Title
I/O Design and Core Power Management Issues in Heterogeneous Multi/Many-Core System-on-Chip

Permalink
https://escholarship.org/uc/item/9g80m9bd

Author
Kim, Myoungseo

Publication Date
2016-01-01

Peer reviewed|Thesis/dissertation
UNIVERSITY OF CALIFORNIA, IRVINE

I/O Design and Core Power Management Issues in Heterogeneous Multi/Many-Core System-on-Chip

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Myoung-Seo Kim

Dissertation Committee:

Professor Jean-Luc Gaudiot, Chair
Professor Alexandru Nicolau, Co-Chair
Professor Alexander Veidenbaum

2016
DEDICATION

To my father and mother,
Youngkyu Kim and Heesook Park
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ix</td>
</tr>
<tr>
<td>CURRICULUM VITAE</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRACT OF THE DISSERTATION</td>
<td>xv</td>
</tr>
</tbody>
</table>

## I DESIGN AUTOMATION FOR CONFIGURABLE I/O INTERFACE CONTROL BLOCK 1

1 Introduction 2

2 Related Work 4

3 Structure of Generic Pin Control Block 6

4 Specification with Formalized Text 9
   4.1 Formalized Text 9
   4.2 Specific Functional Requirement 11
   4.3 Composition of Registers 11

5 Experiment Results 18

6 Conclusions 24

## II SPEED UP MODEL BY OVERHEAD OF DATA PREPARATION 26

7 Introduction 27

8 Reconsidering Speedup Model by Overhead of Data Preparation (ODP) 29
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Case Studies of Our Enhanced Amdahl’s Law Speedup Model</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>9.1 Homogeneous Symmetric Multicore</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>9.2 Homogeneous Asymmetric Multicore</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>9.3 Homogeneous Dynamic Multicore</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>9.4 Heterogeneous CPU-GPU Multicore</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>9.5 Heterogeneous Dynamic CPU-GPU Multicore</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>Conclusions</td>
<td>43</td>
</tr>
<tr>
<td>III</td>
<td>EFFICIENT CORE POWER CONTROL SCHEME</td>
<td>44</td>
</tr>
<tr>
<td>11</td>
<td>Introduction</td>
<td>45</td>
</tr>
<tr>
<td>12</td>
<td>Related Work</td>
<td>47</td>
</tr>
<tr>
<td>13</td>
<td>Architecture</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>13.1 Heterogeneous Many-Core System</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>13.2 Discrete L2 Cache Memory Model</td>
<td>52</td>
</tr>
<tr>
<td>14</td>
<td>3-Bit Power Control Scheme</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>14.1 Active Status</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>14.2 Hot Core Status</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>14.3 Cold Core Status</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>14.4 Idle Status</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>14.5 Powered Down Status</td>
<td>61</td>
</tr>
<tr>
<td>15</td>
<td>Power-Aware Thread Placement</td>
<td>64</td>
</tr>
<tr>
<td>16</td>
<td>Evaluation And Methodology</td>
<td>70</td>
</tr>
<tr>
<td>17</td>
<td>Expanded Works</td>
<td>81</td>
</tr>
<tr>
<td>18</td>
<td>Conclusions</td>
<td>82</td>
</tr>
<tr>
<td>IV</td>
<td>POWER-ENERGY EFFICIENCY MODEL BY OVERHEAD OF DATA PREPARATION</td>
<td>84</td>
</tr>
<tr>
<td>19</td>
<td>Introduction</td>
<td>85</td>
</tr>
<tr>
<td>20</td>
<td>Related Work</td>
<td>87</td>
</tr>
<tr>
<td>21</td>
<td>Power-Energy Efficiency Model of Heterogeneous Multicore System</td>
<td>90</td>
</tr>
<tr>
<td>22</td>
<td>Evaluation and Analysis</td>
<td>92</td>
</tr>
<tr>
<td>23</td>
<td>Conclusions</td>
<td>98</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>A</td>
<td>Sniper: Scalable and Accurate Parallel Multi-Core Simulator</td>
<td>113</td>
</tr>
<tr>
<td>A.1</td>
<td>Intel Nehalem Architecture</td>
<td>114</td>
</tr>
<tr>
<td>A.2</td>
<td>Interval Simulation</td>
<td>117</td>
</tr>
<tr>
<td>A.3</td>
<td>Multi-Core Interval Simulator</td>
<td>118</td>
</tr>
<tr>
<td>A.4</td>
<td>Instruction-Window Centric Core Model</td>
<td>120</td>
</tr>
<tr>
<td>B</td>
<td>Parsec and Splash-2: Benchmark Suite</td>
<td>121</td>
</tr>
<tr>
<td>B.1</td>
<td>Overview of Workloads and the Used Inputs</td>
<td>122</td>
</tr>
<tr>
<td>B.2</td>
<td>Program Characteristics</td>
<td>123</td>
</tr>
<tr>
<td>C</td>
<td>McPAT: Power Analysis Framework for Multi-Core Architectures</td>
<td>125</td>
</tr>
<tr>
<td>C.1</td>
<td>Operation</td>
<td>126</td>
</tr>
<tr>
<td>C.2</td>
<td>Type of Representative Architecture-Level Power Estimator</td>
<td>127</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Core Architecture of a Generic Pin Control Block.</td>
<td>7</td>
</tr>
<tr>
<td>4.1</td>
<td>Functions and Parameters.</td>
<td>10</td>
</tr>
<tr>
<td>4.2</td>
<td>Formalized Description of Our Automated Design Scheme.</td>
<td>12</td>
</tr>
<tr>
<td>4.3</td>
<td>Control Property Definition in a Formalized Text.</td>
<td>13</td>
</tr>
<tr>
<td>4.4</td>
<td>Composition of a Specific Register Group.</td>
<td>14</td>
</tr>
<tr>
<td>4.5</td>
<td>Composition of Port Control Registers.</td>
<td>16</td>
</tr>
<tr>
<td>5.1</td>
<td>An Example of an Execution Model by the Auto-Generator.</td>
<td>19</td>
</tr>
<tr>
<td>5.2</td>
<td>Composition of Multimedia SoC Platforms.</td>
<td>20</td>
</tr>
<tr>
<td>5.3</td>
<td>Quantitative Analysis in a Typical Multimedia SoC Platform.</td>
<td>22</td>
</tr>
<tr>
<td>5.4</td>
<td>Design Volume in Multimedia SoC Platforms about Generic and PAD Pins.</td>
<td>23</td>
</tr>
<tr>
<td>8.1</td>
<td>Normalized Task (Equivalence Time), Split between Computation and Data Preparation.</td>
<td>30</td>
</tr>
<tr>
<td>9.1</td>
<td>Speedup Distribution of Homogeneous Symmetric Multicore where $p_c = 0.6$, $f_h = 0.8$.</td>
<td>37</td>
</tr>
<tr>
<td>9.2</td>
<td>Speedup Distribution of Homogeneous Asymmetric Multicore where $p_c = 0.6$, $f_h = 0.8$.</td>
<td>38</td>
</tr>
<tr>
<td>9.3</td>
<td>Speedup Distribution of Heterogeneous CPU-GPU Multicore where $p_c = 0.6$, $f_h = 0.8$, $i = 4$.</td>
<td>40</td>
</tr>
<tr>
<td>9.4</td>
<td>Speedup Distribution of Heterogeneous Dynamic CPU-GPU Multicore where $p_c = 0.6$, $f_h = 0.8$, $i = 4$.</td>
<td>42</td>
</tr>
<tr>
<td>13.1</td>
<td>Heterogeneous Many-Core Architecture.</td>
<td>53</td>
</tr>
<tr>
<td>13.2</td>
<td>4-Way Cuckoo Directory Structure.</td>
<td>54</td>
</tr>
<tr>
<td>14.1</td>
<td>3-bit Core Power Control Scheme under FSM.</td>
<td>57</td>
</tr>
<tr>
<td>14.2</td>
<td>3-bit Core Power Control Scheme under the Operating Sequence.</td>
<td>58</td>
</tr>
<tr>
<td>14.3</td>
<td>Power and Clock Distribution.</td>
<td>59</td>
</tr>
<tr>
<td>15.1</td>
<td>Hardware-Software Thread Interaction.</td>
<td>66</td>
</tr>
<tr>
<td>15.2</td>
<td>Outline of Heuristic Thread Cosolidation Method.</td>
<td>69</td>
</tr>
<tr>
<td>16.1</td>
<td>Architectural Topology of FFT and FFT-HETERO Test Case. - Generated Results from McPAT framework.</td>
<td>74</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1</td>
<td>Processor Power Design Space</td>
<td>62</td>
</tr>
<tr>
<td>14.2</td>
<td>Each Core Power Approximate Calculation</td>
<td>63</td>
</tr>
<tr>
<td>16.1</td>
<td>Simulation Configuration Parameters</td>
<td>71</td>
</tr>
<tr>
<td>16.2</td>
<td>Feature's Summary of Existing Well-Known Simulators</td>
<td>72</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

First of all, I would like to thank and praise God to give me wisdom, knowledge, and strength, that I make all these possible against every temptation and adversity. I am also deeply respectful and grateful to my advisor and co-advisor, Professor Jean-Luc Gaudiot (IEEE Fellow and 2017 IEEE Computer Society President) and Professor Alexandru Nicolau (IEEE Fellow), for their encouragement, guidance and patience during my study. I was very fortunate to meet them as an advisor and a co-advisor. In addition, I have learned many aspects of computer science and engineering from their incredible and creative insight and been inspired by their passion for research.

I would also like to say ‘thank you’ to my committee member: Professor Alexander Veidenbaum for his kind support, encouragement and trust. I wish to express best regards and blessing to all my colleagues in PArallel Systems & Computer Architecture Lab (PASCAL).

Additional support is provided by the National Science Foundation under Grant No. CCF-1065448. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Special thanks to Korean graduate student members for being my great supporter and for helping me on everything to be enriched the life in Irvine.

Finally, I give my sincerest gratitude and honor to my family who have patiently supported and prayed for me to move forward and to achieve my dream finally.
CURRICULUM VITAE

Myoung-Seo Kim

EDUCATION

Doctor of Philosophy in Computer Science 2012–2016
University of California Irvine, California

* M.S. in Electrical and Computer Engineering at University of California Irvine, California

Master of Science in Computer Science 2003–2005
Yonsei University Seoul, Korea

Bachelor of Science in Computer Science 1999–2003
Bachelor of Science in Electrical and Electronic Engineering
Yonsei University Seoul, Korea

RESEARCH EXPERIENCE

Graduate Student Researcher 2012–2016
Project: National Science Foundation Project
University of California Irvine, California

Research Engineer 2008–2009
Team: Physical SoC Design Team
Apple Inc. Cupertino, California

Research Engineer 2005–2008
Team: Application Processor Development Team
Samsung Electronics, Semiconductor Business Yongin, Korea

Graduate Student Researcher 2003–2005
Project: Brain Korea 21 Project
Yonsei University Seoul, Korea

Undergraduate Student Researcher 1999–2003
Project: Yonsei-Samsung Joint Project
Yonsei University Seoul, Korea
TEACHING EXPERIENCE

Teaching Assistant 2014–2016
Advanced System-on-Chip Design course,
Embedded System Design course,
Computer Organization course,
Data Structure Implementation and Analysis course
University of California Irvine, California

Teaching Assistant 2003–2005
Advanced Computer Architecture course,
Computer Architecture course,
Digital Logic Design course
Yonsei University Seoul, Korea
REFEREED JOURNAL PUBLICATIONS

[1st Author Work: J1]

International Journal of High Performance Computing and Networking (SCOPUS)

[J1-3] Evaluating the Overhead of Data Preparation for Heterogeneous Multicore System 2016 (Under Review)
KSII Transactions on Internet and Information Systems (SCIE/SCOPUS)

IEEE Embedded Systems Letters (SCOPUS)

[J1-1] Design of configurable I/O pin control block for improving reusability in multimedia SoC platforms 2015
Multimedia Tools and Applications (SCIE/SCOPUS/EI)

REVIEWED CONFERENCE PUBLICATIONS

[1st Author Work: C1]

[C1-7] Survey about Smart System-on-Chip of Embedded Devices for Internet of Things January 2016
International Conference on EEECS: Innovation and Convergence

[C1-6] Supercapacitor’s Application for Power Awareness in Internet of Things January 2016
International Conference on EEECS: Innovation and Convergence

International Conference on EEECS: Innovation and Convergence

[C1-4] An Efficient I/O Interface Control Block Design Methodology for Application-Specific MPSoC Platforms May 2014
5th IEEE International Conference on Information Science and Applications

[C1-3] Reduced 44 block intra prediction modes using directional similarity in H.264/AVC Sep 2007
7th ACM International Conference on Multimedia, Internet and Video Technologies
The IEEE International SoC Design Conference

[C1-1] A Slope-Based Search Technique for Block Motion Estimation Jun 2004
The International Conference on Imaging Science, Systems and Technology

[2nd Author Work: C2]

6th IEEE International Conference on Green and Sustainable Computing Conference

[C2-1] Fusion Coherence: Scalable Cache Coherence for Heterogeneous Kilo-Core System Aug 2014
10th Annual Conference on Advanced Computer Architecture

[3rd Author Work: C3]

[C3-3] Implementation of a Low Cost Home Energy Saver Based on OpenWrt Dec 2012
The IEEE International Conference on IT Convergence and Security

The International Conference on Embedded Systems and Applications

[C3-1] An Effective Intra-Prediction Systolic Array Architecture Oct 2004
The IEEE International SoC Design Conference
AWARDS AND ACHIEVEMENTS

MPSoC Architecture Design Contest  May2013
Award
Automotive Semiconductor Global Forum  Oct2012
Award
Yonsei International Foundation  Jun2011
Scholarship
Samsung Innovation  Feb2008
Award
Brain Korea 21  Mar2003
Scholarship

PROFESSIONAL SERVICES

Journal Referee
Elsevier Information Sciences (SCI)

Journal Referee
Elsevier Parallel Computing (SCIE)

Journal Referee
Springer Soft Computing (SCIE)

Technical Program Committee
Global Summit on Computer and Information Technology

Technical Program Committee
International Conference on Architectural Research and Computer-Aided Design

Conference Reviewer
The IEEE International Conference on Computer Technologies Innovations and Applications

Conference Reviewer
The IEEE International Circuits and Systems Symposium

Conference Reviewer
The IEEE International Conference on Mobile and Wireless Technologies

Conference Subreviewer
26th International Symposium on Computer Architecture and High Performance Computing
ABSTRACT OF THE DISSERTATION

I/O Design and Core Power Management Issues in Heterogeneous Multi/Many-Core System-on-Chip

By

Myoung-Seo Kim

Doctor of Philosophy in Computer Science

University of California, Irvine, 2016

Professor Jean-Luc Gaudiot, Chair
Professor Alexandru Nicolau, Co-Chair

Since dark silicon and the end of multicore scaling, multi/many-core system-on-a-chip (SoC) platform designs nowadays are facing some conflicting issues regarding product development. One is induced by increasing design complexity and another is induced by decreasing time-to-market. Hence, designers are seeking a more efficient and reliable methodology in order to design complex multimillion gate SoC under such harsh conditions.

