PHYSICAL PLANNING AND UNCORE POWER MANAGEMENT FOR MULTI-CORE PROCESSORS

A Dissertation

by

XI CHEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Jiang Hu
Co-Chair of Committee,	Paul V. Gratz
Committee Members,	Hank Walker
	Shuguang Cui
Department Head,	Chanan Singh

May 2013

Major Subject: Computer Engineering

ABSTRACT

For the microprocessor technology of today and the foreseeable future, multi-core is a key engine that drives performance growth under very tight power dissipation constraints. While previous research has been mostly focused on individual processor cores, there is a compelling need for studying how to efficiently manage shared resources among cores, including physical space, on-chip communication and on-chip storage.

In managing physical space, floorplanning is the first and most critical step that largely affects communication efficiency and cost-effectiveness of chip designs. We consider floorplanning with regularity constraints that requires identical processing/memory cores to form an array. Such regularity can greatly facilitate design modularity and therefore shorten design turn-around time. Very little attention has been paid to automatic floorplanning considering regularity constraints because manual floorplanning has difficulty handling the complexity as chip core count increases. In this dissertation work, we investigate the regularity constraints in a simulated-annealing based floorplanner for multi/many core processor designs. A simple and effective technique is proposed to encode the regularity constraints in sequence-pair, which is a classic format of data representation in automatic floorplanning. To the best of our knowledge, this is the first work on regularity-constrained floorplanning in the context of multi/many core processor designs.

On-chip communication and shared last level cache (LLC) play a role that is at least as equally important as processor cores in terms of chip performance and power. This dissertation research studies dynamic voltage and frequency scaling for on-chip network and LLC, which forms a single uncore domain of voltage and frequency.

This is in contrast to most previous works where the network and LLC are partitioned and associated with processor cores based on physical proximity. The single shared domain can largely avoid the interfacing overhead across domain boundaries and is practical and very useful for industrial products. Our goal is to minimize uncore energy dissipation with little, e.g., 5% or less, performance degradation. The first part of this study is to identify a metric that can reflect the chip performance determined by uncore voltage/frequency. The second part is about how to monitor this metric with low overhead and high fidelity. The last part is the control policy that decides uncore voltage/frequency based on monitoring results. Our approach is validated through full system simulations on public architecture benchmarks.

DEDICATION

To My Parents and Grandparents

ACKNOWLEDGEMENTS

It has been a long way to walk through the process of pursuing the truth. The difficult part of it is you have to bear the change of different states because of both desperation and hope. Luckily, I have those people around, with their full-hearted encouragment and positive support. Here, I would like to express my sincere appreciation to them, with my gratitude.

First, I want to thank my advisor, Dr. Jiang Hu and my co-advisor, Dr. Paul V. Gratz for their patience and guidance to help me grow up. Dr. Hu opens a door of research for me and leads me into this wonderful world. Since my first day here, Dr. Hu helps to keep me on the right track in the reserach so that I will not lost myself in the sea of reserach. Dr. Gratz opens another door for me. He introduces me into the wonderful world of computer architecure. Meanwhile he also broadens my horizion of research world and guides me to do research on the state-of-the-art topics. Both professors teach me how to do research, from which I also can apply to my daily life that will benefit me for the rest of my life.

I would also like to express my appreciation to my committee members Dr. Hank Walker and Dr. Shuguang Cui for their valuable suggestions in my preliminary examizations. Special thanks will be given to Dr. Laszlo Kish for his suggestions on my defense and dissertation. Many thanks to my fellow friends here at Texas A&M University: Qinghe Du, Xu Zheng, David Kadjo and some other friends. Their help on academic study and my daily life makes my student life complete here.

Last but not least, my greatest gratefullness would be dedicated to my family, for their everlasting and unconditional support during my whole life.

NOMENCLATURE

WL	Wire Length
DVFS	Dynamic Voltage Frequency Scaling
V/F	Voltage/Frequency
NoCs	Network-on-Chips
LLC	Last Level Cache
CMP	Chip Multi-Processor
AMAT	Average Memory Access Time
PID	Proportional Integral Derivative
APHL	Average Per-Hop Latency
APRL	Average Per-Request Latency

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

1.1 Research Motivation

Due to the increasingly stringent chip power budget, the semiconductor industry has changed its development strategy from improving single core performance to the multi-core technology. Indeed, multi-core technology becomes the new engine that drives chip performance growth under tight power budget. How to efficiently manage all kinds of resources in a multi-core chip design is not a well-solved problem yet. For physical resources, multi-core chip floorplanning considering regularity constraints has received little attention before. How to manage power for shared resources, such as on-chip network and last level caches, is another problem that deserves deep study.

1.2 Contribution

1.2.1 Regularity-Constrained Floorplanning

In physical design, floorplanning is the first primary step that decisively affects chip layout, on-chip communication, power, performance and various design concerns [59]. When the core count is keeping increase, manually design will be very difficult to quickly and thoroughly explore different options. Therefore, there is a strong need for automatic floorplanning [19] [57] techniques for many-core CMP designs.

In multi-core chip designs, processcing units and cache blocks are usually placed in arrays. However, such regularity constraint is rarely considered in the conventional floorplanning. In Chapter 2, the work on floorplanning with regularity constraint, which is oriented toward multi-core processor designs is presented. The key contributions are on how to encode the regularity constraint in sequence-pair and how to achieve the regularity in packing procedure. In this work, the proposed approach is compared with a naive method that manually tries multiple placements for array blocks, each of which is followed by conventional floorplanning for non-array blocks while the array blocks are fixed. The experimental results indicate that the proposed approach can achieve an average of 12% less wirelength than the semi-automatic method. At the same time, the proposed approach usually leads to smaller area.

1.2.2 Uncore Power Management for Multi-Core Processors

In Chip-Multiprocessor (CMP) designs, uncore usually refers to on-chip communication fabrics and the shared last level cache (LLC), both of which account for a large portion of chip estate. On-chip communication and LLC are both chip performance bottlenecks as well as large power consumers. This thesis research focuses on Dynamic Voltage and Frequency Scaling (DVFS) of on-chip network and LLC, with attempt to substantially reduce uncore power dissipation with very small performance degradation.

In previous reseach, people usually put core and associated LLC in same voltage/clock domain. However in our work, we argue that placing the shared LLC in one clock domain across the chip is also logical because it is a one large, partitioned structure. Contrary to the individual cores which are running different threads/programs, the LLC banks have a mostly homogeneous load due to the interleaving of cache lines in the system; in this case, voltage/frequency domain partitioning should be more reasonable if placing LLC and interconnect networks in the same V/F domain.

In this work, A new method is proposed and investigated to measure the system performance and effectively transport information in network. Meanwhile, PI controller with a dynamic reference point based on the new metric is adopted to control the whole uncore system. These methods are evaluated in full system simulation on the PARSEC benchmarks [3].

In Chapter 3, the whole power management problem will be addressed. The key contributions of this chapter are as follows: 1.Introducing several new uncore status metrics to predict the impact of DVFS policy on system performance. 2.Proposing a novel, extremely low overhead, uncore status monitoring technique. This technique is composed of the following: 1) Per-node metric sampling, 2) Passive in-network status encoding, no extra packets needed, 3) Metric extrapolation to properly scale value weights. 3.A PI controller with a dynamic reference point is described.

2. REGULARITY-CONSTRAINED FLOORPLANNING FOR MULTI-CORE PROCESSORS

2.1 Introduction

When the Moore's law is near its end, continuing chip performance growth will inevitably rely on the improvement of system level integration. This is evidenced by the popularity of multi-core technology for both microprocessors and embedded processors. In a foreseeable future, current multi-core processors will advance to many-core processors, which allow hundreds of cores on a chip. This trend presents new challenges to the design and design automation technologies. This paper discusses floorplanning problem for multi-core and many-core processors and proposes an algorithmic solution to this problem.

