DESIGNING COOL CHIPS: LOW POWER AND THERMAL-AWARE DESIGN METHODOLOGIES

A Thesis in
Computer Science and Engineering
by
Wei-Lun Hung

© 2006 Wei-Lun Hung

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2006
The thesis of Wei-Lun Hung was reviewed and approved* by the following:

Yuan Xie  
Assistant Professor of Computer Science and Engineering  
Thesis Adviser  
Chair of Committee

Mary Jane Irwin  
Professor of Computer Science and Engineering  
Evan pugh Professor, A. Robert Noll Chair of Engineering

Vijaykrishnan Narayanan  
Associate Professor of Computer Science and Engineering  
Director of Graduate Affairs

Zhiwen Liu  
Assistant Professor of Electrical Engineering

Raj Acharya  
Professor of Computer Science and Engineering  
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.
Abstract

Power is one of the rigid challenges for high performance computer system designs and for the widespread use of portable and wireless electronic systems. With technology continuing to scale down, the power issue is even more prominent than before, with the highly integration capability on a single chip. Power dissipation affects battery life and performance, and greatly affects reliability and cooling costs. Thus, reducing power consumption has become more and more important in the nanometer era.

With the power issues induced from technology scaling, temperature in modern high performance VLSI circuits has moved up dramatically due to smaller feature sizes, higher packing densities, and rising power consumptions. Temperature affects not only the reliability but also the performance, power, and cost of the chip designs. Power-aware design alone is not able to address the temperature challenge. Thus, both low power and thermal-aware techniques need to be adopted jointly to combat the ever-increasing power and thermal related problems.

One power optimization framework based on the genetic algorithm has been proposed. The optimization strategy can simultaneously perform multiple-$V_{dd}$ assignment, multiple-$V_{th}$ assignment, and gate sizing in conjunction with stacking force techniques to minimize total power consumption, while maintaining performance requirements. The effectiveness of the proposed total power optimization framework was validated by conducting various experiments.

A thermal-aware floor planner is advocated to reduce the hot spot temperatures for two-dimensional chips and is further enhanced to tackle the thermal issues of three-dimensional integrated circuits. The uniqueness of the proposed floor planner is that it accounts for the effects of the interconnect power consumption in estimating the peak temperature, while still maintaining good traditional floor planning design metrics.

The thermal-aware floor planner is then applied to System-on-Chip designs to
combat the voltage island partitioning and floorplanning problems. A thermal-aware task allocation and scheduling algorithm is also proposed for embedded systems. The algorithm is used as a sub-routine for hardware/software co-synthesis to reduce the peak temperatures and to achieve a thermally balanced system.

Finally, a thermal-aware IP placement algorithm is proposed for Networks-on-Chip architecture. Since the temperature distribution profile of the chip depends on the IP core virtualization and placement, the algorithm is able to alleviate the thermal issues by taking this factor into consideration. The proposed framework can not only reduce the hot spot temperature, but can also minimize the communication cost through placement, which results better performance.
# Table of Contents

List of Figures viii

List of Tables xi

Chapter 1
Introduction 1

1.1 Motivation .......................................................... 1
1.2 Contributions of the Thesis .................................... 3
1.3 The Organization of the Thesis ................................. 5

Chapter 2
Related Work 6

2.1 Low Power Techniques ............................................ 6
2.2 Temperature Estimation ......................................... 8
2.3 Physical-level Thermal-aware Techniques .................. 11
2.4 System-level Thermal-aware Techniques .................... 12

Chapter 3
Total Power Optimization through Simultaneously Multiple $V_{dd}$, Multiple $V_{th}$ Assignment and Device Sizing with Stack Forcing 14

3.1 Delay Model ......................................................... 17
3.2 Genetic Algorithm ................................................ 19
3.3 Power Optimization Framework ............................... 21

3.3.1 Chromosome encoding ...................................... 21
3.3.2 Fitness function ................................................ 23

3.3.2.1 Power ..................................................... 23
3.3.2.2 Delay .................................................... 23
3.3.2.3 Penalty function .................................. 24
3.4 Experimental Evaluation ................................. 24
3.4.1 Control parameters .................................... 25
3.4.2 Generation and performance ........................... 26
3.4.3 Applying four low power techniques ................... 28
3.4.4 Comparison with other existing combined power techniques . 33
3.4.5 Exploiting parallelism in genetic algorithms ........... 34

Chapter 4
2D Thermal-Aware Floorplanning ......................... 37
4.1 2D Thermal-Aware Floorplanning Framework ............ 37
4.1.1 Chromosome encoding .................................. 38
4.1.2 Fitness function ......................................... 39
  4.1.2.1 Area minimization design ......................... 40
  4.1.2.2 Temperature balanced design ...................... 40
4.1.3 Control parameters .................................... 41
4.2 Experimental Results ................................... 41
  4.2.1 Experiment on MCNC benchmarks ................. 41
  4.2.2 Experiment on a face detection design ........... 45

Chapter 5
3D Interconnect and Thermal-Aware Floorplanning ......... 48
5.1 Experimental Methodology .................................. 50
  5.1.1 Processor model ...................................... 50
  5.1.2 B*-tree floorplan model ............................. 51
  5.1.3 Simulated annealing engine ......................... 52
  5.1.4 Interconnect power distribution .................... 54
  5.1.5 Temperature approximation ......................... 55
5.2 Experimental Results ................................... 56

Chapter 6
Thermal-aware Task Allocation and Scheduling for Embedded Systems ............................................. 61
6.1 Problem Formulation ...................................... 62
6.2 Task Allocation and Scheduling ............................ 64
  6.2.1 Power-aware allocation and scheduling ............ 66
  6.2.2 Thermal-aware allocation and scheduling .......... 68
6.3 Experimental Results ................................... 69
Chapter 7
Thermal-Aware Voltage Islands Architecting in System-on-Chip Design
7.1 Problem Formulation ........................................... 75
7.2 Voltage Island Partitioning and Floorplanning Algorithm ........ 75
7.3 Slicing Tree Floorplan Model .................................. 77
  7.3.1 Thermal-aware floorplanning ............................... 79
7.4 Experimental Results ........................................... 80
  7.4.1 Experiment on chip-level ................................. 81
  7.4.2 Experiment on two-level thermal-aware optimization .... 81

Chapter 8
Thermal-Aware Virtualization and Placement for 2D/3D Networks-on-Chip Architecture
8.1 IP Virtualization and Placement Framework ........................ 87
  8.1.1 Chromosome encoding ................................... 87
  8.1.2 Fitness function ........................................ 88
    8.1.2.1 Thermal balanced design .......................... 88
    8.1.2.2 Power balanced design .............................. 89
    8.1.2.3 Communication cost minimization design .... 89
  8.1.3 Crossover operator and mutation operator ................... 89
8.2 Case Study on LDPC and Experimental Results .................... 90
  8.2.1 A brief introduction on LDPC ............................ 90
  8.2.2 Design methodology .................................. 92
  8.2.3 Experimental results .................................. 94
    8.2.3.1 Thermal-aware IP placement ...................... 95
    8.2.3.2 Thermal-aware IP virtualization and placement .. 97

Chapter 9
Conclusions and Future Directions .............................. 100
9.1 Future Directions ........................................... 102

Bibliography ....................................................... 104
List of Figures

2.1 Stacking force example for an inverter. .......................... 7

3.1 An example circuit with a critical path of darkened gates. ..... 16
3.2 An example circuit with applying four low power techniques. ... 16
3.3 The example of two inverters. ..................................... 18
3.4 The typical optimization flow of a genetic algorithm. .......... 21
3.5 The structure of a chromosome. .................................. 22
3.6 The implementation of a level converter. ......................... 26
3.7 Solution convergences under different timing relaxations. .... 27
3.8 Power reduction as timing constraint relaxed. .................. 28
3.9 Power breakdown for speed-optimized circuits. .................. 29
3.10 Power breakdown for power-optimized circuits. ................ 29
3.11 The profile of used low power techniques. ...................... 30
3.12 Comparison of power consumption with/without stacking force. 30
3.13 Comparison of circuits’ power consumptions with/without stacking force.................................................. 31
3.14 Comparison of different power techniques. ..................... 32
3.15 The distribution of level converters for testbench2 circuit. .... 33
3.16 Power comparison of $V_{dd}+V_{th}$, $V_{dd}+\text{size}$, $V_{dd}+V_{th}+\text{size}$, and all four low power techniques. .............. 34
3.17 Comparison of leakage power from different combined low power techniques................................. 34
3.18 Run time comparison of single processor and dual processors. 35

4.1 The flow of the thermal-aware floorplanning. .................... 38
4.2 The slicing tree representation and its corresponding floorplan. 39
4.3 The encoding of chromosome for floorplanning. ................. 39
4.4 The floorplan of circuit $hp$ with area-optimized only. ......... 43
4.5 The floorplan of circuit $hp$ with thermal-aware optimization. .. 43
4.6 The comparison of temperature and dead space ratio for ami49 under thermal-aware optimization. ..................... 44
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7</td>
<td>The comparison of average temperatures for area optimization and</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>temperature-aware optimization.</td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>The comparison of maximal temperatures (hot spot) for area</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>optimization and temperature-aware optimization.</td>
<td></td>
</tr>
<tr>
<td>4.9</td>
<td>The overall design methodology for face detection application’s</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>power estimation.</td>
<td></td>
</tr>
<tr>
<td>4.10</td>
<td>The comparison of area average and maximal temperatures under</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>thermal-aware optimization.</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Wire length in 2D and 3D architecture configurations.</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Processor model diagram.</td>
<td>51</td>
</tr>
<tr>
<td>5.3</td>
<td>(a) one floorplan example (b) the corresponding B*-tree.</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Thermal profiles of Alpha processor for 2D and 3D architectures.</td>
<td>58</td>
</tr>
<tr>
<td>5.5</td>
<td>Interconnect distribution of Alpha processor under 2D architecture.</td>
<td>59</td>
</tr>
<tr>
<td>5.6</td>
<td>Interconnect distribution of Alpha processor under 3D architecture.</td>
<td>59</td>
</tr>
<tr>
<td>5.7</td>
<td>Temperature profiles of Alpha processor for 160 nm and 90 nm</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>technologies.</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>The flow of the traditional co-synthesis framework.</td>
<td>62</td>
</tr>
<tr>
<td>6.2</td>
<td>(a) An example of a task graph. (b) An example of a general target</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>architecture.</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Outline of the allocation and scheduling procedure (ASP).</td>
<td>65</td>
</tr>
<tr>
<td>6.4</td>
<td>Examples of three different power-aware heuristics.</td>
<td>67</td>
</tr>
<tr>
<td>6.5</td>
<td>The flow of the thermal-aware co-synthesis framework and the</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>thermal-aware platform-based system design.</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>An example of two voltage islands.</td>
<td>74</td>
</tr>
<tr>
<td>7.2</td>
<td>Outline of the voltage island partitioning and floorplanning</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>algorithm.</td>
<td></td>
</tr>
<tr>
<td>7.3</td>
<td>Example of single point crossover in voltage island partitioning.</td>
<td>78</td>
</tr>
<tr>
<td>7.4</td>
<td>Slicing tree representation and its corresponding floorplan.</td>
<td>79</td>
</tr>
<tr>
<td>7.5</td>
<td>Partitioning and floorplanning result of ami49 under TAW optimization.</td>
<td>81</td>
</tr>
<tr>
<td>7.6</td>
<td>On-chip temperature distribution for ami49 with TAW optimization</td>
<td>83</td>
</tr>
<tr>
<td>7.7</td>
<td>On-chip temperature distribution for ami49 with two-level</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>optimization.</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Left - an NoC layout showing the physical placement of the PEs and</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>router. Right - an individual NoC router shown in detail with the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>connections to the rest of the routers.</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>Chromosome encoding for IP placement</td>
<td>88</td>
</tr>
<tr>
<td>8.3</td>
<td>Derivation of the Bipartite Graph from the H-Matrix.</td>
<td>91</td>
</tr>
</tbody>
</table>
8.4 The overall design methodology for NoC LDPC power estimation. 93
8.5 Peak temperature comparison of 1x1 and 2x2 window schemes. 94
8.6 Average temperature comparison of 1x1 and 2x2 window schemes. 95
8.7 Average temperature comparison for a 4x4 LDPC decoder with 5 sets of hardware virtualization under different optimization strategies 96
8.8 Peak temperature comparison for a 4x4 LDPC decoder with 5 sets of hardware virtualization under different optimization strategies 96
8.9 Communication cost comparison for a 4x4 LDPC decoder under different optimization strategies 97
8.10 The experimental result for virtualized mapping of 960 virtual nodes of a LDPC decoder 98
8.11 The comparisons of non-virtualized and virtualized mapping under temperature-based approach 98
8.12 The comparisons of communication cost for virtualized mapping and non-virtualized mapping 99
List of Tables

4.1 Comparisons of area requirements among O-tree, iterative and simulated annealing B*-tree, cluster refinement. (Data taken from [64] for non-slicing approaches.) ................................................. 42

4.2 Power consumption and neuron dimension profile of the face detection application design. ......................................................... 46

5.1 Floorplanning results of 2D architecture. ................................. 56

5.2 Floorplanning results of 3D architecture. ................................. 56

5.3 Interconnect power profile. ...................................................... 59

6.1 A technology library example. ................................................. 63

6.2 The comparisons of different power heuristics under co-synthesis architecture and platform-based target architecture. .................. 71

6.3 The temperature comparisons of the power-aware and the thermal-aware approaches on co-synthesis architecture. ..................... 72

6.4 The temperature comparisons of the power-aware and the thermal-aware approaches on platform-based architecture. ...................... 72

7.1 The comparison of temperatures under different design metrics ... 82

7.2 Results of the two-level thermal-aware approach .......................... 83

8.1 LDPC nodes configuration profile ............................................. 95
Chapter 1

Introduction

1.1 Motivation

Power consumption is a top priority in high performance circuit design today. Many low power techniques have been proposed to tackle the ever serious, highly pressing power consumption problem, which is composed of both dynamic and static power in the nanometer era. Currently, the static power consumption receives even more attention than that of dynamic power consumption, when technology scales below 100 nm. In order to mitigate the aggressive power consumption, various existing low power techniques are often used; however, they are often applied independently or combined with two, or at most three, different techniques together. This is not sufficient to address the escalating power issue. In this thesis, we present a power optimization framework for the minimization of total power consumption through multiple-$V_{dd}$ assignment, multiple-$V_{th}$ assignment, device sizing, and stack forcing, while maintaining performance requirements. These four power reduction techniques are properly encoded into the genetic algorithm and evaluated simultaneously. The overhead imposed by the insertion of level converters is also taken into account. The effectiveness of each power reduction mechanism is verified, as are the combinations of different approaches. Experimental results are presented for a number of 65 nm benchmark circuits that span typical circuit topologies, including inverter chains, SRAM decoders, multiplier, and a 32bit carry
adder. Our experiments show that the combination of four low power techniques is the effective way to achieve a low power budget. The framework is general and can be easily extended to include other design-time low power techniques, such as multiple gate length or multiple gate oxide thickness.

Process scaling and aggressive performance improvements have resulted in a dramatic power consumption increase. Power density directly translates into heat; as a result, the temperature in modern high-performance VLSI circuits increases dramatically due to smaller feature size, higher packing density, and rising power consumption. The hot spot in a modern chip might have a temperature of more than 100°C, while the intra-chip temperature differentials can be larger than 10-20°C [37]. Temperature can have a dramatic impact on circuit performance, power, and reliability: MOS current drive capability decreases approximately 4% for every 10°C temperature increase, and interconnect (Elmore) delay increases approximately 5% for every 10°C increase [38], while the leakage current increases exponentially with the temperature increase. Many chip failure mechanisms (e.g., electromigration and stress migration) are significantly accelerated [38], which leads to an overall decrease in reliability. As a recent report from AMD [20] states, it has discovered a potential heat problem with a small percentage of Opteron chips run under extreme conditions. To address these design challenges, temperature analysis must be included in the design process. The circuit design process must include a better methodology to understand the heat flow from the die through the package and the heat flow within the die.

As projected by the International Technology Roadmap for Semiconductors (ITRS) [39], IC designs will hit the wall of power consumption and heat dissipation below 65 nm. It is critical to address the energy and thermal issues during on-chip system design to meet the urgent need of the semiconductor industry and to enable future technology scaling. Therefore, it is very important to reduce or eliminate hot spots and have a thermal balanced design to go along with technology scaling.

Today’s circuit designs demand higher packaging density and faster performance. The resulting increase in power densities is prevalent in every chip design; thus, temperature gradients across circuits can lead to higher failure rates. For the first time, local chip temperatures are driving overall design methodologies and power management schemes. The magnitude and complexity of this temperature-
power relationship has created within the design community a sense of urgency to address temperature issues. Temperature values within a chip are a function of the materials, their dimensions, package characteristics, and ambient conditions. The high concentration of power in today’s designs results in the self-heating of devices and interconnects, leading to larger temperature variations within the chip. Consequently, there is a critical need for research in this direction.

1.2 Contributions of the Thesis

In this thesis, we first present a power optimization framework in Chapter 3 for the minimization of total power consumption through multiple $V_{dd}$ assignment, multiple $V_{th}$ assignment, device sizing, and stack forcing, while maintaining performance requirements. These four power reduction techniques are properly encoded into the genetic algorithm and evaluated simultaneously. The overhead imposed by the insertion of level converters is also taken into account. The effectiveness of each power reduction mechanism is verified, as are the combinations of different approaches. Experimental results are presented for a number of 65 nm benchmark circuits.

Throughout this thesis, we will demonstrate that the use of thermal analysis at different points of the design process can be beneficial. Thermal analysis in the early design stages is effective in removing hot spots and in equalizing the temperature distribution so that subsequent design implementation steps progressively occur with less steep thermal gradients. Once thermal analysis has been applied to the design flow, repair techniques can be made to improve further the refinement of thermal integrity.

In this thesis, we have applied thermal-aware techniques to different stages of the design flow to tackle the ever-increasing thermal problems. The 2D thermal-aware floorplanning is explored first in Chapter 4. As the 3D integration is widely researched, interconnects are becoming an increasing problem from both performance and power consumption perspectives in future technology nodes. The introduction of 3D chip architectures, with their intrinsic capability of reducing wire length, is one of the promising solutions to mitigate the interconnect problem. While interconnect power consumption is reduced due to the adoption of 3D de-
signs, the stacking of multiple active layers leads to higher power densities. Thus, high peak temperatures are a major concern in 3D designs. Consequently, we present a thermal-aware floorplanner for 3D architectures in Chapter 5.

Temperature affects not only the reliability but also the performance, power, and cost of the embedded system. Thus, we propose a thermal-aware task allocation and scheduling algorithm for embedded systems in Chapter 6. The algorithm is used as a sub-routine for hardware/software co-synthesis to reduce the peak temperature and to achieve a thermally even distribution, while meeting real time constraints. Both power-aware and, thermal-aware allocation and scheduling are explored in this chapter.

The rising on-chip temperatures can also have negative impacts on system-on-chip (SoC) performance, power, and reliability. Thus, a hybrid optimization approach, which aims at temperature reduction and hot spot elimination, is given in Chapter 7. We demonstrate that considerable improvement in the thermal distribution of a design can be achieved through careful voltage island partitioning, voltage level assignment, and voltage island floorplanning.