In particular, the complexity of a generic pin control block in multimedia SoC which implements input/output (I/O) paths for off-chip communication has increased exponentially in recent years. Accordingly, the possibility of introducing human errors in designing such block has grown. Operation of generic-pin control block needs to be validated with a top-level RTL from the early stages of design, which correctly checks full-chip interface. However, generic-pin control block has inherent several design issues since function registers and multi-I/O paths are usually fixed in the relatively late stages of design. Also, the role of a generic pin control block that shares limited pins causes frequent changes in pin assignment. Therefore, current design approaches of a generic pin control block are no longer adequate to meet the challenges of design productivity, design reusability, and shorter time-to-market for design. And, this results in many possible human errors when using a traditional RTL description.
As a response to this problem, we developed a design automation based approach to reduce the possibility of human errors. In the case study presented, we succeeded in auto-generating a generic pin control block in multimedia SoC platforms which has more than 400 general purpose I/O interfaces including both input and output, as well as 1200 PAD pins. Ultimately, we reduced the amount of manual description for generating a generic pin control block by a whopping 98%.

The Overhead of Data Preparation (ODP) is very concerned in the future design of multi/many-core systems on the same chip. Therefore, we considered this issue under the extended Amdahl’s law and apply it to three “traditional” mult/many-core systems scenarios such as homogeneous symmetric, asymmetric, and dynamic. In addition, we expanded it toward two new scenarios spanning heterogeneous and dynamic CPU-GPU multi/many-core systems. Based on our evaluation, we found that potential innovations in heterogeneous system architecture are indispensable to decrease ODP.

Furthermore, providing a solution of low power consumption and the trade off a small decrease in performance and throughput are the main challenges in designing future heterogeneous multi/many-core architecture on a single chip. Our design incorporates heterogeneous cores representing different points in the power-performance design space during an applications execution. Under this circumstance, system software dynamically chooses the most appropriate core to meet specific performance and power requirements. As a response to this finding, we have presented a power-aware core management scheme through tightly-coupled hardware and software interaction: (1) heuristic thread consolidation scheme in software level, (2) 3-bit core power control scheme in hardware level. It is based on efficient methods of the core power management on heterogeneous multi/many-core architecture as a mechanism to reduce huge clock cycles as a latency when a core is powered down to up. Operation is based on distinct scenarios by 3-bit core power control scheme through 5 statuses switching such as active, hot, cold, idle, and powered down. In addition, this kind of status switching
is exactly triggered by referencing two information. One is the collected process ID information which is allocated by OS scheduler. Another is the decision information of heuristic thread consolidation scheme to maximize power-performance efficiency. Experiments prove that the power-performance efficiency of our model presented reduces power on average by 2.3% compared to a system with an efficient power-aware policy and by up to 15% with respect to the basic policy.

At the aspect of energy-efficiency on the same chip, we have proposed a performance-energy efficiency analytical model for the future integrated heterogeneous parallel multi/many-core systems which is promising to be used for big data applications. The model extends the traditional computing-centric model by considering ODP which can not be neglected in heterogeneous multi/many-core systems anymore. The analysis has clearly shown that higher parallelism gained from either computation or data preparation brings greater energy-efficiency. Improving the performance-energy efficiency of data preparation is another promising approach to affect power consumption. Therefore, more informed tradeoffs should be taken when we design a modern heterogeneous multi-many-core systems within limited budget of energy envelope.

*Index Terms - Generic pin control block, Design automation, Multimedia system-on-a-chip, Extended Amdahls Law, Overhead of Data Preparation, Power Management, Heterogeneous Multi/Many-Core Systems, 3-bit Core Power Control Scheme, Power-Aware Thread Placement, Heuristic Thread Consolidation, Energy-Efficiency, Computing-Centric Model.*
Part I

DESIGN AUTOMATION FOR
CONFIGURABLE I/O INTERFACE
CONTROL BLOCK
Chapter 1

Introduction

As the design productivity cannot follow the rapid advance of fabrication technologies, development of a new design methodology to improve design productivity has become necessary. Automating platform integration and verification process is a possible solution [6], [8]. More specifically, in various multimedia domains [1], [5], [10], [13], commercial solutions of design automation for signal multiplexer and PAD blocks have been presented [7], [9]. However, several architectural parameters of such commercial solutions; such as the maximum number of I/O pins [17], PAD control signals [12], and various address regions make it difficult to reuse a generic pin control block. As a result, register transfer level (RTL) designers rely on manual design work which is time-consuming and error-prone in the usually limited timeframe of the design cycle [14].

In this paper, we propose an automated design scheme of a generic pin control block. The key idea of this automated design scheme is the processing of a formalized text through our proposed auto-generator which is made by script programming language. By using this approach, designers can automatically generate RTL blocks to reduce the possibility of human errors and design time, while maintaining consistency among the generated outputs.
The remainder of Part I is organized as follows. We first briefly review related work in Chapter 2. Chapter 3 introduces the structure of generic pin control blocks. Then, the detailed structure and functions of the proposed method are introduced in Chapter 4. Chapter 5 shows the experimental results and our analysis of the proposed method, followed by some conclusions in Chapter 6.
Chapter 2

Related Work

Nowadays, complex SoCs include more peripheral interfaces in the core than can be accessed at one time. This requires a complex scheme having a flexible MUX-ing strategy that allows pins to configure. As a result, peripheral interface designers spend much time designing complex I/O subsystems and then adapting them to inevitable specification changes during the course of the design. Also, this issue has been increased front-end development effort for SoC platforms and derivative designs by more than an order of magnitude, while also dramatically increasing the possibility of human error in the design.

Organized under the entity-based rules of the IEEE standards association corporate program, the P1685 working group will begin work on the current IP-XACT v1.2 specification and complete the work following another expected SPIRIT consortium contribution that covers transaction-level modeling extensions. The IP-XACT (IEEE-1685) standard [3] which was originally progressed by SPIRIT consortium [16] defines a way for the description and handling of multi-sourced Intellectual Property (IP) components. In addition, it enables automated design integration and configuration within multi-vendor design flows. Specifically, the IP-XACT proposed standard today comprises some features. Firstly, it can support full
RTL design, including any component type, hardware description language, configuration, or connection type at this abstraction level. Secondly, it has methods for defining register information, memory maps, and address spaces. Thirdly, it is a basic interface to enable configuration and generation scripts. It is also a mechanism for adding implementation constraints to IP descriptions to help the flow to synthesis. The IP-XACT uses the form of an XML schema to define an IP meta-data description. This meta-data description represents an IP in several aspects such as hardware model, signals, bus interfaces, memory map, address space, and model views. This standard can also be used to describe a hierarchical subsystem resulting from an assembly of sub-components. IP-XACT components, like general IPs supplied by IP providers, can be assembled into a SoC platform directly using IP-XACT compliant tools. It leads to an improvement in the interoperability of different abstraction levels of a SoC platform. This method has made for a more flexible, automated and optimized platform design flow. A number of industrial experiences of applying SPIRIT technology have been proposed [15], and some research has been conducted from the electronic system level (ESL) to the RTL design area [4].
Chapter 3

Structure of Generic Pin Control Block

An SoC contains many IPs and incorporates general interface control blocks to form a top-level RTL design with a tremendously large number of signal connections. However, integrating a generic pin control block with a top-level RTL design is not only time consuming but also vulnerable to human errors. Thus, an automated design approach to improve such tedious work is necessary.

The core architecture of a generic pin control block is shown in Fig. 3.1. This architecture has multi-functional port pins organized in several groups with a similar model. Each port can be easily configured by software to meet various system configuration and design requirements. These multi-functional pins need to be properly configured before they can be used. If a multiplexed pin is not used as a dedicated functional pin, this pin should be configured as a generic input or output port.

When a pin is configured as an input port by clearing the output enable line and setting the input enable line, external signals are written to the Input Data Register. Alternatively, if
General Purpose Interface Control Block

Figure 3.1: Core Architecture of a Generic Pin Control Block.
the pin was configured as an output port, the value in the output data register will drive the General Purpose Interface. Some pins need to be programmed in a special mode such as low-active input, bi-directional, or low-active bi-directional mode. If an appropriate pin in an input mode is not selected, the default pin value will be set to 0. However, in case of a low-active input, the default pin value will be set to 1. The pin in a bi-directional mode, which can be used in an input or output mode, has an output enable signal directly connected to the output enable signal of the generic pin control block. As for the low-active bi-directional mode, the inverted signal of the output enable is connected to the output enable signal of the generic pin control block.

The reuse of generic-pin control blocks is now considered to be the foundation of SoC design. It is the only methodology allowing the design of complex SoCs to meet a limited design timeframe, productivity, and quality requirements. The challenge for designing a generic pin control block nowadays is not whether to adopt the reuse methodology or not, but how to employ it efficiently. Hence, the details of this methodology need to be considered in some aspects of an automated design scheme. More specifically, if architectural redundant models of general interface control blocks can be organized using a formalized text through our proposed auto-generator, it is possible to find a way to significantly reduce both human errors and design time.
Chapter 4

Specification with Formalized Text

In this section, we define the types of special registers according to functional requirements of a generic pin control block and propose a formalized text approach to unify various architectures [11] for the generic pin control block.

Fig. 4.1 shows the summary of functionalities and structural parameters that are essential to implement a generic pin control block. Our goal is to develop a formalized description approach to express all the functions and parameters described in Fig. ??.

4.1 Formalized Text

Our proposed formalized text defines types of interface connections and presents an effective way to map definitions onto physical ports. The signal interface connections include different types of bus interface. More specifically, the standard bus interface uses standard communication protocols such as the AMBA bus interface [2]. Further, a custom bus interface represents customizable communication protocols like DMA and memory interface. Also, a dedicated bus interface represents remaining signals that are regarded as another type of bus
General Purpose Interface Control Block

**Functions**

- Special function register
  - Data I/O through bus interface
  - Register control

- MUX
  - I/O MUX control

- PAD
  - I/O direction control
  - Pull up/down control
  - Drive strength control

**Parameters**

- Special function register

- MUX
  - The number of MUX input
  - Signal MUX register
  - Control MUX input
  - Power control

- PAD
  - The number of control signal
  - PAD control register
  - Control signal name/width
  - Control logic

Figure 4.1: Functions and Parameters.
which includes many other side signals interfacing different IPs and PAD directly.

Each signal in a generic pin control block belongs to its own type of bus as defined above. Hence, our goal is to develop a formalized description approach to express all the functions and parameters with respect to connections of the generic pin control block. The description of this formalized text is depicted in a spreadsheet style input template in Fig. 4.2. A designer can model RTL blocks by using control properties in the proposed formalized text defined in Fig. 4.3. The auto-generator will then automatically convert the proposed formalized text into corresponding logics. In particular, input in the Func0 column indicates a dedicated input to ARM core in each pin and output in the Func1 column indicates a dedicated output from ARM core in each pin. Each field of function from Func2 to Func15, which is composed of such as name, I/O, and mode, indicates IP’s pin name, IP’s I/O type, and IP’s mode in each pin.

### 4.2 Specific Functional Requirement

Except for the basic I/O function of the generic pin control block, our proposed block also has additional functions that can efficiently control an interface of a multimedia SoC platform. This module consists of specific registers used to support additional functions as shown in Fig. 4.4. The base address of each register group is added to the memory information of multimedia SoC platforms.

### 4.3 Composition of Registers

In terms of hardware implementation, it is necessary to define a register type that represents the specific control property. Most pins of a generic pin control block are multiplexed and
<table>
<thead>
<tr>
<th>Port</th>
<th>Func0</th>
<th>Func1</th>
<th>Func2</th>
<th>Func3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
<td>Name</td>
<td>I/O</td>
<td>Mode</td>
</tr>
<tr>
<td>P0.0: cudi10000</td>
<td>input</td>
<td>output</td>
<td>SPI0_CEN</td>
<td>LI</td>
</tr>
<tr>
<td>P0.1: cudia0010</td>
<td>input</td>
<td>output</td>
<td>SPI0_CLK</td>
<td>B</td>
</tr>
<tr>
<td>P0.2: cudi10100</td>
<td>input</td>
<td>output</td>
<td>SPI0_MOSI</td>
<td>B</td>
</tr>
<tr>
<td>P0.3: cudib0110</td>
<td>input</td>
<td>output</td>
<td>SPI0_MISO</td>
<td>B</td>
</tr>
<tr>
<td>P0.4: cudi11000</td>
<td>input</td>
<td>output</td>
<td>UART0_RXD</td>
<td>I</td>
</tr>
<tr>
<td>P0.5: cudic1010</td>
<td>input</td>
<td>output</td>
<td>UART0_TXD</td>
<td>O</td>
</tr>
<tr>
<td>P0.6: cudi01110</td>
<td>input</td>
<td>output</td>
<td>I2S0_MCK_O</td>
<td>O</td>
</tr>
<tr>
<td>P0.7: cudi01110</td>
<td>input</td>
<td>output</td>
<td>I2S0_LRCK_O</td>
<td>O</td>
</tr>
</tbody>
</table>

P#.# indicates P[port number.pin number]. Note that each port consists of 8 pins, one bit per pin.

Figure 4.2: Formalized Description of Our Automated Design Scheme.
Figure 4.3: Control Property Definition in a Formalized Text.
<table>
<thead>
<tr>
<th>Register</th>
<th>Type</th>
<th>Base Address</th>
<th>R/W</th>
<th>Description</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCON#</td>
<td>B</td>
<td>0x000000#0</td>
<td>R/W</td>
<td>Configure the pins of port#</td>
<td>0x00000000</td>
</tr>
<tr>
<td>PDAT#</td>
<td>B</td>
<td>0x000000#4</td>
<td>R/W</td>
<td>Data register for port#</td>
<td>0x000000XX</td>
</tr>
<tr>
<td>PCTL#</td>
<td>S</td>
<td>0x000000#8</td>
<td>R/W</td>
<td>Control the pins of port#</td>
<td>0x00000000</td>
</tr>
<tr>
<td>PPUR#</td>
<td>S</td>
<td>0x000000#C</td>
<td>R/W</td>
<td>Pull-up enable register for port#</td>
<td>0x00000000</td>
</tr>
<tr>
<td>PPDR#</td>
<td>S</td>
<td>0x00000(#+1)00</td>
<td>R/W</td>
<td>Pull-down enable register for port#</td>
<td>0x00000000</td>
</tr>
<tr>
<td>PPIE#</td>
<td>S</td>
<td>0x00000(#+1)04</td>
<td>R/W</td>
<td>Input enable register for port#</td>
<td>0x00000000</td>
</tr>
<tr>
<td>FSEL</td>
<td>S</td>
<td>0x00010000</td>
<td>R/W</td>
<td>Change the specified function</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Symbols: B = Basic Register; and S = Special Register.