Floorplanning is the first primary physical design step that decisively affects chip layout, on-chip communication, power, performance and various design concerns [59]. When the number of cores on a chip is small, the floorplanning can be managed by manual designs, especially for CMP (Chip Multiprocessors). For instance, a 4-core processor can be manually placed in a 2 by 2 array. When the core count exceeds one hundred, the options of floorplans increase dramatically. Then, it would be very difficult for manual design to quickly and thoroughly explore these options. Besides processing cores, a processor chip usually contains cache, I/O blocks and communication fabrics. Further, CMP technology will move from homogeneous to heterogeneous cores [27] like IBM Cell processor. These facts imply heterogeneous entities, which make manual floorplanning even more difficult. Therefore, there is a strong need for automatic floorplanning [19] [57] techniques for many-core CMP designs. Multi-core technology is also widely adopted in SoC (System-on-Chip) designs and leads to the so called MPSoC (Multi-Processor SoC). SoC designs are often targeted to embedded computing and require much shorter design turn-around time than microprocessors. Although conventional floorplanning techniques are applicable to current MPSoC designs, there is a new problem as the system grows from multi-core to many-core. That is, if multiple identical cores are adopted, usually they are preferred to be placed in a regular array. If ever possible, regularity is desired in chip layout for the sake of design simplicity, modularity and easy management of physical resources.

The regularity issue is rarely considered in the conventional floorplanning. One similar case is analog circuit layout [30] [32] [2] [56] [39] [9] where components are often placed in a symmetric fashion. One may want to fulfill the regularity constraint by enforcing the symmetry constraints. Even though symmetry and regularity are related, regularity is actually more complex than symmetry and often more difficult to achieve. A chip with m cores can be placed in a p by q array and there are often multiple ways for the factorization of m = p * q, e.g., m = 30 = 1 * 30 =2 * 15 = 3 * 10 = 5 * 6 =6 * 5 :. 30 * 1. Even for a specific factorization, symmetries to different axes need to be maintained to obtain a regular array. The work of [39] addressed the array-type constraint for analog placement. However, there is a key difference between the array constraint in analog placement and regularity constraint in multi-core processor floorplanning. In analog placement, array blocks of the same type of device are compacted together in order to reduce the effect of spatially-dependent variations. In multi-core processor floorplanning, in contrast, non-array blocks can be placed between array blocks and one group of array blocks can be placed inside of another group of array blocks. By allowing such option, one may have an opportunity to further reduce interconnect delay and congestion. For example, placing the memory controller in the center of an array of processing cores conceivably causes less interconnect congestion than placing it at peripheral regions.

In this chapter, I will present the work on floorplanning with regularity constraint, which is oriented toward multi-core processor designs. This floorplanner is a simulated annealing algorithm using sequence-pair representation. The key contributions are on how to encode the regularity constraint in sequence-pair and how to achieve the regularity in packing procedure. To the best of our knowledge, this is the first work studying regularity-constrained floorplanning for multi-core processors. I compared the proposed approach with a naive method that manually tries multiple placements for array blocks, each of which is followed by conventional floorplanning for non-array blocks while the array blocks are fixed. The experimental results indicate that the proposed approach can achieve an average of 12% less wirelength than the semi-automatic method. At the same time, the proposed approach usually leads to smaller area.

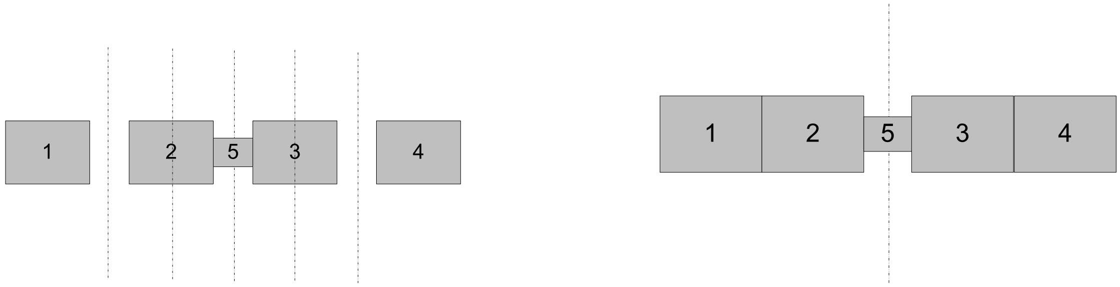
2.2 Preliminaries

Among previous works on floorplanning, those for analog integrated circuits have the closest problem formulation as the regularity-constrained floorplanning work. Since an analog circuit typically has a small number of elements, its placement is often equivalent to the floorplanning of a digital integrated circuit. In analog circuit designs, one important requirement is to place blocks or devices symmetrically with respect to one common axis so as to improve the tolerance to common-mode noise [9].

There are many analog circuit layout works [30, 32, 2, 56, 39, 9, 50, 24, 33, 31, 41] focusing on the symmetry constraints. In [30], TCG-S is used as placement representation to do analog layout under symmetry constraint. In [32], in the context of Silicon on Insulator (SOI), mismatch analysis for analog layout is proposed and tested. In [2], a symmetry-constrained analog block placement method is proposed. This work is based on a typical floorplanning approach – simulated annealing with sequence-pair representation. The symmetry constraint is described through sequence-pairs. For a sequence-pair (α, β), α_A^{-1} denotes the position of block A in sequence. Consider a group of blocks G that must be placed symmetrically around a vertical axis. A sequence-pair (α, β) is symmetric-feasible for G if for any blocks A and B in G

$$\alpha_A^{-1} < \alpha_B^{-1} \Leftrightarrow \beta_\sigma^{-1}(B) < \beta_\sigma^{-1} A \tag{2.1}$$

where $\sigma(A)$ is the block symmetric to A. However, it is pointed out in [24] that condition (2.1) is sufficient but not necessary. More recently, the work of [56] [33] presented another sequence-pair based approach for simultaneously satisfying symmetry and centroid constraints. Another method based on B*-tree is proposed in [31] for handling both 1-D and 2-D symmetry constraints. The other symmetry constrained analog placement work [41] uses O-tree representation. In [60], a symmetry-aware placement work is proposed based on Transitive Closure Graphs (TCG) data structure. To certain extent, regularity constraint can be treated as an extension to the symmetry constraints. However, the extension is not trivial as the number of implicit symmetry constraints embedded in a regularity constraint can be quite large. More specifically, every spatially contiguous subset of an array group needs to follow its own symmetry. Figure 2.1 is a simple example and it has 4 blocks that need to satisfy the regularity array constraint in (a) and symmetry constraint in (b). In (a), all spatially contiguous subsets {1,2}, {2,3}, {3,4}, {1,2,3} and {2,3,4} need to satisfy their own symmetry constraints. Hence, the regularity array constraint implies significantly more symmetry constraints than the case of analog circuit layout like in (b). In [39], the array-type constraint is considered for analog circuit placement. In order to mitigate the effect of PVT (Process, Voltage and Temperature) variations, which are usually spatially correlated, the work of [39] packs array blocks of the same type right next to each other. In multi-core processor designs, however, the problem granularity level and concerns are different. By allowing non-array blocks to be embedded between array blocks, chip interconnects performance and cost can be reduced. Figure 2.2 shows an industrial design where non-array blocks SIU and CCX are placed in between the 2*4 array of L2T blocks and the L2T blocks are placed in between the 2*8 L2D blocks.



(a) Regularity constraint for digital processors.

(b) Symmetry constraint for analog circuit.

Figure 2.1: Comparison between regularity and symmetry constraint.

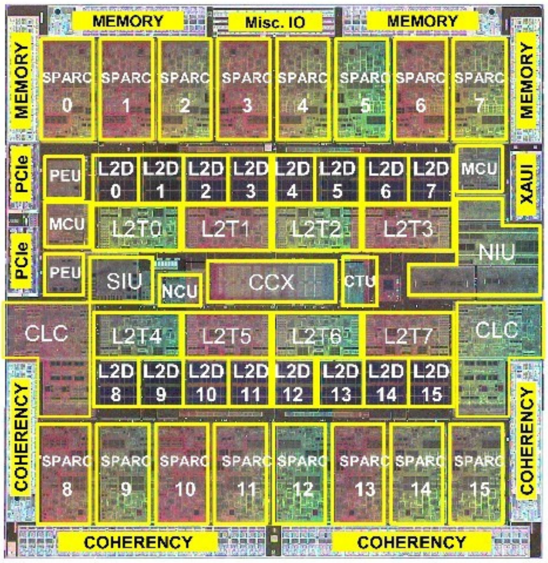


Figure 2.2: Floorplan of SUN Niagara-3 processor.