In Chapter 8, a thermal-aware IP virtualization and placement approach is proposed to tackle the thermal issues in Networks-on-Chip Architecture (NoC). NoC, a new System-on-Chip (SoC) paradigm, has been proposed as a solution to mitigate complex on-chip interconnect problems. NoC architecture consists of a collection of IP cores or processing elements (PEs) interconnected by on-chip switching fabrics or routers. Hardware virtualization, which maps logic processing units onto PEs, affects the power consumption of each PE as well as the communications among PEs. The communication among PEs affects the overall performance and router power consumption, which depends on the placement of PEs. Therefore, the temperature distribution profile of the chip depends on the IP core virtualization and placement. In this chapter, we present an IP virtualization and placement algorithm for generic regular NoC architectures. The algorithm attempts to achieve a thermal-balanced design, while minimizing the communication cost via placement. Our framework can also realize hardware virtualization, which can further accomplish better performance. A case study on Low Density Parity Checks (LDPC) decoder is presented to evaluate our approach.
1.3 The Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 3 presents a low power technique for the minimization of total power consumption via multiple $V_{dd}$ assignment, multiple $V_{th}$ assignment, device sizing and stack forcing, while maintaining performance requirements. Chapter 4 gives a genetic algorithm based thermal-aware floorplanning framework that aims at reducing hot spots and distributing temperature evenly across a chip, while optimizing the traditional design metrics. In Chapter 5, we present a thermal-aware floorplanner for 3D architectures; in addition, the proposed floorplanner also consider the interconnect power consumption. Chapter 6 gives a thermal-aware task allocation and scheduling algorithm for embedded systems, while in Chapter 7, a hybrid optimization approach, which aims at temperature reduction and hot spot elimination, is proposed for system-on-chip designs. In Chapter 8, an IP virtualization and placement algorithm is given for generic regular Network-on-Chip architecture. Thermal effects are also taken into consideration with the proposed algorithm. Finally, the key contributions of this thesis is summarized and possible future research directions are presented.
Related Work

2.1 Low Power Techniques

Due to the quadratic relationship between dynamic power consumption and $V_{dd}$, reducing the supply voltage is the most effective way to lower the dynamic power, at the expense of increasing gate delay. In order to prevent the negative effect on performance, the threshold voltage ($V_{th}$) must be reduced proportionally with the supply voltage, so that a sufficient driving current is maintained. This reduction in the threshold voltage causes an exponential increase in leakage power, which in turn can raise the static power of the device to unacceptable levels.

To counter the loss in performance while improving the power efficiency, multiple-$V_{dd}$ \cite{1} and multiple-$V_{th}$ \cite{3} techniques have been proposed. The gates on critical paths operate at the higher $V_{dd}$ or lower $V_{th}$, while those on non-critical paths operate at the lower $V_{dd}$ or higher $V_{th}$, thereby reducing overall power consumption without performance degradation. These techniques have been successfully implemented. For example, IBM’s ASIC design flow can take advantage of the power-performance tradeoff fully by using their voltage island concept and multiple-$V_{th}$ standard cell library \cite{4}. Gate sizing \cite{5} is another powerful method for power optimization. Logic gates on critical paths may be sized up to meet timing requirements at the expense of higher power consumption, while those on non-critical paths can be sized down to reduce the power consumption. Hamada et al. \cite{6} examined multiple supply voltages, multiple threshold voltages, and transistor sizing individually. They have derived a set of rules of thumb for optimal supply
voltages, threshold voltages, and transistor sizing. A good summary of these three techniques is presented by Brodersen et al. [7].

Besides the multiple-$V_{th}$ technique, another solution, stack forcing, is being used to tackle the ever-increasing waste of leakage power. It has been shown that the stacking of two off transistors can significantly reduce leakage power compared with a single off transistor [8]. Therefore, we can force a non-stack device to a stack of $N$ devices without affecting the input load. Figure 2.1 shows a stacking force example of an inverter with $N=2$. By ensuring the input load unchanged, we guarantee that the previous stage’s delay and dynamic power consumption are not affected. The logic gate with stack forcing has much lower leakage power, however, at the expense of a delay penalty, because the effective device width $W_{eff}$ becomes $W/N^2$ after stack forcing. The idea of stacking force is thus similar to replacing a low-$V_{th}$ device with a high-$V_{th}$ device in a multiple-$V_{th}$ design.

To achieve the most power efficient design, all these power reduction techniques have to be complementarily used. Stojanovic et al. [9] combined gate sizing and supply voltage optimization to minimize power consumption under a delay constraint. Roy et al. [10] presented a heuristic algorithm to combine dual-$V_{dd}$ and dual-$V_{th}$ techniques. Both Srivastava et al. [28], and Augsburger and Nikolic [11] evaluated the effectiveness of multiple supply voltage, transistor sizing, and multiple thresholds independently and in conjunction with one another, showing that the order of application of these techniques determines the final savings in active and leakage power.

Recently, researchers have looked at the joint optimization of these techniques,
since such joint optimization can help to achieve maximum power savings compared to a sequential application of a single variable optimization. Sirichotiyakul et al. [13] presented an algorithm for joint optimization of dual-$V_{th}$ and sizing to reduce leakage power. Karnik et al. [14] developed a heuristic iterative algorithm to do device sizing and dual-$V_{th}$ allocation simultaneously to exploit the timing slack for reduction of total power consumption. They found that joint dual-$V_{th}$ and sizing can reduce the power by 10\% and 25\%, compared with pure $V_{th}$ allocation or pure sizing method, respectively. Srivastava et al. [12] were the first to investigate the effectiveness of simultaneously multiple supply and threshold voltage assignment for total power saving. Their algorithm is based on a linear programming approach. Nguyen et al. [15] developed another linear programming algorithm that can simultaneously perform the threshold voltage assignment and sizing optimization, and then apply the supply voltage optimization as a sequential step. Lee et al. [16] proposed heuristic algorithms for simultaneous state, $V_{th}$, and gate oxide assignment. Srivastava et al. [17] proposed a sensitivity-based algorithm to perform concurrent sizing, $V_{dd}$, and $V_{th}$ assignment. Recently, work [35] started investigating the integration of retiming and simultaneous supply/threshold voltage assignment. The proposed three steps approach showed promising results even when the Flip-Flop delay/power was considered.

### 2.2 Temperature Estimation

In order to consider thermal impact, a compact thermal model is needed to provide the temperature profile. Numerical computing methods (such as FEM and FDM [64]) are most accurate but computational intensive, while the simplified close-form formula [45] is the fastest but inaccurate. Skadron et al. proposed a thermal modeling tool called HotSpot [32], which is based on lumped thermal resistances and thermal capacitances. It is more efficient compared to previous approaches, since it provides temperature estimation of a microprocessor at the functional/IP module level by employing the principle of thermal-electrical duality.

The determination of resulting temperatures is similar in nature to the determination of the voltages in a resistance network, where heat sources are represented as current sources. An RC network of thermal capacitances and resistances of func-
tional modules are constructed and then temperatures at the center of functional modules are calculated by using circuit-solving techniques. The actual operations are performed in a matrix format, where the full thermal resistance matrix is multiplied by a vector containing the power dissipation of each functional module. This allows the temperature of a chip to be tracked on a per-module level.

The inputs to HotSpot are the floorplan and the power consumption number of individual modules, and the specifications of heat spreader and heat sink are also provided to define the heat-removing ability. The temperature of each functional module depends on the power consumption and the position of the functional modules. HotSpot provides a simple compact model, where the heat dissipation within each functional block and the heat flow among blocks are accounted for. The basic idea is that, if we define the transfer thermal resistance $R_{ij}$ of functional module $M_i$ with respect to $M_j$, as the temperature rises at $M_i$ due to one unit of power dissipated at $M_j$. Thus, $R_{ij}$ can be written as

$$R_{ij} = \Delta T_{ij}/\Delta P_j \quad (2.1)$$

Using a similar method, we can obtain a transfer thermal resistance matrix for the entire device. The determination of resulting temperatures is similar in nature to the determination of the voltages in a resistance network in which heat sources are represented as current sources. The actual operations are performed in a matrix format, where the full thermal resistance matrix is multiplied by a vector containing the power dissipation of each functional/IP module. This allows the temperature of a floorplan to be tracked on a per-module level. Below shows a transfer thermal resistance matrix:

$$R^t = \begin{bmatrix} R_{11}^t & R_{12}^t & \cdots & R_{1m}^t \\ R_{21}^t & R_{22}^t & \cdots & R_{2m}^t \\ \vdots & \vdots & \ddots & \vdots \\ R_{m1}^t & R_{m2}^t & \cdots & R_{mm}^t \end{bmatrix}$$

For any power distribution on the floorplan, we can calculate each block’s temperature by applying the following equation:
\[
\begin{bmatrix}
T_1 & R_{11}^t & R_{12}^t & \cdots & R_{1m}^t & P_1 \\
T_2 & R_{21}^t & R_{22}^t & \cdots & R_{2m}^t & P_2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
T_m & R_{m1}^t & R_{m2}^t & \cdots & R_{mn}^t & P_m \\
\end{bmatrix}
\]

where \( P_i \) is the power consumed by functional block \( M_i \) and \( T_i \) is the temperature of the functional module \( M_i \). The transfer thermal resistance matrix can be obtained from HotSpot, given the IP block placement.

While the HotSpot tool was originally intended to be a fast means of modeling temperatures of 2D architectures, as we move from 2D to 3D, HotSpot is no longer capable of generating an accurate temperature profile. As such, we have adopted a new tool called HS3D [62] that includes a variable number of additional levels, each composed of both a silicon layer and an inter-silicon "glue" material. This tool was validated for accuracy by first comparing it to the original HotSpot tool, which showed identical temperature estimates. To validate the multi-layer modeling, HS3D was compared to a commercial FEM tool, Flotherm, which showed an average temperature mis-estimation of 3°C and a maximum deviation of 5°C. This compares well to the validation of many other thermal estimation libraries. In addition, we note that the maximum discrepancies occurred "downstream" of the airflow modeled in Flotherm. HS3D uses a simple resistive model for heat transfer to ambient. Since the heatsink is not modeled, the heat transfer from airflow back to the heatsink may result in a significant portion of the discrepancy.

Unlike HotSpot, which represents each block of the chip as a single node in the thermal resistance matrix, HS3D has a means of automatically imposing a conservative grid structure, in which individual FUs are automatically divided into sub-nodes. This automatic division into sub-nodes occurs based on the edges of floorplan blocks in other levels in the stack, ensuring that accurate heat flow is modeled in the vertical layer. This automatic division also has the side-effect of increasing the horizontal accuracy of the tool due to the increased granularity of the model. In addition, this automatic block division can be called manually, allowing an arbitrary grid size to be superimposed on the chip, for arbitrary resolution.
2.3 Physical-level Thermal-aware Techniques

Area-optimized floorplanning techniques have been explored for a long time. Both slicing floorplanning [56, 58] and non-slicing floorplanning [56, 59] methods all performed well at area minimization. Genetic algorithms can also be used for floorplanning and result in good compact floorplan [60]. Nevertheless, area is not the only constraint considered for floorplanning in future complex chip designs. For example, Yong et al. [61] presented a unified method to handle multiple constraints (such as pre-place constraint, range constraint, and alignment constraint). However, there exists no floorplanning approaches, which considers the thermal impact incurred by extremely high area compaction. In contrast, thermal-aware placement methods for standard cell ASIC design have been investigated. For example, Chu and Wong used a matrix synthesis problem (MSP) to model the thermal placement problem and three algorithms were proposed to solve it [81]. Chen et al. proposed a partition-driven thermal placement model [71] for standard cells, making use of a multigrid-like approach to simplify the thermal problem at each level of finer granularity, facilitating the inclusion of temperature constraints on the placement. Hung et al. [65] proposed a thermal-aware IP placement algorithm for uniform-size Network-on-Chip IP cores. However, none of the above methods can be applied directly to the floorplanning problem because the objects in floorplanning have non-uniform dimension. Goplen and Sapatnekar explored the thermal placement for standard cells with force directed approach in [90] by formulating temperature as another force. Hung et al. [51] proposed the thermal-aware floorplanning by using genetic algorithms, but wire length factor is not included in their cost function evaluation. Tsai and Kang [82] proposed a compact FDM-based temperature model to derive temperature, based on which the standard cell placement and macro cell placement were tackled by a simulated annealing based thermal-driven placement algorithm.

Recent studies on microarchitecture focused on the performance issue. Full chip power modeling, including functional units and interconnects, is rarely considered. In [76], the performance-meeting requirement was involved during the microarchitecture evaluation. However, the power factor was not considered and an unsuitable set of benchmarks for representing actual microprocessor architecture
was used. Ekpanyapong et al. [77] proposed a profile-driven micro-architectural floorplanning in evaluating placement of functional units of the microprocessor with SimpleScalar incorporated. Unfortunately, the only concern of their work is to satisfy performance demand during the floorplanning process.

Interconnect and interconnect buffers are now first-order timing and power considerations in VLSI design [78]. This change has imposed challenges across all design levels. It is no longer possible to produce accurately the power consumption and performance of a design without prior knowledge about its floorplan to predict the structure of its interconnect. A number of researchers have considered the impacts of chip-level interconnect in power and performance aspects [74, 75, 79].

Some researchers have looked at the thermal problem in 2D microarchitectural floorplanning. For example, Han et al. in [92] used an Alpha floorplan; however, they use hypothetical interconnects as opposed to our interconnect information extracted from a real processor design. The tradeoff between performance and temperature is explored in [91]; nevertheless, 5% extra space is required to get a lowering temperature solution with their HotFloorplan tool. Both the above works neglected to investigate the interconnect power consumption and they did not consider floorplanning of 3D architectures. Although [93] targets on the same problem as our work, their approach of 3D partition followed by a mixed ILP floorplanning approach may generate sub-optimal solutions with a prohibitively high runtime overhead.

### 2.4 System-level Thermal-aware Techniques

There have been extensive studies in the literature on the task allocation and scheduling of hardware/software co-synthesis. Xie and Wolf [40] investigated the problem with the conditional branches constraints. The simulated annealing-based scheduling was used by Doboli [41] with an attempt to balance the latency and power consumption incurred by assigning tasks onto different PEs. In [42], Liu et al. targeted a real mission-critical application to demonstrate the effectiveness of maintaining the power budget while satisfying the performance requirements. Shang and Jha [43] tackled the low power HW/SW co-synthesis problem for an embedded system with dynamically reconfigurable FPGA processors and other
system resources.

Recently, several research studies investigated the task scheduling problem for DVS-enabled multiprocessor, real-time embedded systems. For example, Zhang et al. [44] formulated the task scheduling problem with voltage selection together as an integer linear programming (ILP) problem. Shin et al. [48] and Luo et al. [46] proposed the condition-aware DVS tasking scheduling algorithm, which can handle more complicated conditional task graphs. Schmitz et al. [47] used a two-step iterative synthesis approach for the same problem.

One of the major challenges for a successful adoption of the network-on-chip paradigm is to reduce the energy consumed during the interactions between the IPs or PEs [95]. Hu and Marculescu [98, 99] proposed an energy-aware mapping algorithm, which minimizes the total communication cost for a 2-D mesh NoC architecture under real-time performance constraints. Murali et al. [104] proposed an algorithm that maps IP cores onto a mesh NoC architecture under bandwidth constraints, minimizing the average communication delay. The potential bandwidth requirements were reduced by the partitioning of inter-core traffic across multiple paths. Al-Rawi et al. [94] also explored an optimal mapping on a set of LDPC nodes to physical computation units, with the purpose of minimizing the communication between the nodes.
Chapter 3

Total Power Optimization through Simultaneously Multiple $V_{dd}$, Multiple $V_{th}$ Assignment and Device Sizing with Stack Forcing

Process scaling and aggressive performance improvements have resulted in power consumption becoming a first-order design criterion. For example, the current generation Intel Pentium 4 processor (Prescott, 2004) has a power consumption of 103 Watts, almost four times larger than that of the Pentium III (1999). The concern of power for portable devices and battery-powered applications is even more evident, since these applications are normally staying longer in stand-by mode instead of in active mode. Thus, a big portion of power, stand-by power (leakage), is wasted, and thus largely affects the available operating time. In addition to its clear impact on battery lifetime in portable embedded systems, processor power consumption has also become a primary constraint on workstation performance due to cooling and heat dissipation issues. This situation is even worse in high performance computing, where a number of processors are arranged to run in parallel to gain high computing capability; however, such arrangement also has huge adverse effects on power. Therefore, reducing power dissipation is a top priority in modern VLSI design.
Power dissipation in CMOS digital circuits consists of dynamic power, short circuit power, and static power. Short circuit power consumption can be kept within bounds by careful design and tuning the switching characteristics of complementary logic (slope engineering). When the rise and fall times of inputs and outputs are equalized, most power dissipation is associated with the dynamic power, and only a minor fraction (< 10%) is devoted to short circuit currents [5]. Thus, the short circuit power is becoming less important in deep submicron technologies, compared to dynamic power and leakage power. Therefore, we will focus on the latter two sources of power consumption, as indicated by equation 3.1.

\[ P = A \cdot V_{dd} \cdot I_{peak} \cdot t_s + A \cdot C \cdot V_{dd}^2 \cdot f + I_{leak} \cdot V_{dd} \quad (3.1) \]

In this equation, the first term is the dynamic power dissipation and the second term models the static power dissipation due to leakage current \( I_{leak} \). \( A \) is the switching activity factor for dynamic power, \( C \) is the switched capacitance, and \( V_{dd} \) is the supply voltage). Dynamic power was once the dominant power consumption term. However, as the result of technology scaling and \( V_{th} \) (threshold voltage) decreasing, leakage power now accounts for a large portion of total power consumption.

Although there are many techniques to reduce power dissipation, most existing efforts focus on one technique in isolation instead of concurrently applying a number of power minimization techniques. In this chapter, we propose a power optimization framework based on the genetic algorithm. Our optimization strategy combines four power reduction techniques: multiple \( V_{dd} \) assignment, multiple \( V_{th} \) assignment, gate sizing, and stack forcing. It simultaneously applies and evaluates the effects of these techniques to achieve maximum power saving under a hard timing constraint.

Figure 3.1 and 3.2 demonstrate how the proposed approach works. For a given circuit, the critical path of the circuit is first identified and is shown as the darkened gates in Figure 3.1. In Figure 3.2, the gates in the critical path are assigned with high supply voltage, while the others gates are assigned with low supply voltage. For gate A in the non-critical path, the stack forcing is applied to reduce the static power consumption. One level converter is inserted between the output of gate B and the input of gate C to eliminate the undesirable static current.
This current flows, since the logic *High* signal of the low supply voltage driven cell cannot completely turn off the PMOS pull-up network of the following high supply voltage driven cell. Since gate E is assigned with a high threshold voltage, it needs to be sized-up to keep the speed in order to meet the performance requirement. All these low power techniques should be carefully applied; otherwise, the critical path may change due to the complexity of a circuit, thereby increasing the critical path delay.

![Figure 3.1. An example circuit with a critical path of darkened gates.](image1)

![Figure 3.2. An example circuit with applying four low power techniques.](image2)

To this end, we present a GA-based power optimization framework that can simultaneously exploit four power optimization techniques: multiple supply voltage assignment, multiple threshold voltage assignment, gate sizing, and force stacking.
To the best of our knowledge, the use of all four low power techniques has not been attempted before and none of the previous works can do simultaneously joint optimization of more than three techniques. The framework is built on top of the genetic algorithm and the power optimization problem is encoded into chromosome representations which can easily be extended to include other techniques, such as multiple gate oxide assignment. This work had been published in International Symposium on Low Power Electronics and Design [24].

3.1 Delay Model

Since our goal is to reduce the power consumption under the performance constraints in association with a critical path, one kind of delay calculation needs to be assumed. The logical effort-based delay model [19] is adopted to perform the calculation of delay time. The derived technique, gain-based synthesis, has been stated its effectiveness in reducing synthesis-placement iterations and has already been incorporated in design automation flow in many industry companies, for example, IBM and Magma [25, 26]. The advantage of this approach resides in the use of a delay-centric approach to logic optimization which naturally targets timing optimization and the use of gain rather than size, and thus obfuscates the need for a pre-placement wire load model. Since the critical paths in the optimized circuit remain critical in the placement phase, the resultant synthesized circuits are good starting points for subsequent physical design.