Figure 4.4: Composition of a Specific Register Group.
have up to 15 different functions. The function of each pin can be configured by 4 bits of port configuration registers. After reset, all port configuration registers have '0' value in ROM boot mode. Thus, all pins are controlled by port configuration registers as input mode of normal generic pin control blocks. When ports are configured as output ports, data can be written on the corresponding bit of port data registers. When ports are configured as input ports, the data can be read from the corresponding bit of port data registers.

For the port specified by port control registers as shown in Fig. 4.5, the corresponding general purpose interface pin will be set to 1 if the 8 bit-set part is 1 in a port control register, while no action will occur if the set part is 0. The corresponding general purpose interface pin will be set to 0 if the 8 bit-clear part is 1 in a port control register, while no action will occur if the clear part is 0. If set and clear parts are both 1, then the value of the pin is toggled.

This operation is performed only when the corresponding pins are designated as outputs of the port configuration registers. These actions are mainly for the purpose of checking the LCD interface on a field programmable gate array (FPGA). A pull up/down register controls the pull up/down resistor enable/disable of each port group. When the corresponding bit is 1, the pull up/down resistor of the pin is enabled. When the corresponding bit is 0, the pull up/down resistor is disabled. However, it is prohibited to set the pull up register and pull down register to enable at the same time.

For the ports and the pins that are specified by the port number and pin number fields of the function selection register, writing a number to function field of function selection register will copy that number to an appropriate place of the port configuration registers, and immediately change the function of the specified general purpose interface. Writing 1110 and 1111 at function field will cause 2 separate actions. For example, writing 1110 at function field with 4 bits size will set the appropriate bits of the port configuration registers to 0001, and will set the lowest bit of the port data registers to 0. Writing 1111 will set the
Specific Register (PCTL0)

Control Logic Generation (Toggled Condition)

Figure 4.5: Composition of Port Control Registers.
appropriate bits of port configuration registers to 0001, and will set the lowest bit of port
data registers to 1. These actions are for the purpose of checking the interface connection
to PAD to improve design for testability (DFT). This register only has an independent base
address which is not used by other registers. Hence, the control properties of these register
types can be extended in the future if necessary.
Chapter 5

Experiment Results

To demonstrate how our proposed approach using a formalized text can effectively auto-generate a generic pin control block, we used a typical multimedia SoC platform inspired by some Samsung projects based on 65 nm technology. Fig. 5.1 presents an execution model of our proposed RTL blocks auto-generator expressed in the Python script language. The implementation of the auto-generator requires a design decision that specifies the behavioral characteristics of the generic pin control block. Further, such decisions may affect the choice of other features for the integration of the top level. Thus, it is recommended to use the formalized text as an input sequence which can minimize the possibility of restriction on the design decision.

To evaluate the efficiency of our proposed design approach, we applied our automated design to multimedia SoC platforms. We generated an architecture model whose hardware characteristics are summarized in Fig. 5.2. We measured the efficiency ratio of traditional RTL design time to RTL auto-generating time by our proposed RTL blocks auto-generator.

As a result, the amount of manual description used for generate a generic pin control block was reduced by 97%, compared to the traditional RTL description.
def genGpicbFile(self) :
gpicb_0_comments() # code information defines
gpicb_1_define()   # macro variables declaration
gpicb_2_start_of_module_define() # module defines
gpicb_3_make_pin_name_list(self) # pin name generation
gpicb_4_end_of_module_define(self) # endmodule defines
gpicb_5_parameters(self) # parameters define
gpicb_6_scan_gating(self) # scan control logics generation
gpicb_7_register_definition(self) # registers generation
gpicb_8_apb_readdata(self) # APB control signals generation
gpicb_9_input_mux(self) # input mux generation
gpicb_10_output_mux(self) # output mux generation

SubTask
def gpicb_8_apb_readdata(table) :
  ......
  print "always @(psel or pwrite or paddr) for pcon
  in table.pconList : print "or r_pcon%d or
  r_pdat%d"(pcon.pconNumber,
pcon.pconNumber)
  if pcon.thisPCONHasPullup:
    print "or r_ppur%d"(pcon.pconNumber)
  ......
  print "or r_fsel"
  print ")"
  print ""
  print "t case({psel, pwrite, paddr})"
  i = 0
  for pcon in table.pconList :
    print "10'b10_%s_000 : prdata_in = r_pcon%d;"%(d2b(i,5),pcon.pconNumber)
  ......
  if pcon.thisPCONHasPullup:
    print "10'b10_%s_011 : prdata_in = r_ppur%d;"%(d2b(i,5),pcon.pconNumber)
  ......
  print "10'b10_%s_000 : prdata_in = r_fsel;" (d2b(i,5))
<table>
<thead>
<tr>
<th>Platform</th>
<th>Item</th>
<th>MUX Type</th>
<th># PAD Control</th>
<th>Power Control</th>
<th>Data Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable Device</td>
<td>8:1</td>
<td>3</td>
<td>No</td>
<td>Byte</td>
<td></td>
</tr>
<tr>
<td>Smart Phone</td>
<td>16:1</td>
<td>4</td>
<td>Yes</td>
<td>Byte</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.2: Composition of Multimedia SoC Platforms.
As shown in Fig. 5.3, we measured the size of specification input and the size of generated output to check the ratio in efficiency between 42 and 45. This automatic generic pin control block generator is applicable to different platforms with simply changing the input formalized description text format.

The Fig. 5.4 shows the design volume in a typical multimedia SoC, which consists generic and PAD pins within platforms which are applied. Moreover, our proposed approach is able to generate RTL blocks automatically and the possibility of human error can be reduced.
<table>
<thead>
<tr>
<th>Platform</th>
<th>Item</th>
<th>Input Description (Bytes)</th>
<th>Output Description (Bytes)</th>
<th>Efficiency Description (Output/Input)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable Device</td>
<td></td>
<td>39,700</td>
<td>1,758,300</td>
<td>44.3</td>
</tr>
<tr>
<td>Smart Phone</td>
<td></td>
<td>51,300</td>
<td>2,205,884</td>
<td>42.9</td>
</tr>
</tbody>
</table>

Figure 5.3: Quantitative Analysis in a Typical Multimedia SoC Platform.
### Figure 5.4: Design Volume in Multimedia SoC Platforms about Generic and PAD Pins.

<table>
<thead>
<tr>
<th>Portable Device</th>
<th>Smart Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Generic Pin</td>
<td>Number of Generic Pin</td>
</tr>
<tr>
<td>Number of PAD</td>
<td>Number of PAD</td>
</tr>
</tbody>
</table>

- Portable Device: Generic Pin: 300, PAD: 1000
- Smart Phone: Generic Pin: 1200, PAD: 1400
Chapter 6

Conclusions

In this paper, we proposed an automated design scheme for the generic pin control block based on the process of a formalized text through our proposed auto-generator. By this formalized text, designers can specify an arbitrary generic pin control block with minimal amount of description effort. In addition, designers can rapidly obtain RTL blocks by such an automated procedure. It is proven to be beneficial for the efficiency and reliability in the design of a generic pin control block. Proper management of a formalized text enables completely configurable design in multimedia SoC platforms. This can ultimately reduce design time and the possibility of human errors significantly. Once the generic pin control block generation has been done based on this automated design approach, designer successfully can apply this block to the top-level integration with a minimal effort.

Experimental results show that it is possible to describe various generic pin control architecture with the proposed description format. We have applied this automated design approach in several multimedia platforms such as portable device and smart phone. These platforms demonstrate validity and reusability with respect to the design methodology of the generic pin control block. Lastly, the amount of manual description for designing a generic pin con-
trol block was reduced by 98%, compared to the traditional RTL description. The generic pin control block has been applied to typical multimedia SoC platforms and we are able to expand this block for different platforms with a configurable formalized text.
Part II

SPEED UP MODEL BY
OVERHEAD OF DATA
PREPARATION
Chapter 7

Introduction

As multicore processors have become mainstream, it has become crucial to identify performance bounds and performance scaling properties in exploiting the massive parallelism they offer [88]. Besides, processor design has been transiting from the homogeneous multicore model to the heterogeneous model [89]. Furthermore, energy efficiency and scalability are affected by the power constraints imposed on heterogeneous processors [90]. The memory wall [91] and communication issues will continue increasing the gap between the performance of an ideal processor and that of a “practical” processor. Therefore, the overhead of data preparation becomes an unavoidable key parameter. Assume that a fraction $f$ of a program’s execution is infinitely parallelizable without scheduling and synchronizing overhead with $c$ processors (cores), while the remaining fraction $1 - f$ is assumed to be sequential execution. Then, the computational speedup of the system is governed by the well-known equation as (7.1) (Amdahl’s law [92], [93])

$$S_A(f, c) = \frac{1}{(1 - f) + \frac{f}{c}}$$  \hspace{1cm} (7.1)
The equation is correct as long as three key assumptions are verified: 1) the programs to be executed are of fixed-size and the fraction of the programs that is parallelizable remains constant as well; 2) there exists an infinite memory space without extra overhead of switching memory blocks and disk blocks, etc.; and 3) the overhead of preparing the data to be used by computing units, which includes memory access, communication on-chips or off-chips and synchronization among cores, can be neglected. In practical multicore systems, as it turns out, the overhead of data preparation is between 30% and 90% of the total execution time depending on the kind of application considered [94]. In other words, the overhead of transferring data between CPUs and GPUs in a heterogeneous multicore system could be the bottleneck of high performance computing. As we introduce this parameter in the single-threaded multicore system, we propose a set of equations in the following Chapters.
Chapter 8

Reconsidering Speedup Model by Overhead of Data Preparation (ODP)

As proposed by Hill & Marty [93], we rst assume that a multicore chip of a given area and manufacturing technology is composed of at most base core equivalents (BCEs) (a single BCE implements a baseline core). Then, we assume that the resources of BCEs can be used to create a powerful core with sequential performance, while the performance of a single-BCE core is 1 [93]. is an arbitrary function, where \( 1 \leq \text{perf}(r) \leq r. \)

Since an improvement in sequential performance by microarchitecture techniques alone would follow Pollack’s rule [95], \( \text{perf}(r) \) is roughly proportional to the square root of the increase in complexity. At the same time, because of the “memory wall”, the overhead of preparing the data and accessing the memory, of transmitting data on- and off-chip, of transferring data between CPU memories and GPU memories for heterogeneous system, of synchronizing processes, etc., becomes so significant that it cannot be ignored any longer. However, Amdahl’s law only considers the cost of instruction execution. We will thus now assume that the whole cost of executing a program can be split into two independent parts, one is preparing the
data for execution and the other one is running instructions when the required data are ready. Therefore, the overhead of data preparation (ODP) includes the cost of preparing data for execution, to the exclusion of actual execution. As shown in Fig. 8.1, ODP can be considered to produce a new speedup equation; we will call it the “Extended Amdahls law” as expressed in equation 8.1

\[
S_{EA}(f_c, p_c) = \frac{1}{((1 - f_c) + \frac{f_c}{p_c}) \times p_c + (1 - p_c)}
\]  

(8.1)

where \( p_c \) denotes the computation portion, \( 1 - p_c \) denotes the data preparation portion normalized to the computation portion: since the clock frequencies and the ISAs of CPUs, GPUs, off-chip bus and memory arbitrator would likely be different, we should normalize the performance of Data Preparation instructions to that of computing instructions. The is the parallelizable computation portion, \( 1 - f_c \) is the sequential computation portion.

As shown in Fig. 8.1, however, equation 8.1 does not make allowance for the introduction of hardware techniques that would decrease or eliminate the overhead of data preparation. We can further divide the portion of data preparation into three subparts: \( 1 - f_h \), \( \alpha f_h \) and \( (1 - \alpha) f_h \). \( 1 - f_h \) denotes the portion of data preparation which is closely dependent on computing instructions. \( f_h \) denotes the data preparation portion of the program that can be overlapped with “computing” instructions before introducing advanced techniques, where 0
Memory access instructions generally can be executed simultaneously with independent computing instructions. However, it will not be possible to execute all of the data preparation instructions simultaneously with computing instructions. Further, not all operations of data preparation would be executed with computing instructions according to our observations. For example, if a load instruction is independent of the following computing instructions, it can be issued simultaneously with them. While it would not be issued if the queue of issuing load instructions was full.

Hence, we introduce the parameter $\alpha$ to denote the percentage of data preparation instructions which are actually executed simultaneously with computing instructions before using advanced architectural techniques ($0 \leq \alpha \leq 1$). Thus, $\alpha f_h$ denotes the portion of actual parallelized instructions for data preparation and $(1 - \alpha)f_h$ denotes the portion of data preparation instructions which cannot be overlapped with computing instructions without sophisticated architecture techniques.

With the help of advanced architectural techniques such as data prefetching, speculative execution, universal memory, no-copy data transfer, 3-D NoC, etc., $(1 - \alpha)f_h$ could be decreased significantly. Thus, we further introduce the variable to model what percentage of data preparation cannot be overlapped on a $c$ cores system by using new advanced techniques ($0 \leq k_c \leq 1$). After normalization to the computation part in theory, the portion of data preparation instructions which cannot be overlapped on a $c$ core system becomes $f_{ud} = (1 - f_h) + k_c \times (1 - \alpha)f_h$. Assuming that the execution time of parallelizable data preparation is less than the execution time of computation and the serial data preparation time put together, the portion of parallelizable data preparation can be masked by computation and serial data preparation. Therefore, we can extend Amdahls law to the following Enhanced
Amdahl’s law.

\[
S_{EA}'(f_c, c, p_c, f_{ud}) = \frac{1}{((1 - f_c) + \frac{f_c}{c}) \times p_c + f_{ud} \times (1 - p_c)}
\]  \hspace{1cm} (8.2)
Chapter 9

Case Studies of Our Enhanced Amdahl’s Law Speedup Model

As in Hill & Marty [93], we also assume that a multicore chip of a given area and manufacturing technology is composed of $n$ base core equivalents (BCEs). Assumed that, if we consider only hardware techniques, the improved sequential performance obeys Pollacks rule [94], $\text{perf}(r)$ is approximately proportional to the square root of the increasing size of a chip or number of transistors. For simplicity, we also assume that the performance (cost) of data preparation is also roughly proportional to $\text{perf}(r)$.