2.3 Floorplanning with Regularity Constraint

2.3.1 Problem Formulation

The input to floorplanning includes a set of n blocks, each with area Ai where i = 1, 2, ... n, a set of l nets N_1, N_2, ..., N_l among the n blocks, a set of k array groups G_1, G_2, ...,G_k. Each array group is a subset of the given blocks that must be placed in a regular array. If a block is in an array group, it is called an array block; otherwise, it is called a nonarray block. The problem is to construct a floorplan F that satisfies non-overlapping and the regularity constraint, and minimizes the target cost function

$$cost(F) = (1 - \lambda) * area(F) + \lambda * wirelength(F) \tag{2.2}$$

where λ is a weighting factor, area(F) is the total area of F and wirelength(F) is the total wire length of F. In this work, the half-perimeter model (HPWL) for wirelength estimation and outer chip area for area estimation are adopted. The main difference of this work from the conventional floorplanning is the regularity constraint. An array group is composed of blocks with identical size and shape, which are usually processor cores or memory cores in reality. Although the cores are required to be placed in an array, the shape of the array, which is decided by the number of rows and columns, is flexible. For example, for an array group of m blocks, any array of p*q = m is allowed and considered in the floorplanning as long as aspect ratio constraint is satisfied. Moreover, the blocks in an array group do not have to be placed next to each other. The floorplan of SUN Niagara-3 processor in Figure 2.2 demonstrates such example. It contains 16 SPARC processor cores, which are placed in a 2*8 array. However, the two rows are not adjacent to each other and allow other blocks like CCX, CTU to be placed in between.

2.3.2 Regularity in Relative Order

Floorplanning representation is critical to the efficiency of a floorplanning method, especially for the popular simulated-annealing approach. There are two main categories of floorplanning representations: absolute representation and topological representation. In an absolute representation, every block is specified in terms of its absolute coordinates. The solution search in the absolute representations tends to be complex and difficult. Therefore, topological representations become more popular and lead to many research results including sequence-pair [38], O-tree [15], B*-tree [6], Corner Block List (CBL) [16] and Transitive Closure Graphs (TCG) [29]. In this work, focus is placed on sequence-pair based floorplanning as sequence-pair is one of the most known floorplanning representations and has been successfully applied in handling symmetry constraints [2] [56]. In section 3.5, the regularity constraint in other floorplanning representations will be briefly discussed. For a set of blocks, a sequence-pair consists of two sequences of block IDs corresponding to the orders along two diagonal directions. A sequence-pair like (<... i ... j ...>, <... i ... j ...>) implies that block i (j)is to the left (right) of block j (i). Similarly, (<... i ... j ...>, <... j ... i ...>) means that block i (j) is above (below) block j (i). A sequence-pair for 6 blocks (<1 2 4 5 3 6>, <3 6 2 1 4 5>) is demonstrated in Figure 2.3.

The regularity constraint to an array group of m blocks implies that these blocks must be placed in a p*q array, where p*q is a factorization of m. The q blocks in one row must appear as a common subsequence [51] in the sequence-pair.

Definition 1: Alien blocks. For one array block group, all the other blocks that do not belong to this group are called alien blocks of this group. Please note an alien block can be either a non-array block or an array block of another array group.

Definition 2: Common subsequence. A set of q blocks b_1, b_2, ..., b_q form

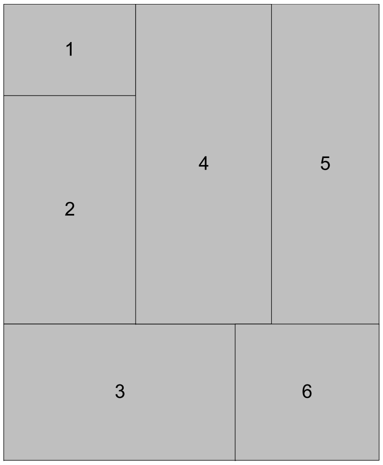


Figure 2.3: Floorplan for the sequence-pair(<1 2 4 5 3 6>,<3 6 2 1 4 5>).

a common subsequence in a sequence-pair (α, β) if $\alpha_1^{-1} < \alpha_2^{-1} < ... < \alpha_q^{-1}$ and $\beta_1^{-1} < \beta_2^{-1} < ... < \beta_q^{-1}$ where $\alpha_i^{-1}(\beta_i^{-1})$ indicates the position of block b_i in sequence $\alpha(\beta)$. For example, the floorplan of Figure 2.4 can be specified by sequence-pair (<0 1 2 3 4 5>, <2 1 0 5 4 3>), which contains 3 common subsequences (0, 3), (1, 4) and (2, 5) for the 3 rows. This concept of common subsequence is similar to H-alignment in [52].

Likewise, the blocks in a column must appear as a reversely common subsequence in the sequence-pair.

Definition 3: Reversely common subsequence. A set of p blocks b_1, b_2 ... b_p form a reversely common subsequence in a sequence-pair (α, β) if $\alpha_1^{-1} < \alpha_2^{-1} < ... < \alpha_q^{-1}$ and $\beta_1^{-1} > \beta_2^{-1} > ... > \beta_q^{-1}$ where $\alpha_i^{-1}(\beta_i^{-1})$ indicates the position of block b_i in sequence $\alpha(\beta)$.

For the example in Figure 2.4 , block 0, 1 and 2 form a reversely common subsequence (<0 1 2>, <2 1 0>) and (<3 4 5>, <5 4 3>).

Lemma 1: The necessary condition that m blocks lead to a p*q array floorplan: the m blocks constitute p common subsequences of length q and q reversely common

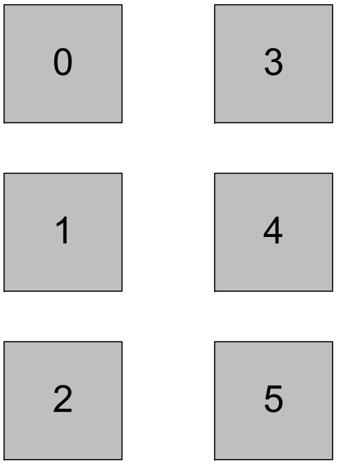


Figure 2.4: An array group placed in a 3*2 array.

subsequences of length p in the sequencepair.

Proof: m blocks can lead to more than one sequencepair representations but a p common subsequences of length q or q reversely common subsequences of length p can only lead to one floorplanning as long as the packing procedure maintains the same. Figure 2.4 is an example, for the total 6 blocks, the floorplanning can be translated into either (<0 1 2 3 4 5>, <2 1 0 5 4 3>) which has 3 common subsequences of length 2 or (<0 3 1 4 2 5>, <2 5 1 4 0 3>) that has 2 common subsequences of length 3. While (<0 1 2 3 4 5>, <2 1 0 5 4 3>) can only be mapped to the floorplanning as Figure 2.4 shows. Thus the above condition is necessary but not sufficient because a sequence-pair specifies only a relative order. Generation of an array-type floorplan also depends on the packing procedure, which will be discussed in Section 2.3.3 .

