatures will have a minimal impact on a given core’s temperature. Please note that this profiling method assumes that there is some sort of emergency throttling method that will prevent the processor from reaching an unsafe temperature during these profiling runs, as OCCAM’s DVFS-based thermal control is not used during these profiling runs.

The processor cores’ temperatures are recorded using the `sensors` command in Linux. Based on `libsensors`, this command reads in the processors’ temperatures from their on-die thermal sensors, providing an accurate assessment of the processor cores’ actual temperature. The sensors have a maximum temperature precision of $1.0^\circ C$. During these runs, the cores’ temperatures are sampled using `sensors` every $0.25$s. Such a time is sufficiently small to accurately capture changes in the cores’ temperatures.

7.3.2 Thermal Model

**Definition 13.** $S_{\text{temp}}$ is the set of all possible temperature phases for the multicore system. It is composed of the Cartesian Product of the individual cores’ possible temperature phases: $S_{\text{temp}} = \{S_{\text{temp}}_0 \times \ldots \times S_{\text{temp}}_{n-1}\}$, $n = \text{NumProcessorCores}$. $S_{\text{temp}}_i$ is the set of all of the thermal phases for each processor: $S_{\text{temp}}_i = \{\text{Set of Thermal Phases}\}$

OCCAM’s thermal model comprises two components: a set of thermal phases (described in Definition 13) that concisely represent the temperatures that the core is likely to see; and a heating/cooling model that predicts the next temperature of the system given the current temperature, DVFS setting, and time duration using a simple log model.

After finishing the offline profiling runs, OCCAM determines the temperature phase values for each processor core. It determines these values using the same techniques used to determine the phases for instruction count and instruction throughput. Each core’s thermal phases is a one-degree Celsius range. A major difference between the thermal model and the
other models of system behavior that OCCAM uses is that the thermal phase predictions are deterministic, rather than stochastic. Thermal phase predictions take the existing phase and project the next temperature phase with 100% probability, rather than as a probability distribution of states. The full probability result is possible because of the high level of accuracy of the thermal heating/cooling model.

The thermal heating/cooling model is determined using the data from the second phase of profiling. The second profiling phase provides a nearly-continuous description of how the processor’s temperature changes for a given DVFS setting. This temperature is fitted into the model of $Temp(t) = A \log(t) + B$, with $A$ and $B$ determined using least squares fitting. At this time, the profile also takes note of the range of temperatures encountered by the different profiles as the interval $[temp_{\text{lower}}, temp_{\text{upper}}]$. This model represents what the curve’s temperature is at a given time $t$. Note that there is a separate model for both the “cool” and “heat” profile runs. The logarithmic model used by OCCAM to model the processor cores’ thermal characteristics is similar to the model used by Yeo and Kim in [102]. In their work, while they acknowledge that the processor’s heating takes on a logarithmic curve, they use an approximation of the thermal model by decomposing the logarithmic curve into three piecewise, linear components. Such approximation was needed because their work involves an online scheduling algorithm that must complete quickly. OCCAM, on the other hand, uses the full logarithmic model because its offline control policy generation allows for using more accurate but more computationally-complex algorithms.

\[
\begin{align*}
Temp_{\text{heat}}(t) &= A_{\text{heat}} \log(t) + B_{\text{heat}} \quad (7.1) \\
t_{\text{heat}}(Temp) &= e^{\frac{Temp - B_{\text{heat}}}{A_{\text{heat}}}} \quad (7.2)
\end{align*}
\]

\[
\begin{align*}
Temp_{\text{cool}}(t) &= A_{\text{cool}} \log(t) + B_{\text{cool}} \quad (7.3) \\
t_{\text{cool}}(Temp) &= e^{\frac{Temp - B_{\text{cool}}}{A_{\text{cool}}}} \quad (7.4)
\end{align*}
\]
\( \text{Time}_{\text{heat}} := t_{\text{heat}}(\text{temp}_{\text{curr}}) \)
\( \text{Time}_{\text{cool}} := t_{\text{cool}}(\text{temp}_{\text{curr}}) \)

\[
\begin{align*}
\text{if } & \text{Time}_{\text{heat}}(\text{Time}_{\text{heat}} + \text{Duration}) \in [\text{temp}_{\text{lower}}, \text{temp}_{\text{upper}}]_{\text{heat}} \text{ then} \\
\text{Temp}_{\text{final}} & := \text{Temp}_{\text{heat}}(\text{Time}_{\text{heat}} + \text{Duration}) \\
\text{else} \\
\text{Temp}_{\text{final}} & := \text{Temp}_{\text{cool}}(\text{Time}_{\text{cool}} + \text{Duration}) \\
\end{align*}
\]

\text{Algorithm 5: Procedure for Predicting the Future Temperature}

Finally, these logarithmic models must be converted into a form that provides a predicted temperature given the current temperature, the DVFS setting, and the Throughput Frame’s duration. The first step is to algebraically transform Equations 7.1 and 7.3 so that they are functions for time and temperature (instead of temperature and time), thus yielding Equations 7.2 and 7.4. Algorithm 5 uses these four equations to predict the next temperature. First, it finds the current “time” (essentially, at what time during the thermal profiling run that temperature was encountered) given the current temperature for that particular model. These “times” are determined for both the “cool” (\( \text{Time}_{\text{heat}} \)) and “heat” (\( \text{Time}_{\text{cool}} \)) thermal profiles. Next, the “end” time is calculated by adding the predicted duration to these time values. These start and end times are checked to see whether they are in the “heat” profile; if it is, then the final temperature is calculated using the “heat” profile’s equation. If the times are not in the “heat” profile’s time interval, then the “cool” profile’s equation is used instead.

7.4 Tractable Control with Heterogeneity

The next issue concerning thermal control is ensuring tractability. Tractability becomes essential when dealing with multicore thermal control because each processor core has its own set of thermal states. As a result, the number of combined states, where these states are a vector of all of the processor cores’ thermal states, grows exponentially with the number of cores if the combined states are formed as a Cartesian Product of the cores’ possible thermal states. Successfully controlling such a system rapidly becomes intractable due to the large number of states that have to be visited in order to profile systems with large core counts, such as the 16 cores in \textit{MULTI}. 
Similarly, non-global DVFS (i.e. where individual cores or groups of cores can have their own separate DVFS settings) compounds the tractability issue.

As a result, the large number of states produced by the aforementioned straightforward, but naive, way of generating combined states has to be significantly reduced. Fortunately, for thermal control, there are a number of opportunities to eliminate many of these states. A key source of potential state-space pruning is to eliminate states that cannot be reached during the normal course of execution. Similarly, it is also possible to prune away a large number of the possible DVFS decisions, which not only reduces the number of possible actions, but also the number of possible, reachable next states.

It can be intuitively shown that there are a great many states that can be generated through the Cartesian Product-based combined state generation that will never be reached in practice. Illustrating this is a simple thermal example. On many, if not most, systems that support non-global DVFS, per-core DVFS is still not supported. For example, on *MULTI*, DVFS can only be done at a per-socket level. As a result, while the application is running in the steady-state (later parts of this section will discuss handling initial conditions in the system), the cores within a given DVFS scaling group will have thermal phases that correspond to all of the cores running at the same DVFS setting at the same time. This means that, for example, if the DVFS scaling group has been running at the highest DVFS setting, the cores will all have similar, high temperatures: there will not be a core that is at its lowest temperature phase while another core is at its highest temperature phase.

It is also possible to remove *a priori* control decisions, particularly DVFS decisions, that are not optimal. As discussed previously, the OCCAM system abstracts away the computer system’s parallel compute resources into a single, virtual compute unit, called the *CPU Aggregate*. The CPU Aggregate measures the combined, additive CPU frequencies as a proxy for system performance. If it is assumed that the CPU cores are at least somewhat homogeneous in their power/thermal characteristics and homogeneous in their performance characteristics (i.e. less than a factor of two difference), the best system DVFS configuration is one where the CPU cores are no more than one
DVFS step away from each other. Doing so significantly reduces the number of possible DVFS settings that need to be investigated. Another heuristic used by OCCAM is to “thermally schedule” the DVFS settings. Such thermal scheduling involves assigning the highest core DVFS settings to the coolest cores, all the way down the line to the lowest DVFS settings to the hottest cores.

DVFS scaling units can also be leveraged not only to reduce the number of possible DVFS decisions that can be made, but also to reduce the number of thermal states that need to be accounted for. Such a reduction is possible by only accounting for the highest temperature core in each DVFS scaling unit when determining the control policy. Such a policy works because we only really care about whether a given thermal threshold has been exceeded by any of the cores. Therefore, it is only necessary to track the hottest core in each group to see whether any of the cores in the group have exceeded the thermal limit. In no situation does the hottest core’s temperature ever get exceeded by any of the other cores’ temperatures.

The final technique, decision pruning, essentially converts the long-horizon policy search into a hybrid greedy (finite horizon of unity) and long-horizon policy search. Essentially, a greedy search through the possible control policies is used to eliminate all but the best few decisions. Such an heuristic is based on the reasoning that even over an infinite horizon, the worst finite horizon decisions will still be the worst infinite horizon decisions.

The aforementioned techniques are combined into the following state reachability procedure. First, a series of “seed” thermal states are obtained by using the profile data that was also used to obtain the thermal phases. The reasoning behind using this data to seed the reachability analysis is that this particular thermal profile at least approximately represents the kind of thermal states that might be realistically seen by the system. These seed thermal states are combined with the instruction count, instruction throughput, and quality phases to provide a combined series of phases.

These seed states are subsequently used as the starting point for generating more reachable states. Algorithm 6 describes the new state search procedure in detail. For each state, the available control decisions are first iterated through to determine the greedy costs of the different decisions.
Reachable States := Seed States
Last Reachable States := Seed States

while size(Last Reachable States) != 0 do
  for Last Reachable State in Last Reachable States do
    for Decision in Decisions do
      Record decision cost
    end for
    Discard all but the lowest cost $n$ decisions
    Compute the next states for these remaining decisions
    Add new, unseen next states to Last Reachable States
  end for
end while

Algorithm 6: Procedure for Searching for Reachable States

Next, only the lowest $n$ decisions are kept ($n$ is a predetermined parameter; the best value for $n$ will be studied later in Chapter 9). The next states for these states is then determined using the $n$ best decisions to get the next states. These next states are then added into the total states. This process repeats itself until there are no more new states are found.
Chapter 8

Safety, Optimality, and Setting Time of OCCAM’s MPC Controller

This chapter discusses three key controller characteristics of OCCAM’s MPC controller: its optimality (discussed in Section 8.1) its safety (discussed in Section ??), and the controller’s settling time (discussed in Section 8.3). Finally, this chapter discusses the effect of the tractability heuristics that reduce the size of the search space on the optimality of the control policy in Section 8.4.