9.1 Homogeneous Symmetric Multicore

We follow Hill & Martys [93] denition of a symmetric multicore architecture and assume that a homogeneous symmetric multicore chip uses one single-BCE core with $r$ BCEs to execute sequentially at performance $\text{perf}(r)$, and that it uses all $c = n/r$ cores to execute in parallel at performance $\text{perf}(r) \times n/r$. Note that performance is driven by the execution
time of a given program on a processor which is only dependent on the technology of the microarchitecture. Therefore, we obtain the following speedup equation for a homogeneous symmetric multicore architecture

\[
S_{EA}^{hs} = \frac{1}{\left(\frac{1-f_c}{\text{perf}(r)} + \frac{f_c \times r}{\text{perf}(r) \times n}\right) \times p_c + \frac{f_d}{\text{perf}(r)} \times (1 - p_c)}
\] (9.1)

Fig. 2 assumes a symmetric multicore chip with a maximum number \( n = 256 \) of BCEs and \( \text{perf}(r) = \sqrt{r} \). The x-axis presents the number of resources (BCEs) to be configured into a core/processor with a total budget of 256 BCEs. For example, if \( x = 4 \), it means that each single core uses 4 BCEs and that it has a total of 64 cores with 4 BCEs each, for a total budget of 256 BCEs resources. The y-axis shows the speedup compared to the baseline of a single-BCE baseline core. We further assume a parameter \( p_c \) with different values for the portion of computation in a program. For instance, \( p_c \) can be 0.2, 0.4, 0.6, and 0.8. In addition, it can be overlapped with data preparation \( (f_h = 0.2, 0.4, 0.6, 0.8) \), and also different values for the parallel fractions in the portion of the computation \( (f_c = 0.5, 0.9, 0.975, 0.99, 0.999) \). We choose a representative \( (p_c = 0.6, f_h = 0.8) \) from 80 similar curve trends as example in the following subsections.

We performed experiments on the Gem5-GPU simulator [38] which is a standard simulator of heterogeneous systems and tested \textit{backprop}, \textit{bfs}, \textit{mum}, and \textit{hs} in the Rodina benchmark [96]. The configuration parameters of simulator is accordant to [32]. We found that the average percentage of sequential load/store operations is about 1/3. Incidentally, we got some similar supportive evidences from other research results. The floating-point operation(fpop) of 052.\textit{alvinn} which belongs to the CFP92 benchmark in the SPEC suite makes up 27.6% of all the operations, memory operations 48.3%, branch operations 5.1%, while others (total-memory-fpop-branch) make up 18.9% [97]. Thus, the summary percentage of computation
is only 27.6%, and the remaining percentage of operations which are relevant to data preparation is about 72.3%. Therefore, we empirically set $k_c$ and $\alpha$ to $1/3$ and $2/3$ respectively. It means that, in a given application, $1/3$ of data preparation cannot be actually executed in parallel, while $2/3$ of data preparation can take place in parallel with computation and other data preparation operations.

As shown in Fig. 2, the speedup will vary with the number of BCEs and reaches the maximum 56.71 when $r = 16$ BCEs, and $f_c = 0.999$. This means that almost 60% of the instructions in a program are parallelizable “computing” instructions, while the remaining 32% are data preparation instructions pertaining to memory access, data transfer, etc. which can be masked.

### 9.2 Homogeneous Asymmetric Multicore

Just as the example proposed by Hill & Marty [92], a homogeneous asymmetric multicore chip is composed of one large core with $r$ BCEs, and the remaining of other $n-r$ BCEs with the same Instruction set architecture (ISA). Assuming that the serial computing part is executed by one large core and the parallel computing part is executed by the remaining $n-r$ one-BCEs cores and the large core simultaneously, the modified Amdahls law with consideration of the ODP becomes

$$S_{EA}^{ha} = \frac{1}{\left(\frac{1-f_c}{\text{perf}(r)} + \frac{f_c}{\text{perf}(r)+(n-r)}\right) \times p_c + \frac{f_{ud}}{\text{perf}(r)} \times (1-p_c)} \quad (9.2)$$

As shown in Fig. 3, the maximum speedup is 70.12 with $r = 128$ BCEs and $f_c = 0.999$. It is higher than the maximum speedup 56.71 shown in Fig. 2. In general, the homogeneous
asymmetric architecture has a potential higher speedup performance. Furthermore, comparing to the homogeneous symmetric architecture, a more powerful core should be built with 128 BCEs to achieve the highest possible speedup in an asymmetric architecture. Further, the maximum speedup with consideration of ODP is almost twice that of a traditional asymmetric architecture. A full length paper will include details.

9.3 Homogeneous Dynamic Multicore

Based on the same assumption as that for a dynamic multicore in [92], the serial part of a program can be executed by a large core dynamically combining up to \( r \) single-BCEs by utilizing some advanced techniques (e.g., thread-level speculation, helper threads, data prefetching). On the other hand, the parallelizable part is executed by all \( n \) base cores with dynamic scheduling techniques such as simultaneous multithreads, dynamic instructions schedules, etc. The modified Amdahl’s law with consideration for ODP thus becomes

\[
S_{EA}^{hd} = \frac{1}{\left( \frac{1-f_c}{\text{perf}(r)} + \frac{f_c}{n} \right) \times p_c + \frac{f_{ud}}{\text{perf}(r)} \times (1 - p_c)}
\]  \hspace{1cm} (9.3)

Based on equation (6), we find that the highest possible speedup is 106.57 for a homogeneous symmetric and dynamic multicore architecture where \( r = 256 \) and \( f_c = 0.999 \). Because it uses ideal dynamic techniques to increase processor performance, the performance will rise directly with the number of cores. Since the dynamic techniques are assumed to eliminate all the overhead from data hazards, control hazards and resource hazards, without considering the overhead caused by these dynamic techniques, the maximum performance improvements are higher than that in homogeneous symmetric architectures.
Figure 9.1: Speedup Distribution of Homogeneous Symmetric Multicore where $p_c = 0.6$, $f_h = 0.8$. 
Figure 9.2: Speedup Distribution of Homogeneous Asymmetric Multicore where $p_c = 0.6$, $f_h = 0.8$. 
9.4 Heterogeneous CPU-GPU Multicore

Just as a homogeneous asymmetric multicore, some cores may be more powerful than others. The more powerful cores are built as CPU processors, and the others are built as GPU cores. Note that the assumption of single-BCE is equivalent to a GPU core here rather than CPU core, if we consider the chip area of a single-BCE core, and a powerful CPU processor is composed of $i$ CPU cores where each one comprises $r$ BCEs. We assume that $i$ CPU processors handle all serial computing at performance $i \times \text{perf}(r)$. The remaining $n - ir$ BCEs are equivalent to $n - ir$ GPU cores, and all the GPUs cores and CPU processors cooperatively execute the parallel portion of the program. In the same manner we calculate the computational performance, thus we assume that the serial performance of data preparation is proportional to the amount of resources (number of BCEs) on the chip by a factor. Consequently, we obtain the speedup equation (7) below.

As shown in Fig. 4, the highest speedup is 40.58 where $r = 32$ and $f_c = 0.999$. In other words, the highest performance heterogeneous CPU-GPU architecture is composed of one CMP with 4 CPU cores, and each CPU core has 32 BCEs. The remaining 128 BCEs are built as 128 GPU cores. However, the highest speedup is less than that of homogeneous asymmetric architecture although the CPU processor also uses 128 BCEs.
Figure 9.3: Speedup Distribution of Heterogeneous CPU-GPU Multicore where $p_c = 0.6$, $f_h = 0.8$, $i = 4$. 
9.5 Heterogeneous Dynamic CPU-GPU Multicore

Similarly to the homogeneous dynamic multicore, we can derive an equation for a heterogeneous dynamic CPU-GPU multicore. While it can also schedule all the computation resources, just as what was assumed in [92], base cores can also execute in parallel either CPUs or GPUs cores. Further, it should be noted that we also assume the cost of data preparation to be the same as that in the general heterogeneous CPU-GPU model. Therefore, the equation becomes

\[
S_{\text{EGD}}^{\text{EA}} = \frac{1}{\left( \frac{1-f_c}{i \times \text{perf}(r)} + \frac{f_c}{n} \right) \times p_c + \frac{f_{ud}}{\text{perf}(r)} \times (1 - p_c)}
\]  

(9.4)
Figure 9.4: Speedup Distribution of Heterogeneous Dynamic CPU-GPU Multicore where \( p_c = 0.6, f_h = 0.8, i = 4 \).
Chapter 10

Conclusions

When one takes into account the ODP, the speedup will not grow linearly with the number of cores. Furthermore, a heterogeneous CPU-GPU architecture would be more beneficial than a homogeneous architecture according to the comparison curves of speedup distribution.

This is a quantitative analysis strictly based on a theoretical and general model. To some extent, however, we have been able to compare the results with Hills theoretical approaches for multicore architectures. We would also compare it with other works in a regular paper. Further research will entail investigating the energy efficiency of multicore processors and multithread processors.
Part III

EFFICIENT CORE POWER

CONTROL SCHEME
Chapter 11

Introduction

In the design of future heterogeneous multi/many-core architecture of a dark silicon era, processor power consumption and heat dissipation have become key challenges as processors continue to increase in performance and speed. In addition, although heterogeneous system architecture (HSA) foundation [18] and international technology roadmap for semiconductor (ITRS) organization [19] including top-tier industry researchers have advocated future many-core chips [20] over heterogeneous multicore systems such as AMD Fusion [21], Intel Haswell [22], Nvidia Denver [23], and ARM big.LITTLE [24], power consumption for future large-scale processors remains a serious concern. Higher costs for thermal packaging, fans, electricity, and even air conditioning are typically caused by increased power consumption and heat dissipation. Higher-power systems can also have a greater likelihood of failures.

Prior chip-level many-core architecture has been proposed using multiple copies of the same core, in our words, homogeneous [25]. Core diversity is of higher value than uniformity for many applications, which offers much greater ability to adapt to the demands of applications [26]. In this study, we present a heterogeneous many-core architecture, in which all cores execute different instruction sets, capabilities and performance levels.
As the scale of cores on a chip increases, there will be more communications and data movements among cores occurred. In addition, when the power dissipation is increased sharply due to the lack of a good solution for it, the entire computing performance will be decreased. Previous work inspired us to eliminate the overhead of power consumption. By designing a 3-bit control scheme for real heterogeneous many-core architecture, we figured out an efficient core power management method.

We have found some issues that typical programs go through phases with different execution characteristics [27], [28]. Hence, during one execution flow, the best core may not fit into for the next execution flow. This observation motivates the ability to switch cores dynamically among execution flow. In addition, rather than left idle, unused cores are completely powered down in this situation. Therefore, unused core suffers from no static leakage or dynamic switching power. However, this approach is subject to a huge latency penalty for powering a new core up. The latency penalty is in approximately one thousand cycles of 2GHz clock [26]. It is more severe in an increasing clock frequency under many cores in the near future.

In this paper, we propose an advanced core power management scheme for the design of heterogeneous many-core architecture. Processing 3-bit core power control scheme through active/hot/cold/idle/powered down status switching is the main idea of this scheme. Per-core can be controlled independently, using this approach, thereby reducing processor power dissipation, while possibly maintaining consistent performance.

The remainder of this Part III is organized as following. Chapter 12 presents the related works, followed by Chapter 13 which introduces the target architecture. Chapter 14 introduces detailed structure and functions of the proposed method. Chapter 15 discusses about power-aware thread placement. Chapter 16 shows the evaluation and methodology based on our previous and current work, followed by future work in Chapter 17. Lastly, Chapter 18 summarizes this work.
Chapter 12

Related Work

Prior work on power-related optimizations for processor design can be classified into four broad categories:

The first category includes work using voltage and frequency scaling of the processor core to reduce power [29]. Because of cubic dependency of dynamic power about frequency scales with voltage squared, dynamic voltage and frequency scaling (DVFS) is a popular method to manage power. The majority of recent work on DVFS control relies on information gathered from performance counters during runtime to identify the best voltage-frequency (V-F) setting. Special compiler support or modifications to the applications can be ignored in this case [41], [42]. Application-level or compiler-level support is required for some approaches of power control via software [43], [44]. Most methods focus on optimizing metrics such as energy, energy-delay-product (EDP) and energy-delay-squared-product (ED^2P). The work of Canturk Isci et al. [45] derives phase categories based on a metric for the memory operation rate (mem/µ-op), where each category is mapped to an optimal V-F setting. In a similar vein, Gaurav Dhiman et al. [46], propose an online learning model for single-core processors. They break down the cycles per instruction (CPI) metric into several components such as
baseline CPI, miss events CPI, and stall CPI, to characterize workloads. Convergence to the optimum V-F setting which uses online learning is guaranteed by this approach. In sum, these approaches focus on energy, EDP or ED²P minimization without taking power caps into consideration.

Although most multicore processors support independent frequencies control for the cores, a common voltage level is usually fixed to support the highest frequency. Extensive design investments in the power-delivery network and the off-chip power regulators are required by independent voltages. Multiple clock domain design and voltage frequency island partitioning have been proposed to increase the granularity of DVFS control [52], [53]. The work of Wonyoung Kim et al. [54] explores designing on-chip regulators and perform core-level DVFS to reduce the overhead of runtime voltage conversion.

The second category includes work that uses gating technique with regard to the ability to turn on and off portions of the core for power management [30]. Power gating, a circuit-level technique, allows to cut off the power supply to a logic macro. It is implemented with the help of a sleep transistor that is inserted as a series header or footer device in the V_{DD}-to-V_{SS} circuit path. This circuit path also includes the targeted macro. Particularly, per-core power gating (PCPG) [71] is becoming an increasingly common knob in recent microprocessors [72], [73], [74]. However, the way to make the most proper use of PCPG remains questionable. For instance, if the core idleness period is not long enough, actuating PCPG each time a core becomes idle may result in negative power-performance benefits. It is evident that processes and software threads must be scheduled accordingly across cores to generate opportunities that are beneficial for PCPG, with a minimal impact on performance.

The third category includes work that facilitates larger degrees of freedom in job scheduling and allocation. Rangan et al. propose a scalable DVFS scheme for multicore systems that enables thread migration among homogeneous cores with heterogeneous power-performance capabilities [47]. Within a given power budget, they assign fixed V-F settings to different
cores and migrate the applications to reach the desired level of performance rather than charging the V-F settings on demand. With respect to applying DVFS under power constraints, the work of Etinski et al. propose a job scheduling policy that optimizes performance for a given power budget [48]. In addition, Isci et al. assess global power management policies with objectives such as prioritization, power balancing and optimized throughput for several benchmark combinations and power budgets [50]. Nevertheless, dynamic adaptation to different workloads is not provided in their approach. A paper by Teodorescu et al. proposes algorithms for power management through scheduling and DVFS under process variations [51].

Lastly, the forth category includes work that generates thread consolidation and motion based on thread migration among cores with different voltage and frequency settings. The work of Cochran et al. [75] is the most closely related to this method. In addition, Tam et al. [76] propose a mechanism for thread clustering based on patterns of data sharing. This mechanism is implemented at operating system (OS) kernel level with information derived from hardware event counters. This work closely relates to ours with respect to the methodology they use, which is also based on dynamic analysis of processor counters. However, they merely tackle performance improvement, whereas we also consider power reduction. Rangan et al. [77] introduce thread motion, a technique capable of fine-grained power management based on thread migration among cores with different V-F settings. Prior studies have also triggered the use of hardware event counters in the context of multi-threaded applications. In addition to the work of Tam et al., Bhattacharjee et al. [78] propose the use of processor counters to predict thread criticality dynamically. A critical thread is the slowest thread in an application, which limits its performance. For load balancing and energy saving purposes, they propose to exploit thread criticality prediction.