The logic effort model is essentially a reformulation of the conventional RC model of CMOS gate delay, and can be simply stated as the follows:

\[ d = \tau \times (p + gh) \] (3.2)

In equation 3.2, \( g \) stands for logical effort, which is the ratio of input capacitance of a gate to the input capacitance of an inverter. The logical effort represents the fact that, for a given load, complex gates have to work harder than an inverter to produce a similar response. In other words, the logical effort tells how much worse it is at producing output current than an inverter. The \( h \) is the electrical effort which is defined as the ratio between the external load and the input capacitance.
of the gate. The $p$ represents the intrinsic delay and depends solely on diffusion capacitance of a gate. It can be seen from this equation that the delay of a gate is independent of $p$ and $g$ when $h$ is fixed. Since the logical effort of an inverter is defined as 1 in the original definition [19], we assume that the smallest size inverter will be given logical effort of 1, and for the rest of other gates, the logical efforts are assigned to be the ratio of input capacitances of these gates to the input capacitance of the smallest inverter.

An example is shown with two inverters in Figure 3.3. Although both of the inverters have the logical effort of 1, the delay of the left inverter is larger than the right inverter, with the reason of higher electrical effort, assuming $p$ is the same for both inverters. The $\tau$ in equation 3.2 is a scaling parameter that characterizes the semiconductor process being used in order to quantify the gate delay. That is, the term, $(p + gh)$ will then be multiplied by a process speed factor to obtain the actual delay time. For 0.25um technology, the process speed is 20 ps, while 15 ps for 0.18um. It approximates 8-11 ps for 65 nm technology, which is our target technology.

One thing to keep in mind is that we do not use the logic effort to calculate the delay of the level converter, because the spice simulation result for one specific implementation of the level converter is sufficient when we try to calculate the delay of a critical path. Furthermore, the level converter can be viewed as a separation point, where the path delay calculation before and after this point are independent.

With the facility of the logical effort, the total delay of a critical path can be defined as the summation of all gates along the path. In addition to the gate delay,
the delay of the level converter also needs to be considered. Thus, the total path delay $D$ should also take the number of level converters on the critical path into consideration and is thus defined as follows:

$$D = \sum_{i=1}^{n} d_i + \#LC \times delay_{LC}$$

(3.3)

By using the logic effort-based delay calculation and BFS STA-like approach, we can easily identify the criticalities of the paths. Note that, although more sophisticated timing analysis approaches can be used to accelerate the path identifying process, it is not the main concern and this is beyond the scope of this chapter. The estimated delay time will then be compared with the performance requirement we give as the input to the algorithm. When there is a discrepancy as the algorithm proceeds, a certain amount of penalties will be added to the fitness score of a chromosome, thus degrading the chance a chromosome can survive till the final round.

### 3.2 Genetic Algorithm

Genetic algorithms (GA) [54] are a class of search and optimization methods that mimic the evolutionary principles in natural selection. The evolution process generally eliminates the bad genes and maintains the good genes to generate better solutions. This concept has been recently applied to solve a range of complex VLSI combinatorial optimization problems [23, 27, 30, 31]. In this kind of approach, the feasible solution is usually encoded into a binary string called chromosome. Instead of working with a single solution, the search begins with a random set of chromosomes called initial population. Specifically, the more populations you have, the quicker you will converge or the more generations are needed with a small initial population. This characteristic can be used for running time controlling in some works. Each chromosome is assigned a fitness score that is directly related to the objective function of the optimization problem.

In general, one very critical aspect of GA is represented by the computation of the fitness function which is the key factor to the successful evolution in GA. The population of chromosomes is modified to a new generation by applying three
operators similar to natural selection operators – reproduction, crossover, and mutation. Reproduction selects good chromosomes based on the fitness score and duplicates them. Crossover picks two chromosomes randomly and some portions of the chromosomes are exchanged with a probability $P_c$.

Finally, a mutation operator changes a 1 to a 0 and vice versa with a small mutation probability $P_m$. A genetic algorithm successively applies these three operators in each generation until a termination criterion is met. One of its advantages is that it can very effectively search a large solution space while ignoring regions of the space that are not useful. That is, the solution space exploration is directed. This algorithmic methodology leads to very time-efficient searches. In general, a genetic algorithm has the following steps:

1. Generation of initial population.
2. Fitness function evaluation.
3. Selection of chromosome.
4. Reproduction, crossover, and mutation operations.

The selection function tries to select the parents’ chromosome with probabilities proportional to their fitness. In this way, the more highly fit chromosome will have a higher number of children in the succeeding generation. The crossover and mutation operation will modify the pattern of a chromosome in order to generate better survivable solutions, so it will be passed on to future generations.

Like most classical research and optimization methods, GA also faces difficulty in handling constraints. To handle this, the most commonly-used strategy is the penalty function method. In our problem, the constraint is the smallest delay time with the lowest power consumption. When the delay time is obtained in each generation, we compare it with the expected delay time. If this constraint is violated, an extra penalty is added to the fitness score of a chromosome. In this way, the reproduction operator discourages the propagation of the unfeasible solutions to future generations.
3.3 Power Optimization Framework

Our power optimization flow uses a genetic algorithm and is shown in Figure 3.4. The circuit configuration information, such as supply voltage assignment and gate sizing, are encoded into a binary string called chromosomes. The optimization flow begins with a random generated initial population, which consists of many randomly generated circuit configurations. The optimization flow is an iterative procedure. The chromosomes with better fitness will survive at each generation and are applied with three different operations (reproduction, crossover, and mutation) to be a new set of chromosomes – or new circuit configuration. The iteration continues until the termination criterion is met.

3.3.1 Chromosome encoding

Given different power reduction techniques, we can encode all the tuning variables into a binary chromosome string. Figure 3.5 shows both the structure of a chromosome and an encoding example, representing N gates in a circuit. This
encoding example is based on the assumption that we use a dual-$V_{dd}$, dual-$V_{th}$ library with four discrete sizes for each type of gate, and the gate has one forced stacking version. For example, zero (0) in voltage means using high $V_{dd}$, one (1) in threshold tells this gate to use low $V_{th}$, and zero (0) in stacking effect stands not to choosing the stacking version of a gate. For transistor sizing, we have used little endian notation; that is, double zero (00) is the minimum size and double one (11) is the maximum size. While the $V_{th}$ allocation and force stacking can be done at the transistor level, for simplicity we assume the granularity is at the gate level. In Figure 3.5, the chromosome shows that Gate 1 is assigned to use lower $V_{dd}$, higher $V_{th}$, and size 1 of the gate with no force stacking. Note that only the tuning variables are encoded into the chromosome. The type of each logic gate and the circuit topology information are known \textit{apriori} to calculate the power and delay based on the chromosome configuration.

This encoding scheme is easy to extend for a more complicated standard cell library. For example, if a standard cell library has more than two supply voltage choices or more than two threshold voltage choices, we just need more bits in the chromosome. Increasing the flexibility of the library simply increases the bits required for each gate. Since the chromosomes are randomly generated and we use the binary bit string representation for chromosomes, in some cases, the chromosome configurations may not be valid. For example, for a library with three threshold voltage choices, we have to use two bits for the $V_{th}$ configuration and
there will be one invalid configuration. We resolve this problem by adding penalties to fitness function, which is described in the next section.

### 3.3.2 Fitness function

The fitness function, which decides the surviving chance for a specific chromosome, is related to the power consumption and the delay of the circuit, as well as the validity of the chromosome. In the following subsections, we will give a detailed explanation for each factor in the objective fitness.

#### 3.3.2.1 Power

In our power optimization framework, the goal is to find a configuration such that the power consumption for the circuit is as low as possible and the timing requirement is met at the same time. Therefore, the fitness of a chromosome should be related to the power consumption of that particular configuration, which can be calculated using equation 3.1. For the dynamic power of the gate, the switching activity were obtained by exhaustive simulation, with the assumption that the input probabilities of being high or low are equal and independent. The gate leakage and sub-threshold leakage were also characterized by similar simulation. Our method is general enough and more accurate power estimates can be used, if more information on probabilities is available.

When using multiple supply voltages in a circuit, level converters are required whenever a logic gate at the lower supply has to drive a gate at the higher voltage [5]. Note that the overall power consumption of the circuit also includes the power consumptions from the level converters.

#### 3.3.2.2 Delay

It should be noted that the power optimization is under a specified timing constraint. If the critical path delay in a circuit is longer than the timing requirement, the configuration is not desirable and the corresponding chromosome should have little chance to survive. The delay calculation of the circuit is based on the logical effort [19] mentioned in section 3.1. The delay from level converters is also taken into account.
3.3.2.3 Penalty function

As we have discussed in section 3.3.1, since we use the binary string to represent a chromosome, when the number of choices for a tuning variable is not \(2^N\) (for example, a gate has six discrete sizes, or three threshold voltage choices), we may encounter an invalid configuration during the population initialization or chromosome operations. For those chromosomes representing invalid configurations, the chance of surviving should be set smaller than those of valid configurations. Based on the above argument, the fitness function can be defined as:

\[
\text{Fitness} = \frac{1}{\text{Total power}} - \text{Penalty}
\]  

(3.4)

where the penalty is a big number if timing is violated or the chromosome is invalid, such that those valid chromosomes with lower power consumption, while meeting timing requirements, have better chances to remain alive.

3.4 Experimental Evaluation

Previous works [12, 2] have shown that by using two supply and threshold voltages, there is substantial improvement in power dissipation. Additionally, the use of additional voltages results in small power improvements. This makes justifying the additional costs associated with multiple supply and threshold voltages difficult. Thus, to validate our proposed framework, we have constructed a dual-\(V_{dd}\) and dual-\(V_{th}\) standard cell library in 65 nm process technology. The logic gates in the library are inverter, NAND2, NOR2, XOR2, and a level converter. The framework was implemented in C. \(V_{th}\) allocation and force stacking can be done either at a transistor level or at a gate level and each type of the gate may have several discrete sizes. For simplicity, we assume the granularity of both \(V_{th}\) assignment and forced stacking are at the gate level and the gates have only two discrete sizes, such that we need only four bits to encode the tuning variables. Note that the flexibility of our approach is that only the length of bits will be affected to reflect the encoding change of the gate’s configuration.

Figure 3.6 shows the adopted level converter implementation [5]. As many researchers predict that it is not practical to provide multiple level converters within
a combinational block, integrating level converters into flip-flops and providing only supply voltage transition are more feasible options. In our framework, the use of the level converters can be constrained and enforced; moreover, the use of the supply voltage transition can also be confined. However, the purpose of our approach is to achieve maximum power savings while satisfying performance demands. Hence, we assume the possibility of arbitrary placement of level converters in our approach. Note that the delay and the power costs incurred from the level converter are accumulated and considered in calculating the path delays and the total power consumption of circuits.

The benchmark circuits we choose to map to our library span typical circuit topologies, including circuits from ISCAS 85 benchmark, an inverter chain, a 32-bit carry-ripple adder, a 8x8 carry-save multiplier, a 8-to-256 SRAM decoder, and three manually generated circuits. The circuit sizes range from 8 gates to 1050 gates. The circuits were first optimized for maximum speed (i.e. using all high $V_{dd}$ and low $V_{th}$ on the critical path with the larger transistor size and no stacking enforced) and then were optimized for lowest power consumption. Note that, if the demand performance can not be met with the above circuit configuration, our framework stops without evaluation. Such rigid performance requirements simply cannot be satisfied with the fastest circuit setting obtainable from our circuit technique combinations. If the reasonable performance is given, the best feasible solutions can be acquired through the evolution process (crossover and mutation), even though the initial population of chromosomes may not have the feasible solutions. We then perform power optimization with timing constraint relaxation. These experiments were carried out on an Intel Pentium 4 processor (2.8 GHz 512MB RAM).

3.4.1 Control parameters

Based on the conclusion in [12] that the optimal second $V_{dd}$ in a dual-$V_{th}$ system should be $\sim50\%$ of the higher supply voltage, the supply voltages for our library are 1V, 0.5V. For NMOS (PMOS) transistors, and the high threshold voltage and the low threshold voltage are $0.22\text{V}$ ($-0.22\text{V}$) and $0.12\text{V}$ ($-0.12\text{V}$) respectively. The library was characterized by using the Berkeley 65 nm BSIM predictive model [29].
The gate leakage and sub-threshold leakage were pre-characterized.

While generating the initial population, we have to set an appropriate population size, and the crossover probability $P_c$, as well as the mutation probability $P_m$. If the population size is too small, the genetic diversity within the population may not increase for many generations. On the other hand, a large population size increases the computation time for each generation; however, it may take fewer generations to find the best solution.

### 3.4.2 Generation and performance

In this subsection, we conducted experiments to explore the goodness of generated results from different aspects. As mentioned in the previous section, the number of generations used in the genetic algorithm has strong impacts on the solution. Figure 3.7 displays this phenomenon. The power consumption of the 32-bit carry-ripple adder (FA32) circuit decreases while the generation increases along 1% timing relaxation curve. We can also see that there is little improvement around 4000 generation and more generations will not help reduce power consumption. Therefore, the maximal generation is set at 4000 for the rest of the experiments.

Figure 3.7 also shows the power reduction process for the FA32 circuit under different timing relaxations. The timing relaxation is done through mitigating the performance requirement. That is, the delay of the critical path is enlarged to lessen the performance demand. As shown in the figure, we can see that for a very tight timing constraint (1% timing relaxation), the algorithm achieves a maximum power saving within a 3600-4000 generation region under this rigid timing
Solution convergences under different timing relaxations.

Figure 3.7. Solution convergences under different timing relaxations.

constraint. Running more generations will not help significantly. If we relax the timing constraint to 20% timing relaxation, the convergence can be achieved more quickly (within 1200 generation). With this performance degradation, the power saving is larger than that of 1% timing relaxation. Thus, we know that the performance and the power consumption of a design needs to be traded off cautiously in order to reach the point of design equilibrium.

Figure 3.8 shows the changes in power composition with the application of our algorithm on the FA32 benchmark. The timing constraint (cycle time) and power consumption are both normalized to the fastest speed setting. We can see that when the timing constraint is relaxed to be two times larger than the fastest speed, the power reduction we can achieve is about 45%. Further relaxation contributes to 65% power saving when the timing constraint is four times larger than that of the fastest performance speed. We can also see that most of the power reduction comes from the static power reduction. The implication behind this result is that with the less aggressive performance requirement, the algorithm is more effective in delivering low power budget solutions by applying four low power techniques. For the targeted 65 nm process technology, the static power is thus reduced more obviously than the dynamic power, as shown in the figure. One more observation from this result is that the dynamic power does not decrease as linearly as equation 3.1 indicates, when the timing constraint keeps relaxing. That is, the dynamic power should decrease at a per 1/T timing relaxation. The
reason for this phenomenon is that the goal of our fitness function tries to minimize the total power consumption instead of individual static or dynamic power. As a result, the reductions of the total power are more obvious from the figure.

### 3.4.3 Applying four low power techniques

Figures 3.9 and 3.10 present the static power and dynamic power breakdown for a maximal speed optimized circuit and a minimal power optimized circuit, respectively. With our 65 nm library, the static power accounts for an average 74% of the total power, while the dynamic power accounts for 26% of the overall power. After applying all four power reduction techniques without any timing constraints, the static power accounts for approximately 20% of the overall power, while dynamic power accounts for 80%. A significant part of overall power reduction is from the static power reduction by using these four techniques simultaneously. Overall, 84% of the power reduction is from the static power reduction.

Figure 3.11 shows the profile of power reduction techniques that are used in the FA32 circuit when the timing constraint is relaxed iteratively. The number of gates used for a specific configuration is normalized to the fastest-speed optimized case. It is obvious that as the timing constraint is relaxed from 0% to 400%, the gates tend to use the less power consuming configurations, i.e., low $V_{dd}$, high $V_{th}$, small gate size, and stacking force. The interaction of these four tuning variables ($V_{dd}$, $V_{th}$, sizing, and stacking force) can also be discovered from our experiment,
Figure 3.9. Power breakdown for speed-optimized circuits.

Figure 3.10. Power breakdown for power-optimized circuits.

which is represented by the fluctuations of four curves.

Figure 3.12 shows the power consumptions between configurations with stacking force and without stacking force. The comparison is conducted with all four low power techniques available and all techniques except stacking force. The power saving is 20% under the fastest-speed performance demand between stacking and non-stacking approaches. As the timing constraint keeps relaxing, the stacking approach still maintains a reasonable level of power savings, for example, 17.8% in 2X and 17.4% in 3X timing relaxations. The overall saving is 18.6% for all timing relaxations. From this result, we observe that stacking force is an effective mechanism to reduce power consumption.
Figure 3.11. The profile of used low power techniques.

![Graph showing distribution of used low power techniques (FA32)](image)

Figure 3.12. Comparison of power consumption with/without stacking force.

![Graph showing stacking vs. non-stacking (FA32)](image)

Based on the previous result, we know that the stacking force mechanism can be employed to lower the power consumption further; thus, we adopt this technique on all of the benchmarks. Figure 3.13 shows the comparison between power consumptions with and without stacking effect for all circuits. As can be seen from the figure, the stacking effect is effective in lowering the power consumptions under the same rigid performance demands. The power reduction can be as large as 20% for circuit FA32 and is 11% for circuit MUL, while the overall average power reduction is 6% for all seven circuits. This reduction is not as surprising as we anticipate. We have uncovered some reasons for this if we exclude two smallest benchmarks. First, a benchmark is an important factor in deciding the length of
the critical path. Unlike FA32 and MUL circuits, testbenches 1 to 3 have a fairly large proportion of gates lying on the critical paths. The gates on the critical paths are normally enforced not to use the stacking effect, in order to maintain their driving strength while allowing the rest of the gates to use this facility freely. However, as a large number of gates constitute the critical paths, only a small number of gates can fully take advantage of using the stacking force to save power. Second, although the stacking force can lower the static power, it requires other compensating techniques (high $V_{dd}$ or bigger transistor size) to recover the loss of the driving strength, which brings the dynamic power consumption higher. As the total power consumption is set for fitness function, the power of the stacking effect is thus cancelled out with each power consumption.

The reason why circuit INV100 is not included is that with the same rigorous performance requisite, this circuit simply cannot use the stacking force along the chain; otherwise, the performance demands cannot be met. Thus, under the same performance requirement, two configurations will result in the same power consumption. From this result, we know that the stacking force effect can lower power consumption even further when applied to the circuits with more than one path.

Figure 3.14 shows the comparison of power consumption with only one low power technique available at a time to one equipped with four low power techniques. The reason for some curves not showing the point from the beginning is that they

![Power consumption: stacking vs non-stacking](image)
simply can not meet timing considered at that point. We can easily observe from the figure that the dual-$V_{dd}$ is the most effective way to reduce power, while the dual-$V_{th}$ is the less effective one. But dual-$V_{th}$ can still maintain good timing constraint achievements compared to the other three techniques. The curve of all available techniques can easily maintain solutions with low power consumption and without incurring performance degradation.

The distribution of the level converter of testbench2 circuit is shown in Figure 3.15. With the fastest-speed performance requirement, the gates in the critical path are all assigned with high supply voltage; therefore, there are no needs for the level converter under this circumstance. In contrast, as the timing constraint keeps being relaxed, the use of the level converter increases. The reason behind this is that only some gates in the critical path need to be assigned with high supply voltage, while the rest of them are driven with low supply voltage with the purpose of saving power.