8.1 Optimality

The MDP solver techniques are optimal for the model because the infinite-horizon MDP techniques, such as policy iteration, are proven to be optimal techniques for solving MDPs. As a result, for the model that the MPC uses, it can determine the optimal policy. Note that while the control policy is optimal, it can still suffer from modeling error that can make the actual overall control decision not be the optimal for the system. Also, since OCCAM’s Model Predictive Controller is based on prediction, it cannot always predict with 100% accuracy what the system is going to be like, as doing so would violate causality.

8.2 Safety

This section provides a proof of the safety of the controller. The ensuing proof consists of several components. This proof first demonstrates that the table-based controller can take any input, whether bounded or unbounded, and convert it to a bounded value, due to the finiteness of
the tables used. Next, the safety of the values measured for the computer system are demonstrated,
by showing how the Instruction Throughput has a definite upper and lower bound. Finally, the
safety of the OCCAM applications themselves is shown by discussing how the Instruction Count
and Quality values are bounded.

8.2.1 Overview of OCCAM's MPC-based controller

OCCAM-MPC's controller uses a two-level hierarchy of tables. The first set of tables con-
verts the continuous-valued $y_t$ values into discrete-values $y'_t$ values. $y_t$ comprises the following
components:

1. $y_{inscnt}$ – the Instruction Count value for the system. Consists of the sum of the instruction
   counts for all of the processors for the past Throughput Task. $y_{inscnt} \in [0, \infty)$

2. $y_{insthr}$ – the Instruction Throughput value for the system. $y_{insthr} = \frac{y_{inscnt}}{t_{task}}$. $y_{insthr} \in
   [0, \infty)$

3. $y_{qual}$ – the Quality value for the results for the past Throughput Task. $y_{qual} \in [-\infty, \infty]$ 

4. $y_{temp}$ – a vector containing the highest temperature values for the hottest core in each
   scaling unit. $y_{temp} \in (Absolute\ Zero, \infty)$
These \( y_t \) values are converted into \( y'_t \) values by looking up in the respective conversion tables which phase value’s range the \( y_t \) value belongs to. All of the \( y'_t \) components, and therefore \( y'_t \) itself, are bounded between two finite values, \( A_t \) and \( B_t \), because the phase conversion table maps the continuous \( y_x \) values to a finite set of values in the set of \( \{States_x\} \), which is the Cartesian product of \( \{States_{inscnt}\}, \{States_{insthr}\}, \{States_{qual}\}, \) and \( \{States_{temp}\} \).

1. \( y'_{inscnt} = Conv_{inscnt}(y_{inscnt}), A_{inscnt} < y'_{inscnt} < B_{inscnt}, y'_{inscnt} \in \{States_{inscnt}\} \)
2. \( y'_{insthr} = Conv_{insthr}(y_{insthr}), A_{insthr} < y'_{insthr} < B_{insthr}, y'_{insthr} \in \{States_{insthr}\} \)
3. \( y'_{qual} = Conv_{qual}(y_{qual}), A_{qual} < y'_{qual} < B_{qual}, y'_{qual} \in \{States_{qual}\} \)
4. \( y'_{temp} = Conv_{temp}(y_{temp}), A_{temp} < y'_{temp} < B_{temp}, y'_{temp} \in \{States_{temp}\} \)

8.2.2 Safety of the Computer System

The two key parameters measured that are intrinsic to the Computer System component of the OCCAM system are the Instruction Throughput and the Core Temperature. In order to demonstrate the safety of the Computer System components, it is necessary to show that \( 0 < y_{insthr} < \text{Upper}_{insthr} < \infty \) and \( < y_{temp} < \text{Upper}_{temp} < \infty \). The highest possible value for \( \text{Upper}_{insthr} \) is \( \text{Upper}_{insthr} = IPC_{\text{max}} \times \text{Freq}_{\text{max}} \times \text{Num\_cores} \). Moreover, assuming that the system is deadlock-free, \( y_{insthr} \) should also be greater than zero. Showing that \( \text{Upper}_{temp} \) is finite and bounded because the system assumes the presence of emergency thermal throttling that prevents thermal runaway from happening.

8.2.3 Safety of OCCAM Applications

The two key parameters measured that are intrinsic to the OCCAM Application component of the OCCAM system are the Instruction Count and Quality. In order to demonstrate the safety of the OCCAM Application components, it is necessary to show that \( -\infty < \text{Lower}_{qual} < y_{qual} < \infty \).
$Upper_{qual} < \infty$ and $0 < \text{yinscnt} < Upper_{inscnt}$ < $\infty$. Note that, as long as the OCCAM Application is actually doing something (e.g. it is not deadlocked or livelocked), $y_{inscnt}$ will be greater than zero. As a relatively generic programming framework designed to accommodate as wide of a variety of applications possible, it is possible to write an OCCAM application that never completes (i.e. has an infinite, unbounded Instruction Count for every Throughput Task) and has a quality function that can output a $\|y_{qual}\| = \infty$, the OCCAM system cannot be proven stable in $y_{qual}$ and $y_{inscnt}$. All seven of the OCCAM benchmarks, however, are safe in these parameters, even though their safety was not taken into account when they were developed for the OCCAM System.

8.3 Settling Time

The final control characteristic discussed in this chapter is settling time. Settling time is defined as the amount of time it takes for the controller to settle on a given setpoint if the input does not change. Since OCCAM-MPC only makes control decisions on Throughput Task boundaries, OCCAM-MPC has a settling time of essentially zero (just the small amount of time it takes to look up the next control decision to make) over the time horizon of a single Throughput Task. The settling time as measured across multiple Throughput Tasks, however, is considerably more difficult to demonstrate, and determining it is not within the scope of this dissertation.

8.4 Effect of Tractability Optimizations on Optimality

This next section provides an analysis of the effects on the optimality of the different tractability heuristics. Since the settling time and safety are affected by the final overall structure of the discrete, table-based controller, none of these optimizations have any effect on the safety or settling time of the controller. What they may potentially affect, however, is the optimality of the controller because they essentially, in various ways, reduce the state space to be searched and/or the set of possible decisions to search.

The first technique used to improve tractability is reachability analysis. Reachability analysis
should have no effect on the final control policy because all it is doing is removing states that should not be encountered by the system. In practice, however, reachability analysis may affect the quality of the control policy if certain possible states that can occur in practice are not covered in the profiling run. As a result, it is important to ensure excellent profiling coverage when using the reachability analysis.

Next, thermal pre-scheduling may lead to a less-optimal set of control decisions to be made. Thermal pre-scheduling is a technique where the higher-frequency DVFS decisions are put on the cooler-running processor cores, while the lower-frequency DVFS decisions are put on the hotter-running processor cores. This policy is determined when the control policy is generated by using the thermal models to determine which processors are the hottest. Thermal pre-scheduling is not an optimal heuristic when the system is not in a steady-state behavior, i.e. an OCCAM-controlled application is not running for awhile, and the system is in some other thermal state. One potential alternative to the static schedule is to dynamically determine which processors get which DVFS settings based on their current temperatures. There are several drawbacks to such a technique, however. First, this technique requires dynamically modeling the system’s temperature at run time. Performing thermal modeling at run time may add significant amounts of overhead to the otherwise-low overhead of OCCAM’s table-based controllers. Second, performing the thermal scheduling at run time breaks OCCAM’s holistic view of the control system, where thermal decisions are made simultaneously with the other decisions.

The next \( n \) decisions tractability heuristic can also lead to sub-optimal decisions. This heuristic, where the infinite horizon control policy only considers the best \( n \) greedy control decisions, is essentially a hybrid of a finite horizon of unity control policy search algorithm and the infinite-horizon control policy. The best case result is that the control policy will be optimal, while the worst-case result is that the \( n^{th} \)-best greedy control decision will be used.

Finally, the hottest core-only heuristic, where the control generation algorithm only models the temperature of the hottest core, is optimal as long as several conditions are met. First, all of the processor cores must have the same thermal limits. This property is obeyed on all of the
processors studied, since the temperature limit is a function of the processor’s manufacturing. Next, all of the cores must be at the same DVFS setting. This property holds because the hottest core only scheduling is only used for collections of processor cores whose frequencies must be scaled together as a group. Finally, the processors’ thermal behavior must be such that there is a hottest core, and for all possible circumstances, that core is the hottest. In other words, \( \text{Core}_\text{hottest} > \text{Core}_\text{others} \) must always hold. For real multicore processors, this property will hold as long as the processors are in their steady state of execution. This property may not hold if, for example, one of the \( \text{Core}_\text{others} \) has been processing large quantities of work and \( \text{Core}_\text{hottest} \) has not (and likely was put into a sleep state by the operating system). As a result, \( \text{Core}_\text{others} > \text{Core}_\text{hottest} \). During the steady state of OCCAM execution, however, this condition should not hold, as all of the work should be distributed among the cores evenly.
Chapter 9

Thermal Control Results

This chapter presents the experimental results for the thermal controller, *OCCAM-THERMAL*, across several sections. First, Section 9.1 provides an overview and summary of the experimental results. Next, Section 9.2 describes the changes made to the experimental methodology from the test setup used in Chapter 6. Next, Section 9.3 presents the thermal cost results. These experimental results compare the non-thermal-aware *OCCAM-MPC* to the thermal-aware *OCCAM-THERMAL* controller. Next, Section 9.4 compares how different horizon lengths in the control policy generation affects the final resulting policy. Finally, Section 9.5 discusses the effect of the control decision pruning, where, to improve tractability, the number of control decisions is reduced to the best $n$ greedy decisions.

9.1 Overview

Overall, *OCCAM-THERMAL* provides superior control of the computer system than does the non-thermal-aware *OCCAM-MPC* controller when thermal constraints are considered. Moreover, *OCCAM-THERMAL*, by holistically taking into account the system’s temperature and adjusting not only the CPUs’ power consumption through the DVFS settings, but also optimizing the application’s parameters allows for superior control over a setup that optimizes the components separately.