Some of the recently proposed techniques have explicitly focused on meeting power budgets or peak power constraints at runtime as the need for power capping and peak power man-
agement has increased Cebrian et al. propose a power balancing strategy which dynamically adapts the per-core power budgets depending on the workload characteristics [55]. However, this strategy would not perform well because it relies on borrowing power budgets from cores that consume lower power than others, in case of balanced workloads which have even power consumption among cores in the Parsec benchmark suite [34]. Sartori et al. propose a peak power management technique for multicore systems by choosing the power state for each core that meets the power constraints [56].

Gandhi et al. propose a power capping strategy which meets the power budget by inserting idle cycles during execution [57]. This approach aims to control the average power consumption, but does not guarantee peak power. Through hardware reconfiguration, many approaches meet the power budgets. Meng et al. propose a power management strategy through dynamic reconfiguration of cores by cache resizing [58]. Kontorinis et al. propose a table-driven adaptive core reconfiguration technique which configures core resources such as floating point units and load-store queues to meet peak power constraints [59].

Recently, processor and system vendors have started to provide peak power management features in commercial products. AMD has introduced PowerCap Manager for 45nm Opteron processors [60]. Both HP and Intel offer a power capping technique which adjusts power caps according to busy/idle states of the nodes for data center power management [40]. This technique utilizes the DVFS states and the throttling capabilities for idle cycle insertion at the chip-level. Besides sleep modes, power nap modes, where the system can enter and exit from low-power modes in milliseconds, have also been proposed to deal with the demand variation patterns in data centers [61].

Our many-core heterogeneous architecture does not preclude the use of these techniques and can potentially address the weaknesses of these techniques by enabling much greater power savings.
Chapter 13

Architecture

This Chapter provides an overview of a potential heterogeneous many-core architecture and discrete second level cache memory model.

13.1 Heterogeneous Many-Core System

Fig. 13.1 illustrates a heterogeneous many-core architecture which consists of 3-level computing elements such as core, quart, and tile. The cluster is characterized by a group of cores which contains CPU and GPU. The heterogeneity of core in such architecture can be made 4 CPU cores and 12 GPU cores sharing adaptive L2 cache among each type of cores. As a default, each CPU and GPU core has its private L1 cache independently. Then, each cluster is a quarter of a tile and each quarter shares their data by input and output queues over high throughput network on chip. Thus, each tile has 16 CPU and 48 GPU cores. This composition is similar to the architecture introduced by Johnsonl [25] and Pei [32].
13.2 Discrete L2 Cache Memory Model

Increasing core counts have underscored the need for scalable on-chip cache coherence mechanisms. The growing number of on-chip cores exposes the power and area costs of scaling the directories. Thus, for separate L2 cache for CPUs and GPUs cluster, we used a 4-way cuckoo directory [31] (see Fig. 13.2) for each CPU and GPU cluster to decrease the possibility of replaced block due to the high reusable frequency of interior data blocks. All ways are looked up in parallel using hashed values of the searched address to find an element in the cuckoo directory. Inserting an entry into the directory necessitates a lookup followed by a write of an entry in one of the ways. If the write replaces a valid directory entry, the insertion procedure is repeated for the victim entry, which iterates until an insertion finds a vacant location.
Core#: $2^4 (=16)$
Quart#: $2^2 (=4)$
Tile#: $2^4 (=16)$

Total#: $2^{10} (=1024)$

Figure 13.1: Heterogeneous Many-Core Architecture.
Figure 13.2: 4-Way Cuckoo Directory Structure.
Chapter 14

3-Bit Power Control Scheme

We have discussed about our heterogeneous architecture model using many of small cores so far. Now we discuss how the heterogeneous many-core architecture can be fitted in an affordable power envelope. Using voltage scaling is the best option to reduce power with minimal impact on performance. Although applying voltage scaling indiscriminately to the entire architecture would lower power, it may not be optimal. At the aspect of utilizing many cores, each core can be 3-bit power controlled separately, therefore employing fine grain power management.

Each core can be scaled with DVFS in the possible range. But this could be cumbersome and difficult. Also, different core running at different voltages and frequency would create asynchronous interface, additional latency, meta-stability, which would require a complex power delivery scheme. We propose a much simpler method of 3-bit core power management scheme in Fig. 14.1 and Fig. 14.2. As illustrated in the finite state machine (FSM) in Fig. 14.1, On/Off indicators around arrows between statuses indicate that they are triggered by the OS. A core operates at one of the five statuses: active, hot, cold, idle and powered down. The 3-bit core power management scheme assigns a 3-bit counter to each core whenever each
application program switches. The counter value is between 0 and $2^n-1$, where $n$ is 3 bits. These 3 bits are used to encode the 5 states in this architecture. The power status is either active or hot/cold core when the counter is greater than or equal to one-half of its maximum value ($2^n-1$), otherwise, it is idle or powered down. In a counter operation, the counters are incremented when the status is On and decremented when the status is Off; the counters saturate at 000 or 100.

Fig. 14.3 shows the power and clock distribution used in the heterogeneous many-core architecture, under the assumption that upper four cores (see red colored cores, namely hot cores) are all frequently used and that lower four cores (see blue colored cores, namely cold cores) are all less frequently used. In addition, two main contributions can be observed from peripherals: the upper line which goes to cores represents the power line and the lower line which goes to cores indicates the clock line.

This scheme focuses on predictively transitioning the power status that some hot cores will change soon again from slow active status to active status. Thus, the strategy avoids power dissipation due to the latency and associated leakage penalty incurred by accessing from an idle status. The OS predictively decides on a power status of hot cores as the slow active for preparing the future usage, since sequentiality among jobs running on core is the norm in execution flow in heterogeneous many-core architecture. Moreover, based on both the cores usability profiling information and real time sensing information getting from the temperature sensor of each core, hot core can be exactly decided by the OS. Hot spot on the chip provides a clue for making a decision more clear with regard to hot cores.

Table 14.1 summarizes the various core statuses which are supported by 3-bit core power control scheme of the heterogeneous many-core architecture and various hardware related to our scheme. The various actions that are performed in the various core statuses supported by the 3-bit core power control scheme of our heterogeneous many-core architecture are described in the following.
* A: Active / HC: Hot Core / CC: Cold Core / I: Idle / PD: Powered Down

Figure 14.1: 3-bit Core Power Control Scheme under FSM.
A_{100}: Operational state, Core fully powered up.

HC_{011}: Keep Core main internal clock slow if it is the hot core.

CC_{010}: Stop Core main internal clock if it is the cold core, but the PLL keeps running.

I_{001}: Stops Core main internal clock and the PLL. (It might spend 400 cycles in 2GHz).

PD_{000}: Core fully powered down. (It might spend 1000 cycles in 2GHz).

Figure 14.2: 3-bit Core Power Control Scheme under the Operating Sequence.
* PMU: Power Management Unit / PLL: Phase-Locked Loop as a clock generator

Figure 14.3: Power and Clock Distribution.
14.1 Active Status

When the core is in the active status, the core is also defined to be in status A as an acronym. In status A, depending on the current job scheduling, instructions are actively executed by a particular core or all cores.

14.2 Hot Core Status

When the core is in the hot core status, the core is also defined to be in status HC as an acronym. If the core is decided to be the hot core by OS, some instructions are executed with half-reduced slow clock to stay alive. Based on the cores usability profiling information and real time sensing information provided from the temperature sensor of an each core, we can predict that this core might be reused in the near future.

14.3 Cold Core Status

When the core is in the cold core status, the core is also defined to be in status CC as an acronym. If the core is decided to be the cold core by OS, no instructions are executed. In addition, the clocks of all clock trees pertaining to the cold cores are turned off. The phase-locked loop (PLL) for the core is still alive in this status, even though the clocks are off by utilizing a technique known as clock gating. Therefore, we can significantly reduce the latency without changing the status from Idle to Active. The only power which remains to be consumed is caused by leakage current. However, it is compensated by the power saving from the latency reduction.
14.4 Idle Status

When the core is in the idle status, the core is also defined to be in status I as an acronym. While in idle status, the PLL for the core is turned off, and the core cache is flushed. A core in this status is considered an inactive core. The wakeup time for this status is significantly longer than in slow active because the core cache must be restored. In addition, the PLL requires time to be stabilized for generating the correct frequency.

14.5 Powered Down Status

When the core is in the powered down status, the core is also defined to be in status PD as an acronym. While in powered down status, the PLL for the core is turned off, and the core cache is also flushed. Additionally, the core status is saved to the L3 cache, namely the last level cache (LLC) here. Further, the core is power gated to reduce power consumption to the core to approximately zero watts. A core in this status is considered an inactive core. The wakeup time for this status is the longest because the core status must be restored from the L3, the core PLL must be stabilized, the power gating must be deactivated, and core clock must be turned back on.

An interesting situation may occur in idle and powered down status. Because I and PD make significant latency for powering up to A status in this architecture model, the energy cost to transit to and from these statuses is very high, especially in PD status. An extreme energy loss can be resulted by frequent transition in and out of these statuses. To prevent this, our 3-bit core power control scheme is very useful to determine when SA status savings justify the energy cost of transitioning into I and PD statuses and then transition back to A. The power management unit (PMU) requests the OS to stay I status if there is not enough justification to transition to PD status.
Table 14.1: Processor Power Design Space

<table>
<thead>
<tr>
<th>Core Clock</th>
<th>Active</th>
<th>Hot(^1) Core</th>
<th>Cold(^2) Core</th>
<th>Idle</th>
<th>Powered Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLL</td>
<td>On</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>Core Power</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>PMU</td>
<td>On</td>
<td>On</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
</tr>
<tr>
<td>Core Caches</td>
<td>Keep</td>
<td>Keep</td>
<td>Keep</td>
<td>Flushed</td>
<td>Flushed</td>
</tr>
<tr>
<td>Shared Cache</td>
<td>Keep</td>
<td>Keep</td>
<td>Keep</td>
<td>Keep</td>
<td>Keep</td>
</tr>
<tr>
<td>Wakeup Time</td>
<td>Active</td>
<td>Few Cycles</td>
<td>Few Cycles</td>
<td>Few Cycles</td>
<td>Few Cycles</td>
</tr>
</tbody>
</table>

\(^1\)Hot core: the frequently used core  
\(^2\)Cold core: the less frequently used core

Table 14.2 shows fine grain power models along with each core status such as active, standby, total, and wakeup power.

We assumed that the power consumption per cycle is approximately 200W at 2GHz. In addition, each job was characterized in the same operating conditions and no competing tasks were performed during its execution. Furthermore, Equation (14.1) shows the full chip level power consumption by combining the summation of all cores power consumption (i.e., \(1 \leq i \leq n\), \(n\) is the total number of cores) with some peripherals power consumption.

\[
P_{\text{chip}} = \sum_{i=1}^{n} P_{\text{core}(i)} + P_{\text{peripherals}}
\]  

(14.1)
<table>
<thead>
<tr>
<th></th>
<th>Active</th>
<th>Hot Core</th>
<th>Cold Core</th>
<th>Idle</th>
<th>Powered Down</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Active (Dynamic)</strong>&lt;br&gt;CV^2F</td>
<td>Max_D</td>
<td>Max_D x 50%</td>
<td>N/A</td>
<td>N/A</td>
<td>None</td>
</tr>
<tr>
<td><strong>Standby (Leakage)</strong>&lt;br&gt;VI_{off}</td>
<td>N/A</td>
<td>N/A</td>
<td>Max_L</td>
<td>Max_L x 60-80%</td>
<td>None</td>
</tr>
<tr>
<td><strong>Total</strong>^1</td>
<td>Max_D ≥ (Max_D x 50%)</td>
<td>Max_L</td>
<td>≥ (Max_L x 60-80%)</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td><strong>Wakeup</strong>^2</td>
<td>N/A</td>
<td>Few Cycles x 200W^3</td>
<td>Few Ten Cycles x 200W^3</td>
<td>Few Hundred Cycles x 200W^3</td>
<td>Few Thousand Cycles x 200W^3</td>
</tr>
</tbody>
</table>

^1Total = Active + Standby <br>
^2Wakeup = Cycle count x Power per cycle <br>
^3Power per cycle: Approximately 200Watts at 2GHz
Chapter 15

Power-Aware Thread Placement

This Chapter elaborates our 3-bit core power control scheme in hardware. In Chapter 14, we started it was stated that the status changing between several core statuses is triggered by OS. As shown in Fig. 14.1, namely the FSM of 3-bit core power control scheme, such type of status transition is created by core’s On/Off signal. The meaning of “core status changing is triggered by OS” is exactly based on the power-aware thread placement. Nowadays, thread placement is performed in a largely power-aware fashion. Consolidating of active threads into fewer cores particularly exposes opportunities for power savings. The opportunity to save power is especially high in this situation because PCPG is becoming viable. This can be applied to unused cores that are excluded from thread consolidation.

The hardware-software interaction is significantly affected by incorporating multiple threads and cores into the same chip. It is a software-level responsibility to allocate software threads across available hardware threads, with little to no hardware control. Hence, programmers rely on OS schedulers to allocate software threads across hardware threads. In addition, preferred thread allocations can be explicitly established by setting affinities between software and hardware threads.
A desired power-performance efficiency needs to be achieved by using the optimum combination of core-wise simultaneous multi-threading (SMT) level and number of active cores. To the best of my knowledge, this type of approach has not been explored in previous work nor implemented as part of the OS task scheduler. However, this approach has a limitation. The best thread allocation policy to improve performance and how thread allocation could impact power consumption are not always evident under multi-threaded workloads. However, it is an inevitable trend for this research field. Thus, this Chapter aims to provide an optimal solution.

In Fig. 15.1, configuration “4x2” allows us to switch the four cores that are left unused to idle state. By setting CPU affinities, software (SW) threads are pinned to specific hardware (HW) threads. Total HW threads (T) are generated by multiplying the number of cores (C) by SMT threads per core (S). For instance, in this case, 16 is to multiply 8 by 2. We benefit from unused cores to reduce power consumption (e.g. by switching them to an hot/cold core status or idle/powered-down status). Through the coherent fabric, in addition, each core can effectively access shared data that are located in remote L2 and L3 caches. When configuration “4x2” is adopted, execution time is just increased by 5%, chip power consumption is reduced by slightly more than 20% [68], [69]. The significant inter-thread data sharing present in this application accounts for such small degradation in performance when the number of cores is halved. The number of accesses to remote cache regions in the chip significantly decreases when software threads are executed closer (e.g., sharing the same cache hierarchy). Very few prior work has discussed the idea of thread consolidation in SMT-enabled Chip Multiprocessors (CMP). Among the very few, the best work we think is as follows [75]. It packed software threads onto a variable number of cores to fit a given power budget, in conjunction with DVFS. The work examined DVFS configurations and thread packing to maximize performance within variable power caps. However, the actual software-hardware thread mapping was not taken into account and SMT was deliberately disabled.
Figure 15.1: Hardware-Software Thread Interaction.
Multi-threaded applications may not benefit from statically pinning software threads to fewer cores. It strongly depends on the applications characteristics during its different execution phases. Therefore, to maximize power-performance efficiency, we need to dynamically find sweet spots that represent particular software thread placements. It is convenient to use a thread consolidation which is the method of using fewer cores at higher SMT levels.