As can be observed in the figure, there are two sharp increases of the level converter, which are at 2X and 3.6X timing relaxation points. The reasoning for the first steep increase is that the increasing use of level converter on the non-critical paths may contribute to this phenomenon. For the second sharp increase of the level converter, the increasing use of the level converter on the non-critical path is the first reason, which is the same as the previous explanation, while the second reason is the relaxing timing. With less aggressive performance requirements, some
gates of a circuit may be assigned to be driven by low-supply voltage, and then
drive the following high-supply voltage gates, which require the level converter
between them.

3.4.4 Comparison with other existing combined power techniques

In this subsection, we conducted experiments to compare the existing proposed
approaches with our proposed techniques in terms of exploring the power-saving
issue. The best known method to save power is the use of multiple supply and
threshold voltages for a design. Thus, experiments are performed with both total
power and leakage power savings recorded to demonstrate the effectiveness of our
approach.

Total power is an important concern for high performance systems and portable
devices. Based on this concern, a comparison of power consumptions of different
combined low-power techniques is performed. Figure 3.16 shows the results of total
power consumptions from different combined low power techniques, which are dual-
$V_{dd}+V_{th}$, dual-$V_{dd}$+size, dual-$V_{dd}+V_{th}$+size, and all four techniques, respectively.
From the figure, the combined all four low power techniques outperform the other
three combined approaches in terms of total power saving. The dual-$V_{dd}+V_{th}$ also
functions better than dual-$V_{dd}$+size one. The rationale behind this is that the
leakage power is sensitive to the threshold voltage and the leakage power is the preeminent power in 65 nm process technology. With the help of dual threshold voltages, the dual-$V_{dd}+V_{th}$ approach can save more power from the leakage power aspect.

![Power comparison of different combined low power techniques](image)

**Figure 3.16.** Power comparison of $V_{dd}+V_{th}$, $V_{dd}+$size, $V_{dd}+V_{th}+$size, and all four low power techniques.

As we mentioned before, leakage is a dominant portion of the total power consumption; thus the comparison of leakage power on different combined techniques is evaluated. The experimental result is presented in Figure 3.17. As shown in the figure, the dual-$V_{dd}+V_{th}$ has the lower leakage power than that of the dual-$V_{dd}+$size approach, but applying all four low power techniques can result in the lowest leakage power consumption when compared with its counterparts. The implication from this result states that, although the leakage power can be effectively reduced by applying dual supply and dual threshold voltages with the strong relationship between them, applying stacking force, in fact, can further improve the leakage power by 16% of all benchmarks, in average.

### 3.4.5 Exploiting parallelism in genetic algorithms

Compared to the ILP approach that was used by [12, 15], one advantage of our GA approach is that the parallel nature of genetic algorithms suggests parallel processing as the natural route to explore. We implemented a parallel version of our algorithm on a dual Intel Xeon processors (3.2 GHz, 2GB RAM) machine
Figure 3.17. Comparison of leakage power from different combined low power techniques.

Figure 3.18. Run time comparison of single processor and dual processors.

running Linux, by dividing the population processing between dual processors. In Figure 3.18, we can see that using dual processors for parallel version of GA can effectively reduce the amount of running time needed in a single processor by almost 50%. We noticed that there was an average of 1.97X run-time speed-up on a 2-processor workstation against the single-processor version of our algorithm. The reason that we cannot achieve a 2X speedup is the interaction overhead between parallel processes. Note that the small circuits (C17, SRAM Decoder, INV100) cannot make use of the parallel advantage because the overhead incurred from the dual processors dissipates the speedup of parallelism.
Although it is beneficial to adopt the parallelized version of GAs to improve run-time efficiency, other techniques can also be incorporated. For instance, intelligence can be employed to identify critical paths of the circuit at the phase of generating the initial population. This way, the feasibility of convergence to the global optimal solution is guaranteed and the efficiency of the algorithm is also effectively improved.

Keep in mind also that the computation complexity seems to be a little bit high from the above figure. As we mentioned in section 3.4.1, the best population to be used should be in the range of 20 to 30 from the previous study [63]; however, the population of 100 was used in conducting the experiments. Moreover, we have used the number of 0.001% as the improvement indicator, which instructs the algorithm for early termination if the improvement does not exceed this number over a predefined generation. This improvement indicator is a relatively small number and thus has little possibility of early algorithm termination to save the computational time. Therefore, we believe that the run-time of our framework can be effectively reduced by carefully choosing these two numbers.

The last thing we want to emphasize is the scaling ability of the genetic algorithm. As far as scaling is concerned, usually, the top priority is to find out how the GA behaves on problems in which the problem size is exponentially scaled. Lobo et al. [33] conducted a theoretical and empirical analysis of the time complexity of genetic algorithms. As indicated in that work, genetic algorithms with perfect mixing have a time complexity of $O(m^2)$ for cases of exponential scaling. To that end, although we did not perform a thorough theoretical and empirical analysis on our framework, we believe that the formation of the power optimization problem in GA is still suitably efficient as the problem size grows.
2D Thermal-Aware Floorplanning

Floorplanning is an important step in VLSI physical design. It can roughly estimate the layout of a given set of functional blocks and this problem is known to be NP-complete [56]. The primary objective for floorplanning is to minimize the total area required to accommodate all the functional blocks on a chip. In this chapter, a thermal-aware floorplanning algorithm is proposed, which can reduce the hot spot temperature and achieve a thermal balanced design. This work had been published in International Symposium on Quality Electronic Design, 2005 [51].

4.1 2D Thermal-Aware Floorplanning Framework

Our thermal-aware floorplan optimization flow is based on the genetic algorithm as described in Section 3.2 and is shown in Figure 4.1. The representation of a floorplan design is encoded into integer and bit mixed strings called chromosomes. The optimization flow begins with a randomly generated initial population, which consists of many randomly generated chromosomes (floorplans). The orientation of each block and the polish expression of a slicing tree are obtained from individual rotate and slicing strings. The area estimation function and HotSpot are invoked to calculate dead space and temperature. The fitness is assigned to each population based on the evaluation of these two variables. The optimization flow is an iterative procedure. The chromosomes with better fitness will survive at each generation and the three different operations (reproduction, crossover, and mutation) are invoked to derive a new set of chromosomes – or new floorplans. The iteration continues.
until the termination criterion is met.

Figure 4.1. The flow of the thermal-aware floorplanning.

4.1.1 Chromosome encoding

The representation of a floorplan can be divided into two categories. One is slicing and the other is non-slicing representations. Although a slicing floorplan can be sub-optimal, empirical evidence shows that it is still quite efficient in tightly packing modules, as evident in the work of Young and Wong [57]. Because of the simplicity and the effectiveness of the slicing method [58], we use a slicing tree representation to be incorporated with the genetic algorithm. A slicing floorplan is a rectangular area that is sliced recursively by a horizontal or vertical slicing line into a set of rectangular regions to accommodate a set of functional modules. A slicing floorplan can also be represented by Polish expression [58]. Figure 4.2 shows an example of a slicing tree, the polish expression, and its corresponding floorplan. The slicing tree is a binary tree and is bottom-up constructed.

One example of the chromosome encoding is shown in Figure 4.3. The slicing array contains four unique integers which represent four functional blocks and two slicing operators, "*" and "+". The "*" operator is used for vertical cutting while "+" operator represents cutting horizontally. There is another bit array called rotate in our encoding, which is used to decide whether a block should rotate, that is, switch its width and height. Each integer maps to a fixed block to which it associates with the width and height measurements. The purpose of
accounts for chromosome A, while the floorplan on the left represents the floorplan information and power consumptions associated with each block. A fitness function is then used to evaluate the suitability of the chromosome, based on different optimization objectives.

**Figure 4.2.** The slicing tree representation and its corresponding floorplan.

bit array is to add more flexibility in finding good solutions (minimal area) as the evolution progresses. For example, the floorplan as shown on the right in Figure 4.3 accounts for chromosome A, while the floorplan on the left represents chromosome B. With this representation, the exact location and orientation of each block on a chip can be extracted easily. After obtaining a valid chromosome, the temperature calculation is invoked by using HotSpot with the floorplan information and power consumptions associated with each block. A fitness function is then used to evaluate the suitability of the chromosome, based on different optimization objectives.

**Figure 4.3.** The encoding of chromosome for floorplanning.

4.1.2 Fitness function

The fitness function, which decides the surviving chance for a specific chromosome, is related to the mapping goals. Depending on the optimization goal, the
fitness function of the genetic algorithm is different. Below we show two different optimization goals.

4.1.2.1 Area minimization design

This optimization strategy attempts to achieve a minimum total chip area to accommodate a set of modules. The fitness of a chromosome is defined in Equation 4.1. The \( \text{area}(B) \) is the total area of blocks and \( \text{area}(C) \) is the total area of the floorplan with respect to the chromosome. The difference of \( \text{area}(C) \) and \( \text{area}(B) \) is the so-called dead space. The fitness calculates the proportion of the dead space of a floorplan and normalizes it to the total modules’ areas. Higher fitness implies a better chance for the chromosome to survive to the next generation.

In a area-based approach, there is no information about the temperature profile; thus, uneven temperature distribution and several hot spots might appear due to packing high power density blocks tightly together.

\[
\text{Fitness} = \frac{1}{(\text{area}(C) - \text{area}(B))/\text{area}(B)} \quad (4.1)
\]

4.1.2.2 Temperature balanced design

The goal of thermal-aware floorplanning is to distribute temperature evenly across a chip and to minimize the hot spot temperature under a specified area constraint. The easiest way is to separate the heat sources as far as possible in order to allow the heat drives away. However, that may result in large dead space and/or total chip area. In order to consider area and temperature constraints together, we first choose those floorplans that satisfy the given dead space ratio by Equation 4.1. For those unsatisfied chromosomes, a penalty is given to make them less unfavorable.

The fitness for thermal effect is defined as:

\[
\text{Fitness} = \frac{1}{(T_{\text{max}} - T_{\text{avg}})/T_{\text{max}}} - \text{penalty} \quad (4.2)
\]

where \( T_{\text{avg}} \) is the average temperature for all modules and \( T_{\text{max}} \) represents the maximal temperature produced by this floorplan. The penalty is a large value if the dead space ratio exceeds a user-defined threshold, such that those valid chromosomes with satisfactory dead space ratio and with lower average and maximum
temperatures have better fitness to survive.

4.1.3 Control parameters

While generating the initial population, we have to set an appropriate population size, and the crossover probability $P_c$, as well as the mutation probability $P_m$. If the population size is too small, the genetic diversity within the population may not increase for many generations. On the other hand, a large population size increases the computation time for each generation; however, fewer generations may be necessary to find the best solution. Schaffer et al. [63] had conducted extensive simulation on a wide range of functions and concluded that a small population of size 20 to 30, a crossover probability in the range of 0.75 to 0.95, and a mutation probability in the range of 0.005 to 0.01 perform very well. In our implementation, we set the population size to be 100, the crossover probability $P_c$ to be 0.8, and the mutation probability $P_m$ to be 0.02. The termination of the iterative evolution can be user-defined. We set a maximum generation to be 100,000 and specify that if the fitness improvement is less than 0.001% during the last 1000 generations, the evolution stops without going through all generations.

4.2 Experimental Results

To evaluate our genetic algorithm based approach, we implemented the algorithm in C, and the experiment was conducted on a dual Intel Xeon processors (3.2 GHz, 2GB RAM) machine running Linux. We performed experiments on a set of MCNC benchmarks and one real application for face detection. The runtime of 100,000 generations is about 900 seconds for the largest benchmark.

4.2.1 Experiment on MCNC benchmarks

We first use area optimization to demonstrate that our approach is comparable to non-slicing approaches. Table 4.1 lists the names of the circuits, the number of modules and the required area for different approaches. From the table we can see that the slicing approach still performed comparable to non-slicing approaches.
The required area for ami49 circuit is a little larger than that for other methods, while that of the rest of the benchmark circuits are analogous.

To conduct experiments on temperature-balanced optimization, the power values from 0.05\(mW\) to 3\(W\) were randomly assigned to the blocks in different benchmark due to the lack of information on the internals of each block. In addition, a real application with accurate power estimation will be shown in the next subsection.

<table>
<thead>
<tr>
<th>Circuit</th>
<th># of cells</th>
<th>Cell area (mm(^2))</th>
<th>O-tree</th>
<th>B*-tree (Iter.)</th>
<th>B*-tree (SA)</th>
<th>Cluster refinement</th>
<th>Slicing</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>9</td>
<td>46.56</td>
<td>47.46</td>
<td>46.92</td>
<td>46.92</td>
<td>48.40</td>
<td>47.52</td>
</tr>
<tr>
<td>xerox</td>
<td>10</td>
<td>19.35</td>
<td>20.18</td>
<td>20.06</td>
<td>19.83</td>
<td>20.30</td>
<td>20.26</td>
</tr>
<tr>
<td>hp</td>
<td>11</td>
<td>8.83</td>
<td>9.49</td>
<td>9.17</td>
<td>8.95</td>
<td>9.58</td>
<td>9.44</td>
</tr>
<tr>
<td>ami33</td>
<td>33</td>
<td>1.15</td>
<td>1.25</td>
<td>1.27</td>
<td>1.27</td>
<td>1.21</td>
<td>1.27</td>
</tr>
<tr>
<td>ami49</td>
<td>49</td>
<td>35.44</td>
<td>38.60</td>
<td>37.43</td>
<td>36.80</td>
<td>37.70</td>
<td>39.16</td>
</tr>
</tbody>
</table>

Table 4.1. Comparisons of area requirements among O-tree, iterative and simulated annealing B*-tree, cluster refinement. (Data taken from [64] for non-slicing approaches.)

The area-optimized floorplan of example hp is shown in Figure 4.4. The black areas depict the dead space, while the numbered blocks represent different modules. The shaded blocks are the top four modules that had the largest power consumption. To obtain a thermal balanced design, we use the minimum area achieved in the area-optimized approach as a constraint and apply the thermal balanced optimization by using Equation 4.2. Figure 4.5 shows the floorplan of the same example circuit hp. The hot spot temperature (maximum temperature) drops from 123\(^o\)C to 120\(^o\)C, and the average temperature drops from 111\(^o\)C to 110\(^o\)C, while the resulting area is the same. Comparing Figures 4.4 and 4.5, we can see that the thermal-aware floorplanning spreads out the modules with high power consumption, such that it can acquire a lower hot spot temperature and thermal-balanced floorplan.

By relaxing the area constraint, the thermal-aware optimization approach can achieve temperature reduction further for two reasons: First, the heat in hotter regions can flow through more dead areas that consume no power; second, it has more flexibility for allocating high-power consuming modules into less hot regions. The
following results confirm our rationale. The maximal and average temperatures of ami49 under different dead space ratios are shown in Figure 4.6. Temperature decreases with gradually increasing dead space ratio.

Figure 4.7 and Figure 4.8 show the average and maximal temperatures of all five MCNC benchmarks with/without thermal consideration. The figures show that the area optimization results in higher average and maximal temperature for all benchmarks. The temperature can be effectively reduced through thermal-aware optimization combined with area constraint, such that lower temperature is achieved with the same compact floorplan. The overall average temperature
Figure 4.6. The comparison of temperature and dead space ratio for ami49 under thermal-aware optimization.

is reduced around 1~5°C, as shown in Figure 4.7. The hot spot temperature (peak temperature) is reduced by as much as 24°C in example apte in Figure 4.8. However, some benchmarks could not achieve large temperature reduction. For example, the hot spot temperature reduction for ami33 is only 1°C. The reason behind this is that the power density is relatively high for a limited small area and thus there is not much flexibility for the algorithm to discover a good solution.

Figure 4.7. The comparison of average temperatures for area optimization and temperature-aware optimization.
4.2.2 Experiment on a face detection design

To demonstrate our algorithm, we mapped a neural network used in detecting the angle of rotation on face images. The network is part of an on-chip face detection application [66] and consists of three layers of neurons: input layer neurons, hidden layer neurons, and output layer neurons. There are 15 input layer neurons, 15 hidden layer neurons, and 36 output layer neurons. The three types of neurons differ from one another, resulting in different area and power consumption over a single detection.

To obtain the power consumption and dimensions of each neuron type, we first designed and synthesized the neural network on an on-chip network using Verilog HDL and Synopsys Design Compiler. We then simulated the entire network operation using a Network-On-Chip simulator [72], which we used to obtain the network traffic, including addressing and control data. We used this network traffic as input to Synopsys Power Compiler, thus simulating the actual switching activity of the neurons to obtain the average power consumption for a single detection. The overall simulation procedure is shown in Figure 4.9. Table 4.2 shows the resulting power consumption and dimension for different neuron blocks.

In Figure 4.10, the result from applying thermal-aware floorplanning to the face detection application is shown. The area, average, and maximal temperatures are normalized. As can be seen from the figure, the maximal temperature reduces
Figure 4.9. The overall design methodology for face detection application’s power estimation.

<table>
<thead>
<tr>
<th></th>
<th>Type A</th>
<th>Type B</th>
<th>Type C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Power (mw)</strong></td>
<td>214.53</td>
<td>52.11</td>
<td>14.17</td>
</tr>
<tr>
<td><strong>Dimens (um²)</strong></td>
<td>1200x450</td>
<td>500x260</td>
<td>200x400</td>
</tr>
<tr>
<td><strong>Number</strong></td>
<td>15</td>
<td>15</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 4.2. Power consumption and neuron dimension profile of the face detection application design.

sharper than average temperatures before 0.2 dead space ratio. After that point, the maximal temperature reduction is small. This experimental result shows that, with minor area penalties, both the hot spot and the average temperatures can be effectively reduced.
Figure 4.10. The comparison of area average and maximal temperatures under thermal-aware optimization.
Aggressive scaling of process technologies has enabled feature sizes to shrink continuously. While the performance of gates has been improving, there are concerns about the performance of wires in scaled technologies. As Ho et al. stated in [73], there are two kinds of wires. One category of wires is local wires inside logic modules that scale in length; another category is long wires that do not scale accordingly with technologies. In order to keep the delays of these long wires tractable, repeaters and flip-flops are inserted to prevent performance degradation. However, these additional components have detrimental impacts on interconnect power dissipation. Consequently, intermediate and global interconnects of current microprocessors contribute to a major portion of power consumption and also serve as impediments for better performance. Hence, many research efforts are devoted to seeking solutions which can overcome the limitation of wiring requirements for present and future chip designs.

There are various reasons why the interconnect has become the center of attention in terms of power consumption and performance of a chip. First, interconnects, unlike transistors, have not scaled down exponentially as we move to the nano-meter era, and thus, a larger portion of total chip capacitance comes from the interconnect capacitance. Second, long interconnects, compared to the scaled transistors, are becoming exceptionally long. As a result, performance degradation is inevitable. Finally, as Kapur et al. states in [74], that with the introduction of
repeaters and vias to compensate the performance lost, interconnect power consumption almost doubles.

One technique being actively researched to alleviate the problems of interconnects is the use of three dimensional (3D) integrated circuits. There are numerous novel 3D architectures under development. In this chapter, we have considered one of the promising styles of 3D technologies: wafer-bonding technology [89]. Wafer-bonding technology holds active device layers after processing each active device layer separately and the connections are provided by 3D vertical vias. 3D architectures are effective in reducing wire length from the geometric point of view. This situation is schematically shown in Figure 5.1. The net length between modules $b_1$ and $b_2$ can be made shorter in 3D configuration due to the added flexibility of wiring in the third dimension. If modules were carefully placed on a 3D chip, many of the long interconnects and the large power consumption associated with them can be reduced. A major concern in the adoption of 3D architecture is the increased power densities that can result from placing one computational block over another in the multi-layered 3D stack. Since power densities are already a major concern in 2D architectures, the move to 3D architectures could accentuate the thermal problem. However, 3D chips could offer some respite due to reduced interconnect power consumption because of the shortening of many long wires. Consequently, an investigation of how the increased power densities of the stacked blocks and the reduced interconnect power consumption combine to influence the thermal behavior is essential.