Presented are several different results. The first set of results presents the overall cost function. These cost function-based results show how well *OCCAM-THERMAL* and *OCCAM-MPC*
can minimize the cost function when thermal constraints are taken into account. It presents an overall cost score that takes into account not only whether the temperature stays within the thermal constraints, but also how well the controller optimizes the other performance requirements within the limits of the thermal constraints. The next set of results examines the temperature of the hottest CPU cores across time. This result only examines how well the controller keeps the temperature within the temperature constraints. Here as well, \textit{OCCAM-thermal} compares favorably to \textit{OCCAM-MPC}, due to its ability to control for temperature.

### 9.2 Test Setup

The test setup is mostly identical to the test setup used with \textit{OCCAM-MPC} in Chapter 6. There are several key differences, however. First, there is a comparison made against using the Linux kernel-based passive thermal control. This technique is used by the Linux kernel to scale back DVFS settings based on the temperature using a custom, essentially \textit{ad hoc} PID controller. This controller is built into the kernel; however after Linux kernel version 2.6.21, the ability to change the thermal trip settings was removed. As a result, in order to get this feature into the kernel version used on the OCCAM test platforms (Linux kernel version 2.6.28), it was necessary to modify the kernel source code and compile another custom kernel.

The cost function is measured in two ways for the thermal results. The first way is with the thermal control put into it. This cost function shows the cost penalty for exceeding the thermal limit, which is set at a high cost \((1 \times 10^{12})\) in order to strongly discourage such actions. The other way the cost is shown is using the same cost function that is used by \textit{OCCAM-MPC}; that is, the cost function without any sort of thermal penalty. These cost values show how the different thermal control techniques affect the ability of the OCCAM controller to meet the performance requirements of the application and computer system.

Only two of the three computer systems previously studied, \textit{HI}, and \textit{MULTI}, are studied. \textit{LO} is not studied because the Intel Atom processor in it does not experience much temperature change with different DVFS settings. As a result, thermal control is both unnecessary and essentially
impossible on this system. Next, for the two computer systems studied, experimental thermal constraints were chosen so that they can be realistically exceeded by the seven benchmarks while still being high enough that the lowest DVFS settings on the processors allow the application to run below the thermal limit of the processors.

Finally, in order to create a scenario where the microprocessors’ temperatures on **MULTI** can be exceeded, for the temperature and cost tests, the input datasets of all of the benchmarks except **hearing Aid** were significantly increased (**hearing Aid**, as a digital hearing aid application, does not make sense to increase the input dataset) in order to make the very high performance 16-core system less overprovisioned for the workloads. As a result, the applications have more success at increasing the temperature of the processors so that the scalability of OCCAMs thermal control can be successfully scaled to large numbers of processors.

### 9.3 Thermal Control Results

This section presents and discusses the thermal control results. These temperature plots consist of measuring the temperature of the hottest processor core (the reasoning is based on if the hottest processor core is below its thermal constraint, then it follows that all of the other processors are below their thermal constraints as well).

#### 9.3.1 **HI**’s Results

Overall, **OCCAM-thermal**’s thermal control does a significantly better job keeping the temperature within its thermal limits than does the reactive, kernel-based thermal controller. This advantage is primarily due to its predictive thermal control that take into account the system’s future behavior as well as the future behavior’s effect on the temperature. The better thermal control of **OCCAM-thermal** comes at a minimal cost to the performance of the rest of the system. This is due to the fact that **OCCAM-thermal**’s ability to intimately know what parameters can be traded off to produce the minimum performance degradation. **tachyon**, shown in Figure 9.14, also demonstrates the situation described in Chapter 1, where the holistic controller can actually
lead to significantly better control of the system beyond just keeping the temperature within its thermal limits.

Four different types of results are compared. The first type of result, called *None*, is a system that is essentially controlled like *OCCAM-MPC*: the Data Resolution and DVFS are controlled for performance, but the processor cores’ temperatures are not taken into account. Next, *Linux* uses the same *OCCAM-MPC*-like controller as used in *None*, but thermal control is provided by the thermal throttling controller in the Linux kernel. The next test setup, *OCCAM*, is a full implementation of the *OCCAM-THERMAL* controller. Finally, *FIXED* is a version of *OCCAM-STOCHASTIC* where the CPU frequency is held at $1.2 \text{GHz}$. $1.2 \text{GHz}$ has been empirically determined on *HI* to be the highest DVFS setting that is guaranteed to not exceed the 70 degree thermal threshold.

The results are analyzed in several different ways. First, presented in Figure 9.1, is a comparison of the overall costs of the different configurations, using a cost function that takes into account how well the quality, timing, and temperature performance requirements are met. Next, the temperatures are plotted across time, with a system-level thermal constraint of 70 degrees Celsius, as shown in Figures 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, and 9.8. In the aforementioned figures, the Quality and Timing constraint levels are integer values. Note that a small offset of 0.1 is added to the *None* plots of Quality and Timing, and a small offset of $-0.1$ is added to the *OCCAM* results to improve readability of the graphs.

Overall, it is worth noting that in general, the *Linux* and *None* configurations will usually meet the Quality and Timing performance requirements better than *OCCAM* or *FIXED*. These two configurations are better able to meet these two performance requirements because they typically run the CPU cores at a higher DVFS setting. *None* is able to run the CPU cores at a higher DVFS setting because the control policy is not thermally constrained. As a result, *None* exceeds the thermal performance requirement on all of the benchmarks except for *hearing_aid*.

The *Linux* setup is similar in that it can often meet the Quality and Timing performance requirements better than can *OCCAM* and *FIXED* because it typically runs the CPU cores faster than
do OCCAM and FIXED. Overall, however, Linux underperforms OCCAM and FIXED because it does not consistently keep the temperature below the thermal performance requirement. While the Linux controller does observe the thermal constraint, it struggles to keep the temperature below this limit. Such behavior likely occurs for several reasons. First, Linux is designed to be a universal controller for performing thermal management on the system. As a result, while Linux functions essentially as a PID-based controller, its settings are a compromise between responsiveness and portability between systems. As a result, it changes the CPU frequency very slowly. An example of this slow response time can be seen in the train results for seismic, where there is a long period where Linux exceeds the thermal requirement, and then Linux successfully puts it back below the performance requirement.

It is important to note that FIXED often provides results whose quality is comparable to OCCAM. Such similar behavior on many of the benchmarks exists for several reasons. First, FIXED functions as a manual selection of the DVFS settings. As a result, the manual selection, which uses the highest DVFS setting that, for all of the benchmarks, does not exceed the thermal requirement. Moreover, the PID controller that chooses the Data Resolution was meticulously hand-tuned by the author, a difficult process that took many hours for all of the benchmarks. The manual selection of the temperature works reasonably well for the case where the ambient temperature is constant; in a situation where the ambient temperature increases or decreases significantly, this frequency selection will not be optimal.

Another issue with FIXED is that it performs relatively poorly with datasets that are significantly different from the ones used to tune the PID controller. This effect demonstrates itself in tachyon and stereo_vision, where difference between OCCAM and FIXED are much higher for the run results than they are for the train results.

9.3.2 MULTI’s Results

Next, in Figures 9.10, 9.9, and 9.14, are the thermal control results for a selected subset of the benchmarks for the MULTI results. These results show how OCCAM-thermal can successfully
Figure 9.1: Summary cost comparison of the benchmarks. The first six Throughput Frames were not counted, in order to give the system time to settle. Overall, OCCAM-THERMAL provides comparable, if not superior, control over either FIXED or Linux. The advantage of OCCAM-THERMAL widens for the run data. Such behavior is likely due to the flexibility provided by the MPC controller over the manually-tuned PID controller.
Figure 9.2: Cost, temperature, quality, and timing plot on HI across time for histogram. OCCAM-THERMAL does not provide an appreciable improvement over the FIXED policy because histogram is a highly predictable workload with virtually no change in its characteristics across time or between data sets. Differences in the executions of histogram are entirely due to different initial conditions in the system.
Figure 9.3: Temperature and cost plot on $HI$ across time for stereo_vision. In this benchmark, OCCAM-THERMAL has a minor advantage over FIXED. In the train data, Linux keeps the processors’ temperature too high, thus not meeting the temperature requirement. For the the run data, on the other hand, Linux keeps the processors’ frequencies too low, leading to a worse Timing outcome.
Figure 9.4: Temperature and cost plot on HI across time for lr. lr demonstrates how OCCAM-THERMAL’s superior knowledge of the system’s requirements and behavior provides better results, both with thermal control as well as by better meeting the timing and quality performance requirements.
Figure 9.5: Temperature and cost plot on HI across time for wordcount. Save for a few instances where OCCAM-THERMAL briefly exceeds (and by a small amount) the thermal threshold, OCCAM-THERMAL better meets the system’s performance requirements. The brief instances in the train dataset where the temperature exceeds the thermal threshold is likely due to small thermal modeling errors. In the run data, the MPC-based controllers make a different trade-off between Quality and Timing than the PID controller.
Figure 9.6: Temperature and cost plot on HI across time for hearing aid. OCCAM-THERMAL, as it is based on a Model Predictive Controller, can more quickly react to the system’s parameters and adjust the decisions than can the PID controller used in FIXED.
Figure 9.7: Temperature and cost plot on HI across time for tachyon. Overall, due to OCCAM- THERMAL’s better model-based knowledge of the application and its requirements, it can make better decisions adapting the application and computer system. This benchmark also provides a good test of the respective controllers’ abilities to adapt to run data that is radically different from the train data.
Figure 9.8: Temperature and cost plot on HI across time for seismic. Note that the MPC-based controllers, particularly OCCAM-THERMAL, have a difficult time controlling this application well due to the rapidly-varying stochastic behavior of the seismic application. The Quality values are identical for seismic because seismic does not measure Quality, as seismic does not use the Data Resolution.
scale to the thermal control of large numbers of processors or other control parameters. Note that these results do not contain the Linux kernel-based thermal controller results, due to the lack of proper ACPI support for this feature in the Sun Fire system tested. Overall, the results are similar to the ones obtained for HI, where OCCAM-THERMAL does a superior job of controlling the system than does the non-thermal-aware OCCAM-MPC.

### 9.4 Comparison of Different Horizon Lengths

As discussed in Chapter 7, thermal control involves making decisions that not only affect the processors’ temperatures in the next Throughput Frame, but these decisions also affect the temperatures of the processors in future Throughput Frames as well. As a result, in order to obtain an optimal decision, it is necessary to account for future Throughput Frames as well as for the next Throughput Frame. This section examines different-length finite horizon search (search policies that take into account 1, 10, and 100 Throughput Frames into the future) policies and compares them against the infinite-horizon policy iteration MDP search technique. The search policies are compared against the policy iteration technique by quantifying the percentage of different control decisions between the two techniques.
Figure 9.10: Cost plot on **MULTI** across time for **stereo_vision** with the **train** data.