Our objective, during an applications execution, is to detect the timing when it is possible to consolidate threads with minimal or zero impact on performance. Similarly, if threads placed in the same core face high inter-thread conflicts, the heuristic unconsolidates them to mitigate the situation. To build and evaluate a heuristic capable of finding the most power-performance efficient thread mappings at runtime, we required to have a metric to quantify such efficiency. As shown in Equation (15.1), this is an efficiency of thread consolidation ($E_{TC}$). It is also used in [70]. If $E_{TC}$ is larger than 1, the new mapping (mapping A) provides larger power-performance efficiency than the previous one (mapping B). Ideally, we are interested in actions (consolidations or unconsolidations) with $E_{TC}$ values that are larger than one. If $E_{TC}$ is smaller than 1, the new mapping is less power-performance efficient than the previous one. Finally, if $E_{TC}$ is equal to 1, both mappings equally perform power-performance efficiency. For instance, assuming that consolidation reduces chip power consumption by 20% or 50% and, at the same time, degrades performance by 5% or 40%. $E_{TC}$ is 1.20 either 0.95/0.80 or 0.60/0.50, which is almost the same. Even if the new mapping is more power-performance efficient (with 50%), such a severe performance degradation (with 40%) is not acceptable.

$$E_{TC} = \frac{Perf(mapping_A)/Perf(mapping_B)}{Power(mapping_A)/Power(mapping_B)}$$

(15.1)

To the best of my knowledge by this metric, we figured out heuristic thread consolidation
approach in Fig. 15.2. This approach aims to benefit from thread consolidation at run-time to minimize power consumption with minimal (or zero) impact on performance. Also, it evaluates $E_{TC}$ during run time with respect to application’s performance and power consumption. In this situation, performance indicates throughput representing the total number of instructions completed by all the threads per cycle, and power means chip power consumption. It is triggered every $T$ milliseconds, but makes decisions only if it is enabled.

This heuristic approach sets the most unconsolidated mapping for that particular thread bucket to not harm application performance in the initial step. Before making any consolidation and unconsolidation decision, heuristic thread consolidation method check a cluster of available hardware threads. Based on the number of current software thread ($N$), a bunch of threads (bucket) is defined as the minimum number of hardware threads (power of 2) that are required to support the current number of software threads. For instance, if the current number of software threads is $N = 4$, the bucket to be used is 4, or in case $N = 13$, the bucket to be used is 16. Heuristic thread consolidation method can identify what the possible thread mappings it can play with are by choosing the right bucket. All possible thread buckets in POWER7 [74] are surveyed as 1, 2, 4, 8, 16 and 32 threads. Once a bucket is chosen, the heuristic thread consolidation method sets the most unconsolidated mapping for that particular bucket to not harm application performance. Throughout the example, we will refer to this mapping as $\text{mapping}_{\text{prev}}$. Consolidation is the first action of the heuristic thread consolidation method. Right after selecting a bucket, no power-performance history exists because the previous bucket might have a completely different power-performance footprint. Therefore, the heuristic thread consolidation method begins evaluating the effects of consolidation on power-performance efficiency. We refer to this new mapping as $\text{mapping}_{\text{next}}$. $E_{TC}$ is then computed and analyzed to compare $\text{mapping}_{\text{next}}$ versus $\text{mapping}_{\text{prev}}$. If $E_{TC}$ is larger than the average profiled $E_{TC}$, the heuristic thread consolidation method assumes that the application is traversing a consolidation-friendly phase, and further consolidates threads when possible. However, if, $E_{TC}$ is smaller than the average profiled $E_{TC}$, the heuristic
Figure 15.2: Outline of Heuristic Thread Consolidation Method.

The thread consolidation method considers that mapping $next$ is less power-performance efficient than mapping $prev$, and returns to mapping $prev$. Heuristic thread consolidation method continues in this way, by analyzing the latest $E_{TC}$, and makes a new decision based on that. Whenever if $E_{TC}$ is not changed, the heuristic thread consolidation method does not make a decision.
Chapter 16

Evaluation And Methodology

In our previous work [32], we simulated the execution of 10 benchmarks from the Rodinia benchmark suite [33]. Benchmarks are simulated using the recent architectural simulator, namely gem5GPU [38], which combined with gem5 [36] and GPGPU-Sim [37] on an unmodified x86 64bits Linux 2.6. They simulated 256 in-order x86 CPU cores at 2GHz and distributed into 16 tiles, and 768 GPU cores at 600MHz and also distributed into 16 tiles. The gem5GPU system obeys traditional MOESI cache coherence protocol [39], and the GPU caches are write-through and obey the VI-based cache coherence protocol. The detail parameters of the simulation infrastructure are reported in Table 16.1.

We get some benefits from this work [32], especially focusing on the design of fusion coherence by two-level directory. It is directly accessing uniformed L3 data cache, which sharply decrease the execution time and latency of accessing memory system. It is due to bypassing L3 data cache guided by fusion directory, directly accessing memory space instead of copying data between each other and transmitting over system bus. As a result, the average speedup is 2.4X and the maximum speedup is more than 4X. However, some benchmarks do not achieve the high speedup. These benchmarks are mainly computing-intensive, and spend a
significant execution time on CPU core instead of transferring a large mount of data between CPU and GPU memory.

Emerging manycore processors have also brought new challenges under available power budgets to the architecture research community. Manycore processors are highly integrated complex system-on-chips with complicated core and uncore subsystems. The core subsystems can consist of a large number of traditional and asymmetric cores. The uncore subsystems have also become unprecedentedly powerful and complex with deeper cache hierarchies, advanced on-chip interconnects, and high-performance memory controllers. In order to conduct research for emerging manycore processor systems, a microarchitecture-level and cycle-level manycore simulation infrastructure is needed. Numerous processor and system simulators are already available. All of these simulators have their own merits and serve their different purposes well. However, according to our specific research goal with regard to power awareness, we try to compare the existing two remarkable simulators such as gem5GPU and Sniper with various dimensions of features as shown in Table 16.2.

Based on Table 16.2, we are able to know that Sniper provides the detail necessary to estimate power across many-core architecture. Sniper [79], [80], [81] is also satisfied with fast and scalable simulation under two major trends in high-performance computing, namely, large numbers of cores and the growing size of on-chip cache memory. By bringing together
Table 16.2: Feature’s Summary of Existing Well-Known Simulators

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>Which one is more proper?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Simulation Speed</td>
<td>Somewhat slow(+)</td>
<td>Faster(+++)</td>
</tr>
<tr>
<td>2</td>
<td>Simulation Accuracy</td>
<td>N/Av</td>
<td>Within 25%</td>
</tr>
<tr>
<td>3</td>
<td>Power Estimation</td>
<td>N/Ap</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Many-Core Support</td>
<td>P/S</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Caches</td>
<td>Y/A</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Heterogeneous Configuration Support</td>
<td>Y/A</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Advanced Visuable Support</td>
<td>N/Ap</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* N/Ap: Not Applicable
* N/Av: Not Available
* P/S: Partially Support
* Y/A: Yes with Advanced Future Support

Accurate high-abstraction analytical models with fast parallel simulation, architects can trade off accuracy with simulation speed to allow for longer application runs, covering a larger portion of the hardware design space. In addition, Sniper is integrated with McPAT [82], an integrated power, area, and timing modeling framework that supports comprehensive design space exploration for many-core architecture under processor configurations ranging from 90nm to 22nm and beyond. An extended description of these simulators categorized by specific features is introduced, which sections in Appendix A and B on the last part of the dissertation.

In our work, we utilize the data from offline profiling of the Splash-2 benchmark suite [35] and implement a binning approach to categorize unknown threads as their nearest neighbor in the Splash-2 benchmark suite. Furthermore, we plan to extend this profiling by using the Parsec benchmark suite for considering a variety of exceptional cases. Based on some test cases
such as fft, and fft-hetero, we have shown an architectural topology, power consumption, and cycle per instruction (CPI) stack (see Fig. 16.1, 16.2, and 16.3) under visualization support to gain insight into lost cycles. Furthermore, we have simulated our ideas by using the above mentioned benchmark and also we have compared the results with the default setting of Sniper. In the context of the benchmark suite, the power consumption is reduced by up to 2.3% (averaged across applications) which our ideas is adopted (see Fig. 16.4, and 16.5). In addition, the power consumption in consideration of the performance impact is also compared and analyzed with an each program of the Splash-2 benchmark suite as shown in Fig. 16.6 and Fig. 16.7.

Based on our experiment, we are able to expect better results under real chip implementation with thermal sensor’s information and PLL’s working model in analog level. It becomes more evident as the number of cores and threads keep growing, which makes the hardware-software tight cooperation necessary for power benefits.
Figure 16.1: Architectural Topology of FFT and FFT-HETERO Test Case. - Generated Results from McPAT framework
Figure 16.2: Power Consumption of FFT and FFT-HETERO Test Case. - Generated Results from McPAT framework
Figure 16.3: CPI Stack of FFT and FFT-HETERO Test Case. - Generated Results from McPAT framework
Figure 16.4: Simulated Power and Energy Consumption of Each Unit of Cores - 8 and 16 Cores

<table>
<thead>
<tr>
<th>Core</th>
<th>8 Cores</th>
<th>16 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>12.35</td>
<td>0.31</td>
</tr>
<tr>
<td>ifetch</td>
<td>3.87</td>
<td>0.10</td>
</tr>
<tr>
<td>alu</td>
<td>2.33</td>
<td>0.06</td>
</tr>
<tr>
<td>int</td>
<td>3.92</td>
<td>0.10</td>
</tr>
<tr>
<td>fp</td>
<td>5.94</td>
<td>0.15</td>
</tr>
<tr>
<td>mem</td>
<td>3.35</td>
<td>0.08</td>
</tr>
<tr>
<td>ru/lsu</td>
<td>7.59</td>
<td>0.19</td>
</tr>
<tr>
<td>core all</td>
<td>39.35</td>
<td>1.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core</th>
<th>8 Cores</th>
<th>16 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>10.75</td>
<td>0.27</td>
</tr>
<tr>
<td>ifetch</td>
<td>3.85</td>
<td>0.10</td>
</tr>
<tr>
<td>alu</td>
<td>2.31</td>
<td>0.06</td>
</tr>
<tr>
<td>int</td>
<td>3.93</td>
<td>0.10</td>
</tr>
<tr>
<td>fp</td>
<td>5.94</td>
<td>0.15</td>
</tr>
<tr>
<td>mem</td>
<td>3.31</td>
<td>0.08</td>
</tr>
<tr>
<td>ru/lsu</td>
<td>7.54</td>
<td>0.19</td>
</tr>
<tr>
<td>core all</td>
<td>37.63</td>
<td>0.94</td>
</tr>
</tbody>
</table>
Figure 16.5: Graphical Results of Simulated Total Power and Energy Consumption of Cores - 8 and 16 Cores
Figure 16.6: Core Power Consumption of Each Program of Splash-2 Benchmark - 8 and 16 Cores
Figure 16.7: Speedup in Execution Time of Each Program of Splah-2 Benchmark - 8 and 16 Cores
Chapter 17

Expanded Works

Power depends not only on the configuration of a processor, but also on the circuit design style and process parameters. Also actual power dissipation varies with activity, though the degree of variability again depends on the technology parameters as well as the gating style used. No existing architecture-level power modeling framework accounts for all of these factors. In addition, our implemented model in Sniper has clear limitations in clock control scheme such as phase locked loop and clock reset generator. Therefore, we should apply our ideas to real chip implementation in conjunction with top-tier semiconductor company, analyze the result, and compare the result with our simulated result in Sniper. We think that this approach is fairly meaningful trial to extract the best experimental value out of the silicon, and a firm foundationl for long-term research impact.
Chapter 18

Conclusions

In this Part III, we have introduced and sought to gain some insights into the power benefits available for the future architecture, that of a heterogeneous many-core architecture on the same chip, using a 3-bit core power control scheme and heuristic thread consolidation approach. The particular opportunity examined is for powering a new core up from powered down when we have an application switching among cores. As a result, it reduces a huge latency and a power dissipation for power the core up from powered down. Operation is based on distinct scenarios by 3-bit core power control scheme through 5 statuses switching. In addition, for more elaborated control, this kind of status switching is exactly triggered by using both the collected process ID information which is allocated by OS scheduler and the decision information of heuristic thread consolidation scheme to maximize power-performance efficiency. We choose to evaluate our ideas on programs from the Splash-2 benchmark suite. The benchmark suite showed an average power saving of 2.3% compared to a system with an efficient power-aware policy and by up to 15% with respect to the basic policy. The larger power savings of the proposed techniques are most likely due to the optimal interaction through hardware and software between a 3-bit core power control scheme and heuristic thread consolidation approach based on the maximum possibility of thread consolidation of
the simulated cases. Expanding these classes of ideas in the power domain will open up new research possibilities for embedded systems community in such areas as automobiles, smartphones, cameras, tablets, PCs, and medical systems to high-performance computing systems such as cloud servers and supercomputers.
Part IV

POWER-ENERGY EFFICIENCY MODEL BY OVERHEAD OF DATA PREPARATION
Chapter 19

Introduction

As multicore processors have become mainstream, it has thus become crucial to identify performance bounds and performance scaling properties in exploiting the massive parallelism they may offer [88]. Computer architecture has been transiting from the homogeneous multicore era into the heterogeneous era [89], which means that the memory wall [90] and communication issues will increase the gap between the performance of an ideal processor and that of a practical processor because the overhead of data preparation becomes an unavoidable key parameter.

Furthermore, energy efficiency is one of the most challenging issues as large-scale increase trend of integration CMOS devices which has led to fused architectures with superscalar central processing units (CPUs) and light-weighted streaming processor units (e.g. graphics processing units (GPUs), FPGA accelerators and ARM cores). Multicore systems with heterogeneous processing elements are becoming the mainstream in the field of future processor design. Recent examples include Intels MIC [98], AMDs Kabini [99], and NVIDIA’s Project Denver [23], etc. Especially, as the rapid development of manufacture process and promising technologies, it becomes probable to make heterogeneous kilo-core system by integrating the
general-purpose (nongraphics or data flow) computing units and GPGPU on a single chip in the near future.