Definition 4: Regularity subsequence-pair (RSP). A contiguous subsequence of length m that satisfies Lemma 1 in a sequence-pair is called regularity subsequence-pair. In Figure 2.4, there are no other blocks in the middle of 3*2 array and the corresponding sequence-pair is a RSP. In fact, the mapping from an array

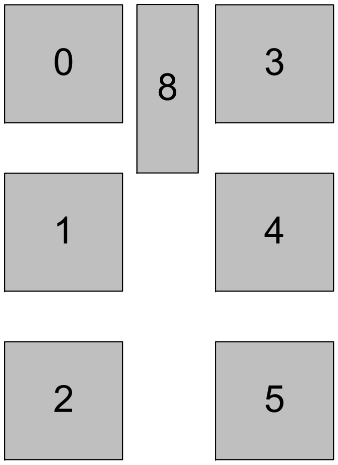


Figure 2.5: Block 8 is placed in the middle of the regularity subsequency(<0 1 2 8 3 4 5>, <2 1 0 8 5 4 3>.

floorplan to sequence-pair is not unique. For instance, the floorplan in Figure 2.4 can be alternatively specified by sequence-pair (<0 3 1 4 2 5>, <2 5 1 4 0 3>), where each row is in a contiguous subsequence. This is in contrast to (<0 1 2 3 4 5>, <2 1 0 5 4 3>) where each column is a contiguous subsequence.

Definition 5: Row (column) based regularity subsequence-pair is a regularity subsequence-pair where each (reversely) common subsequence corresponding a row (column) is contiguous. In the regularity subsequence-pair, alien blocks are allowed in the middle of an array, their block ID can be embedded within the corresponding RSP as long as complying with the following 2 rules. One such example is given in Figure 2.5 where block 8 is inside of both subsequence of the RSP.

Rule 1: An alien block can be inside both or neither of α and β sequences of a RSP. An alien block cannot be inside one of α and β sequences but outside of the other for a RSP. In the example of Figure 2.6, for sequence-pair (α,β), we do not allow (<0 1 2 8 3 4 5>, <8 2 1 0 5 4 3>). Here alien block 8 is outside the α part of RSP (0 1 2 3 4 5), but inside the β part. This is a violation of Rule 1 as it will lead

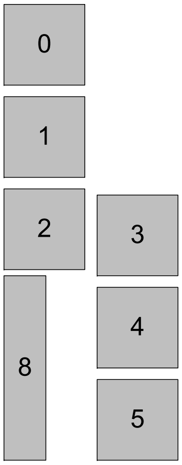


Figure 2.6: Misalignment due to violation of Rule 1.

to misalignment in later packing procedure. A packing result of this sequence-pair is depicted in Figure 2.6. We can see that α_8^{-1} and β_8^{-1} is " $<$ " than both α_5^{-1} and β_5^{-1} which means that block 8 is always on the left of block 5. But α_8^{-1} is " $>$ " than α_2^{-1} and β_8^{-1} is " $<$ " than β_2^{-1} which also implies that block 8 is under the block 2. Ideally, we hope to make block 2 and block 5 symmetrical about vertical axis, however failing to obey rule 1 causes the result unexpected.

Rule 2: An alien block can be inside both or neither of α and β part of a contiguous (reversely) common subsequence in a row (column) based RSP and it must be between the same array blocks when it is in both α and β sequence. An alien block cannot be inside one of α and β part but outside of the other for a contiguous (reversely) common subsequence in a row (column) based RSP. For example, (<0 1 2 8 3 4 5>, < 2 8 1 0 5 4 3>) is not allowed as alien block 8 is outside the α part of the reversely common subsequence (0 1 2), but inside its β part. Another example (<0 a 1 2 3 4 5>, <2 a 1 0 5 4 3>) is also not allowed, because even "a" is between (0 1 2) in both parts, it is not between the same array blocks. In α part, "a" is between block 0 and 1, while in β part, "a" is between block 2 and 1.

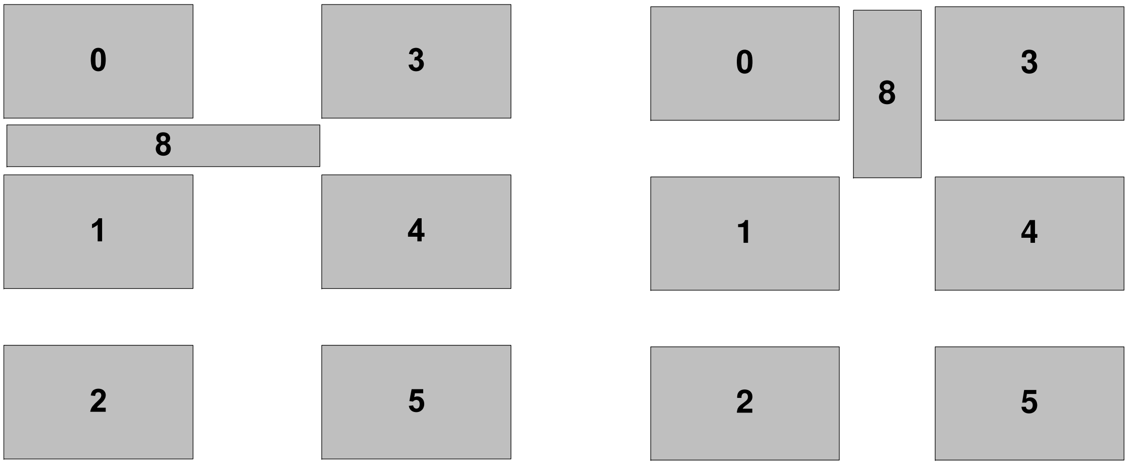
The reason of rule 2 is the same as rule 1 that it may cause the misplacement during the packing procedure. Rule 1 and rule 2 are enforced in the floorplanning algorithm in order to avoid ambiguity for subsequent packing that may lead to misalignment.

The enforcement of rule 2 implies that an alien block cannot be placed between two columns (rows) for a row (column) based RSP. Consequently, row-based and column-based RSPs may lead to different floorplan if alien blocks are allowed between array blocks. Figure 2.7 is an example. In Figure 2.7a, block 8 lies between row 1 (block 0,3), and row 2 (block 1,4), it is hard to represent the floorplanning with column-based RSP, thus row-based representation (<0 3 8 1 4 2 5>, <2 5 1 4 8 0 3>) is necessary. On the contrary, in Figure 2.7b, block 8 is placed between 2 array columns (block 0 1 2 and block 3 4 5), so it is almost impossible to represent with row-based representation without violating rule 2 above, column-based RSP (<0 1 2 8 3 4 5>, <2 1 0 8 5 4 3>) is necessary under this situation.

In order to explore the complete space, both rowbased and column-based RSP need to be separately examined during the annealing process.

2.3.3 Regularity in Packing

In floorplanning, a sequence-pair only specifies a relative order for the blocks and the absolute locations for the blocks need to be further decided through a packing procedure. In [38], horizontal and vertical constraint graphs are constructed for a sequence-pair. Then, the packing is obtained by performing the longest path algorithm on the graphs. Later, a faster packing algorithm based on longest common sequence (LCS) is proposed in [51]. In this work, the packing approach of [51] is adopted. Regularity in the packing implies the alignment and spacing constraints. Array blocks of each row (column) must be horizontally (vertically) aligned. Some-



(a) Row-based RSP.

(b) Column-based RSP.

Figure 2.7: Row-based and column-based RSP

times, one may additionally prefer identical spacing between rows (columns). This is illustrated in Figure 2.8 . The alignment and spacing constraints can be expressed according to the block locations. For example,

$$X_{i,j} - X_{i,j-1} = X_{i,j+1} - X_{i,j} \tag{2.3}$$

$$Y_{i,j} - Y_{i-1,j} = Y_{i+1,j} - Y_{i,j} \tag{2.4}$$

where X, Y are x coordinate and y coordinate of the lower-left corner of an array block, and i(j) represents row (column) index.

During the packing process, if there is no alien block inside an array, i.e., no alien block is inside a RSP, then the array can be pre-packed into a single object, with or without spacing. Then, the LCS (Longest Common Subsequence) packing algorithm [51] can be applied directly by treating pre-packed array blocks as one big block.