Figure 9.11: Cost plot on **MULTI** across time for **lr** with the **train** data.
Figure 9.12: Cost plot on *MULTI* across time for *wordcount* with the *train* data.

Figure 9.13: Cost plot on *MULTI* across time for *hearing_aid* with the *train* data.
Figure 9.14: Cost plot on \textit{MULTI} across time for \texttt{tachyon} with the \textit{train} data.

Figure 9.15: Cost plot on \textit{MULTI} across time for \texttt{seismic} with the \textit{train} data.
Figures 9.16, 9.17, 9.18, 9.19, 9.20, 9.21, and 9.22 compare the finite horizon search results to the infinite horizon policy search technique. Overall, for the seven benchmarks, there are some key, general findings worth noting across all seven of the benchmarks. First, the differences between the different horizon lengths are more pronounced when the CPUs’ temperatures are more heavily constrained. This observation demonstrates that it is primarily the temperature that needs an infinite horizon search policy. Another general, overall observation is that while there is substantial difference between the infinite horizon policy search and the finite horizon search policies that search 1 or 10 Throughput Frames into the future, the 100 Throughput Frame finite horizon search policy is almost identical to the infinite horizon search policy in all of the benchmarks. Nevertheless, policy iteration is still a better choice, as the MDP Toolboxs policy iteration converges in less time than the 100 Throughput Frame finite horizon policy search.

9.5 Action Pruning Results

A major issue for OCCAM’s MDP-based MPC controller is tractability. As discussed in Chapter 7, tractability becomes a serious issue for control policy generation once thermal control
Figure 9.17: Horizon length comparison for histogram.

Figure 9.18: Horizon length comparison for lr.
Figure 9.19: Horizon length comparison for stereo_vision.

Figure 9.20: Horizon length comparison for seismic.
Figure 9.21: Horizon length comparison for tachyon.

Figure 9.22: Horizon length comparison for wordcount.
is factored in and the resultant search space size increases exponentially. This exponential growth in the size of the search space is due to the fact that each DVFS control group is another degree of freedom that needs to be accounted for. As a result, several different techniques are employed to reduce the search space. A key technique is decision pruning, where only the best \( n \) greedy decisions (greedy decisions are the best decisions when only accounting for the decisions effect on the next Throughput Frame) at each state are used by the infinite horizon control policy search algorithm to determine the final control policy.

The action pruning results are presented in a similar manner to the horizon length results in Figures 9.23, 9.24, 9.25, 9.26, 9.27, 9.28, 9.29. The control policies generated with various numbers of preserved decisions are compared to the non-action-pruned control policy. Graphed are the percentage of different control decisions between the baseline no pruning control policy and the different levels of action pruning. Overall, while it is beneficial to not produce a greedy control policy (keeping only one greedy control decision is equivalent to a greedy control policy), increasing the number of possible greedy decisions to choose from improves the quality of the results while still significantly reducing the number of possible control decisions. For most of the benchmarks, limiting the number of control decisions to five provides a good balance between control policy quality and tractability.

9.6 Runtime Overhead

\textit{OCCAM-thermal}'s runtime controller overhead, measured using \texttt{gprof}, is also reasonably low as a percentage of the system's overall run time, with an average overhead of 6.0\%. The highest overheads measured are for \texttt{wordcount} at 19.24\% and \texttt{seismic} at 9.11\%, and the lowest overheads are 0.1\% and 0.21\% for \texttt{hearing aid} and \texttt{histogram} respectively. The relatively high overhead for \texttt{seismic} is a result of the technique used to render only the screen tiles that change. For \texttt{wordcount}, the high overhead is due to its very short-duration Throughput Tasks, which cause the runtime’s relative overhead to be significantly higher (the absolute overhead of the \textit{OCCAM} run-time is relatively constant regardless of the execution time of a Throughput...
Figure 9.23: Decision pruning comparison for stereo vision.

Figure 9.24: Decision pruning comparison for histogram.
Comparison of Decision Pruning Amounts

Figure 9.25: Decision pruning comparison for hearing aid.

Figure 9.26: Decision pruning comparison for lr.
Figure 9.27: Decision pruning comparison for seismic.

Figure 9.28: Decision pruning comparison for tachyon.
Figure 9.29: Decision pruning comparison for wordcount.
Task).
Chapter 10

Model Checking

This chapter discusses characterizing OCCAM’s MDP-based Model Predictive Controller using a model checking-like framework. Section 10.1 first provides a brief overview of probabilistic model checking and how OCCAM uses it to characterize the system’s behavior. Next, Section 10.2 provides a more detailed overview of probabilistic model checking. Finally, Section 10.3 discusses how model checking is used with OCCAM to provide an overview of how well the system works.

10.1 Overview

Model checking is a technique where the system determines properties about a system that is modeled as a finite state machine. The purpose of this model checking study is to characterize the degree of robustness (robustness is a measure of how well the controller performs in the midst of measurement and modelling error) of the controller.

Probabilistic model checking is the technique used to determine both of these parameters. While standard model checking checks a finite state machine with deterministic state transitions, probabilistic model checking operates on a finite state machine that describes the next state as a probability distribution. Model checking such a setup is possible by determining probabilistic properties, such as the probability that the next state will belong to a certain set of states.

For this model checking study, the model checker was a “probabilistic” model checker, where Monte Carlo analysis is used to obtain a probabilistic evaluation of the particular model property.
Such a method was used because the model creation part of model checking consumed more memory than was available on any of the systems studied.

10.2 **Background: Probabilistic Model Checking**

Probabilistic model checking is a type of model checking where the system operates with probabilistic quantities. This probability can be used in one of two ways. The first way is to determine, with a “yes” or “no” answer, whether some event occurs with some probability. Take, for example, (NOTE: this is borrowed from an example at the PRISM Model Checker’s Web site [3]):

\[ P \geq 1 \ [ F \ "terminate" ] \]

This command means “the algorithm eventually terminates successfully with probability 1”.

Probabilistic model checking can also be used to determine the probability that an event will occur. Take this example from the PRISM Model Checker’s Web site:

\[ P = ? \ [ !proc2_terminate \ U \ proc1_terminate ] \]

This command means “the probability that process 1 terminates before process 2 does”.

Probabilistic model checking is an excellent fit for studying the model and the controller used by OCCAM’s MDP-based Model Predictive Controller. Probabilistic model checking is a good choice for OCCAM because OCCAM models a stochastic system as a Markov Decision Process-based system with discrete control parameters. Probabilistic model checking is also useful because it allows for determining the probability and frequencies that certain sets of states will be visited during a given execution run with a given control policy. If each of these sets corresponds to the states with a certain level of constraints, then it is possible to determine the likelihood of encountering various constraint levels.

Using probabilistic model checking in order to anticipate how well the system will perform has several uses. First, this model checking-based approach potentially allows for obtaining a
snapshot of the system controller performance without having to actually run the application on a given computer system. Another use of the model checker is to help determine \textit{a priori} what control policies are actually better.

10.3 Applying Model Checking and OCCAM

Applying model checking to the system provides a way to characterize and measure the system model’s performance. In order to perform this model checking, the Markov Decision Process-based model used by the controller to determine its control policy is translated into a format usable by the model checker. Providing this translation involves several steps. First, the MDP is “lowered” into a Discrete Time Markov Process, or DTMC, by applying the control policy being studied to the MDP.

This DTMC is then formatted for use by the model checker. Originally, the plan was to study the model using the PRISM Model Checker [40] [61]. PRISM was chosen because it is a well-known, mature model checker. Ultimately, the final model checking infrastructure did not use PRISM, for two reasons. First, the models generated by OCCAM were too large; as a result, PRISM would run out of memory when trying to generate the models, even on a system with 16GB of memory. As a result, the regular model checker could not be used. Fortunately, PRISM also contains a simulator that can perform a Monte Carlo analysis of the model by examining a large number of simulated paths. While the PRISM simulator seemed promising, the large amount of code needed to translate the DTMC into a format usable by the PRISM simulator became excessive. As a result, the final model checking study used a custom Monte Carlo analysis tool written in Python for analyzing the models.
Chapter 11

Model Checking Robustness Results

This chapter discusses the robustness results obtained empirically for OCCAM using the Monte Carlo-based probabilistic model checking discussed in Chapter 10. These results are performed for the HI system only for the sake of brevity. The number of paths considered in the Monte Carlo-based model checking is $n = 1e5$ for all of these studies. Section 11.1 discusses the robustness of the Markovian probabilities for the various states, while Section 11.2 discusses the robustness of the measured inputs. For all of the robustness model checking results, the values are given a uniformly-distributed probability distribution with a lower and upper bound equal to the original value $\mu$ equal to the original value and $\sigma$ equal to one of the following factors: 0.05, 0.1, 0.25, 0.5, 0.75, and 1.0. The Monte Carlo analysis shows that OCCAM’s controller is substantially robust, most likely due to the discrete nature of the controller and the performance requirements, where the discrete states are determined using the actual system’s behavior.

The graphs presented in this chapter display the results in terms of a “constraint distribution”. The constraint distribution shows what percentage of Throughput Tasks were within which constraint bounds. Displayed are graphs for each of the different constraints within the OCCAM system: Throughput Task completion time (i.e. a measurement of the real-time throughput), Quality, and Temperature. The Throughput Task completion time and the Quality are constraints that are part of the OCCAM application, while the CPU temperature thermal constraints are part of the Computer System.
11.1 Markov Probability Robustness

Presented in Figures 11.1, 11.2, 11.3, 11.6, 11.5, and 11.4 are the Markov probability robustness results. These results show how the performance of the controller is affected by changes in the next-state probability distribution. For all of the benchmarks (note that hearing aid is not presented here because the Monte Carlo model checker ran out of memory), the system is substantially robust to changes in the Markovian probabilities.