Here, we classify the processors(cores) in heterogeneous mutlicore system as big processor and little core respectively. As the density on multicore chip increase, future heterogeneous multicore parallel system will have to take seriously consideration on how to achieve higher performance and manage hardware/software resources while maintaining their power(energy) consumption within a limited budget. This challenge will stimulate multicore processor architect to develop new approaches that pursuit better performance per watt rather than simply yielding higher performance.

In this paper we model computer system, from the view of data ow computing model, by extending the fraction of data preparation rather than the state-of-art Amdahls law on basis of computing-centric system which never takes into account potential cost of data preparation. Our contributions are mainly (1) Extending the performance model by considering the overhead of data preparation, and then making new equations to evaluate performance of heterogeneous parallel multicore system; (2) Building a performance-energy efficiency model and evaluating heterogeneous parallel multicore system; (3) Comparing the results to that of traditional Amdahl’s law.

This Part IV is organized as follows. Chapter 20 presents some related works. Chapter 21 presents a performance model of integrated heterogeneous multicore system by considering the overhead of data preparation after revisiting traditional Amdahl’s law. And also, it proposes performance-energy efficiency model for heterogeneous multicore system and creates a new equation of describing performance-energy ratio by referring to some basic variables from [100], [110]. Chapter 22 evaluates the performance-energy of heterogeneous multicore system by comparing to the results of traditional Amdahl’s law. Chapter 23 concludes our work and future missions.
Chapter 20

Related Work

There are a lot of research achievements in theory to harness power(energy) consumption [100], [90], [101], [102], [103], [104]. Woo and Lee [100] extend Amahl’s law for energy-efficient computing of many-core, who classified many-core design styles as three types: symmetric superscalar processor tagged with P, symmetric smaller power-efficient core tagged with c, and asymmetric many-core processor with superscalar processor and many smaller cores tagged with P + c. The research results show that heterogeneous architecture is better than symmetric system to save power. Similarly, Marowka[8] extends Amdahls law for heterogenous computing, and investigated how energy efficiency and scalability are affected by the power constraints for three kind of heterogeneous computer system, i.e., symmetric, asymmetric and simultaneous asymmetric. The analysis shows clearly that greater parallelism is the most important factor affecting power consumption [90]. Karanikolaou, et al evaluate experimental energy consumption based on distributed and many-core platforms. They evaluated for the power their processors demand at the idle and fully utilized state. In proportion to the parallelized percentage each time, the estimations of the theoretical model were compared to the experimental results achieved on the basis of the performance/power and performance/energy ratio metrics [101]. Kim et al focus on the energy efficiency of the se-
quential part acceleration, and how to determine the optimal frequency boosting ratio which maximize energy efficiency. According to the results, energy efficiency of the acceleration increases with the number of cores and an optimal frequency boosting ratio can be determined. Accelerating the sequential part of a program is a promising approach to improve overall performance in parallel processors [102]. Londono et al present a study about the potential dynamic energy improvement that can be achieved when hardware parallelization is used to increase the energy efficiency of the system rather than to increase performance. They model the potential dynamic energy improvement, optimal frequency and voltage allocation of a multicore system in terms of extending Amdahls law [103]. Ge et al propose a power-aware speedup model to predict the scaled execution time of power-aware clusters by isolating the performance effects of changing processor frequencies and the number of nodes. By decomposing the workload with DOP and ON/OFF-chip characteristics, this model takes into account the effects of both parallelism and power aware techniques on speedup [104].

In eld of practical products or implementations, low power techniques and algorithms for multicore system, such as dynamic voltage/frequency scaling (DVFS) and heterogeneous microarchitectures, are recommend to reduce power (energy) by lowering the voltage and frequency or by migrating execution to a more efficient, but smaller size of chip [105], [106], [32], [107], [108], [109]. Sawalha and Barnes demonstrate that significant reduction in energy consumption can be achieved by dynamically adjusting mapping as application behavior changes with new program phases. Significant energy reduction over random scheduling of programs within a heterogeneous multicore processor [105]. Nowak employs the Convey HC-1, a heterogeneous system equipped with four user-programmable FPGAs, for our investigations toward energy-efficient computing. He nd that heterogeneous systems based on reconfigurable hardware, efficient data exchange mechanisms, data-driven and component-based programming, and task-parallel execution can help achieve power-efficient exascale systems in future [106]. Lukefahr et al developed an ofine analysis tool to study the potential energy efficiency of ne-grained DVFS and heterogeneous microarchitectures, as well as a
hybrid approach [107]. Our prior work on designing a fused cache with compacted cache directories and a framework of accessing unified memory address space of heterogeneous parallel multicore system [32], [107], [108]. Wang and Ren coordinate inter-processor work distribution and per-processors frequency scaling to minimize energy consumption under a given scheduling length constraint. Through several evaluations on a real CPU-GPU system, our results gain 14% energy reduction compared with static mapping strategy [109].
Chapter 21

Power-Energy Efficiency Model of Heterogeneous Multicore System

Four decades ago, Gene Amdhal mainly focused on performance of computer for a special case of using multiple processors in parallel when he argued for the single-processor approach’s validity for achieving large-scale computing capabilities [93]. Here, we are more interested in the power-efficiency or energy-efficiency of future integrated heterogeneous multi-core processor system. Hence, we develop analytical power models of integrated heterogeneous multicore and formulate metrics to evaluate energy-efficiency on the basis of performance with considering the overhead of data preparation.

We adopt the variables $s_c$, $w_c$, $k$ and $k_c$ from [100], where $s_c$ represents a little cores performance normalized to that of a big processor ($0 \leq s_c \leq 1$), $w_c$ represents an active little core’s power consumption relative to that of an active big processor ($0 \leq w_c \leq 1$), $k$ represents the fraction of power that the big processor consumes in idle state ($0 \leq k \leq 1$), and $k_c$ represents the fraction of an little core’s idle power normalized to the same cores overall power consumption ($0 \leq k_c \leq 1$). Assumed that the big processor in active state consumes a
During the sequential fraction of executing a given program, the amount of power that the big processor consumes is 1, and the amount of power that \( c \) little cores at idle state consume is \( c \times w_c \times k_c \). During the parallel fraction of executing a given program, the big processor consumes \( k \), and the \( c \) little cores consume \( c \times w_c \). Because the cost (performance) of executing sequential and parallel portion are in equation 21.1 respectively, the average power is in equation of Fig. 21.1.

\[
pw_s = (1 - f_c) \times p_c + f_{ud} \times (1p_c), \quad pw_p = \frac{f_c}{c \times s_c} \times p_c + f_{pd} \times (1 - p_c) \quad (21.1)
\]

We can also model, similar to [100], the performance (speedup) per watt (S/W) equation which represents the performance achievable at an average power (W) in equation of Fig. 21.2.
Chapter 22

Evaluation and Analysis

Assumed that the integrated heterogeneous multicore system consists of one big processor (e.g., superscalar multicore CPU processor) and \( c \) little cores (e.g., GPU cores). In order to compare the results of S/W with that in [100], we also set \( s_c, w_c, k \) and \( k_c \) as 0.5, 0.25, 0.3 and 0.2 respectively. We set the variables \( p_c, f_h \), and \( d_c \) as 0.6, 0.8, 0.699 and 0.333 respectively.

Figure 22.1 shows the analytical results of performance for asymmetric multicore system. Figure 22.1a shows that the maximum relative performance of \( P + c \) is 56.39, where \( f = 0.99 \) and \( c = 256 \). As shown 5 curves in Figure 22.1a, the relative performance dramatically increases as the number of little cores increase when \( f = 0.99 \). It means that a little bit more percentage of parallel computation would bring huge performance gain. For example, the curve of \( f = 0.99 \) and \( f = 0.9 \), only 10% increase on parallel computation gets more than 5 times increase of performance where \( c = 256 \). However, the performance increase of each curve in Figure 22.1b is much more gently than that in Figure 2(a), and the maximum of relative performance is just only 3.35 where \( f = 0.99 \) and \( c = 256 \). The reason is that much more cost would be consumed by data preparation, and the increase of relative performance
is the result of balancing computation and data preparation. The Figure 22.1b also shows that a program with low percentage of parallel computation, such as $f = 0.3$, can not get great relative performance increase even continuously increasing the number of little cores up to 256. Unfortunately, even though the percentage of parallel computation is higher than 90%, it can not get dramatic increase as that in Fig. 22.1a.

As shown in the Fig. 22.2a and Fig. 22.2b, for each kind number of little cores, higher fraction of parallelizable computation in a given program gets higher relative performance per watt. For example, as shown in the Fig. 22.2b, the values of relative performance per watt are respectively 0.5397, 0.5877, 0.6452, 0.7517 and 0.7517 where $f = 0.3$, $f = 0.5$, $f = 0.7$, $f = 0.9$ and $f = 0.99$ in the case of $c = 16$. It means that a program consisting of higher fraction of parallel execution consumes more energy. We can easily observe that the trend of curves in the Fig. 22.2a are also similar to that in the Fig. 22.2b. The overall situation of state-of-art asymmetric multicore system is identical to that of integrated heterogeneous parallel multicore system.

In Fig. 22.2a, the values of relative performance per watt decrease after $c = 4$ ($f = 0.9$ and $f = 0.99$) or $c = 8$ ($f = 0.3$, $f = 0.5$, and $f = 0.7$). It means that as the number of little cores increase after the critical value of $c$, on average, the performance would become worse and worse within a xed budget of energy. The reason is as the number of little cores increases for a xed fraction of parallelizable computation in a given program, the performance of little cores will saturate. Therefore, the performance per watt decrease after the critical value of little cores. More little cores would consume more energy to maintain the synchronization, communication, cache coherence, etc rather than improve performance.

However, if a program with higher fraction of computing parallelism, the slop of performance decrease becomes less slowly. For example, the curve corresponding to $f = 0.99$ is over the curve corresponding to $f = 0.9$. The reason is that higher fraction of parallel computation will have enough computing tasks to be executed on more little cores in parallel.
(a) Woos State-of-Art Asymmetric Multicore System P + c.

(b) Integrated Heterogeneous Parallel Multicore System with considering ODP where $p_c = 0.6$, $f_h = 0.8$, $s_c = 0.699$, and $d_c = 0.333$.

Figure 22.1: Scalable Performance Distribution of Heterogeneous Asymmetric Multicore (HAM) where $s_c = 0.5$, $w_c = 0.25$, $k = 0.3$, and $k_c = 0.2$. 
(a) Woos State-of-Art Asymmetric Multicore System $P + c$.

(b) Integrated Heterogeneous Parallel Multicore System with considering ODP where $p_c = 0.6$, $f_h = 0.8$, $q = 0.699$, and $d_c = 0.333$.

Figure 22.2: Scalable Performance per Watt Distribution of Heterogeneous Asymmetric Multicore (HAM) where $s_c = 0.5$, $w_c = 0.25$, $k = 0.3$, and $k_c = 0.2$. 
There is a special case in the Fig. 22.2b, the value of performance per watt is 1 where \( f = 0.3 \) and \( c = 1 \). Here, it’s the heterogeneous multicore system built with only one big processor and only one little core. The value of performance per watt discloses that the sequential cost of computation and data preparation consumes the same amount of energy to that of parallel computation and data preparation. The interesting thing is that the point is \( f = 0.3 \) which means the energy of executing 30% parallel computation in a given program is approximately identical to the energy of implementing 70% data preparation.

Comparing Fig. 22.2a with Fig. 22.2b, we observe that the ve curves in the Fig. 22.2b are more identical than that in the Fig. 22.2a, and they are approximate to the point around 0.1. Because some parts of sequential data preparation are executed by the big processor, and some parts of parallelizable data preparation are executed by little cores, whether the fraction of parallel computation is high or not, which can make a stable balance between sequential tasks and parallel tasks.

The maximum value of relative performance per watt for a given case (e.g. the number of little cores is 4) in Fig. 22.2a and Fig. 22.2b are 1.53 and 1.13 respectively. It means that the integrated heterogeneous multicore system would sacrifice 26% computing performance to implement data preparation within a fixed budget of energy, comparing to the performance in the state-of-art asymmetric multicore system based on traditional Amdahl’s law.

After considering the overhead of data preparation, the relative performance per watt will decrease much more sharply that that in the Fig. 22.2a as the number of little cores increases. The reason is that the energy consumed by data preparation tasks is not taken consideration in terms of legacy performance formula of Amdahl’s law. In practical computer system, as number of little cores increase, there would be more costs of executing data preparation, such as synchronization, communication, data transformation, etc. However, the time consumed by executing data preparation can not gain speedup performance according to traditional Amdahl’s law rather than waste energy. As a result, the Fig. 22.2b highly reects the practical
relative performance per watt than that in Fig. 22.2a.
Chapter 23

Conclusions

We reevaluate the performance-energy efficiency for integrated heterogeneous parallel multicore system based on the mathematical model of data ow, which splits a program(task) into computation and data preparation rather than only computation in the state-of-art Amdahls law. By deriving the new equation of performance-energy based on the analytic model of integrated heterogeneous multicore system with considering the overhead of data preparation, we get a new result from it and compare it with that of Woos work.

The results show that our model is much more closely reecting practical heterogeneous multicore system than that in traditional Amdahls model. According to the results, more little cores would not bring signicant performance gain after passing the critical point (e.g. c = 4 or c = 8) other than consuming more energy. Therefore, as for the heterogeneous multicore system built with one big core and many little cores, which has limited budget of energy to execute a program(task). Its gain of performance per watt is increased by improving the fraction of parallel task including parallel computation and parallel data preparation, rather than by increasing the number of little cores. Generally, the fraction of parallel computation in a given program is hard to be improved by instructions level parallelism (ILP),
threads level parallelism (TLP) nowadays. However, I have another approach to accelerate performance by improving the fraction of parallel data preparation, such as data prefetching, no-copy data transfer, solid-state storage, computing on memory and other emerging technologies.

We will keep on investigating the performance gain within limited energy (or power) budget for supercomputer or cluster built with multiple big processors and many little cores while considering the overhead of data preparation.
Bibliography


102


[78] A. Bhattacharjee, and M. Martonosi, “Thread Criticality Predictors for Dynamic Per-


Appendix A

Sniper: Scalable and Accurate Parallel Multi-Core Simulator

Sniper is a next generation parallel, high-speed and accurate x86 simulator. This multi-core simulator is based on the interval core model [83] and the Graphite simulation infrastructure [84], allowing for fast and accurate simulation and for trading off simulation speed for accuracy to allow a range of flexible simulation options when exploring different homogeneous and heterogeneous multi-core architectures.

The Sniper simulator allows one to perform timing simulations for both multi-program workloads and multi-threaded, shared-memory applications with 10s to 100+ cores, at a high speed when compared to existing simulators. The main feature of the simulator is its core model which is based on interval simulation, a fast mechanistic core model. Interval simulation raises the level of abstraction in architectural simulation which allows for faster simulator development and evaluation times; it does so by ‘jumping’ between miss events, called intervals. Sniper has been validated against multi-socket Intel Core2 and Nehalem systems and provides average performance prediction errors within 25% at a simulation speed of up to
several MIPS.