If there is any alien block inside an array, then the minimum uniform spacing

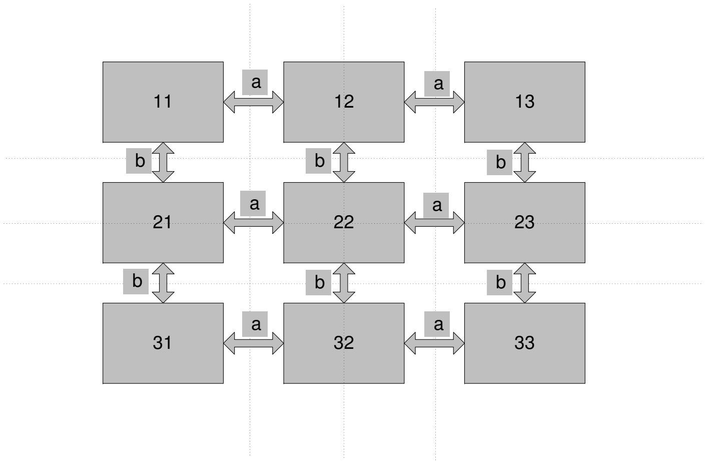
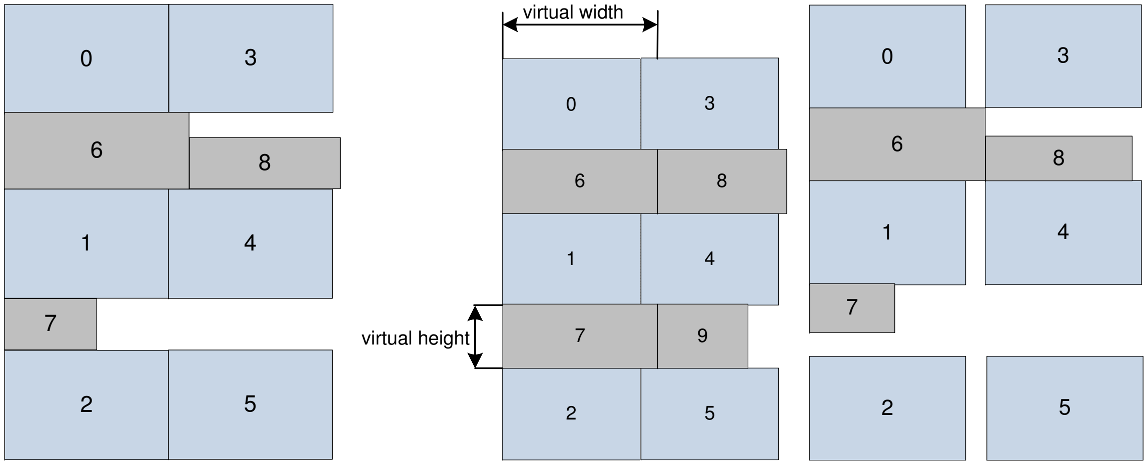


Figure 2.8: Alignment and uniform spacing.

between array blocks is decided by the maximum height of all alien blocks embedded within the array group and the maximum width of all alien blocks embedded within the array group except the rightmost ones. Before calling the LCS engine, a preprocessing that temporarily expands the dimensions of the alien blocks is performed. Consider the example in Figure 2.9 which illustrates the entire packing procedure. First, there are 3 alien blocks: 6, 7, 8 and 6 array blocks in Figure 2.9a , and the sequence pair is (<0 3 6 8 1 4 7 2 5>, <2 5 7 1 4 6 8 0 3>). Then, there is one space between two regular array blocks 4 and 5. Therefore, a virtual block (block 9) is placed between array blocks 4 and 5, so the sequence pair becomes (<0 3 6 8 1 4 7 9 2 5>, <2 5 7 9 1 4 6 8 0 3>). Next, the heights of 4 alien blocks (6 7 8 9) are expanded to the largest height of all 3 alien blocks (6 7 8), which is the height of block 6. This height is called virtual height. Then the next step is to expand the width to the largest width (virtual width) of all 2 alien blocks 6 and 7 (excluding the rightmost column alien blocks), which is the width of block 6, as Figure 2.9b shows. Notice that there is no need to expand the width of block 8 and 9 as they are in the rightmost column of the array. An expansion of these two blocks is unnecessary



(a) packingA. (b) PackingB. (c) PackingC.

Figure 2.9: Packing Procedure

and may affect blocks outside of this array. If there is no alien block between two adjacent array blocks, this method can still work by imaging a virtual block there, which has the virtual height and width. After the expansion is finished, the LCS algorithm is performed to do the packing. Once the packing result is obtained, the alien blocks are restored to their original dimensions. Finally, the result is shown in Figure 2.9c with the sequence pair (<0 3 6 8 1 4 7 2 5>, <2 5 7 1 4 6 8 0 3>).

Currently, the proposed method allows at most one alien block between two adjacent array blocks. The scenarios of allowing multiple alien blocks can be very complex and will be studied in the future research.

LCS-Regularity Packing Pseudo Code

```
1. LCS-REGULARITY(X, Y, Position, array blocks)
2. If (no alien blocks among array blocks)
3. Packing array blocks into one object;
4. Else
5. Begin
```

```
If (a space between two array-blocks is empty)
    One virtual block is generated and filled in;
    If (Row-based RSP)
    Begin
        Find the max height of all alien blocks in between array blocks;
        Find the max width of all alien blocks except those in the rightmost column between array blocks;
        Expand all alien blocks among array blocks to virtual height;
        Expand all alien blocks within array blocks except those in the rightmost column to virtual width;
    End
    If (Column-based RSP)
    Begin
        Find the max width of all alien blocks in between array blocks;
        Find the max height of all alien blocks except those in the lowest row between array blocks;
        Expand all alien blocks among array blocks to virtual width;
        Expand all alien blocks within array blocks except those in the lowest row to virtual height;
    End
End
Start normal LCS packing procedure;
Restore expanded alien blocks to their original dimensions and delete virtual block;
```

2.3.4 The Floorplanning Algorithm

Once the regularity constraint is encoded in sequence-pairs, the floorplanning algorithm is a straightforward extension of conventional simulated annealing based approach. Initially, a sequence-pair that satisfies Lemma 1 for each array group is generaged. For each array group with m blocks, a random factorization m = p*q is generated first. Then, the m cores are randomly allocated to p subgroups. Each subgroup, which corresponds to a row/column, is arranged as a common subsequence in the initial sequence-pair. Next, the p common subsequences are arranged in a reversely common order so that a regularity subsequence-pair is obtained for the array group. After the initial sequence-pair is obtained, a packing procedure (as described in Section 2.3.3) is performed. The result is evaluated according to the cost function cost(F)=(1-λ) * area(F) + λ * wirelength(F) , which is defined in Section 2.3.1 .

The initial solution is then iteratively improved by simulated annealing with the following moves.

- Changing the factorization of an array group. For example, a 2*3 array can be changed to a 3*2 array (Figure 2.10) or other factorizations.
- Changing the RSP for an array group between row-based and column-based.
- Moving an alien block into (or outside) a RSP (Regularity Subsequence-pair).
- Moving smaller RSP into larger RSP.
- Swapping two alien blocks.
- Rotating an alien block.

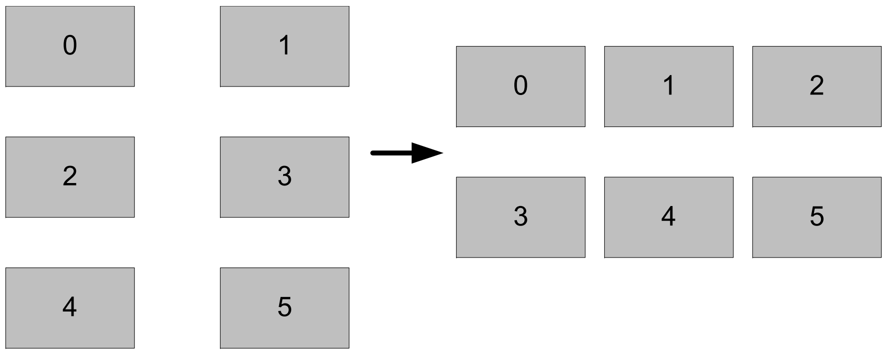


Figure 2.10: Factorization change(<0 1 2 3 4 5>,<4 5 2 3 0 1>)⟹(<0 1 2 3 4 5>,<3 4 5 0 1 2>).