11.2 Input Measurement Robustness

Presented in Figures 11.7, 11.8, 11.9, 11.12, 11.11, and 11.10 are the input robustness results. These results show how the performance of the controller is affected by changes in the inputs. For all of the benchmarks (note that hearing aid is not presented here because the Monte Carlo model checker ran out of memory), the system is substantially robust to changes in the Markovian probabilities, albeit not as robust as the Markov probabilities are.
Figure 11.1: Markov Robustness comparison for the histogram benchmark.
Figure 11.2: Markov Robustness comparison for the $lr$ benchmark.
Figure 11.3: Markov Robustness comparison for the stereo_vision benchmark.
Figure 11.4: Markov Robustness comparison for the wordcount benchmark.
Figure 11.5: Markov Robustness comparison for the \textit{tachyon} benchmark.
Figure 11.6: Markov Robustness comparison for the seismic benchmark.
Figure 11.7: Input Robustness comparison for the histogram benchmark.
Figure 11.8: Input Robustness comparison for the $1r$ benchmark.
Figure 11.9: Input Robustness comparison for the stereo_vision benchmark.
Figure 11.10: Input Robustness comparison for the wordcount benchmark.
Figure 11.11: Input Robustness comparison for the tachyon benchmark.
Figure 11.12: Input Robustness comparison for the seismic benchmark.
Chapter 12

Related Work

This chapter provides a survey overview of the various work related to OCCAM. Since the OCCAM System embodies work from a large variety of different disciplines, ranging from programming design patterns to computer system architecture to control theory, this chapter is divided up into several sections. Section 12.1 first describes work related to application adaptation, which corresponds to the design time programming framework component of OCCAM. Next, Section 12.2 delves into detail about other ways to implement functionality similar to the Data Resolution, as well as providing a detailed comparison of OCCAM against a similar work called Code Perforation. After that, Section 12.3 surveys work involving various techniques used to keep processors temperature within its limits. This work inspired OCCAM’s thermal control by describing other techniques for providing thermal control to computer systems. Section 12.4 follows Section 12.3 by surveying various program phase characterization and prediction techniques. These techniques are important for OCCAM because they form the basis of OCCAM’s system identification and predictive control techniques. Next, Section 12.5 discusses various techniques developed for controlling the processors frequency (a.k.a. Dynamic Voltage Scaling (DVS) or Dynamic Voltage and Frequency Scaling (DVFS)). Most of these techniques leverage DVFS in various ways to improve the power/energy consumption of the computer system while only impacting the performance in some limited way. Immediately following Section 12.5 is Section 12.6, which discusses the use of DVFS in the context of real-time scheduling, where the goal is to minimize power/energy consumption while ensuring that the applications’ real-time performance
requirements are met. Finally, Section 12.7 discusses various other work that motivates the need for OCCAM’s functionality.

12.1 Application Adaptation

Application adaptation has been widely studied in the past, and continues to be studied to this day. Many have researched application-specific application adaptation, such as h.264 video encoding [82], mobile content-based image retrieval [59], and audio recorders [57]. While their work demonstrates the usefulness of application adaptation, their methods are essentially *ad hoc* in the sense that there is not a systematic methodology for adapting the applications.

The Illinois GRACE Project [5] developed a system that allows for adapting applications using cross-layer adaptation. Cross-layer adaptation, as described by GRACE, involves optimizing the application within both the application itself as well as by simultaneously optimizing the run-time system. They demonstrated this cross-layer adaptation by executing a videoconferencing application which minimized power consumption by trading off encoding complexity for network bandwidth. OCCAM differs from GRACE in two ways. First, OCCAM provides a comprehensive programming framework for developing adaptable applications. Second, OCCAM provides multicore application adaptation.

CMU’s Odyssey [76] project also developed a system for application adaptation. Odyssey strove to optimize mobile systems by trading off how much work gets done on a remote server versus computed locally on the device. Initially, they used their *Coda* file system, a distributed file system that provides application-transparent adaptability, and later added application-aware adaptability. Like GRACE, Odyssey focused heavily on adapting applications by trading network bandwidth for local processing. OCCAM differs from Odyssey in that Odyssey provided applications with an interface that informed the application about what resources were available to it. Odyssey, however, did not tackle the actual process of providing a framework for adaptable application development.

The work in Peddersen and Parameswaran [78] presents another methodology for applica-
tion adaptation. In this work, the authors present several different programming techniques for providing application self-adaptation: conditional selection of code and/or functions, and various kinds of loop modification. They then modified several multimedia applications with these techniques to allow the applications to adapt themselves for a fixed energy budget. OCCAM differs from this work in two key ways. First, OCCAM is not a framework for writing self-adaptable applications; rather, it is a framework for allowing a run-time system to adapt applications. Such an approach is more flexible in that it allows the application to be written once and later adapted to meet different constraints and goals. Second, OCCAM’s adaptation methodology is designed to provide scalable parallel adaptability.

Another important set of application adaptation work targets cluster and server workloads, such as search engines. The first two papers, by Keleher, et. al [54] and the Active Harmony [90] project, discuss how to develop adaptable applications by allowing for different versions of library routines to be used. These different library routines trade off result quality for computational requirements as well as make different trade-offs between space and execution time by utilizing different data structures. The Green Framework [12] is similar to the Harmony work in that it provides high-level application adaptation of the computer system(s) via function approximation (similar to the selecting different-fidelity library routines technique used by Harmony), as well as loop approximation, where loops that iteratively improve a result are exited early to save time. Green augments Harmony with a focus on not only Quality of Service, but also on energy consumption.

12.2 Data Resolution

There are two key works that discuss different techniques similar to OCCAM’s Data Resolution design pattern. The one with the greatest similarity is a project at MIT called Code Perforation [6] [72]. Code Perforation involves using special compiler techniques to essentially skip iterations in a loop. Their initial techniques involved using the compiler to automatically determine which loops were most profitable to perforate both from the perspective of reducing computation
complexity as well as from having a relatively small effect on the performance loss.

Code Perforation differs from OCCAM in several key ways. First, it performs the application adaptation automatically by finding loops whose iterations can be skipped. While automatically skipping loop iterations is appealing for many different types of applications (particularly the server/cluster types of applications targeted by Code Perforation), skipping loop iterations is not well-suited for many of the CPS applications targeted by OCCAM. Many DSP applications, such as hearing aid, do not function well when simply skipping loop iterations due to several factors. First, with sum-of-products transformations like Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters, the final result needs to at least be scaled up by a factor of the proportion of loop iterations what were skipped. Moreover, skipping loop iterations is akin to lowering the sampling rate; when lowering the sampling rate, the filter tap coefficients need to be changed as well as using a different low-pass filter to remove high frequency content above the Nyquist limit in order to prevent aliasing. Similarly, tachyon cannot heavily benefit from loop perforation, as the main way to trade off result quality for performance is by skipping rendering pixels. However, merely skipping pixels leads to a poor-quality image; some sort of interpolation must also be employed to “fill in” the skipped pixels. Similar result scaling issues also exist for stereo vision.

Note that while Code Perforation automatically determines which and how much to perfo-
rate loops automatically, it is still necessary to provide to the Code Perforation system a way to measure the result quality. Similarly, it is necessary to insert calls to their Heartbeat API [70] (their equivalent of the Throughput Frame). As a result, Code Perforation is not a completely automated solution, as it is still necessary for the application designer to provide a way to quantify the result quality. Note that later Code Perforation-related research, such as the Dynamic Knobs work [71], does not use the automatic Loop Perforation techniques espoused in the previous work, but instead develops a dynamic way to alter command-line parameters for the various PARSEC [14] benchmarks studied.

Another important aspect of OCCAM and Code Perforation is how they control the application and computer system at run time, since both platforms adapt the application and computer system at run time. The Code Perforation control system (called PowerDial) uses a simplistic model of the computer system that is based on the average performance of the benchmark across multiple heartbeats. OCCAM, on the other hand, uses a Markov Decision Process-based model to account for changing input data and system behavior so that the controller can predictively control the system to get the best results in the kinds of rapidly-changing environments that CPS often face. PowerDial, on the other hand, is a slow-changing, reactive controller that adjusts the application adaptation parameters after twenty or more heartbeats have passed.

In addition to Code Perforation, there are other works that investigate how specifically to make adaptable applications. The work done by Rinard [80], for example, discusses how to trade off in clustered applications computational complexity for performance by discarding some tasks that are to be executed. Other work (e.g. using adaptive precision reduction for physics computation [101], using variable precision floating point to improve the efficiency of CPS applications [31], and developing and using variable precision floating-point modules for FPGAs [13]) has investigated various techniques to vary the precision of either fixed point or floating-point arithmetic as a way to implement similar performance-result quality tradeoffs. While variable precision arithmetic is appealing for a variety of reasons (straightforward to implement in software, fine-grained precision variability is possible), it is difficult to implement in realizable hardware,
and fine-grained variable precision is not supported by most current hardware.

12.3 Thermal Control Techniques

Thermal control has been extensively studied throughout the literature. There are two key aspects of thermal control: how to control the hardware so as to prevent thermal emergencies; and given a certain set of hardware control techniques, what are the best strategies to control the system while minimizing performance loss.

The following papers describe a variety of different basic techniques for ensuring that the processor(s) stay within its thermal limit. First, Gomaa, et. al. [33] describe a technique for thermal management called “Heat and run”. “Heat and run” is a technique where the operating system schedules thermally “hot” tasks on one core for awhile until the temperature increases, and then moves them to another core once the temperature of the current core gets too hot. In a similar vein, Ghiasi and Grunwald [32] explored dual core thermal management as a way to keep a processor within its thermal threshold. This paper explored both symmetric and asymmetric dual core designs as ways to perform thermal management. At a finer-grained level, Seondmoo, Barr, and Asanović [39] studied performing dynamic thermal management by moving around computation across multiple replicated units. The goal of this technique is to reduce specific thermal hotspots within the processor. An interesting aspect of this paper is that it looks at migrating work between fine-grained replicated microarchitectural features.

The next two papers investigate fine-grained microarchitectural techniques for quickly reducing processor power consumption to keep a processor within its thermal limits. First, Skadron, Abdelzaher, and Stan [88] studied using certain fast-acting microarchitectural adaptation techniques, such as fetch toggling, as a way to keep the processor’s temperature within its thermal limits. Also included was a “control theoretic” part of the paper, which examined using Proportional-Integral-Derivative (PID) controllers to control this parameter. Next, Skadron [87] discusses combining fetch gating with DVFS as a way to improve the thermal management of a microprocessor, with the idea that fetch gating is more effective for mild thermal stress, and DVFS is more effective for
more serious thermal stress.