![Figure 5.1. Wire length in 2D and 3D architecture configurations.](image)

In this chapter, we explore a floorplanning algorithm that attempts to reduce the peak temperatures of a 3D design. High temperature has adverse impacts on circuit performance. The interconnect resistance becomes larger, while the driv-
ing strength of a transistor decreases with increasing temperature. In addition, leakage power has an exponential dependence on the temperature and can even result in thermal runaways. Finally, higher temperatures are known to accelerate failure mechanisms and to reduce the lifetime of the device. While there have been prior proposals to do thermal aware floorplanning, the interconnect power consumption is often not considered. Our results in this work indicate that excluding interconnect power can result in peak temperatures to be underestimated by as much as 15°C in 90 nm technology. In order to capture the actual influence of the interconnect, a detailed model of a microprocessor is used to model the interaction between the functional modules and relative interconnects. This work had been published in Internation Symposium on Quality Electronic Design, 2006 [67].

5.1 Experimental Methodology

In this section, we describe our experimental methodology. First, we introduce the Alpha-like, detailed microprocessor architecture. This architecture model is then mapped to OKI 160 nm and TSMC 90 nm libraries by Design Compiler. Then it is placed and routed through First Encounter to extract power consumption numbers of functional modules, interconnects among modules, and area of each module as well. This information will be used next in guiding the algorithm to generate a solution with the lowered hot spot temperatures from our 2D/3D floorplanner.

5.1.1 Processor model

In order to explore the architecture-level interconnect power consumption of a modern microprocessor effectively, we need a detailed model which can act for the current-generation high-performance microprocessor designs. Based upon this requirement, we have used IVM [87], a Verilog implementation of an Alpha-like architecture (denoted as Alpha in the rest of this chapter) at register-transfer-level, to evaluate the impacts of both interconnect and module power consumptions at the granularity of a functional module level. A diagram of the processor is shown in Figure 5.2. Each functional block in Figure 5.2 represents a module used in our floorplanner. The registers between pipeline stages are also modeled but not shown in the figure.
5.1.2  B*-tree floorplan model

The floorplanner used in this work is based on the B*-tree representation. The B*-tree is proposed by Chang et al. in [86] and will be described briefly here.

A B*-tree is an ordered binary tree which can represent a non-slicing admissible floorplan. Figure 5.3 shows a B*-tree representation and its corresponding floorplan. The root of a B*-tree is located at the bottom-left corner. A B*-tree of an admissible floorplan can be obtained by a depth-first-search procedure in a recursive fashion. Starting from the root, the left subtree is visited first then followed by the right subtree, recursively. The left child of node \( n_i \) is the lowest unvisited module in \( R_i \), which denotes the set of modules adjacent to the right boundary of module \( b_i \). On the other hand, the module, representing the right child of \( n_i \), is adjacent to and above module \( b_i \). We assume the coordinates of the root node are always (0,0) residing at the bottom-left corner. The geometric relationship between two modules in a B*-tree is maintained as follows: the x-coordinate of node \( n_j \) is \( x_i + w_i \), if node \( n_j \) is the left child of node \( n_i \). That is, \( b_j \) is located and is adjacent to the right-hand side of \( b_i \). For the right child \( n_k \) of \( n_i \), its x-coordinate is the same as that of \( n_i \), with module \( b_k \) sitting adjacent to and above \( b_i \). There are several operations used in perturbing a B*-tree floorplan and they are listed.
below:

![Floorplan and B*-tree](image)

**Figure 5.3.** (a) one floorplan example (b) the corresponding B*-tree

While the original B*-tree structure was developed and used for the 2D floorplanning problem, we modified the perturbation function to handle 3D floorplans in this work. There are six perturbation operations used in our algorithm and they are listed below:

1. Node swap, which swaps two modules.
2. Rotation, which rotates a module.
3. Move, which moves a module.
4. Resize, which adjusts the aspect-ratio of a soft module.
5. Interlayer swap, which swaps two modules at different layers.
6. Interlayer move, which moves a module to a different layer.

The first three perturbations are the original moves defined in [86]. Since these moves only have influence on the floorplan in a single layer, more interlayer moves, (5) and (6) are needed to explore the 3D floorplan solution space.

### 5.1.3 Simulated annealing engine

A simulated annealing engine is used to perturb floorplanning solutions. The inputs to our floorplanning algorithm are the area of all functional modules as shown in Figure 5.2 and interconnects among modules. However, the actual dimension of each module is unknown *a priori* except for its area before placement. That is, we have to treat them as *soft* modules. Thus, we provide the choice of adjusting
the aspect-ratio as one perturbation operation. During a simulated process, each module dynamically adjusts its aspect-ratio to fit closely with the adjacent modules, that is, with no dead space between two modules. A traditional weighted cost representing optimization costs (area and wire length) is generated after each perturbation.

Different from 2D floorplanning, our 3D floorplanner uses a two-stage approach. The first stage tries to partition the blocks into the appropriate layers and also tries to minimize the packed area difference between layers and total wire length, using all perturbation operations (one through six listed in previous subsection). However, due to the fact the first stage was trying to balance the packed areas of the different layers, the floorplan of some of the layers may not be compactly packed. The second stage is intended to overcome this problem. Thus, in the second stage, we start with the partitioning solution generated by the first stage and focus on adjusting the floorplan of each layer simultaneously with the first four operations. At this point, there are no interlayer operations to disturb module partition of each layer obtained from stage one.

One problem of 3D floorplanning is the final packed area of each layer must match to avoid penalties of the chip area. For example, assuming two layers, L1 and L2, if the final width of packed modules of L1 is larger than the final width of packed modules of L2 and the height of L1 is smaller than that of L2, a significant portion of chip area is wasted due to the need for the layer dimensions to match for manufacturing. Thus, care must also be taken in both stages of our algorithm so that the dimension of each layer will be compatible. Thus, we adopt the concept of dimension deviation \( dev(F) \) in [83]. The goal is to minimize \( dev(F) \), which tells the deviation of the upper-right corner of a floorplan from the average \( Ave_x \), \( Ave_y \) values. The value, \( Ave_x \) can be calculated by \( \sum ux(f_i)/k \); where \( ux(f_i) \) is the x-coordinate of upper-right corner of floorplan \( i \) and \( k \), indicating the number of layers. The value \( Ave_y \), can be obtained in similar manner. Thus, \( dev(F) \) is formulated as \( \sum_{\# \text{ layers}} |Ave_x - ux(f_i)| + |Ave_y - uy(f_i)| \). The modified cost function for 3D floorplanner can be written as

\[
\text{cost} = \alpha \times \text{area} + \beta \times \text{wl} + \gamma \times dev(F) \tag{5.1}
\]
where \textit{area} and \textit{wl} are chip area and wire length, respectively.

### 5.1.4 Interconnect power distribution

With the detailed model of the Alpha-like microprocessor in Verilog, the actual power consumptions of functional modules and interconnects are extracted by Design Compiler and First Encounter, and are used in our floorplanning algorithm. After running our 2D floorplanner, we have the total interconnect length for 2D architecture. Since there is still no available tool that can be used to model accurately the power dissipation of all interconnects in 3D architecture, one simple way to obtain this number is through scaling. The approximation is done by scaling the 3D interconnect length with the interconnect length from 2D architecture. Thereby, the total interconnect power consumption in 3D is formulated as equation 5.2,

\[
3DP_{int} = 2DP_{int} \times (3DL_{int}/2DL_{int})
\]

where \(3DP_{int}\) represents the power consumption of all nets in 3D and \(3DL_{int}\) indicates the total net length accumulated through 3D floorplanner. Since HS3D only models thermal effects at the per-module level, we need a mechanism to account for those interconnect-induced power consumptions. That is, an approach to distribute the power consumed by each net to the modules is required. We accomplish this goal by using the intuition that power consumption is relative to capacitance and capacitance is proportional to the module area. Thus, from equation (2), the power value of each net, \(n_i\), is the ratio of the net length of \(n_i\) to the total net length multiplied by total net power either in 2D or 3D. Finally, the amount of net power contributing to the connecting module \(b_j\) is the ratio of the area of \(b_j\) to the total area of \(n_i\)'s connecting modules multiplied by the power of \(n_i\), and can be stated as follows:

\[
Net_{ij} = \left( \frac{n_i}{3DL_{int}} \right) \times 3DP_{int} \times \left( \frac{A_{b_j}}{TBA_{n_i}} \right)
\]

where \(Net_{ij}\) indicates the amount of power from net \(n_i\) contributing to module \(b_j\), \(A_{b_j}\) represents the area of functional module \(b_j\), and \(TBA_{n_i}\) tells the total area of connected modules of \(n_i\).
5.1.5 Temperature approximation

Although HS3D can be used for providing temperature feedbacks, when evaluating a large number of solutions during simulated procedure, it is not wise to involve the time-consuming temperature calculation every time. Other than using the actual temperature values, we have adopted the power density metric as a thermal-conscious mechanism in our floorplanner. The temperature is heavily dependent on power density based on a general temperature-power equation: $T = P \cdot R = P \cdot (t/k \cdot A) = (P/A) \cdot (t/k) = d \cdot (t/k)$, where $t$ is the thickness of the chip, $k$ is the thermal conductivity of the material, $R$ is the thermal resistance, and $d$ is the power density. Thus, we can substitute the temperature and adopt the power density, according to the equation above, to approximate the 3-tie temperature function, $C_T = (T - T_o)/T_o$, proposed in [80] to reflect the thermal effect on a chip. As such, the 3-tie power density function is defined as $P = (P_{max} - P_{avg})/P_{avg}$, where $P_{max}$ is the maximum power density of all modules while $P_{avg}$ is the average power density of all modules. The cost function for 2D architecture used in simulated annealing can be written as:

$$\text{cost} = \alpha \cdot \text{area} + \beta \cdot \text{wl} + \gamma \cdot P$$  \hspace{1cm} (5.4)

For 3D architectures, we also adopt the same temperature approximation for each layer as horizontal thermal consideration. However, since there are multiple layers in 3D architecture, the horizontal consideration alone is not enough to capture the coupling effect of heat. The vertical relation among modules also needs to be involved and is defined as: $OP(TPm) = \sum (Pm + Pm_i) \cdot \text{overlap area}$, where $OP(TPm)$ stands for the summation of the power density of module, $Pm$, and all overlapping module $m_i$, with module $m$ and their relative power densities multiplying their corresponding overlapped areas.

The rationale for this procedure is that for a module with relatively high power density in one layer, we want to minimize its accumulated power density from overlapping modules located in different layers. We can define the set of modules to be inspected, so the total overlap power density is $TOP = \sum OP(TP_i)$, for all modules in this set. The cost function for 3D architecture is thus modified as
Table 5.1. Floorplanning results of 2D architecture.

\[ \text{cost} = \alpha \ast \text{area} + \beta \ast \text{wl} + \phi \ast \text{dev}(F) + \gamma \ast P + \delta \ast \text{T OP} \quad (5.5) \]

At the end of the algorithm execution, the actual temperature profile is reported by the HS3D tool.

## 5.2 Experimental Results

Table 5.2. Floorplanning results of 3D architecture.

We implemented the proposed floorplanning algorithm in C++. The thermal model is based on the HS3D. The implementation of microprocessor has been mapped to OKI 160 nm library by Design Compiler and placed and routed by First Encounter under 1GHz performance requirement. We have also synthesized the microprocessor design to TSMC 90 nm low power library in order to exhibit
the significance of interconnect power when technologies were scaled. There are
totally 34 functional modules and 168 netlists extracted from the processor design.
The area and power consumptions from the actual layout served as inputs to our
algorithm. The experiment was run on a dual Intel Xeon (3.2 GHz, 2GB RAM)
machine running Linux. Other than an Alpha processor, we have also used MCNC
benchmarks to verify our approach. A similar approach in [82] is used to assign the
average power density for each module in the range of $2.2 \times 10^4$ (W/m²) and $2.4 \times 10^6$
(W/m²). The total net power is assumed to be 30% of total power of modules due
to lack of information for the MCNC benchmarks and the total wire length used
to be scaled during floorplanning is the average number from 100 test runs with
the consideration of area factor alone. The widely used method of a half-perimeter
bounding box model is adopted to estimate the wire length. Throughout the
experiments, two-layer 3D architecture was assumed, due to a limited number of
functional modules and excessively high power density beyond two layers; however,
our approach is capable of dealing with multiple-layer architecture.

Tables 5.1 and 5.2 show the experiment results of our approach when consider-
ing traditional metrics (area and wire) and thermal effect. The dead space columns
in Table 5.1 demonstrate the effectiveness of our tightly compact floorplanner with
the ability of adjusting the aspect-ratio for soft modules. The peak temperature
results from these tables also reiterate the importance of including the intercon-
nect power consumption, which most prior works ignore. The difference between
peak temperatures not considering interconnect power (peakT) and considering it
(peakT(Int)) is 6°C on the average. When taking thermal effect into account, our
thermal-aware floorplanner can reduce the peak temperature by 7% on the aver-
age, while increasing wirelength by 18%, and providing a comparable chip area as
compared to the floorplan generated using traditional metrics. Note that the tem-
perature estimation doubles the floorplanner execution time for the thermal-aware
case.

When we move to 3D architectures, the peak temperatures increased by 18%
(on the average) as compared with the 2D floorplan due to the increased power
density. However, the wire length and chip area were reduced by 32% and 50%,
respectively. The adverse effect of the increased power density in the 3D de-
sign can be mitigated by our thermal-aware 3D floorplanner. We lower the peak
temperature by 6% (peakT(Int)) with little area increase as compared to the 3D floorplanner, which does not account for thermal behavior.

Figure 5.4 shows the temperature profile of the peak and the average temperatures with (peakT(Int), avgT(Int)) and without (peakT, avgT) the interconnect power for both 2D and 3D architecture for the Alpha design. As expected, the chip temperature is higher when we step from 2D to 3D architecture without thermal consideration. Although the wire length is reduced when moving to 3D and thus accordingly reduces interconnect power consumption, the temperature for 3D architecture is still relatively high, due to the accumulated power densities from different layers and the diminished chip area. After applying our thermal-aware floorplanner, the peak temperature is lowered to a moderate level through the separation of high power density modules in different layers.

![Temperature Profile](image)

**Figure 5.4.** Thermal profiles of Alpha processor for 2D and 3D architectures.

Figures 5.5 and 5.6 show the interconnect distribution of Alpha processor for 2D and 3D architectures. As can be seen from Figure 5.5, almost half of interconnect lengths fall into the 400~2400 microns range in 2D, while the range is 400~1600 microns in 3D in Figure 5.6. As such, the 3D architecture is effective in lowering interconnect length due to its geometric advantage and, hence, the interconnect power consumption.

Table 5.3 shows the ratio of interconnect power to total power consumption. In the 3D case, the interconnect power becomes a less dominant part of the overall power consumption, accounting for only 15% of overall power in both cases, as
Figure 5.5. Interconnect distribution of Alpha processor under 2D architecture.

Figure 5.6. Interconnect distribution of Alpha processor under 3D architecture.

<table>
<thead>
<tr>
<th></th>
<th>2D</th>
<th>2D (Ther)</th>
<th>3D</th>
<th>3D (Ther)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>20.92</td>
<td>26.00</td>
<td>15.15</td>
<td>15.37</td>
</tr>
<tr>
<td>xerox</td>
<td>23.16</td>
<td>23.19</td>
<td>14.17</td>
<td>14.04</td>
</tr>
<tr>
<td>hp</td>
<td>27.96</td>
<td>27.79</td>
<td>19.97</td>
<td>18.08</td>
</tr>
<tr>
<td>ami33</td>
<td>13.44</td>
<td>15.63</td>
<td>8.5</td>
<td>8.3</td>
</tr>
<tr>
<td>ami49</td>
<td>22.46</td>
<td>27.96</td>
<td>19.32</td>
<td>19.55</td>
</tr>
<tr>
<td>Average</td>
<td>21.58</td>
<td>24.11</td>
<td>15.42</td>
<td>15.06</td>
</tr>
</tbody>
</table>

Table 5.3. Interconnect power profile.
compared to approximately 21% and 24% (Ther) for the 2D cases. Note that these numbers are lower than 30%, because the total wire length used for scaling is obtained prior to including the wire factor in the cost function.

In the next experiment, we demonstrate the importance of the impact of interconnect when moving to a more advanced technology. Figure 5.7 shows the temperature difference between 160 nm and 90 nm technologies under 2D and 3D architecture configurations. As can be seen from the figure, the temperature difference without considering interconnect power can be as high as 15°C in 90 nm, which is higher than those (6∼8°C) in 160 nm technology. Based on this result, we state that more attention should be drawn to interconnect in future technologies and it is also imperative to include interconnect power estimates in guiding any thermal-aware floorplanning.

Figure 5.7. Temperature profiles of Alpha processor for 160 nm and 90 nm technologies.
Chapter 6

Thermal-aware Task Allocation and Scheduling for Embedded Systems

Hardware/software co-synthesis is typically the first step in an embedded design procedure. It partitions the system specification into hardware and software modules to meet performance, power, and cost goals [49]. During the co-synthesis process, the architecture space is explored and the partition of tasks on software (CPUs) and hardware (ASICs) is generated. Then, a task allocation and scheduling routine is used in the inner loop of co-synthesis to determine the mapping and execution schedules of tasks on processing elements (PEs) and evaluate the quality of the co-synthesis.

As technology scales, temperature in modern high-performance VLSI circuits has moved up dramatically due to smaller feature sizes, higher packing densities and rising power consumptions. Temperature affects not only the reliability but also the performance, power, and cost of the embedded system. However, traditional allocation and scheduling routines as the core of the hardware/software co-synthesis use performance or power as the design metric [40, 41, 42]. Therefore, it is very important to reduce hot spot temperature and have a thermal balanced design through a thermal-aware design methodology. In view of this, we propose a thermal-aware task allocation and scheduling algorithm for embedded systems. The algorithm is used as a sub-routine for hardware/software co-synthesis to reduce the peak temperature and achieve a thermally even distribution while meeting real time constraints. We investigate both power-aware and thermal-aware approaches.
to task allocation and scheduling.

In this chapter, we investigate both power-aware and thermal-aware approaches for task allocation and scheduling. Unlike the previous studies focusing on reducing the power consumption and/or the cost of entire system that can result in higher chip temperatures, our allocation and scheduling algorithm aims at reducing the hot spot temperature while achieving other design goals. The experimental results show that thermal-aware approach outperforms the power-aware schemes in terms of maximal and average temperature reductions. This work had been published in Design, Automation and Test in Europe, 2005 [69].

6.1 Problem Formulation

The traditional co-synthesis flow is shown in Figure 6.1. The co-synthesis framework takes the technology library and the task graph as inputs. Typically, a co-synthesis framework consists of three steps: the construction of target architecture with a number of PEs from technology library, the allocation of tasks on PEs, and the scheduling of execution sequence of tasks on each PE. The co-synthesis is an iterative process in which the underlying architecture is improved, until the performance constraint is met while system cost and/or power are minimized.

![Figure 6.1. The flow of the traditional co-synthesis framework.](image-url)
is a common application model to describe the specification of a real time periodic application. Figure 6.2(a) shows an example of the task graph. Each node in the figure represents a task while the directed edge connects two nodes. The edge is also referred to as the precedence of connected tasks of this link. The precedence usually accounts for the data or control dependencies between two nodes. That is, a task can not be executed unless its predecessors finish. There is a scalar associated with each edge describing the amount of data needed to be transferred between two connected tasks. The tasks come at a period and must finish before a deadline (real time constraint).