- Rotating all blocks in an array group.
- Swapping two blocks in the same array group.

The proposed method can simultaneously handle multiple array groups and permits one group to be interleaved with another under certain conditions. When the sequence pair is row-based, alien blocks of another array can be placed between two array block rows. Thus, when the factorization of the small size array group has less number of rows than that of the larger size array group, our method allows the smaller size group to be interleaved with the larger size group. Figure 2.13 is an example where the smaller regularity group (5*2=10 blocks) is embedded in the larger regularity group (6*6=36 blocks). Likewise, when the sequence pair is column-based, alien blocks of another array can be placed between two array blocks column. Overall, only when both row and column counts of an array group are less than another array group, the smaller size group to be interleaved within the larger size group is allowed. This condition guarantees that the interleaving is feasible.

During the annealing process, when array blocks need to change their factorizations, all alien blocks are moved out of the array blocks and randomly put around.

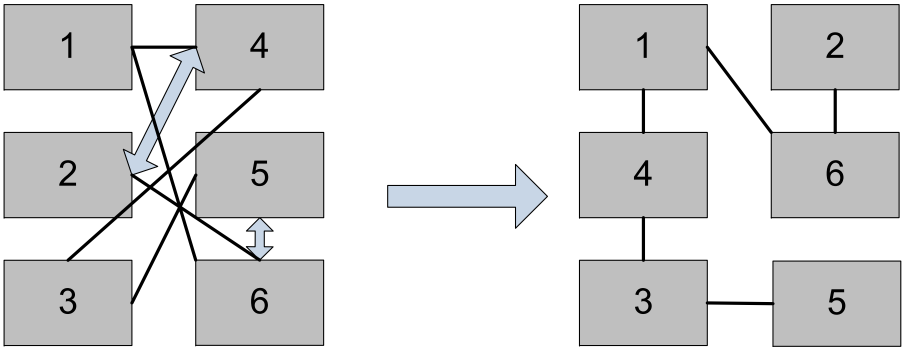


Figure 2.11: Swapping array blocks.

After the array blocks change their factorization, the algorithm can decide randomly whether to put alien blocks back or not.

Most of the moves are first performed on the sequence-pair and then the packing is performed. The cost function for each packing result is evaluated after packing procedure. The last type of move, which swaps two blocks in the same array group, needs further discussion. Since any two array blocks in the same array group have the same dimension and orientation, swapping them does not affect area and non-overlapping constraint, but sometimes may reduce wirelength as Figure 2.11 indicates. This swap affects only wirelength and therefore can be skipped if λ =0, i.e., the floorplanning is only for area minimization. Since the interconnect within an array group is typically symmetric, this swap affects more on wires between the array group and blocks outside this group.

2.3.5 Other Floorplan Representations

Besides sequence-pair, there are numerous other floorplan representations: B*-Tree, O-Tree, Corner Block List (CBL) and Transitive Closure Graph (TCG), to name a few. Many of them, including B*- Tree, O-Tree and CBL, are for compacted floorplanning. The compacting nature of these representations often conflicts

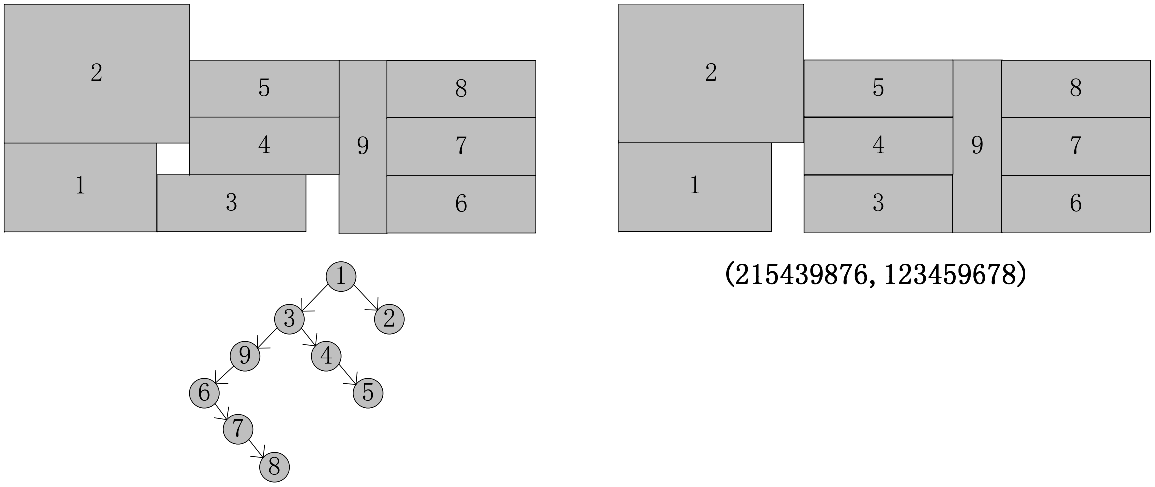


Figure 2.12: Floorplan compaction from B*-Tree and sequence-pair.

with the alignment requirement for multi-core processor designs. On the left of Figure 2.12, B*-Tree forces block 3 right next to block 1. Consequently, block 3 is not aligned with block 4 and 5 like the sequence-pair result on the right. If moving the whole block 3 and its offspring to left node of block 2, then all blocks (3,4,5,6,7,8,9) will be above block 1, which increases area of floorplanning. Of course, one can redo the compaction but this nullifies a main advantage of these representations, compaction. TCG is not a compacted floorplan representation and is similar to sequence-pair in nature.

2.4 Experimental Results

To the best of our knowledge, there is no previous work on regularity-constrained floorplanning for multi-core processors. The most-related works are the analog placement methods considering certain symmetry constraints. A direct comparison with the analog placement methods can hardly lead to conclusive observations. First, the problem formulations on symmetry constraints and regularity constraints are significantly different. Second, it is not obvious how to set analog-layout-like symmetry constraints for multi-core processors in a way that is fair for comparison.

Here attemptption is made to emulate what a designer might do his/her best with existing EDA tools. More specifically, the proposed approach is compared with a semimanual/ semi-automatic method, called multi-prefix method. The multi-prefix method tries multiple manual placements of the array blocks in order to satisfy the regularity constraints. Each of the placements is followed by a conventional simulated annealing based floorplanning for the alien blocks while the array blocks stay to follow maintain their regularity. The best result among the multiple attempts is selected as the final solution. The manual First, multiple different array factorizations are explored and the one that leads to the best result is selected in the manual prefix method. Second, the relative position between an array and the other blocks can be changed during the simulated annealing part of the manual prefix method. Moreover, the manual prefix method allows blocks within an array to be swapped.

The input to both methods includes a set of blocks containing preidentified array groups. The blocks in the same array group should have the same size and shape. In the multi-prefix method, different factorization options of an array group are explored. For instance, if a chip has an array group of 16 blocks (like Figure 2.2), the multi-prefix approach is performed 5 times with arrays of 1*16, 2*8, 4*4, 8*2 and

16*1, respectively. Then each array group can be treated as a single big block which maintains regularity during the floorplanning process. In the multi-prefix method, blocks in the same array group can be swapped to get smaller wirelength if necessary. The best result among these options is selected to be compared with our approach. When the runtime of the multi-prefix method is reported, it is the total runtime of the multiple runs.

Existing public domain benchmarks are mostly from old designs and do not have cases for multi-core processors. Therefore, slightly modifications are made to the MCNC and GSRC floorplanning benchmark by converting a subset of blocks into array groups. In the benchmarks, I randomly choose those blocks with similar dimensions and modify their width and length to be the same. Those large or small blocks are kept unchanged so that the change to the benchmarks is limited.

The floorplanning algorithms are implemented in C++ and the experiments are performed on a Windows based machine with a 2.5GHz Intel Core 2 Duo processor and 2GB memory.