The next set of papers study other, miscellaneous techniques for providing thermal control. First, Merkel and Bellosa [69] present a scheduling technique for scheduling hotter and cooler tasks on a multiprocessor so as to maximize the performance while keeping the processors below their thermal threshold. The temperature characteristics are determined using profiling that measures the performance counters on processors. Next, Donald and Martonosi [29] explores various techniques for performing DVFS control on a multicore system. They find that distributed, per-core DVFS provides performance improvements. Moreover, when there is no per-core DVFS available, thermal-aware thread scheduling provides improvements as well. Finally, a multi-loop, control-theoretic DVFS approach improves performance as well. Next, Huang, et. al [43] studied combining a series of different energy/thermal management techniques for minimizing energy consumption while keeping the processor within its thermal limits. Finally, Chen, Dick, and Shang [20] presented a more accurate thermal modeling technique based on fourth-order Runge-Kutta methods.

The next set of papers present various scheduling techniques for providing thermal management. First, Yang, et. al. [100] present a scheduling technique for hot and cool tasks whose goal is to keep the processor below a certain temperature threshold. An interesting contribution of this paper is the observation that the most effective schedule is one where the processor temperature is kept right below the thermal threshold temperature. This property holds because heat removal is greatest at higher temperatures. Next, Kursun, et. al. [60] discuss various techniques for performing dynamic task scheduling so as to meet the processor’s thermal requirements. This paper mostly just performs thread scheduling and presents some promising results. Next, Yeo and Kim [102] provide another technique for scheduling threads on multicore processors to keep the processor within its thermal limits. Its main contributions are a clustering-based technique for grouping together the thermal behavior of different processes as well as a piecewise modeling technique that is the inspiration for the thermal modeling in OCCAM. Finally, Liu, et. al. [66] present the somewhat-surprising conclusion that minimizing energy consumption and keeping the
system within its thermal limits are not the same.

The final set of papers surveyed in this section discuss various control-based thermal management techniques. The first paper, by Srinivasan and Adve [89], is a GRACE-related paper. This paper targets dynamic thermal management for multimedia applications by proactively controlling the temperature. It uses predictive control techniques that attempt to model how the system will look in the future as a way to proactively apply hardware adaptation to the system. Next, Isci, et. al. [49] evaluated a series of different control techniques for performing global DVFS on a multicore processor. This control involved various feedback-based techniques to optimize overall system performance while keeping the multicore within its thermal envelope. The next paper, by Weissel and Bellosa [91], studies control techniques in the data center, by presenting a control-based approach to dynamic thermal management for data centers. It uses event counters on the processors to identify which tasks are influencing the temperature the most. Finally, it introduces an operating system framework to manage all of this. Finally, Jung and Pedram [53] present a hotspot alerting technique that involves determining the current state of the processor using a Kalman filter and a Partially Observable Markov Decision Process (POMDP) to determine what the system actually behaves like and what kinds of hot spots there are. This is somewhat similar to my work.

12.4 Phase Prediction

Phase prediction research involves a key component of OCCAM, its offline profiling, clustering-based system identification, and Markovian prediction scheme. Two key papers are discussed. First, Isci, Contreras, and Martonosi [50] study implementing in the processor a simple phase prediction table that allows for predicting the next program phase. This phase prediction mechanism is then used to adjust the DVFS mechanism accordingly in order to minimize processor energy consumption. Next, Sherwood, Sair, and Calder [84] describes a generic phase classification and phase prediction scheme. They develop a phase classifier that is simple enough that can be implemented in hardware, as well studying two types of phase predictors: a Markov-based run length
phase predictor that is better able to predict the future phases than just predicting that the next phase will be the same as the current phase.

12.5 DVFS-Based Techniques

The computer system hardware adaptation technique used by OCCAM is Dynamic Voltage and Frequency Scaling, or DVFS. DVFS was chosen to be the computer system adaptation technique for OCCAM because it is a powerful technique for trading off power and energy consumption for performance. DVFS is also a potentially difficult technique to use successfully in many control-based systems due to two key issues. The first issue is that DVFS’ relationship between performance (as described by processor clock frequency) and power consumption is highly nonlinear, ranging from a quadratic to cubic relationship depending on the exact implementation. Second, DVFS control is a highly discrete control problem. Many computer systems have a limited number of widely-spaced, discrete DVFS settings to choose from, so as a result, many of the continuous controller techniques, such as Proportional-Integral-Derivative (PID) systems, are difficult and/or infeasible to adapt to the DVFS control problem.

Various work within this section describes different ways in which DVFS is used to conserve energy. One issue with using DVFS in a multicore system concerns the issue of local vs. global DVFS: while local DVFS can potentially provide better-quality, more power-efficient execution of applications, it is considerably more difficult to successfully control all of the clock domains separately. Other work discusses controlling DVFS with various soft real-time applications, such as multimedia applications, while predicting the workload characteristics in order to make good decisions.

The following set of papers examines how to optimize a system to minimize energy/power consumption when there are multiple separate clock domains. The first paper, by Hughes, Srinivasan, and Adve [48], investigates combining global DVFS techniques with fine-grained microarchitectural techniques to provide adaptation of the hardware system for use by multimedia systems. The adaptation algorithm combined profiling to characterize the application and computer system
along with a predictor for predicting the system’s behavior. Next, Wu, et. al. [93] investigate how to control a microprocessor with multiple controllable clock domains. This work formally models the processor as a queue network and subsequently models the DVFS control problem as a feedback control problem and linearizes the behavior. Next, Xian, Lu, and Li [95] explored globally-optimal DVFS scheduling for multicore. The main contribution of this work was to develop a polynomial-time alternative to the NP-hard optimal scheduling problem.

The next set of papers examine formal control theory-based approaches to the problem of controlling DVFS in multiple clock domain microprocessors. First, Wu, et. al. [94] investigated how to optimally scale the DVFS settings of different regions of a microprocessor when the processor is architectured to have multiple clock domains. Their optimization is based on a type of state-space-based controller. Next, Juan, et. al. [52] developed a global method for controlling per core DVFS in multicore systems. It develops this control method by modeling the system using queuing theory that utilized a series of distributed PID controllers to control the system locally as well as to provide global control. Next, Yang, Chen, and Kuo [98] presented and evaluated an algorithm for scheduling on a chip multiprocessor with homogeneous CMP. The CMP components can either be off or on, and there is one global DVFS setting for the entire chip. Finally, Ogras, Marculescu, and Marculescu [77] provide a queuing theory-based technique where the authors model a Network-on-Chip multicore as a series of interconnected queues. This model is then used to control the system by creating a state space model and subsequently using that to control the system. Of note is that this controller also takes into account process variation of the NoC as well.

The following papers use either a control-theoretic framework or a framework very similar to a control-theoretic one to adjust the computer system’s DVFS setting(s) while allowing the application(s) to meet their real-time deadline(s). The first paper by Xu, Melem, and Mossé [96], compare various DVFS control methods against a stochastic DVFS method. The paper presents several different categories of DVFS-based task control: inter-task DVFS, intra-task DVFS, and a hybrid of the two. The paper also presents the notion of a “frame”, which is a periodically executing collection of tasks that periodically execute in a fixed order.
The next paper, by Hughes and Adve [46], is part of the GRACE project, and uses formal techniques to optimize the adaptation control. It uses Lagrange multipliers to adapt the system in order to minimize energy usage while ensuring that the multimedia application meets its real-time deadlines. Similarly, Sinha and Chandrakasan [86] determine how to adjust the DVFS settings by predicting what the actual characteristics of what the workload actually are. It performs this prediction by modeling the workloads using a Markov-like process and an adaptive FIR algorithm. Next, Lu, et. al. [68] describe a formal control-theoretic approach to controlling DVFS with multimedia applications, while Simunic, et. al. [85] use an algorithm based on detecting the multimedia application’s needs in order to adjust the DVFS.

The next set of papers investigates using a static DVFS control policy to determine the DVFS policy. First, Hsu and Ulrich [41] used the compiler to statically analyze a program’s code so as to insert DVFS decisions in the code. The key feature used by this compiler technique is to slow down the processor during execution points when the application is primarily memory-bound. In the next paper, Huang and Ghiasi [45] present a compiler design technique where the code is statically analyzed and optimal combined DVFS and ABB decisions are made. Finally, Hsu, Ulrich, and Hsiao [42] provide compiler-directed DVFS control setup that statically analyzes the code and attempts to minimize slack and maximize energy efficiency by controlling the DVFS of the processor.

12.6 Real-Time Power-Aware Scheduling

This section discusses DVFS-based real-time-specific scheduling techniques. These techniques can be considered part of the traditional real-time scheduling domain. Most of these techniques involve scheduling various tasks together in different ways along with changing the DVFS setting(s) of the processor(s).

The first set of papers are related to the Illinois GRACE project. This project encompasses many of the different techniques used in OCCAM. The first paper, by Yuan and Nahrstedt [103], discusses GRACE-OS, whose job is not only to schedule tasks, but also to determine what CPU
frequency is to be used for a given task. An interesting aspect of this paper is that they add in the notion of stochastic scheduling, which is intended to leverage the probabilistic execution probabilities of multimedia applications. Next, Yuan and Nahrstedt [104] also present a journal article that discusses the DVFS algorithms used by GRACE to minimize energy consumption. Finally, the paper by Sanska, Hughes, and Adve [81] make several different contributions. First, it focuses on adapting applications between different frames in multimedia applications as a coarse-grained adaptation point. This paper also combines global DVFS adaptation with faster-changing energy-adaptation algorithms such as clock throttling and instruction window resizing. The faster adaptation algorithms are used for adapting each frame, while the DVFS adaptation is used to adapt the system in between frames. Various control algorithms are also evaluated in the context of this adaptation.

The next three papers discuss slack reclamation-based techniques and procrastination scheduling. First, Kim, Kim, and Min [56] discuss characterizing and reclaiming the slack in a real-time system with periodic tasks. This reclaimed slack is then used to lower the DVFS settings. Next, Zhu, Melhem, and Childers [107] present a slack reclamation scheduling algorithm. In addition to characterizing and scheduling for slack, they also put in the DVFS adjustment delay into the scheduling algorithm. Finally, Jejurikar and Gupta [51] discuss using procrastination scheduling to coalesce together large blocks of idle time as a way to reduce leakage power. Procrastination scheduling is a technique where the tasks are scheduled as late as possible, before their respective deadlines.

The next two papers present various techniques for performing DVFS scaling on a hard real-time system. The first paper, by Krishna and Lee [58], describe a technique for performing adaptive voltage scaling on hard real-time systems. Next, Aydin, et. al. [10] present an $O(n^2 \log(n))$ algorithm for scheduling hard real-time tasks so as to provide a DVFS schedule.