![Task Graph](image)

**Figure 6.2.** (a) An example of a task graph. (b) An example of a general target architecture.

The task graph model [49] is a common application model to describe the specification of a real time periodic application. Figure 6.2(a) shows an example of the task graph. Each node in the figure represents a task while the directed edge connects two nodes. The edge is also referred to as the precedence of connected tasks of this link. The precedence usually accounts for the data or control dependencies between two nodes. That is, a task can not be executed unless its predecessors finish. There is a scalar associated with each edge describing the amount of data needed to be transferred between two connected tasks. The tasks come at a period and must finish before a deadline (real time constraint).

### Table 6.1. A technology library example.

<table>
<thead>
<tr>
<th>Task</th>
<th>PE1(ASIC)</th>
<th>PE2(CPU)</th>
<th>PE3(ASIC)</th>
<th>PE4(CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WCET</td>
<td>WCPC</td>
<td>WCET</td>
<td>WCPC</td>
</tr>
<tr>
<td>T1</td>
<td>20</td>
<td>0.1</td>
<td>30</td>
<td>0.5</td>
</tr>
<tr>
<td>T2</td>
<td>–</td>
<td>–</td>
<td>150</td>
<td>2</td>
</tr>
<tr>
<td>T3</td>
<td>–</td>
<td>–</td>
<td>60</td>
<td>1.8</td>
</tr>
<tr>
<td>T4</td>
<td>–</td>
<td>–</td>
<td>22</td>
<td>1.1</td>
</tr>
</tbody>
</table>

The target library stores the worst case power consumptions (WCPC) and worst case execution times (WCET) for a task executed on different PEs. Table 6.1 shows an example of four tasks executed on four different PEs. Since some PEs are applications specific, they can only execute a certain types of tasks. For
example, tasks 2, 3 and 4 can never be placed on PE1, because it is an ASIC designed for T1.

The target architecture is normally a heterogeneous architecture which has a number of processing elements (PEs). The PE may be a CPU, DSP, ASIC, or FPGA depending on the need of performance, cost, and/or power constraints. An architecture example is shown in Figure 6.2(b). This architecture consists of two CPUs and two ASICs connected by a bus. Normally, there are two ways to construct an embedded system. One is the platform-based approach, and the other is the customized architecture generated by the HW/SW co-synthesis procedure. In the platform-based approach, the applications of the embedded systems are mapped to the pre-defined platform. In contrast, for customized architecture, the components of the architecture are iteratively composed by the co-synthesis process. Note that the customized architecture generated by the co-synthesis work varies from one application to another. For the platform-based architecture, there are various existing platforms. For example, the IBM PowerPC 440 platform architecture contains two CPU cores and several ASIC IP cores; a newly announced symmetric multiprocessing ARM [50] platform can support as many as four ARMv6-compliant CPU.

The problem tackled in this chapter is defined as: given a task graph and a technology library, the co-synthesis algorithm should produce a feasible task mapping and an embedded system architecture, which satisfies the deadline requirements of an application and reduces the hot spot and the average temperatures. The problem is the same for platform-based design, except that the target architecture is pre-defined.

### 6.2 Task Allocation and Scheduling

For either a platform-based or customized architecture, the task Allocation and Scheduling Procedures (ASP) is critical to produce good solutions. The selection of PEs and the assignment of tasks are both guided by ASP. Our task allocation and scheduling procedure is similar to the one proposed by Xie and Wolf [40]. Figure 6.3 outlines the allocation and scheduling procedure (ASP). The procedure takes the task graph and architecture (either a pre-defined platform architecture
or a customized architecture generated via co-synthesis) as input, and generates the task mapping and scheduling on the target architecture.

**Allocation and Scheduling**(task\_graph, architecture)

begin

*for each task, calculate its static criticality;*

*while the ready list of task is not empty;*

begin

*pick the first task in the ready list;*

*for each PE with respect to current task*

*calculate the dynamic criticality upon current PE;*

*schedule the current task with the maximal dynamic criticality;*

*update the ready list by adding current task’s successors to the ready list;*

end

end

**Figure 6.3.** Outline of the allocation and scheduling procedure (ASP).

The static criticality (SC) for each task is calculated as the maximum distance from current task to the end task in a task graph. This is similar to the priority ordering in some list schedulers. The dynamic criticality (DC) calculation is based on three different factors and is defined as follows:

\[
DC(task_i, PE_j) = SC(task_i) - WCET(task_i, PE_j) - \max(\text{avl.} PE_j, \text{ready task}_i)
\]

The first term stands for the static criticality of the task\(_i\); the second term retrieves the WCET of this task\(_i\) executed on PE\(_j\) from the technology library, and the third term takes the maximum of PE’s available time and the ready time of task\(_i\). The algorithm will prefer the pair of the task\(_i\) and the corresponding PE\(_j\) with the largest DC and schedule task\(_i\) on PE\(_j\) first.

The allocation and scheduling algorithm is effective on finding the task mapping and scheduling that satisfy the deadline requirement. However, it neglects the temperature impacts during the process and can potentially allocate tasks to PEs in such a way that some PEs have much higher temperature than other PEs due
to uneven power density. To account for this problem, we introduce power/energy aware ASP and thermal-aware ASP.

### 6.2.1 Power-aware allocation and scheduling

Since the level of temperature depends heavily on the power density, in power-aware allocation and scheduling, the power/energy factor is involved in the process of calculating dynamic criticality. The intuition is that we want to reduce and balance the power consumption for each PE when we allocate and schedule a task. Therefore, the DC equation is modified as follows:

$$DC(task_i, PE_j) = SC(task_i) - WCET(task_i, PE_j)$$

$$-\max(\text{avl.}_PE_j, \text{ready}\_task_i) - \text{Pow}$$

The last term (Pow) captures the effect of power/energy which can be interpreted by the following three heuristics.

**Heuristic 1**: Minimize power consumption of the current task.

This heuristic attempts to minimize the power consumption of the task to be allocated and scheduled. It makes use of the fact that a task can have different power consumptions on different processing elements. When we try to assign a task to one specific PE, we first examine its power consumption on this PE. This number contributes the last term in the DC calculation. For example, in Figure 6.4(a), when considering scheduling task G with the assumption of the other terms staying equal in the equation, it will be placed on PE1 because of the lowest power consumption.

**Heuristic 2**: Minimize cumulative average power of processing element.

This heuristic attempts to capture the historic cumulative average power consumption of each PE as far as current execution time. If one PE has been used frequently, it will have a higher power density, and thus have a higher chance to become a hot spot. The goal is to balance the average power consumption evenly among all available PEs. In Figure 6.4(b), the cumulative average power consump-
tions (including the power of task G) of PE1 are 0.8W, 0.76W for PE2, and 0.72W for PE3. The task G will be placed on a PE having the lowest cumulative average power, which is PE3.

**Heuristic 3**: Minimize energy of current task.

This heuristic tries to minimize the energy of the task to be allocated and scheduled on the system. It makes use of the fact that a task can have different energy consumptions on different processing elements. When we try to assign a task to one specific PE, we first examine its energy consumption on this PE. From the Figure 6.4(c), we can see that when the task G is placed on PE3, it will have the lowest energy value compared to those of the other two PEs.

![Figure 6.4](image)

**Figure 6.4.** Examples of three different power-aware heuristics.
6.2.2 Thermal-aware allocation and scheduling

Although the above three heuristics include the power/energy effect, they address the temperature issue indirectly. The proposed thermal-aware ASP addresses this issue by taking the temperature into consideration. The temperature of an embedded system depends on the power consumption of each processing element (PE), its dimension and relative location on the embedded system platform.

Skadron et al. [32] proposed a thermal modeling tool called HotSpot, which is easy to use and computationally efficient for modeling thermal effects at the block level. HotSpot provides a simple compact model, where the heat dissipation within each PE and the heat flow among PEs are accounted for. HotSpot takes a system floorplanning and the power consumption for each function block as input, and generates an accurate temperature estimation for each block. Thus, we can easily calculate each PE’s temperature by using HotSpot with the given architecture floorplanning of an embedded system and the power consumption of each PE.

It is the same as either the platform-based or customized architecture; the proposed thermal-aware scheduling is crucial to delivering thermally-balanced task assignments. For the thermal-aware ASP, we first pass the cumulating power consumptions of each PE along with the consuming power incurred by current scheduled tasks to the HotSpot. The temperatures returned from the HotSpot are averaged and then used in calculating dynamic criticality, which is defined below:

\[
DC(task_i, PE_j) = SC(task_i) - WCET(task_i, PE_j) - \max(\text{avl.}_PE_j, \text{ready TASK}_i) - Avg_Temp
\]

The flow of our thermal-aware co-synthesis framework is shown in Figure 6.5(a). Unlike the flow shown in Figure 6.1, the allocation and scheduling procedure executes and then activates the thermal-aware floorplanning [51] when considering placing a task on one specific PE. The HotSpot tool interacts with the floorplanning procedure to provide temperature information. The co-synthesis framework keeps executing until there is no improvement on current architecture configuration. For the platform-based thermal-aware design, the target architecture and the task graph are given, and the HotSpot is activated by the modified ASP with...
thermal inquires. This flow is depicted in Figure 6.5(b).

![Figure 6.5. The flow of the thermal-aware co-synthesis framework and the thermal-aware platform-based system design.](image)

### 6.3 Experimental Results

The co-synthesis work is modified to have the ability to handle both the pre-defined platform architecture and the newly proposed scheduling procedures. The algorithm is implemented in C++ and the experiments are conducted on a dual Intel Xeon processors (3.2GHz, 2GB RAM) machine running Linux. The benchmarks with relative power consumptions are generated by the modified TGFF tool [52] and are used to evaluate our approach. The runtime of our approach is time-efficient and is below 250 seconds for all of the benchmarks.

The first experiment we conduct compared the temperature differences from different power heuristics, when using the co-synthesis to decide the selection of PEs and when using the platform-based architecture. The architecture generated by the co-synthesis work is referred to as the customized architecture, which tries to reduce the system cost. The experimental results are shown in Table 6.2. The three columns under the co-synthesis are the results of the traditional co-synthesis
work, while the other three columns represent the results from the platform-based target architecture.

The very first row of a group of four rows indicates the characteristics of each benchmark and is the baseline case that does not take power into consideration. The following three rows represent three power heuristics. As can be seen from the table, when considering power only, the third power heuristic outperforms the other two heuristics and the baseline approach. This result indicates that minimizing the energy of a task executed on one specific PE achieves the best temperature result among all three heuristics. Thus, the third power heuristic will be used in the following experiments.

Note that Table 6.2 shows that platform-based architecture has lower hot spot temperatures than the customized architecture generated by co-synthesis. With the platform-based architecture, the tasks can be distributed among all PEs on the pre-defined platform. In contrast, the primary goal of co-synthesis is to minimize the cost (using as fewer PEs as possible), while meeting the performance deadline. Thus, the chance of having hot spot and un-even temperature distribution for platform-based design is lower. The customized architecture via co-synthesis tends to have higher power densities than the platform-based architecture, even though the number of PEs used in the architecture may be fewer than the pre-defined architecture.

The second experiment demonstrates the effectiveness of our thermal-aware approach in terms of lowering both the peak and the average temperatures. We take the best results of customized architecture and platform-based architecture from the first experiment for comparison. The power-aware and thermal-aware customized architecture comparison is shown in Table 6.3. As we mentioned in an earlier section, the peak temperature is reduced while trying to reach the lowest average temperature. From the results, the customized architecture with a thermal-aware approach demonstrates that it can effectively reduce the divergence between hot spot and average temperatures and lower both of them. The total average temperature reduction is 10.9°C and 6.95°C for the maximal and the average, respectively. This result indicates that observing the average temperature of all approaches using PEs while doing task scheduling, is beneficial to control the temperature of an embedded system.
<table>
<thead>
<tr>
<th>name/task/edge/</th>
<th>Co-synthesis</th>
<th>Platform-based arch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bm1/10/19/90</td>
<td>16.60</td>
<td>118.18</td>
</tr>
<tr>
<td>Heuristic1</td>
<td>16.14</td>
<td>121.70</td>
</tr>
<tr>
<td>Heuristic2</td>
<td>16.60</td>
<td>118.18</td>
</tr>
<tr>
<td>Heuristic3</td>
<td>15.56</td>
<td>113.29</td>
</tr>
<tr>
<td>Bm2/35/40/1500</td>
<td>29.47</td>
<td>121.44</td>
</tr>
<tr>
<td>Heuristic1</td>
<td>28.55</td>
<td>115.21</td>
</tr>
<tr>
<td>Heuristic2</td>
<td>29.47</td>
<td>121.44</td>
</tr>
<tr>
<td>Heuristic3</td>
<td>28.27</td>
<td>112.82</td>
</tr>
<tr>
<td>Bm3/34/39/1450</td>
<td>28.75</td>
<td>122.63</td>
</tr>
<tr>
<td>Heuristic1</td>
<td>28.60</td>
<td>121.57</td>
</tr>
<tr>
<td>Heuristic2</td>
<td>28.75</td>
<td>122.63</td>
</tr>
<tr>
<td>Heuristic3</td>
<td>27.25</td>
<td>114.22</td>
</tr>
<tr>
<td>Bm4/39/43/1650</td>
<td>28.84</td>
<td>113.58</td>
</tr>
<tr>
<td>Heuristic1</td>
<td>27.75</td>
<td>110.33</td>
</tr>
<tr>
<td>Heuristic2</td>
<td>29.35</td>
<td>110.49</td>
</tr>
<tr>
<td>Heuristic3</td>
<td>28.20</td>
<td>109.96</td>
</tr>
<tr>
<td>Bm5/51/60/2000</td>
<td>44.99</td>
<td>122.09</td>
</tr>
<tr>
<td>Heuristic1</td>
<td>46.99</td>
<td>122.28</td>
</tr>
<tr>
<td>Heuristic2</td>
<td>44.99</td>
<td>117.86</td>
</tr>
<tr>
<td>Heuristic3</td>
<td>43.34</td>
<td>118.68</td>
</tr>
<tr>
<td>Bm6/54/63/2800</td>
<td>40.49</td>
<td>121.47</td>
</tr>
<tr>
<td>Heuristic1</td>
<td>38.54</td>
<td>119.44</td>
</tr>
<tr>
<td>Heuristic2</td>
<td>39.59</td>
<td>114.52</td>
</tr>
<tr>
<td>Heuristic3</td>
<td>38.16</td>
<td>110.86</td>
</tr>
</tbody>
</table>

Table 6.2. The comparisons of different power heuristics under co-synthesis architecture and platform-based target architecture.

As for the platform-based architecture, the proposed thermal-aware approach outperforms the power-aware approach in both temperature attempts. From the results in Table 4, we can see that, under thermal-aware approach, both the maximal and average temperatures are lower than that of the corresponding power-aware approach, approximately by 9.75°C and 5.02°C, respectively.

The results from Tables 6.3 and 6.4 show that, with the platform-based architecture, the thermal ASP can balance the workloads of all PEs, and thus can deliver a lower peak and average temperature task mapping than temperatures in customized architecture.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bm1</td>
<td>15.56</td>
<td>113.29</td>
<td>104.49</td>
<td>12.48</td>
<td>87.11</td>
<td>86.13</td>
</tr>
<tr>
<td>Bm2</td>
<td>28.27</td>
<td>112.82</td>
<td>105.42</td>
<td>24.64</td>
<td>106.38</td>
<td>99.84</td>
</tr>
<tr>
<td>Bm3</td>
<td>27.25</td>
<td>114.22</td>
<td>107.39</td>
<td>27.19</td>
<td>108.00</td>
<td>103.20</td>
</tr>
<tr>
<td>Bm4</td>
<td>28.20</td>
<td>109.96</td>
<td>100.15</td>
<td>26.51</td>
<td>102.08</td>
<td>96.28</td>
</tr>
<tr>
<td>Bm5</td>
<td>43.34</td>
<td>118.68</td>
<td>109.87</td>
<td>42.41</td>
<td>106.32</td>
<td>102.48</td>
</tr>
<tr>
<td>Bm6</td>
<td>38.16</td>
<td>110.86</td>
<td>101.05</td>
<td>35.74</td>
<td>104.39</td>
<td>98.72</td>
</tr>
</tbody>
</table>

Table 6.3. The temperature comparisons of the power-aware and the thermal-aware approaches on co-synthesis architecture.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bm1</td>
<td>10.40</td>
<td>85.88</td>
<td>75.58</td>
<td>6.37</td>
<td>65.71</td>
<td>61.16</td>
</tr>
<tr>
<td>Bm2</td>
<td>24.09</td>
<td>106.63</td>
<td>97.40</td>
<td>22.37</td>
<td>96.33</td>
<td>93.47</td>
</tr>
<tr>
<td>Bm3</td>
<td>24.09</td>
<td>110.65</td>
<td>99.95</td>
<td>22.95</td>
<td>96.98</td>
<td>93.63</td>
</tr>
<tr>
<td>Bm4</td>
<td>25.20</td>
<td>103.95</td>
<td>94.69</td>
<td>24.98</td>
<td>103.03</td>
<td>94.59</td>
</tr>
<tr>
<td>Bm5</td>
<td>41.64</td>
<td>100.42</td>
<td>89.24</td>
<td>38.54</td>
<td>94.85</td>
<td>85.76</td>
</tr>
<tr>
<td>Bm6</td>
<td>35.48</td>
<td>108.57</td>
<td>97.73</td>
<td>33.11</td>
<td>100.66</td>
<td>95.82</td>
</tr>
</tbody>
</table>

Table 6.4. The temperature comparisons of the power-aware and the thermal-aware approaches on platform-based architecture.
Aggressive scaling of process technologies has enabled designers to pack more functionality onto a single die. The higher levels of integration due to scaling, along with the advent of highly complex re-usable IP (Intellectual Property) blocks, has spurred an increase in core-based SoC (System-on-Chip) design techniques. However, the increased level of integration within a single die imposes rigid constraints on the power consumption budget. Among various approaches proposed to reduce the power consumption, the use of multiple voltage islands [53] is one of the most attractive approaches for core-based SOC designs.

The use of multiple voltage islands exploits the concept of using lower supply voltages for parts of the design that are not in the critical path to reduce both dynamic and leakage power. In order to alleviate the cost of supplying multiple voltages to different parts of the chip and the cost of level conversion when communicating across different voltage levels, the voltage island approach clusters a group of cores operating at the same voltage and provides one single voltage level for all modules inside this island.

Even though the voltage island technique can help mitigate power problems, accordingly, it complicates the chip design process in terms of power routing, floor-planning, and timing closure with the additional overheads of level converters with
respect to area and delay. Thus, how to group effectively the compatible cores together with the same supply voltage without disturbing other design metrics, such as wire length and critical path timing, is a crucial issue.

Figure 7.1 shows an example of a SoC design with two voltage islands. Each module has a list of operating voltages from which it can choose. For example, module \( b_6 \) can operate at 1.2, 1.3 or 1.4 volts. The chip level supply voltage is 1.4 volt, so there is no need of level converter for modules \( b_1, b_2, \) and \( b_3 \). Level converters, however, are needed for islands A and B in order to communicate with components in other islands or cores operating at chip-level supply voltage. Typically, the modules may be soft or hard, free to rotate, and also there may exist some constraints associated with the SoC design, such as boundary constraints, range constraints, and/or performance constraints.