The main results on the MCNC benchmark circuits are reported in Table 2.1. Here, the value of λ is 0.5, i.e., the weighting factors for the area and wirelength are the same. In this experiment, the aim is to find an optimized result with the balance between area and wirelength. Each circuit has one array group. The second column lists the array factorization that leads to the minimum cost among multi-prefix methods. The third and fourth column lists the final area and total wirelength of the best prefixed method. The fifth column "CPU(s)" is the aggregation running time of the all prefixed method. The proposed approach can reduce wirelength by 12% on average. At the same time, our approach achieves the same or less area and mostly faster runtime.

Circuit	# of	Multi-Prefix				Our Approach				
	Blocks	Min Cost Array	Area	WL	CPU(s)	Area	Area Reduction	WL	WL Reduction	CPU(s)
Apte	9	4*1	48	486	38	48	0%	472	2.9%	22
Hp	10	1*4	10	300	47	9	9.2%	279	7.0%	27
Xerox	11	1*4	25	1016	293	25	1.1%	687	32.3%	102
Ami33	33	4*2	1	83	1572	1	2.5%	77	7%	474
Ami49	49	4*4	50	1763	4960	49	2.6%	1559	11.6%	1354

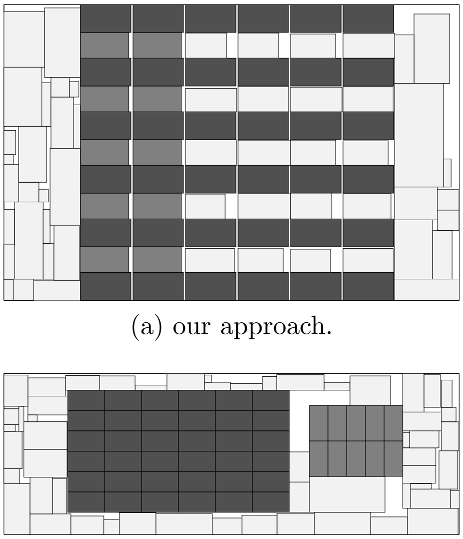
Table 2.1: Experimental results of multi-prefix and our approach on MCNC circuits, with λ=0.5.

Comparison of the two approaches for only area-driven formulation is made. The results are summarized in Table 2.2. The third column lists the number of blocks in each array group. The GSRC cases are relatively large and each case has two array groups. Except Apte, the proposed approach always results in better area usage (or less dead-space). The area advantage from the proposed approach is more obvious for the larger cases from GSRC benchmark. In Figure 2.13, visualization on the floorplanning result on GSRC circuit n100 is provided. Figure 2.13a indicates that the proposed floorplanning approach allows alien blocks inside an array. Further, the proposed approach allows one array to be embedded in another array.

Circuit	No. of Blocks	No. of Arrays	Multi-Prefix			Our Approach	
			Min Area Arrays	Area Usage	CPU(s)	Area Usage	CPU(s)
Apte	9	1(4)	4*1	95.56	32	96.56	3
Hp	10	1(4)	2*2	90.63	22	90.64	16
Xerox	11	1(4)	1*4	96.71	14	97.13	29
Ami33	33	1(8)	2*4	94.63	379	95.42	331
Ami49	49	1(16)	8*2	93.69	713	93.80	231
n50	50	2(16,12)	4*4, 4*3	88.06	71	93.05	42
n70	70	2(49,9)	4*6, 3*3	87.02	149	90.53	465
n100	100	2(36,10)	6*6, 2*5	90.16	461	92.20	259
n200	200	2(56,21)	7*8, 7*3	84.11	3016	92.89	5007
n300	300	2(81,40)	9*9, 10*4	86.25	5429	89.82	6380

Table 2.2: Area-driven floorplanning results on MCNC and GSRC circuits.

The methods for wirelength-driven floorplanning is also tested and the results are



(a) our approach.

(b) the manual prefix method.

Figure 2.13: Floorplan of n100

shown in Table 2.3. For the multi-prefix method, the results of all different array factorizations are listed. Compared to the multiprefix method, the proposed method achieves the same or less wirelength and usually costs less runtime.

Circuit	No. of Blocks	No. of Arrays	Multi-Prefix		Our Approach	
			WL	CPU(s)	WL	CPU(s)
Apte	9	1(4)	473	72	437	65
Hp	10	1(4)	192	39	191	104
Xerox	11	1(4)	442	81	439	445
Ami33	33	1(8)	68	724	68	831
Ami49	49	1(16)	1264	1316	1247	2679

Table 2.3: Wirelength-driven result on MCNC circuits.

At last, an experiment is conducted to see the effect of allowing alien blocks to be embedded between array blocks. This experiment is performed on the GSRC benchmark circuits as they have larger sizes than the MCNC benchmarks. The results are summarized in Table 2.4. One can see that allowing alien blocks among an array often leads to significant area usage improvement. This benefit can also be seen in Figure 2.13.

Circuit	No. of Blocks	No. of Arrays	Disallowing Other Blocks in Array Blocks		Our approach	
			Area Usage	CPU(s)	Area Usage	CPU(s)
n50	50	2(16,12)	87.83	35	93.05	42
n70	70	2(24,9)	80.93	202	90.53	465
n100	100	2(36,10)	86.62	132	92.20	259
n200	200	2(56,21)	80.56	3496	92.89	5007
n300	300	2(81,40)	88.27	5739	89.82	6370

Table 2.4: Comparison between allowing and disallowing other blocks in between array blocks.

3. UNCORE DYNAMIC VOLTAGE AND FREQUENCY SCALING FOR MULTI-CORE PROCESSORS*

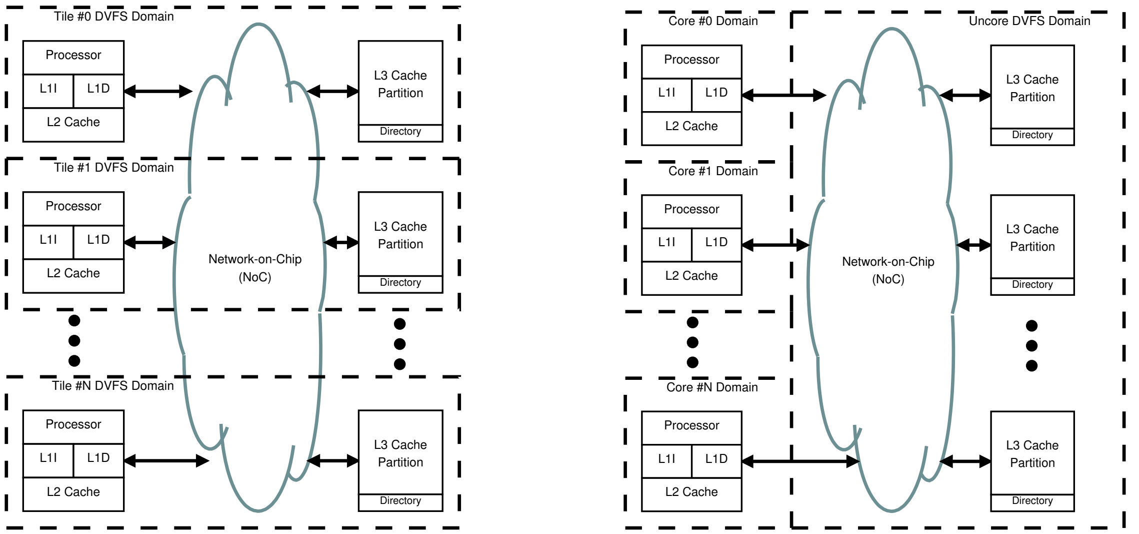
3.1 Introduction

The progress of chip design technology faces two related challenges: power and on-chip communication [35]. A recent study by Google [1] shows that, as power-efficiency improves for server processors, the interconnection network is becoming a major power consumer in the datacenter. Likewise, on-chip communication now forms a power bottleneck in chip multiprocessors (CMPs) given the considerable progress on processor core power-efficiency. Recent designs have resorted to increasing cache size to circumvent the off-chip memory bottleneck. A large cache in turn demands an increase of on-chip communication bandwidth. Indeed, on-chip communication fabrics and shared, last-level caches (LLCs) have grown to occupy a large portion of the overall die area, as much as 30% of chip area in recent Intel chip multiprocessors [25]. Such growth inevitably exacerbates the power challenge. In his speech at International Conference on Computer-Aided Design 2011, Chris Malachowsky, a co-founder of Nvidia, pointed out that the energy expended delivering data on chip has far exceeded the energy in computation operations. Dynamic voltage and frequency scaling (DVFS) is an effective and popular low-power technique. This chapter presents techniques that facilitate efficient DVFS for NoC (Networks-on-Chip).