The final set of papers describe various other DVFS and power-aware scheduling techniques. The first paper, by Lorch and Smith [67], describes the PACE system. This system provides a set of algorithms that adjust the system’s DVFS settings in such a way as to allow for meeting the
real-time deadlines of various tasks. Their system is unique in that it progressively raises the DVFS settings as a task’s deadline approaches. Next, Heath, Pinheiro, and Bianchini [37] present techniques for transforming applications so as to maximize the duration of idle times. Lengthening the duration of the idle times allows for reducing power by enabling devices to sleep/power off more. The next paper, by Yang, Chen, and Kuo [99], study how to use the multiframe model discussed in Mok and Chen’s work [73] to schedule the system so as to reduce energy consumption using DVFS.

Next, Aydin and Yang [11] discuss a technique for performing multiprocessor task scheduling and DVFS control. This technique’s key feature is partitioning, where the tasks are assigned to processors first, and then well-established DVFS scheduling techniques are used for each processor. After that, Gruian [35] discusses a stochastic technique for determining the task schedule and DVFS schedule for hard real-time systems. Finally, Aydi, et. al. [9] describe a set of speed adjustment algorithms. The purpose of these algorithms is to first provide a static speed schedule based on WCET. Next are a series of dynamic adjustments that are used to adapt to differing actual execution characteristics. Finally, Andrew, Lin, and Wierman [7] formally discuss DVFS-aware scheduling from a robustness, fairness, and optimality standpoint.

12.7 Hardware/Software Trends

This final section of this chapter discusses other miscellaneous papers that describe trends that played an important role in the development of OCCAM. The first paper, by Hughes, et. al. [47], tests the conjecture that dynamic execution microarchitectural features cause execution performance variability in multimedia applications. It concludes that they do not for multimedia applications, with most of the variability being caused by algorithm and input data-related variability. Next, Borkar, et. al. [15] provides a summary discussion of increasing process variation in future microprocessors and how it will affect microprocessor design, such as causing different processors to have different thermal characteristics.

The next set of papers show different aspects of OCCAM being studied by other researchers.
First, Kestor, et. al. [55] describe a Recognition, Mining, and Synthesis (RMS) benchmark suite designed to test the performance of transactional memory systems, which is similar to the benchmarks used to test OCCAM. Next, Huang and Ghiasi [44] present a technique for using both Adaptative Body Biasing and DVFS to optimally reduce both the dynamic as well as static power consumption. Such a technique provides a way for OCCAM’s DVFS control can be used in a situation where transistor leakage is a significant problem by also adjusting the transistor body biasing to trade off performance for reduced leakage. Next, Mok and Chen [73] cast aside the traditional periodic task scheduling model of Liu and Layland’s rate monotonic scheduling in favor of dividing applications’ execution characteristics into a series of different frames with different execution times. This technique allows for greater scheduling flexibility in that the system does not have to pessimistically schedule for only the worst-case overall execution time. Finally, Brooks and Martonosi [16] point out that the average and maximum power consumption in a processor varies more widely in the past because of the aggressive use of clock gating techniques. This paper makes a key contribution by laying out the theoretical groundwork for future dynamic thermal management by directing where future thermal management work should go: trigger selection, allowing chip designers to focus on average power, trigger activation time, lightweight policies are effective, and to look for thermal management techniques that improve power consumption more than they hurt performance.
Chapter 13

Summary And Conclusions

This thesis presented OCCAM, a robust framework that enables Cyber-Physical Systems applications to meet their performance requirements while optimizing resource usage in the midst of changing system dynamics. It provides these facilities by systematically exploring and rigorously applying portable, application-specific run-time optimization with low overhead, with an average overhead of 6.0%. By providing a programming framework to allow the developer to provide information about the performance requirements and application-specific adaptation, OCCAM can provide portability between different computer systems with different resource availabilities. OCCAM’s control-based runtime system uses a stochastic Model Predictive Controller (MPC) to handle the highly variable and nonlinear system dynamics. This controller is generated offline, using profiling. The MPC distills the system’s behavior into a compact, Markov chain-based representation. Finally, using a cost function based on the design time performance requirements as well as based on the run time resource utilization, the MPC determines the best optimization decisions to make.

Successfully providing robust, heterogeneous thermal control to OCCAM requires two things: accurate thermal models of the processing cores, and a discrete, Markov Decision Process (MDP)-based model of the system that is small enough to be solved in a tractable manner. For the thermal modeling, this thesis demonstrated that a logarithmic model of the processor cores’ heating and cooling model can successfully predict future processor temperatures. Heterogeneous thermal control support to OCCAM’s MPC can be provided by using several different techniques for reducing
the number of states in the MDP’s search space. The first technique reduces the number of thermal states that need to be accounted for by only considering the temperature of the hottest core in each of the frequency scaling units. To further reduce the search space, the frequency scaling decisions are “prescheduled” so that higher frequency scaling decisions are placed on hotter scaling units first. Finally, before using the MDP solver to determine a final, infinite-horizon solution, the search space is pruned to only use a small subset of the possible decisions.

The result is a system that allows an application, written once, to efficiently meet its performance requirements on a large range of systems. If the system is highly resource-constrained, OCCAM can use developer-specified relaxed performance requirements to allow the CPS application to run in a correct, but lower fidelity manner. This thesis successfully demonstrated that OCCAM can effectively adapt applications at run time to optimize system resource usage while meeting the system’s performance requirements on systems ranging from an energy-constrained, single-core system to a high performance, power-constrained 16-core system without rewriting the application.

This system successfully scales to controlling a 16-core computer system while enjoying low overhead, ranging from 0.1% in histogram to 19.2% in wordcount, with an average overhead of 6.0%. Using OCCAM’s thermal-aware Model Predictive Controller, the system reduces the number of frames that exceed a thermal limit for most of the benchmarks on both a dual-core notebook system as well as on a 16-core server system. Furthermore, this thermal control is accomplished with a modest impact to the non-thermal cost function. Moreover, this thesis showed that, by leveraging application-specific knowledge, OCCAM makes better online adaptation decisions, resulting in faster adaptation, and better success at meeting application performance requirements. Finally, this thesis showed that OCCAM can successfully control applications running on systems with insufficient compute resources by relaxing the application’s performance requirements in a structured manner. Finally, this thesis presented a simulation-based, stochastic model checking-based framework that allows for quantifying the robustness of the controller.
Appendix A

Benchmark Constraints

This appendix provides an explicit list of the different constraint levels for the seven different benchmarks. Listed are three different values: the timing constraints, the quality constraints, and the cost. The timing constraints are the maximum amount of time that a Throughput Task can take to execute and still meet a given constraint level. Likewise, Quality measurements provide the highest Quality level (as determined by the developer-provided getQuality() method) that allows for meeting a given constraint level. Note that when determining constraint levels, the “worst” constraint level is chosen as the constraint level for that Throughput Task. Finally, the cost measurement is used by the Model Predictive Controller to determine the cost of meeting a certain constraint level. This cost value is determined by the application developer because only the developer knows the relative importance of the system meeting various constraint levels.

<table>
<thead>
<tr>
<th>Timing Constraints (s)</th>
<th>Quality Constraints</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9300000</td>
<td>0.00E+000</td>
<td>0.50E+009</td>
</tr>
<tr>
<td>1.8600000</td>
<td>1.00E+000</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>3.7200000</td>
<td>1.00E+000</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>7.4400000</td>
<td>2.00E+000</td>
<td>3.00E+009</td>
</tr>
</tbody>
</table>

Table A.1: Constraints for hearing aid.
<table>
<thead>
<tr>
<th>Timing Constraints (s)</th>
<th>Quality Constraints</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>-4.00E-005</td>
<td>2.50E+008</td>
</tr>
<tr>
<td>0.15</td>
<td>-3.90E-005</td>
<td>5.00E+008</td>
</tr>
<tr>
<td>0.2</td>
<td>-3.80E-005</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>0.3</td>
<td>-3.70E-005</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>0.4</td>
<td>-3.60E-005</td>
<td>3.00E+009</td>
</tr>
<tr>
<td>0.5</td>
<td>-3.50E-005</td>
<td>4.00E+009</td>
</tr>
<tr>
<td>0.6</td>
<td>-3.40E-005</td>
<td>5.00E+009</td>
</tr>
<tr>
<td>0.7</td>
<td>-3.30E-005</td>
<td>6.00E+009</td>
</tr>
<tr>
<td>0.8</td>
<td>-3.20E-005</td>
<td>7.00E+009</td>
</tr>
<tr>
<td>0.9</td>
<td>-3.10E-005</td>
<td>8.00E+009</td>
</tr>
<tr>
<td>1</td>
<td>-3.00E-005</td>
<td>9.00E+009</td>
</tr>
<tr>
<td>1.1</td>
<td>-2.90E-005</td>
<td>1.00E+010</td>
</tr>
<tr>
<td>1.2</td>
<td>-2.80E-005</td>
<td>1.10E+010</td>
</tr>
<tr>
<td>1.3</td>
<td>-2.70E-005</td>
<td>1.20E+010</td>
</tr>
<tr>
<td>1.4</td>
<td>-2.60E-005</td>
<td>1.30E+010</td>
</tr>
<tr>
<td>1.5</td>
<td>-2.50E-005</td>
<td>1.40E+010</td>
</tr>
<tr>
<td>1.6</td>
<td>-2.50E-005</td>
<td>1.50E+010</td>
</tr>
<tr>
<td>1.7</td>
<td>-2.50E-005</td>
<td>1.60E+010</td>
</tr>
<tr>
<td>1.8</td>
<td>-2.50E-005</td>
<td>1.70E+010</td>
</tr>
<tr>
<td>1.9</td>
<td>-2.50E-005</td>
<td>1.80E+010</td>
</tr>
<tr>
<td>2.1</td>
<td>-2.50E-005</td>
<td>1.90E+010</td>
</tr>
<tr>
<td>2.2</td>
<td>-2.50E-005</td>
<td>2.00E+010</td>
</tr>
<tr>
<td>2.3</td>
<td>-2.50E-005</td>
<td>2.10E+010</td>
</tr>
<tr>
<td>2.4</td>
<td>-2.50E-005</td>
<td>2.20E+010</td>
</tr>
<tr>
<td>2.5</td>
<td>-2.50E-005</td>
<td>2.30E+010</td>
</tr>
<tr>
<td>2.6</td>
<td>-2.50E-005</td>
<td>2.40E+010</td>
</tr>
<tr>
<td>2.7</td>
<td>-2.50E-005</td>
<td>2.50E+010</td>
</tr>
<tr>
<td>2.8</td>
<td>-2.50E-005</td>
<td>2.60E+010</td>
</tr>
<tr>
<td>2.9</td>
<td>-2.50E-005</td>
<td>2.70E+010</td>
</tr>
<tr>
<td>3.0</td>
<td>-2.50E-005</td>
<td>2.80E+010</td>
</tr>
<tr>
<td>3.1</td>
<td>-2.50E-005</td>
<td>2.90E+010</td>
</tr>
<tr>
<td>3.2</td>
<td>-2.50E-005</td>
<td>3.00E+010</td>
</tr>
</tbody>
</table>

Table A.2: Constraints for histogram.