This simulator, and the interval core model, is useful for uncore and system-level studies that require more detail than the typical one-IPC models, but for which cycle-accurate simulators are too slow to allow workloads of meaningful sizes to be simulated. As an added benefit, the interval core model allows the generation of CPI stacks, which show the number of cycles lost due to different characteristics of the system, like the cache hierarchy or branch predictor, and leads to a better understanding of each component’s effect on total system performance. This extends the use for Sniper to application characterization and hardware/software co-design.

A.1 Intel Nehalem Architecture

Nehalem-based microprocessors, as can be seen in Figure A.1 and Figure A.2, use the 45nm process, run at higher clock speeds, and are more energy-efficient than the older microprocessor. Hyper-threading is reintroduced, along with a reduction in L2 cache size, as well as an enlarged L3 cache that is shared among all cores. It involved some technologies as shown in Table A.1.

It has been reported that Nehalem has a focus on performance, thus the increased core size. Compared to the older microprocessor, Nehalem has 10-25% better single-threaded performance / 20-100% better multithreaded performance at the same power level 30% lower power consumption for the same performance. On average, Nehalem provides a 15-20% clock-for-clock increase in performance per core. Overclocking is possible. Nehalem processors incorporate SSE 4.2 SIMD instructions, adding seven new instructions to the SSE 4.1 set in the Core 2 series. The Nehalem architecture reduces atomic operation latency by 50% in an attempt to eliminate overhead on atomic operations.
Figure A.1: Nehalem Microarchitecture. - Image from Nehalem Processor at Intel.com
Figure A.2: Nehalem Core Die. - Image from Nehalem Processor at Intel.com
Table A.1: Technology of Nehalem

<table>
<thead>
<tr>
<th>Technology</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyper-Threading</td>
<td>Reintroduced</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td>64KB Per Core</td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td>256KB Per Core</td>
<td></td>
</tr>
<tr>
<td>L3</td>
<td>4MB to 24MB Shared</td>
<td></td>
</tr>
<tr>
<td>Second Level</td>
<td>Branch Predictor</td>
<td></td>
</tr>
<tr>
<td>Level</td>
<td>Translation Lookaside Buffer (TLB)</td>
<td></td>
</tr>
<tr>
<td>Interconnect</td>
<td>Quickpath</td>
<td></td>
</tr>
<tr>
<td>ETC</td>
<td>Integration of PCI Express and DMI Integrated Memory Controller</td>
<td>20 to 24 Pipeline Stages</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TLB Sizes</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTLB</td>
<td>1st</td>
<td>64</td>
</tr>
<tr>
<td>ITLB</td>
<td>1st</td>
<td>128</td>
</tr>
<tr>
<td>STLB</td>
<td>2nd</td>
<td>512</td>
</tr>
</tbody>
</table>

A.2 Interval Simulation

Interval simulation is a recently proposed simulation approach for simulating multi-core and multiprocessor systems at a higher level of abstraction compared to current practice of detailed cycle-accurate simulation. This technique leverages a mechanistic analytical model to abstract core performance by driving the timing simulation of an individual core without the detailed tracking of individual instructions through the core’s pipeline stages. As shown in Fig. A.3, the foundation of the model is that miss events (like branch mispredictions, cache and TLB misses, serialization instructions, etc.) divide the smooth streaming of instructions through the pipeline into intervals. Branch predictor, memory hierarchy, cache coherence and interconnection network simulators determine the miss events; the analytical model derives
the timing for each interval. The cooperation between the mechanistic analytical model and the miss event simulators enables the modeling of the tight performance entanglement between co-executing threads on multi-core processors.

**A.3 Multi-Core Interval Simulator**

Fig. A.4 shows the multi-core interval simulator which models the timing for the individual cores. The simulator maintains a window of instructions for each simulated core. This window of instructions corresponds to the reorder buffer of a superscalar out-of-order processor, and is used to determine miss events that are overlapped by long-latency load misses. The functional simulator feeds instructions into this window at the window tail. Core-level progress (i.e., timing simulation) is derived by considering the instruction at the window head. In case of an I-cache miss, the core simulated time is increased by the miss latency. In case of a branch misprediction, the branch resolution time plus the front-end pipeline depth is added to the core simulated time, i.e., this is to model the penalty for executing the
chain of dependent instructions leading to the mispredicted branch plus the number of cycles needed to refill the front-end pipeline. In case of a long-latency load (i.e., a last-level cache miss or cache coherence miss), we add the miss latency to the core simulated time, and we scan the window for independent miss events (cache misses and branch mispredictions) that are overlapped by the long-latency load-second-order effects. For a serializing instruction, we add the window drain time to the simulated core time. If none of the above cases applies, we dispatch instructions at the effective dispatch rate, which takes into account inter-instruction dependencies as well as their execution latencies.
A.4 Instruction-Window Centric Core Model

Large core counts and complex cache hierarchies are increasing the burden placed on commonly used simulation and modeling techniques. Although analytical models provide fast results, they do not apply to complex, many-core shared-memory systems. In contrast, detailed cycle-level simulation can be accurate but also tends to be slow, which limits the number of configurations that can be evaluated. A middle ground is needed that provides for fast simulation of complex many-core processors while still providing accurate results.

In the group of snipersim.org, they explore, analyze, and compare the accuracy and simulation speed of high-abstraction core models as a potential solution to slow cycle-level simulation. We describe a number of enhancements to interval simulation to improve its accuracy while maintaining simulation speed. In addition, we introduce the instruction-window centric (IW-centric) core model, a new mechanistic core model that bridges the gap between interval simulation and cycle-accurate simulation by enabling high-speed simulations with higher levels of detail. We also show that using accurate core models like these are important for memory subsystem studies, and that simple, naive models, like a one-IPC core model, can lead to misleading and incorrect results and conclusions in practical design studies. Validation against real hardware shows good accuracy, with an average single-core error of 11.1% and a maximum of 18.8% for the IW-centric model with a 1.5X slowdown compared to interval simulation.
Appendix B

Parsec and Splash-2: Benchmark Suite

The Parsec and Splash-2 benchmark suites have been adopted by Sniper simulator within a short amount of time. In addition, these benchmark suites are major input resources for our experimental evaluation in Part III. The detail of these workloads is not yet fully understood by researchers. Comparing the Splash-2 and Parsec benchmark suites with each other, we are able to gain insights into differences and similarities between these two benchmark suites. We have surveyed some related documents and have looked into Sniper to analyze the suites for characteristics. Our analysis shows that Parsec workloads are fundamentally different from Splash-2 benchmarks. The observed differences can be explained here with a measurable impact on workload behavior. We found that no single reason for the differences could be identified. Furthermore, Parsec is the more diverse suite in direct comparison.
Table B.1: Parsec Workloads and the Simlarge Input Set

<table>
<thead>
<tr>
<th>Program</th>
<th>Application Domain</th>
<th>Problem Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>blackscholes</td>
<td>Finalcial Analysis</td>
</tr>
<tr>
<td>2</td>
<td>bodytrack</td>
<td>Computer Vision</td>
</tr>
<tr>
<td>3</td>
<td>canneal</td>
<td>Engineering</td>
</tr>
<tr>
<td>4</td>
<td>dedup</td>
<td>Enterprise Storage</td>
</tr>
<tr>
<td>5</td>
<td>facesim</td>
<td>Animation</td>
</tr>
<tr>
<td>6</td>
<td>ferret</td>
<td>Similarity Search</td>
</tr>
<tr>
<td>7</td>
<td>fluidanimate</td>
<td>Animation</td>
</tr>
<tr>
<td>8</td>
<td>freqmine</td>
<td>Data Mining</td>
</tr>
<tr>
<td>9</td>
<td>streamcluster</td>
<td>Data Mining</td>
</tr>
<tr>
<td>10</td>
<td>swaptions</td>
<td>Financial Analysis</td>
</tr>
<tr>
<td>11</td>
<td>vips</td>
<td>Media Processing</td>
</tr>
<tr>
<td>12</td>
<td>x264</td>
<td>Media Processing</td>
</tr>
</tbody>
</table>

B.1 Overview of Workloads and the Used Inputs

Parsec is a new benchmark suite that was released since 2008. It was rapidly adopted by researchers around the world with a download over 500 times. Also, The Splash-2 suite is one of the most widely used collections of multithreaded workloads. It is composed of eleven workloads, three of which come in two implementations that feature different optimizations. As shown in Table B.1 and Table B.2, we provide an overview of Parsec and Splash-2. The workload composition of the Parsec suite differs significantly from Splash-2. Since the release of Splash-2 parallel computing has reached the mainstream. The wide availability of CMPs has turned multiprocessor machines from an expensive niche product into a commodity that is used for problems from an increasingly wide range of application domains. This fact has influenced the Parsec program selection. The suite includes benchmarks from many different areas such as enterprise servers, data mining and animation.
Table B.2: Splash-2 Workloads and the Used Inputs

<table>
<thead>
<tr>
<th>Program</th>
<th>Application Domain</th>
<th>Problem Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>barnes</td>
<td>High-Performance Computing</td>
<td>65,536 particles</td>
</tr>
<tr>
<td>cholesky</td>
<td>High-Performance Computing</td>
<td>tk29.O</td>
</tr>
<tr>
<td>fft</td>
<td>Signal Processing</td>
<td>4,194,304 data points</td>
</tr>
<tr>
<td>fmm</td>
<td>High-Performance Computing</td>
<td>65,536 particles</td>
</tr>
<tr>
<td>lu</td>
<td>High-Performance Computing</td>
<td>1024 x 1024 matrix, 64 x 64 blocks</td>
</tr>
<tr>
<td>ocean</td>
<td>High-Performance Computing</td>
<td>514 x 514 grid</td>
</tr>
<tr>
<td>radiusy</td>
<td>Graphics</td>
<td>large room</td>
</tr>
<tr>
<td>radix</td>
<td>General</td>
<td>8,388,608 integers</td>
</tr>
<tr>
<td>raytrace</td>
<td>Graphics</td>
<td>car</td>
</tr>
<tr>
<td>volrend</td>
<td>Graphics</td>
<td>head</td>
</tr>
<tr>
<td>water</td>
<td>High-Performance Computing</td>
<td>4096 molecules</td>
</tr>
</tbody>
</table>

B.2 Program Characteristics

To capture the fundamental program properties we included a set of four instruction characteristics that were normalized to the total number of instructions: The number of floating point operations, ALU instructions, branches and memory accesses. Threads running on the Sniper use shared caches to communicate and share data with each other. Another ve characteristics were thus chosen that rect properties related to data usage and communication such as the total working set size or how intensely the program works with the shared data. These characteristics are the data cache miss rate, what percentage of all cache lines is shared for reading and what percentage for writing, the ratio of memory references that reads from shared cache lines and the ratio that writes to them. Other characteristics rect properties which are related to data usage and communication such as the total working set size or how intensely the program works with the shared data. These characteristics are the data cache miss rate, what percentage of all cache lines is shared for reading and what percentage for writing, the ratio of memory references that reads from shared cache lines and the ratio that writes to them. A thorough study would require the analysis of the workloads.
on a wide range of designs, since the heterogeneous system architectures have not matured yet.
Appendix C

McPAT: Power Analysis Framework for Multi-Core Architectures

Sniper integrates the Multicore Power, Area, and Timing (McPAT) framework for power and area specific modeling for of manycore architectures. The McPAT that is installed with Sniper is based on McPAT which is available from Hewlett-Packard Labs, for increased performance and functionality. McPAT is a new power, area, and timing modeling framework that enables architects to estimate new metrics combining performance with both power and area, which are useful to quantify the cost of new architectural ideas. At the microarchitectural level, McPAT includes models for the fundamental components of a chip multiprocessor, including in-order and out-of-order processor cores, networks-on-chip, shared caches, integrated memory controllers, and multiple-domain clocking. At the circuit and technology levels, McPAT supports critical-path timing modeling, area modeling, and dynamic, short-circuit, and leakage power modeling for each of the device types forecast in the ITRS roadmap including bulk CMOS, SOI, and double-gate transistors. McPAT has a flexible XML interface to facilitate its use with many performance simulators. Combined with a performance simulator, McPAT enables architects to consistently quantify the cost of new ideas.
and assess tradeoffs of different architectures using new metrics like energy-delay-area\(^2\) product (EDA\(^2\)P) and energy-delay-area product (EDAP). This paper explores the interconnect options of future manycore processors by varying the degree of clustering over generations of process technologies. Clustering will bring interesting tradeoffs between area and performance because the interconnects needed to group cores into clusters incur area overhead, but many applications can make good use of them due to synergies of cache sharing. Combining power, area, and timing results of McPAT with performance simulation of Parsec benchmarks at the 22nm technology node for both common in-order and out-of-order manycore designs shows that when die cost is not taken into account clustering 8 cores together gives the best energy-delay product, whereas when cost is taken into account configuring clusters with 4 cores gives the best EDA\(^2\)P and EDAP.

C.1 Operation

Rather than being hardwired to Sniper simulator, McPAT uses an XML-based interface with the performance simulator. This interface allows both the specification of the static microarchitecture configuration parameters and the passing of dynamic activity statistics generated by the performance simulator. McPAT can also send runtime power dissipation back to the performance simulator through the XML-based interface, so that the performance simulator can react to power (or even temperature) data. This approach makes McPAT very flexible and easily ported to other performance simulators. McPAT runs separately from a simulator and only reads performance statistics from it. Performance simulator overhead is minor - only the possible addition of some performance counters. Since McPAT provides complete hierarchical models from the architecture to the technology level, the XML interface also contains circuit implementation style and technology parameters that are specific to a particular target processor. Examples are array types, crossbar types, and the CMOS
technology generation with associated voltage and device types.

C.2 Type of Representative Architecture-Level Power
Estimator

Cacti [86] was the longest tool to explain the need for fast power, area, and timing estimates for computer architecture research, focusing on RAM-based structures. Cacti uses the approach of logical effort to size transistors. It contains optimization features that enable the tool to find a configuration with minimal power consumption, given constraints on area and timing. Using generic circuit models for pipeline stages, it estimates the RC delay for each stage and determines the critical path.

Wattch [85] is a largely-used processor power estimation tool within the portable and familiar SimpleScalar framework. Wattch calculates dynamic power dissipation from switching events obtained from an architectural simulation and capacitance models of components of the microarchitecture. Wattch has enabled the computer architecture research community to explore power-efficient design options, as technology has progressed; however, limitations of Wattch have become apparent. First, Wattch models power without considering timing and area. Second, Wattch only models dynamic power consumption; the Hot Leakage package partially addressed this deficiency by adding models for subthreshold leakage. Third, Wattch uses simple linear scaling models based on 80nm technology that are inaccurate to make predictions for current and future deep-submicron technology nodes.

Orion [87] is a tool for modeling power in networks-on-chip (NoC). Orion includes models for area, dynamic power, and gate leakage, but does not consider short-circuit power or timing.