To minimize the total power consumption of a voltage island-based SoC, the most intuitive way is to assign each core with the lowest voltage from its supply voltage list. However, this may not be feasible in practice. For example, in Figure 7.1, if we form the voltage island by combining module \( b_3 \) and \( b_9 \) and assigning both of them to a single voltage island, it will result in large waste of dead space and may increase the wire length during the floorplanning process. Moreover, there will be a total of four voltage islands in this configuration, resulting in the need of larger numbers of level converters compared to the partition in Figure 7.1, which
has only two voltage islands. Furthermore, it is possible that some timing critical cores might have timing violations when operated with certain voltage supplies.

Based on the facts above, the additional imposed constraints make the thermal-aware voltage island partitioning and floorplanning a unique and interesting problem. For this work, we present a hybrid optimization approach which targets peak temperature reduction and elimination of hot spots. We demonstrate that by carefully partitioning the cores, assigning voltage levels, and allocating the islands, the thermal distribution of the design can be improved. This work had been published in International Conference on Computer Design, 2005 [68].

7.1 Problem Formulation

We assume that the SoC design contains a set of IP modules. Let $B = b_1, b_2, ..., b_m$ be the set of rectangular IP modules embedded on the SoC, each of which has width $w_i$, height $h_i$, and area $a_i$ associated with it. Different IP modules might have different supply voltage levels, so the related supporting supply voltages and their corresponding average power consumptions are also provided. A voltage island partition and floorplan is mapped for each block $b_i$ into distinct voltage islands, such that no IP modules fall into two islands. The goal of thermal-aware voltage island partitioning and floorplanning is to minimize both peak and average temperatures across the SoC system and other design metrics, such as area, wire length, and power budget, all of which can also be imposed in the evaluation process.

7.2 Voltage Island Partitioning and Floorplanning Algorithm

The voltage island partition and floorplan framework is composed of two parts: a genetic algorithm (GA) based voltage island partitioning algorithm and a simulated annealing (SA) based floorplanning algorithm. GAs [54] are a class of search and optimization methods that mimic the evolutionary principles in natural selection and have been used actively in recent VLSI optimization problems. In GAs,
Algorithm

Begin

Given modules’ information and the power values;
GA is applied to generate the initial population;

While (the termination criteria is not met)
  For each chromosome in the population
    For each voltage island encoded in chromosome
      *Use SA to floorplan the current voltage
        island partition;
  EndFor
EndFor

* *Use SA to floorplan all voltage islands at the
chip level;

GA fitness function evaluation;
Apply three operators (selection, crossover, and
mutation) to produce new chromosomes;

EndWhile

End

Figure 7.2. Outline of the voltage island partitioning and floorplanning algorithm.

the voltage partition solution is encoded into an integer string called a chromosome. Instead of working with a single solution, as in the simulated annealing algorithm, the GA search process begins with a random set of chromosomes called initial population. Each chromosome is assigned a fitness score that is directly related to the objective function of the optimization problem. The population of chromosomes is modified to a new generation by applying three operators similar to natural selection operators – selection, crossover and mutation.

As shown in Figure 7.2, the initial modules’ information and their relative power values are given as the inputs to the proposed algorithm. The GA first generates a population of random voltage partitions and then each valid voltage partition contained in the chromosome is evaluated. That is, each voltage island involves another floorplanning step. The simulated annealing based floorplanner is activated to iteratively improve the quality of the floorplanning solution, which is done by the solution perturbation. The cost function used by an SA-based floorplanner can be area minimization, balancing the thermal profile of a chip, or both. More details about balancing the thermal profiles will be given in the
following section. For the goal of area minimization, it will not only help reduce the dead space, but also implicitly reduce the wire length of connections between modules due to the resulting tight area compaction. After floorplanning each voltage island, a chip-level floorplanning is performed with a weighted cost of the optimized objectives and their constraints. The cost function can be written as:

\[
\text{cost} = \alpha \cdot C_{\text{area}} + (1 - \alpha) \cdot C_{\text{wire}} + \beta \cdot C_{\text{temp}}
\]  
(7.1)

where \( C_{\text{area}} \) and \( C_{\text{wire}} \) represent the chip area and wire length, while \( C_{\text{temp}} \) indicates the chip temperature. The fitness function is thus assigned to be the reciprocal of this cost function (i.e. fitness = \( 1/\text{cost} \)). Other factors in the design process can be easily incorporated into our approach by adding more design variables and adjusting the relative weights. Note that the voltage partition should always be valid and compatible, which is taken care of in GA before we pass the modules’ information to the SA-based floorplanner. After assigning the fitness value to the current island partitions represented by a chromosome, the new chromosomes are those which will likely have higher fitness than the old ones. That is, the voltage island partitions will have lower area and total wire length. Thus, this process is oriented toward the optimal solution.

In order to generate a new chromosome (voltage island partition), three operations are applied and are listed in the following:

1. Selection, which is done by stochastic roulette-wheel approach;
2. Crossover, which is done by single point crossover and is illustrated in Figure 7.3;
3. Mutation, which is accomplished by the swap operation.

Usually, the crossover rate is higher than the mutation rate, so that evolution will not become a random search algorithm.

### 7.3 Slicing Tree Floorplan Model

The slicing tree structure [58] is used as the floorplan model and is incorporated with the simulated annealing algorithm [55]. Note that other floorplanners can be adopted in our approach. The four operations used in perturbing a slicing floorplan are listed below:
Figure 7.3. Example of single point crossover in voltage island partitioning.

(1) Rotation, which rotates a module;
(2) Leaf node swap, which swaps two modules;
(3) Flip, which flips a slicing operator;
(4) Sub-tree swap, which swaps two sub-trees in a slicing tree.

A slicing floorplan is a rectangular area that is sliced recursively by a horizontal or vertical slicing line (operator) into a set of rectangular regions to accommodate a set of functional modules. A slicing floorplan can also be represented by Polish expression. Figure 7.4 shows an example of a slicing tree, the polish expression, and its corresponding floorplan. The slicing tree is a binary tree constructed in a bottom-up fashion in which each internal node in the binary tree contains the size and coordinate of the enclosing rectangle and those of two children. This tree structure gives the advantage of fast extracting the coordinates of the dead spaces, used in the calculation of the chip temperature profile.

The procedure of dead space coordinate calculation is as follows: After constructing the slicing tree, the internal node contains the width and height of the enclosing rectangle where its left and right children are two modules combined together, either by a vertical or horizontal slicing operator. For example, modules 2 and 3 in Figure 7.4 are combined first by a vertical slicing operator and then this combined one is horizontally sliced to module 1. Since the width of module 1 is smaller than that of the combined one, an unavoidable dead space is generated. The bottom-left coordinate of this dead space (marked by dark color) is obtained by comparing the width of module 1 and that of combined one; thus, coordinate x is 8, while y coordinate is 12. To ensure accurate thermal modeling, these dead
spaces are passed as inputs to the thermal model, allowing heat to flow across and through them as it would in an actual chip.

### 7.3.1 Thermal-aware floorplanning

The traditional simulated annealing based floorplanner only considers area and wire length minimizations (i.e. only considers the first two terms in Equation 7.1). If two hottest blocks are adjacent to each other, the temperature is higher than when they are placed far away from each other. In order to take into account the temperature impact in the floorplanning process, a thermal-aware SA floorplanner should use the same cost function as Equation 7.1.

Our thermal-aware floorplanning can be conducted using two different approaches: the first one is called the chip-level approach (which only takes temperature into consideration at chip level), and the second one is called the two-level approach (which considers temperature at both chip level and voltage island level). The main objective of both approaches is to distribute temperatures across a chip evenly and minimize the hot spot temperatures by using the thermal-aware floorplanning.

The chip-level approach uses a traditional SA floorplanner (minimizing area and wire length) in the voltage island level floorplanning (line * of Figure 7.2). The thermal optimization is performed at chip-level, where the voltage islands are placed simultaneously using a thermal-aware SA floorplanner (line ** of Figure 7.2).

The two-level approach performs thermal optimization at both chip level and
voltage island level. In this approach, both voltage island level floorplanning (line * of Figure 7.2) and chip level floorplanning (line ** of Figure 7.2) are performed using a thermal-aware SA floorplanner. Compared to the chip-level approach, in which the voltage island level floorplanning is guided by traditional area and wire length metrics, the two-level approach incurs longer run-time, because it needs to call temperature estimation tool at the voltage island floorplanning level, while the chip-level approach only invokes temperature estimation at chip-level floorplanning. However, the two-level approach may achieve better thermal reduction, because temperature is considered at a finer granularity. Since an enormous number of configurations will be evaluated during the thermal aware SA-based floorplanning process, the fast retrieval of temperature profiles is necessary, which is the reason why we speed up the HotSpot computation.

7.4 Experimental Results

We implemented the proposed thermal-aware voltage island partitioning and floorplanning algorithm in C++. The thermal model and tool are based on the enhanced HotSpot source code. The experiment was run on a dual Intel Xeon (3.2 GHz, 2GB RAM) machine running Linux. We applied our algorithm to conduct experiments on a set of MCNC benchmarks. For voltage island setting, there are a total of five voltage levels available, which range from 1.0V to 1.4V, with the chip-level supply voltage assigned as 1.4V. For instance, $b_2$ can be operated at 1.1V, 1.2V, and 1.4V, while $b_4$ can be operated at 1.3V and 1.4V. We also assume the power consumption of the level converter is negligible. Although the number of voltage islands can be constrained in our framework, we allowed the algorithm handle the voltage level assignment in an attempt to achieve overall temperature minimization. We use an approach similar to the one in [82] to assign the average power density for each IP module in the range of $2.2 \times 10^4$ (W/m$^2$) and $2.4 \times 10^6$ (W/m$^2$). The net length is estimated by using the general half-perimeter bounding box model.
7.4.1 Experiment on chip-level

Table 7.1 summarizes the experiments on three different approaches: area+wire (AW), power+area+wire (PAW), and temperature+area+wire (TAW) optimizations. Time is reported in minutes. The waste area columns below each optimization indicate the ratio of total dead space to the cell area. As can be seen from the table, by including the thermal effect the traditional design metrics, area and wire length, the peak and average temperatures are effectively reduced. The average maximal and average temperature reductions from applying TAW, compared with AW, can be as much as 26°C and 9°C, respectively, while with PAW are 11°C and 1.9°C. The PAW approach used more voltage islands than the other two approaches in seeking a low power solution. Although power is also an important factor affecting the temperature distribution, without the physical location information of each module, the PAW optimization can not capture the thermal coupling effects as accurately as its counterpart, TAW optimization. Figure 7.5 shows the floorplanning result of ami49 partitioned into three voltage islands.

![Figure 7.5. Partitioning and floorplanning result of ami49 under TAW optimization.](image)

7.4.2 Experiment on two-level thermal-aware optimization

Table 7.2 shows the result of experiments when considering thermal effect in both chip level and island level. Comparing the best results in Table 7.1, the two-level approach can reduce the peak and average temperatures further by as much as
5°C and 6°C, respectively. The floorplanning with the thermal profile inside the voltage island is helpful in separating the hot IP modules away from one another.

Figures 7.6 and 7.7 show the distributions of temperatures across a chip. From Figure 7.7, we see the number of IP modules with temperatures ranges of 80~85°C and 85~90°C are reduced compared to those in Figure 7.6, while the number of IP modules in the range of 75~80°C increases. The peak temperature also dropped from 96°C to 92°C. This re-distribution of temperatures is beneficial in reducing both the peak and average temperatures with the reason of decreasing effects of hot core coupling. However, the area and wire length also increase accordingly, albeit slightly, due to the finer level of thermal-aware floorplanning. Notably, the runtime penalty for including thermal optimization is small, with an average performance penalty of only 17 percent over the benchmarks tested. Even for larger benchmarks, such as ami49, obtaining temperature values requires only 10ms. Without the optimized thermal tool, the performance penalty is 107%. As such, the inclusion of temperature optimization at the floorplanning stage is a low-overhead means of

### Table 7.1. The comparison of temperatures under different design metrics

<table>
<thead>
<tr>
<th>Circuit</th>
<th># cells</th>
<th>Cell area</th>
<th>maxT</th>
<th>avgT</th>
<th>wire</th>
<th>area</th>
<th>area vi</th>
<th># vi</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>9</td>
<td>45.56</td>
<td>110.46</td>
<td>71.79</td>
<td>499</td>
<td>4.1%</td>
<td>3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>xerox</td>
<td>10</td>
<td>19.35</td>
<td>116.67</td>
<td>84.68</td>
<td>450</td>
<td>9.1%</td>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>hp</td>
<td>11</td>
<td>8.83</td>
<td>139.80</td>
<td>113.74</td>
<td>168</td>
<td>8.9%</td>
<td>3</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ami33</td>
<td>33</td>
<td>1.15</td>
<td>112.62</td>
<td>111.10</td>
<td>87</td>
<td>13.8%</td>
<td>4</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>ami49</td>
<td>49</td>
<td>35.44</td>
<td>113.02</td>
<td>87.79</td>
<td>1544</td>
<td>16.6%</td>
<td>4</td>
<td>49</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit</th>
<th># cells</th>
<th>Cell area</th>
<th>maxT</th>
<th>avgT</th>
<th>wire</th>
<th>area</th>
<th>area vi</th>
<th># vi</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>9</td>
<td>45.56</td>
<td>87.95</td>
<td>68.18</td>
<td>500</td>
<td>10.5%</td>
<td>3</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>xerox</td>
<td>10</td>
<td>19.35</td>
<td>97.09</td>
<td>79.01</td>
<td>511</td>
<td>12.7%</td>
<td>3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>hp</td>
<td>11</td>
<td>8.83</td>
<td>118.43</td>
<td>97.81</td>
<td>170</td>
<td>15.3%</td>
<td>4</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>ami33</td>
<td>33</td>
<td>1.15</td>
<td>103.37</td>
<td>102.02</td>
<td>87</td>
<td>17.3%</td>
<td>4</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>ami49</td>
<td>49</td>
<td>35.44</td>
<td>112.46</td>
<td>85.12</td>
<td>1606</td>
<td>24.2%</td>
<td>4</td>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit</th>
<th># cells</th>
<th>Cell area</th>
<th>maxT</th>
<th>avgT</th>
<th>wire</th>
<th>area</th>
<th>area vi</th>
<th># vi</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>9</td>
<td>45.56</td>
<td>80.67</td>
<td>66.93</td>
<td>580</td>
<td>4.6%</td>
<td>3</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>xerox</td>
<td>10</td>
<td>19.35</td>
<td>86.65</td>
<td>78.47</td>
<td>610</td>
<td>14.7%</td>
<td>2</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>hp</td>
<td>11</td>
<td>8.83</td>
<td>98.77</td>
<td>95.41</td>
<td>203</td>
<td>15.1%</td>
<td>3</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>ami33</td>
<td>33</td>
<td>1.15</td>
<td>99.58</td>
<td>97.50</td>
<td>95</td>
<td>19.3%</td>
<td>4</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>ami49</td>
<td>49</td>
<td>35.44</td>
<td>96.03</td>
<td>84.50</td>
<td>1569</td>
<td>19.1%</td>
<td>3</td>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

MCNC Benchmarks
Area and wire
(AW)

Power-aware with area and wire
(PAW)

Thermal-aware with area and wire
(TAW)
improving resulting designs.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>maxT</th>
<th>avgT</th>
<th>wire</th>
<th>waste area</th>
<th># vi</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>apte</td>
<td>77.29</td>
<td>60.97</td>
<td>803</td>
<td>5%</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>xerox</td>
<td>81.49</td>
<td>75.17</td>
<td>874</td>
<td>14.7%</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>hp</td>
<td>97.79</td>
<td>93.59</td>
<td>274</td>
<td>19.7%</td>
<td>4</td>
<td>29</td>
</tr>
<tr>
<td>ami33</td>
<td>97.01</td>
<td>95.21</td>
<td>101</td>
<td>22.1%</td>
<td>4</td>
<td>46</td>
</tr>
<tr>
<td>ami49</td>
<td>91.57</td>
<td>82.18</td>
<td>2112</td>
<td>20.3%</td>
<td>3</td>
<td>59</td>
</tr>
</tbody>
</table>

Table 7.2. Results of the two-level thermal-aware approach

Figure 7.6. On-chip temperature distribution for ami49 with TAW optimization
Figure 7.7. On-chip temperature distribution for ami49 with two-level optimization
Chapter 8

Thermal-Aware Virtualization and Placement for 2D/3D Networks-on-Chip Architecture

Network-on-chip architecture has been proposed as a potential solution for the interconnect demands that arise with the nanometer era [95]. In the network-on-chip architecture, a general purpose on-chip interconnection network replaces the traditional design-specific global on-chip wiring, by the use of switching fabric or routers to connect IP cores or processing elements (PEs). The PEs communicate with one another by sending messages in packets through the routers. This is usually called packet-based interconnect. An example implementation is shown in Figure 8.1. The diagram on the left illustrates a tiled single PE per node architecture arranged in a 2-D mesh network, whereas the diagram on the right illustrates the underlying interconnect, in which each router communicates to the PE via a local port, and to its neighboring routers via four global ports.

A particular property of Network-on-chip architecture is the concept of hardware virtualization, which maps one or more logical processing units onto a single PE, thus allowing the PE to virtually perform the computation of one or more (depending on the degree of virtualization) logical processing units. This method requires the presence of additional logic and memory as part of the PE hardware in order to identify the particular computation that is performed at any given time, and to read/write the required data per computation. It is possible that
computations on virtualized PEs happen out of order; as a result, synchronization mechanisms are implemented within the PEs in order to allow for out of order computation, as well as to synchronize the completed computations with the rest of the PEs. This contributes to a larger overall PE, which consequently consumes more power. However, it also increases the amount of logical units that can be mapped on the chip, which, depending on the computation type, can result in computation performance gain [94].

After the hardware virtualization (i.e., mapping of the computation onto a single PE), the energy consumption and the computation time for each PE is fixed. However, the communication delay between PEs depends on the PE locations. For example, a message from PE1 has to go through at least 7 routers to reach PE16 in Figure 8.1, which means it needs 6 hops. On the other hand, the communication energy consumption depends on the location of the PE as well as the message volume sent through routers and the distance between PEs. Consequently, the performance and the overall energy consumption are determined by the IP placement. If two PEs exchange several packets, it is better to place them closer together to reduce both the communication energy and latency.

As the technology scales, the temperature in modern high-performance VLSI circuit increases dramatically due to smaller feature size, higher packing density, and rising power consumption. The hot spot in a modern chip might have a tem-
perature of more than 100°C, while the intra-chip temperature differentials can be larger than 10~20°C. Temperature can have dramatic impacts on circuit behavior. For example, interconnect (Elmore) delay increases approximately 5% for every 10°C increase, and the leakage current increases exponentially with the temperature increase. Therefore, it is very important to reduce or eliminate hot spots and have a thermally balanced design. For NoC architecture, the thermal distribution profile of a design is largely determined by the PE locations; consequently, the IP placement algorithm is the key to achieving the thermal balance design goals.

In this chapter, we present a thermal aware IP virtualization and placement algorithm based on genetic algorithm. We demonstrate that a careful IP virtualization and placement can reduce the hot spot temperature and provide a thermally balanced design. This work had been published in International Conference on Computer Design, 2004 [65].

8.1 IP Virtualization and Placement Framework

The proposed NoC mapping optimization flow uses genetic algorithms as described in Chapter 3. The IP virtualization and placement information is encoded into integer strings called chromosomes. The optimization flow begins with a randomly generated initial population, which consists of many randomly generated IP placements. The optimization flow is an iterative procedure. The chromosomes with better fitness will survive at each generation and are operated on with three different operations (reproduction, crossover, and mutation) to form a new set of chromosomes, each of which represents a new IP virtualization and placement. The iteration continues until the termination criterion is met.