<table>
<thead>
<tr>
<th>Timing Constraints (s)</th>
<th>Quality Constraints</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.17</td>
<td>-2.00E+008</td>
<td>0.50E+009</td>
</tr>
<tr>
<td>0.34</td>
<td>-5.00E+007</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>0.68</td>
<td>-2.00E+007</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>1.36</td>
<td>0.00E+000</td>
<td>3.00E+009</td>
</tr>
</tbody>
</table>

Table A.3: Constraints for lr.
### Table A.4: Constraints for stereo.vision.

<table>
<thead>
<tr>
<th>Timing Constraints (s)</th>
<th>Quality Constraints</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3.00E-002</td>
<td>0.50E+008</td>
</tr>
<tr>
<td>4</td>
<td>4.00E-002</td>
<td>1.00E+008</td>
</tr>
<tr>
<td>4.2</td>
<td>5.00E-002</td>
<td>2.00E+008</td>
</tr>
<tr>
<td>4.4</td>
<td>6.00E-002</td>
<td>3.00E+008</td>
</tr>
<tr>
<td>4.6</td>
<td>7.00E-002</td>
<td>4.00E+008</td>
</tr>
<tr>
<td>4.8</td>
<td>8.00E-002</td>
<td>5.00E+008</td>
</tr>
<tr>
<td>4.9</td>
<td>9.00E-002</td>
<td>6.00E+008</td>
</tr>
<tr>
<td>5</td>
<td>10.00E-002</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>6</td>
<td>11.00E-002</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>7</td>
<td>12.00E-002</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>8</td>
<td>13.00E-002</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>9</td>
<td>14.00E-002</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>10</td>
<td>15.00E-002</td>
<td>3.00E+009</td>
</tr>
</tbody>
</table>

### Table A.5: Constraints for tachyon.

<table>
<thead>
<tr>
<th>Timing Constraints (s)</th>
<th>Quality Constraints</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>2.50E-001</td>
<td>0.50E+009</td>
</tr>
<tr>
<td>1.00</td>
<td>1.25E-001</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>2.00</td>
<td>1.25E-001</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>4.00</td>
<td>1.25E-001</td>
<td>3.00E+009</td>
</tr>
</tbody>
</table>

Table A.4: Constraints for `stereo.vision`.

Table A.5: Constraints for `tachyon`. 
### Table A.6: Constraints for seismic.

<table>
<thead>
<tr>
<th>Timing Constraints (s)</th>
<th>Quality Constraints</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0100000</td>
<td>0.00E+000</td>
<td>0.06E+009</td>
</tr>
<tr>
<td>0.0200000</td>
<td>1.00E+000</td>
<td>0.07E+009</td>
</tr>
<tr>
<td>0.0300000</td>
<td>1.00E+000</td>
<td>0.08E+009</td>
</tr>
<tr>
<td>0.0400000</td>
<td>1.00E+000</td>
<td>0.09E+009</td>
</tr>
<tr>
<td>0.0500000</td>
<td>1.00E+000</td>
<td>0.10E+009</td>
</tr>
<tr>
<td>0.0600000</td>
<td>1.00E+000</td>
<td>0.20E+009</td>
</tr>
<tr>
<td>0.0700000</td>
<td>1.00E+000</td>
<td>0.30E+009</td>
</tr>
<tr>
<td>0.0800000</td>
<td>1.00E+000</td>
<td>0.50E+009</td>
</tr>
<tr>
<td>0.0900000</td>
<td>1.00E+000</td>
<td>0.60E+009</td>
</tr>
<tr>
<td>0.1000000</td>
<td>1.00E+000</td>
<td>0.70E+009</td>
</tr>
<tr>
<td>0.1100000</td>
<td>1.00E+000</td>
<td>0.80E+009</td>
</tr>
<tr>
<td>0.1200000</td>
<td>1.00E+000</td>
<td>0.90E+009</td>
</tr>
<tr>
<td>0.1334000</td>
<td>1.00E+000</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>0.2668000</td>
<td>1.00E+000</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>0.5336000</td>
<td>1.00E+000</td>
<td>3.00E+009</td>
</tr>
<tr>
<td>1.0672000</td>
<td>1.00E+000</td>
<td>4.00E+009</td>
</tr>
<tr>
<td>2.1344000</td>
<td>1.00E+000</td>
<td>5.00E+009</td>
</tr>
</tbody>
</table>

### Table A.7: Constraints for tachyon.

<table>
<thead>
<tr>
<th>Timing Constraints (s)</th>
<th>Quality Constraints</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.27</td>
<td>2.00E-005</td>
<td>0.50E+009</td>
</tr>
<tr>
<td>0.54</td>
<td>2.00E-005</td>
<td>1.00E+009</td>
</tr>
<tr>
<td>1.08</td>
<td>2.00E-005</td>
<td>2.00E+009</td>
</tr>
<tr>
<td>2.16</td>
<td>2.00E-005</td>
<td>3.00E+009</td>
</tr>
</tbody>
</table>
Appendix B

Thermal Logmodel Graphs

The purpose of these graphs is to study how well the logarithmic model, which fits the observed temperature data to a log function via least squares, performs in the presence of thermal throttling. This logarithmic model is used by OCCAM-THERMAL to predict the future temperatures of the various processor cores. OCCAM-THERMAL assumes the existence of some sort of thermal throttling, where if OCCAM-THERMAL cannot keep the CPU cores temperatures within their thermal limits, then some sort of hardware technique, such as clock gating, power gating, or instruction fetch toggling, is quickly employed to bring down the temperature. A major issue with thermal throttling, however, is that it may harm the ability of the logarithmic model to obtain an accurate model of the thermal behavior. As a result, the logarithmic models are also compared with setups where the thermal constraints are simulated by removing thermal values above a given thermal threshold and subsequently fitting the remaining data to the logarithmic model.

The thermal model is examined on the dual core system, HI, which is a dual core Intel Core 2 Duo notebook. The processor cores have four possible DVFS settings: \(800 \text{MHz}\), \(1200 \text{MHz}\), \(1600 \text{MHz}\), and \(2000 \text{MHz}\). For each of the seven benchmarks, for each of the two CPU cores, each frequency is tested in two ways. The first way is called cool: cool models the temperature after a benchmark run at the processors lowest frequencies. The second way is called heat: heat models the temperature after a benchmark run at the processors highest frequencies. Both models are generated in order to ensure that the entire possible temperature range of the processor cores is covered by at least one of the models.
Figure B.1: Thermal model comparison for histogram on CPU1 (cool)
Figure B.2: Thermal model comparison for histogram on CPU1 (heat)
$CPU = 1$, cool, $Frequency = 800MHz$

$CPU = 1$, heat, $Frequency = 800MHz$

$CPU = 1$, cool, $Frequency = 1200MHz$
$CPU = 1$, heat, $Frequency = 1200MHz$

$CPU = 1$, cool, $Frequency = 1600MHz$

$CPU = 1$, heat, $Frequency = 1600MHz$
CPU = 1, cool, Frequency = 2000MHz

CPU = 1, heat, Frequency = 2000MHz
Appendix C

Setpoints for Cascaded PID Controllers

This appendix provides the constants and setpoints that were chosen for the cascaded PID controller, called *OCCAM-STOCHASTIC*, used for the initial studies of OCCAM’s runtime. These PID values were determined (i.e., the PID controller was tuned) manually. It is worth noting that the PID (Proportional-Integral-Derivative) controllers are really PI (Proportional-Integral) controllers, as the Derivative factor was found to not be particularly useful.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hearing_aid</td>
<td>(0.0, 0.0, 0.0)</td>
<td>(2.5, 1.0, 0.0)</td>
<td>(0.08, 0.0, 0.0)</td>
</tr>
<tr>
<td>histogram</td>
<td>(0.75 × 10^6, 0.25 × 10^6, 0.0)</td>
<td>(20.0, 0.0, 0.0)</td>
<td>(0.08, 0.0, 0.0)</td>
</tr>
<tr>
<td>lr</td>
<td>(2.0 × 10^-8, 0.0, 0.0)</td>
<td>(20.0, 0.0, 0.0)</td>
<td>(0.25, 0.1, 0.0)</td>
</tr>
<tr>
<td>stereo_vision</td>
<td>(50.0, 20.0, 0.0)</td>
<td>(2.0, 0.5, 0.0)</td>
<td>(0.05, 0.01, 0.0)</td>
</tr>
<tr>
<td>seismic</td>
<td>(0.0, 0.0, 0.0)</td>
<td>(15.0, 5.0, 0.0)</td>
<td>(0.25, 0.0, 0.0)</td>
</tr>
<tr>
<td>tachyon</td>
<td>(10.0, 2.0, 0.0)</td>
<td>(3.0, 1.0, 0.0)</td>
<td>(4.0, 0.01, 0.0)</td>
</tr>
<tr>
<td>wordcount</td>
<td>(1.5 × 10^5, 0.0, 0.0)</td>
<td>(20.0, 1.0, 0.0)</td>
<td>(0.25, 0.1, 0.0)</td>
</tr>
</tbody>
</table>

Table C.1: P, I, and D values for the different PID controllers.
Bibliography

[1] Libxtract.


[57] Younghyun Kim, Youngjin Cho, Naehyuck Chang, Chaitali Chakrabarti, and Nam Ik Cho. Extending the lifetime of media recorders constrained by battery and flash memory size.
In ISLPED ’08: Proceeding of the thirteenth international symposium on Low power electronics and design, pages 159–164, New York, NY, USA, 2008. ACM.


[82] Muhammad Shafique, Lars Bauer, and Jörg Henkel. 3-tier dynamically adaptive power-aware motion estimator for h.264/avc video encoding. In ISLPED ’08: Proceeding of the thirteenth international symposium on Low power electronics and design, pages 147–152, New York, NY, USA, 2008. ACM.