8.1.1 Chromosome encoding

One example of the chromosome encoding is shown in Figure 8.2. It contains 16 unique integers, which represent the 16 IP cores (physical PEs). The position of each integer indicates its placement location. For example, chromosome A in Figure 8.2 represents the placement as shown in Figure 8.1, while chromosome B represents the placement where the PE 1 and PE 6 in Figure 8.2 are swapped.
With virtualization, logical processing units 22, 73, 19, and 31 are clustered into physical PE 4, while logical processing units 44, 37, 56, and 85 are grouped together in physical PE 11 in chromosome A. With this representation, the optimizations of mapping and virtualization can be done simultaneously.

\[
\text{chromosome A} \\
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
22 & 73 & 19 & 31 \\
44 & 37 & 56 & 85
\end{array}
\]

\[
\text{chromosome B} \\
\begin{array}{cccccccccc}
6 & 2 & 3 & 4 & 5 & 1 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16
\end{array}
\]

Figure 8.2. Chromosome encoding for IP placement

8.1.2 Fitness function

The fitness function, which decides the survival chance for a specific chromosome, is related to the mapping goals. Depending on the optimization goal, the fitness function of the genetic algorithm is different.

8.1.2.1 Thermal balanced design

The goal of thermal placement is to distribute temperature evenly across a chip and to minimize the hot spot temperature. We use an approach similar to that proposed by Chu et al. [81]. Their work tried to solve the problem of thermal placement for gate arrays. Here, we model our NoC architecture as an \( m \times n \) matrix with the given temperatures, such that the maximum sum among all \( t \times t \) submatrices is minimized. The number assigned to each cell in the matrix is a non-negative temperature value. The parameter \( t \) is used to account for the heat transfer ability. Increasing \( t \) means that the heat transfer is good, so the number of affected cells near the heat source cell will be larger. For any matrix \( M \); let \( St(M) \) be the set of all \( t \times t \) submatrices, the fitness of a solution can be defined as follows:

\[
\text{Fitness} = \frac{1}{\text{Max}(S_t)} \quad (8.1)
\]
8.1.2.2 Power balanced design

This optimization strategy is to achieve a balanced power distribution on the chip. Basically, we use the same approach to calculate the fitness of a chromosome here as in the thermal balanced design. The only difference is the numbers in the matrix. For a power-balanced design, we provide the power consumption of each IP and its corresponding router. The power consumption for the IP is fixed when the hardware virtualization is done. However, the router power consumption depends on the communication links. The temperature in one IP location will not be equal to that of another IP location, since the heat can flow to the adjacent IPs. But power consumption of each IP will remain the same, even when placed in different physical locations. Thus, we can observe the different temperature results from power and thermal balanced designs.

8.1.2.3 Communication cost minimization design

The communication cost is given by:

\[ Commcost = \sum_{i=1}^{|E|} vol(d^i) \times dist(source(d^i), dest(d^i)) \] (8.2)

where the \( d_i \) is any communication between two IPs (where the \( source(d_i) \) is the source IP core and the \( dest(d_i) \) is the destination IP core) and \( vol(d_i) \) is the message volume that has to flow between these two IP cores. The \( dist \) is the number of hops that the messages have to go through. To minimize the communication cost, the fitness function of our algorithm is given by:

\[ Fitness = \frac{1}{Commcost} \] (8.3)

8.1.3 Crossover operator and mutation operator

Due to the nature of genetic algorithms, the number in a chromosome will be generated randomly. For our placement problem, we map each physical IP as a non-negative unique number as the encoding method. When doing a crossover operation, the offspring’s chromosome is generated from the mating parents’ chromosomes. At this stage, there are possibilities that some numbers are redundant.
It is important to guarantee that each number should exist only once in a chromosome to make the evaluation proceed. As for the mutation operation, we have two different operators to explore more solution spaces. One is the mutation by swapping and another is mutation by shifting. Either operator can be used with a random probability.

8.2 Case Study on LDPC and Experimental Results

In this section, we present a case study of implementing a Low Density Parity Check (LDPC) decoder on networks-on-chip architecture and evaluate our algorithm using this real application.

8.2.1 A brief introduction on LDPC

Low Density Parity Check (LDPC) codes are a form of iterative error correction codes similar to Turbo codes that can achieve near Shannon-limit communication channel capacity [97, 102]. They offer excellent decoding performance and good block error performance. The most notable advantage of LDPC codes is their suitability for parallel hardware implementation. An LDPC code is a linear message encoding technique defined by a set of two very sparse parity check matrices, G and H. The message to be sent is encoded using the G matrix. When it reaches its destination, it is decoded using the H-matrix. The LDPC decoding algorithm consists of a series of intensive computations derived from a message-passing iterative bipartite graph, as shown in Figure 8.3. The bipartite graph consists of two types of nodes: the bit node and the check node [97]. Connections between the two nodes in the bipartite graph depend on the row and column weight of the H-Matrix, where the weight is the number of 1-entries in the row/column. Columns represent the number of bit nodes and rows represent the number of check nodes. The 1 in the $ij_{th}$ entry of the H-Matrix represents an edge between the $i_{th}$ check and the $j_{th}$ bit nodes as shown in Figure 8.3.

Message passing iterations are performed by the two computation units – the bit node and the check node [102]. Each type of node interacts with a number
of other nodes, all of the opposite type, to decode a word. The number of nodes involved in the computation depends on the desired block size.

The H-matrix is usually sparse and needs to be large, in order to decode large blocks of data; this consequently implies a relatively large amount of edges in the bipartite graph \[102\]. Hence, the two major challenges identified when designing LDPC decoders are the interconnect structure between the nodes and the amount of memory required for computation as well as configuration purposes per node \[105\].

![Parity check matrix, H](image)

**Figure 8.3.** Derivation of the Bipartite Graph from the H-Matrix.

For the reasons explained, we therefore use the LDPC decoder as an example to evaluate our mapping algorithms for NoC architecture. The underlying interconnections between the LDPC nodes are implemented as an NoC architecture. We explore the mapping of the check and bit nodes on the NoC architecture in such a way as to obtain an even temperature distribution and reduce the overall communication on-chip. It has been shown \[101\] that, while architectural modifications in the PE can reduce the overall power consumption, most of the chip power is consumed in the communication links and the routers which are constantly active. In addition, the LDPC nodes consume different amounts of power because of the variations in the number of connections per node \[102\]; hence, there is a thermal distribution problem. As a result, an emphasis is also placed on reducing both the number of hops (defined as a message transmission between two on-chip routers) as well as the number of messages transmitted overall. The mapping and
placement problems are magnified by the hardware constraints in terms of area, however. Initially we have the H-matrix, which provides the total number of nodes as well as the connections between them and their degrees (# of inputs). Based on the area constraints, we then have a limited number of processing units that we can place on the chip. The problem therefore becomes one of mapping the pool of computation nodes given by the H-Matrix into the limited area we have on the chip. The overall computation can be enhanced by hardware virtualization; this is again a technique that thrives from the NoC architecture. Using hardware virtualization, we can have a physical PE function as two or more logical PEs (depending on the virtualization factor). The tradeoff comes in both the PE area and power consumption, which increase by a small factor to incorporate the extra memory and logic required for node identification.

An optimal mapping of the LDPC nodes into the physical PEs of the NoC architecture provides potential reduction both in the communication (# of hops) between the PEs as well as the number of messages transmitted. The proposed NoC architecture provides a fast and reliable underlying structure, allowing the bit and check nodes to communicate with one another effectively.

### 8.2.2 Design methodology

In order to obtain the power models for each PE, we used the following tool flow. Firstly, the LDPC Software \[102\] was used in order to generate both the H-Matrix and the encoded messages. We used three types of LDPC codes: a (7, 4) Hamming code, a (2000, 1000) LDPC code with 3 checks per bit and 6 bits per check, and a (10000, 5000) LDPC code with 3 checks per bit and 6 bits per check. For the (7, 4) Hamming code, we transmitted the message using a Binary Symmetric Channel (with error probability 5%) and for all three codes, we transmitted the messages using Additive White Gaussian Noise channels, with noise standard deviation of 5%. Once the bipartite graph was obtained, we then used NOCSim \[102\] to set up either a 4x4 or 5x5 2-D mesh network, with different physical bit nodes and check nodes and an I/O communication-oriented node.

NOCSim sets up a predefined network of PEs and routers, and in addition, provides mechanisms to handle virtualized nodes. NOCSim takes as input the Net-
work topology defined by the H-Matrix and the encoded message to be decoded. It then simulates the Network, generating real network traffic by packetizing data and headers and simulating cycle-accurate data transmission between routers and PEs. NOCSim generates the entire network traffic between physical nodes, taking into consideration potential mapping of more than one virtual node on a physical PE (virtualization). NOCSim outputs include the number of messages from PE to PE and the routing path they follow in the form of the real binary data that travels across the network. In parallel with NOCSim simulations, we created and synthesized the application in commercial 160 nm technology using Synopsys Design Compiler, Verilog models of each PE as well as the entire underlying NoC architecture.

These NOCSim outputs were then used as our test vectors for the synthesized models of the physical PEs, and Synopsys Power Compiler was used to give the power models of the individual PEs. The operating clock frequency was at 500MHz with a Vdd of 1.8V. Figure 8.4 shows the overall modeling methodology.

Figure 8.4. The overall design methodology for NoC LDPC power estimation.
In the genetic algorithm, we use an intelligent crossover mechanism which guarantees that bit nodes are always swapped with bit nodes only and similarly for check nodes. Also, in the case of virtualization, a logical PE node is decoupled from its physical PE mapping in the previous generation during crossover and mutation, i.e. the physical to logical mapping is not static.

8.2.3 Experimental results

First we implement a 10x10 NoC architecture with 100 PEs on the chip to choose the window parameter value $t$. Each PE has a size of 1 mm x 0.8 mm in a commercial 160nm standard cell library. Based on the physical layout, we use HotSpot to obtain the transfer thermal resistance matrix and to estimate the temperature for each IP block, as described in Chapter 2.

Figures 8.5 and Figure 8.6 show the peak and average temperatures, respectively, by using two different windows (a 1x1 window and a 2x2 window, as defined in section 8.1.2). From the figures we can see that the 1x1 window performs better in both, reducing peak and average temperatures. For the following experiments, we use $t=1$ to obtain better results. The IP virtualization and mapping algorithm is implemented in C and the experiment is done on an Intel Pentium 4 processor (2.8 GHz and 512MB RAM) running Linux. The runtime is about 15∼18 minutes for 3000 generations in 10x10 NoC examples.

![Peak Temperature Comparison (10x10)](image_url)

Figure 8.5. Peak temperature comparison of 1x1 and 2x2 window schemes.
8.2.3.1 Thermal-aware IP placement

The first experiment we conduct is to do the hardware virtualization manually using a custom algorithm that sequentially places virtual nodes to the same physical PE and use the proposed algorithm to do thermal aware IP placement. Table 8.1 shows the set up data for the LDPC nodes used in our experiment. The degree of a node is the number of outgoing connection edges to complementary nodes. Each node in the setup has two different degree values. The virtualization is done through manual assignments before the mapping process.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Bit Node Degrees</th>
<th>Check Node Degrees</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>set 1</td>
<td>3,4</td>
<td>6,8</td>
<td>16087000</td>
</tr>
<tr>
<td>set 2</td>
<td>3,4</td>
<td>8,10</td>
<td>13315840</td>
</tr>
<tr>
<td>set 3</td>
<td>3,4</td>
<td>8,12</td>
<td>14128000</td>
</tr>
<tr>
<td>set 4</td>
<td>5,6</td>
<td>6,12</td>
<td>20591360</td>
</tr>
<tr>
<td>set 5</td>
<td>3,9</td>
<td>6,9</td>
<td>16473088</td>
</tr>
</tbody>
</table>

Table 8.1. LDPC nodes configuration profile

Figures 8.7, 8.8, and 8.9 show the comparisons of five different sets of LDPC codes implemented on a 4x4 NoC.

Figure 8.7 shows the average temperature for the NoC chip. We can see that the temperature optimization approach performs better than all the others. The
running times for these experiments are 12~15 seconds for 1000 generations. Beyond 1000 generation, there is little improvement.

Figure 8.7. Average temperature comparison for a 4x4 LDPC decoder with 5 sets of hardware virtualization under different optimization strategies

Figure 8.8 shows the peak temperature, or hot spot temperature. It shows that among all three optimization strategies, the temperature optimization strategy results in the lowest peak temperature, with the average difference of 4°C for all the sets of optimization approaches.

Figure 8.9 shows the communication cost for different optimization strategies. It is obvious that the communication cost minimization strategy is the best choice.
in a communication critical environment. The conclusion we draw from our experimental results for thermal-aware mapping with manual hardware virtualization is that, to reduce the temperature of the hot spot, we should use the thermal balanced optimization strategy.

![Figure 8.9. Communication cost comparison for a 4x4 LDPC decoder under different optimization strategies](image)

8.2.3.2 Thermal-aware IP virtualization and placement

Using five different LDPC codes which had different bit/check node connectivities, we performed a comparison of simultaneously performing virtualization and mapping (denoted Virtualized), shown in Figure 8.10. As shown in the figure the Temp. approach still outperforms the other two approaches in both average and peak temperatures. This confirms the results shown in Figures 8.7 and 8.8.

Figure 8.11 shows the comparisons of our custom virtualization followed by our genetic algorithm based placement approach (denoted non-virtualized) and virtualized approach for temperature optimization. The number beside the name of a data set is the size of the IP array required for this mapping configuration. The effectiveness of performing simultaneous virtualization and placement is established here, as it reduces both the peak and average temperature by 2~3°C as compared to the non-virtualized approach. The reason that virtualized mapping is better than non-virtualized mapping is that we consider the connection of virtual nodes at a finer granularity that can place closely related nodes onto the same physical
Figure 8.10. The experimental result for virtualized mapping of 960 virtual nodes of a LDPC decoder PE to further reduce the number of communication links; as a consequence, we achieve further temperature reduction.

Figure 8.11. The comparisons of non-virtualized and virtualized mapping under temperature-based approach

The virtualized mapping comparison of virtualized mapping and non-virtualized mapping is shown in Figure 8.12. We can see that the virtualized approach has about 10% of the average communication cost reduction. The runtime for the experiments of doing virtualization and placement concurrently is about 9~13 minutes for 5000 generations.
Figure 8.12. The comparisons of communication cost for virtualized mapping and non-virtualized mapping
Chapter 9

Conclusions and Future Directions

As transistor counts keep increasing and clock frequencies rise, high power consumption is becoming one of the most important obstacles, preventing further scaling and performance improvements. While many low power techniques have been proposed, those techniques are normally applied independently or combined with at most three low power techniques, which is not sufficient to tackle the ever serious power issue. In this thesis, we present a power optimization framework based on the genetic algorithm. The optimization strategy can simultaneously perform multiple-$V_{dd}$ assignment, multiple-$V_{th}$ assignment, and gate sizing in conjunction with stacking force technique to minimize total power consumption, while maintaining performance requirements. We demonstrated the effectiveness of our total power optimization framework by conducting various experiments. The comparisons to the differently combined, low power techniques have also been conducted and the results confirm that our approach is valid and is comparable to other approaches.

In technology nodes of 100nm and below, and in high current applications, temperature gradients have a significant and growing impact on both performance and reliability of the product. As more transistors are packed on a die, power density varies considerably, causing local hot spots and non-trivial temperature gradients within the semiconductor chip. These high temperatures have a profound impact on signal timing, clock skew, reliability, power consumption, and median-time-to-failure of the manufactured chips. Because of temperature effects, existing methodologies are deviating further away from the reality of how today’s silicon
behaves. Thus, the assumption of constant on-chip temperatures can no longer be trusted.

This thesis discusses the possibilities of adding thermal-aware capabilities in the traditional design flow to combat the ever-increasing thermal problems. First, both 2D and 3D thermal-aware floorplanning algorithms are proposed and demonstrated to be effective in reducing hot spot temperatures. The interconnect power is also taken into account in our 3D floorplanner, as interconnect is already an important aspect in determining both the performance and the power consumption of a design in current and future technology nodes.

The technique of adding thermal consideration is also applied to three system-level design approaches. The first one is the thermal-aware allocation and scheduling sub-routine used in hardware/software co-synthesis. Unlike previous studies focusing on reducing the power consumption and/or the cost of entire system, which can result in higher chip temperatures, our allocation and scheduling algorithm targets reducing the hot spot temperature directly while achieving other design goals.

The second one targets System-on-Chip designs. There have been many techniques proposed to tackle the standard cell placement and floorplanning problem; however, these thermal-aware standard cell placement and floorplanning techniques cannot be applied directly to the targeted problem because the cores in voltage island domain are typically of a non-uniform size; in addition, the voltage island partitioning problem also prevents us from using traditional thermal-aware placers. Our approach integrates the thermal impact into floorplanning, partitioning, and voltage assignment, and thus results in a thermally balanced SoC design.

The third one focuses on mapping applications onto a NoC architecture with the goal of achieving a lower hot spot temperatures mapping. We compare different optimization strategies (power-balanced placement, thermal-balanced placement as well as communication cost minimization placement), and the experimental result indicates that with the thermal-balanced placement, we can achieve the best thermal distribution profile for the NoC architecture.
9.1 Future Directions

While this thesis discusses a number of points where thermal-aware analysis can be deployed, there are many new emerging technologies and new effects that are result of advanced technologies which challenge the already severe thermal problems. There are two possible future research topics in which we are particularly interested. The first one is to explore the thermal effect on three-dimensional (3D) architectural designs while the second one is to investigate the interaction between process variation and thermal effect.

There are numerous and novel 3D integrated technologies under development and one of the promising styles of 3D technologies is *wafer-bonding technology*. In the wafer-bonding technology, 3D integrated circuits are formed by vertical stacking of multiple strata where each stratum is an active device layer and is processed independently, and 3D vias provide die-to-die connections.

While 3D architecture can provide potential high packing density and performance, it also exacerbates the high power density issue of designing a microprocessor. Since the traditional two-dimensional (2D) design flow can not be applied directly, researchers have been actively investigating physical designs and automated tool designs for the 3D technology. However, microarchitecture evaluations can not be directly applied to the 3D technology due to the existence of through-wafer vias. There are a few works focusing on the evaluation of 3D microarchitecture from a performance angle; however, how to design each individual component of a microprocessor in a thermal-balanced way is unclear. Essentially, there is a clear need for one to investigate how to cleverly design a microprocessor such that the thermal impact is minimized in 3D architectures.

Aggressive technology scaling presents challenges to fabricate small feature size transistors, and results in significant variations in transistor parameters such as channel length, gate-oxide thickness, and threshold voltage across identically designed neighboring transistors (*intra-die variation*) and across different identically designed chips (*inter-die variation*). Process variations are primarily due to uncertainty in the device and interconnect characteristics, such as gate length, the thickness of gate oxide, and doping concentrations.

One of the biggest challenges related to thermal issues under process variation...
is the uncertainty of leakage power. Process variation make many manufacture parameters unstable under the influence of variation and thus makes leakage a variable. Leakage has emerged as the most critical design challenge for current and future integrated circuits because it has great impacts limiting the frequency, power, and thermal tolerance of circuit designs. Therefore, we believe that there is an urgent need to explore the interaction between process variation and thermal impact.
Bibliography


[37] http://www.isonics.com


Vita

Wei-Lun Hung

Wei-Lun Hung was born in Taiwan in May, 1975. He received his B.S and M.S degrees from National Taiwan University of Science and Technology and National Tsing Hua University, in 2000 and 2002, respectively. He has been in the Ph.D program in the department of Computer Science and Engineering at Pennsylvania State University since August 2002. He had worked as a teaching assistant in the Fall of 2003 and as a research assistant thereafter. The focus of his Ph.D work has been thermal-aware EDA tool designs, including, but not restricted to, low power VLSI design, computer architecture, and computer arithmetic, and process variation induced issues.