Networks-on-Chip (NOC) are recognized as a scalable approach to addressing the increasing demand for on-chip communication bandwidth. One study shows

*Reprint with permission from "Dynamic voltage and Frequency Scaling for Shared Resources in Multicore Processor Design" by Xi Chen, DAC 2013, Austin, Copyright 2013 by ACM Digital Library that NOCs can achieve 82% energy savings compared to conventional bus design in a 16-core system [23]. Nonetheless, the NOC still accounts for a considerable portion of total chip power, e.g., 36% in MIT RAW architecture [54]. When the workload is small, some cores can be shut down to save leakage power. However, NOC and LLC need to stay active for serving the small workload and therefore their power proportion is even greater. The power-efficiency of NOC and LLC can be improved by Dynamic Voltage and Frequency Scaling (DVFS), with the rationale that power should be provided based on dynamic need instead of a constant level. DVFS has been intensively studied for individual microprocessor cores as well as the NOC [46, 28, 49, 40, 14, 36, 43, 5, 7]. Much of this prior work assumes a core-centric voltage/frequency (V/F) domain partitioning. The shared resources (NOC and/or LLC) are then divided and allocated to the core-based partitions according to physical proximity. While such configuration allows a large freedom of V/F tunings, the inter-domain interfacing overhead can be quite large. Furthermore, as these shared resources are utilized as a whole, with cache line interleaving homogenizing traffic and cache slice occupancy, per-slice V/F tunings makes little sense.

In this chapter, focus will be placed on DVFS for NoCs of CMPs. Previous work in DVFS for NoCs and CMPs have focused on per-core or per-router DVFS policies, as shown in Figure 3.1a. Unlike much prior work, we consider a realistic scenario wherein the entire NoC and shared last-level cache (LLC) forms a single voltage/frequency domain, separate from the domains of the cores (see Figure 3.1b). We argue placing the shared LLC in one clock domain across the chip is logical because it is, in fact, one large, partitioned structure. Allowing some portions of the address space to see a penalty in performance due to a given LLC bank being clocked slower relative to other portions would impact performance determinism and could make the performance of active threads hostage to the DVFS state of idle threads.



(a) CMP with V/F domains by tile.

(b) Separate V/F domain for the uncore and cores.

Figure 3.1: Logical CMP diagrams highlighting Voltage/Frequency domain partitions.

Unlike the individual cores which are running different threads/programs, the LLC banks have a mostly homogeneous load due to the interleaving of cache lines in the system; in this case, the voltage/frequency domain partitioning like Figure 3.1a can be inefficient. For example, if one core is active and makes many LLC requests while the other cores are idle, then, according to the partition in Figure 3.1a, only the V/F domain for the active core is in high V/F mode. However, this is not sufficient as its data travels in other domains which are in low V/F modes. Therefore, Figure 3.1b is a more reasonable V/F domain partitioning for a CMP with shared LLC. Latency is a critical characteristic in CMP NoCs [13, 12]. Synchronizing across clock domains is expensive in cycles per hop; placing many clock domain crossings in the interconnect makes the design unscalable by imposing a high cost in latency per hop [44].

Performing DVFS on an entire NoC system is, however, more difficult than doing it partially as in previous work such as DVFS for each link [46] or each core [55]. In these cases, voltage/frequency levels are determined by local information (e.g. link congestion [46] or core workload [55]). In contrast, sharing V/F level over the entire network requires information on the load/activity of many different entities. Therefore, an efficient network monitoring technique which accurately collects information from each tile without impacting performance of the network by flooding it with status information messages is required.

This work focuses on a realistic scenario where the entire NOC and LLC belong to a single V/F domain. As such, the interfacing overhead can be largely prevented and there is a coherent policy covering the whole of these shared resources. To the best of our knowledge, only two works [28, 7] have addressed DVFS for such scenario. Liang and Jantsch propose a rule-based control scheme, using network load as the measured system performance metric [28]. Although Liang et al.'s work demonstrates the benefit of DVFS, there are two critical hurdles that have not been well solved. First, the impact of the NOC/LLC V/F level on the chip energy-performance tradeoff is not straightforward. These prior works shy away from this problem by evaluating only parts of the chip system. Second, the chip energy-performance tradeoff is dynamic at runtime while the controls of these prior approaches are based on fixed reference points.

In this paper, we present remarkable progress on overcoming these hurdles. New methods are proposed and investigated. First, a throughput driven controller with dynamic reference point is examined. Second is a model assisted PI controller based on a new metric that bridges the gap between the NOC/LLC V/F level and the chip energy-performance tradeoff. The last one is a PI controller with a dynamic reference point based on the new metric.

In this chapter we address these questions. The key contributions of this work are as follows:

- We introduce several new uncore status metrics to predict the impact of DVFS policy on system performance.
- We propose a novel, extremely low overhead, uncore status monitoring technique. This technique is composed of the following: 1) Per-node metric sampling, 2) Passive in-network status encoding, no extra packets needed, 3) Metric extrapolation to properly scale value weights.
- We introduce an uncore DVFS policy based upon PID (Proportional-Integral-Derivative) control with a dynamic reference point based on the new metric.

These methods are evaluated in full system simulation on the PARSEC benchmarks [3]. The experimental results show that our techniques can reduce NOC/LLC energy by $\sim 80\%$ and $\sim 50\%$ compared to a baseline fixed V/F level (no power management) and the state-of-the-art prior work [7], respectively. Simultaneously, we achieve our target of $\leq 5\%$ performance loss. Compared to the static reference point design [7], the energy-delay product is reduced by 56%.

3.2 Preliminaries

This section introduces the basics of shared, distributed, last-level caches and their NoC interconnect in CMPs (which we collectively describe as the "uncore"). We then introduce the basic concepts in NoC performance monitoring and discuss the power and performance constraints on the uncore. Finally we discuss power management using DVFS in the uncore.

3.2.1 Related Work

Shang et al. present a pioneering work on dynamic voltage scaling in NOCs [46]. They tune voltage levels for individual links separately according to their utilization history. Mishra et al. propose DVFS techniques for NOC routers [36]. They monitor input queue occupancy of a router, based on which the upstream router changes its V/F level. Son et al. perform DVFS on both CPUs and network links [49]. They target to parallel linear system solving and the V/F levels are decided according to task criticality. Guang et al. propose a voltage island based approach [14], where router queue occupancies are monitored and island V/F levels are tuned accordingly. Rahimi et al. take a similar rule-based approach according to link utilization and queue occupancy [43]. Ogras et al. propose a formal state-space control approach also for voltage island based designs [40]. Bogdan et al. introduce an optimal control method using a fractional state model [5]. In drowsy caches, dynamic voltage scaling is applied at certain cache lines at a time for reducing leakage power [10]. To the best of our knowledge, there is no published work on DVFS which focuses on shared caches in multicore chips.

Liang and Jantsch present a DVFS controller that attempts to maintain the network load near its saturation point, where the load is the number of flits in the network [28]. At each control interval, its policy is to increase (decrease) network

V/F level by one step if the network load is significantly greater (less) than the saturation point. This method neglects the fact that chip performance also depends on the distribution besides the amount of network load. A non-uniform distribution may imply certain congestion hot-spots, which may significantly degrade chip performance. Even with consideration of the distribution, network load does not always matter. For example, many *store* operations induce large network load but they are not critical to the overall chip performance. Liang and Jantsch's method may respond slowly for bursty traffic as only one step V/F change is allowed in each control interval.

In this chapter, we will first introduce a PI controller based on AMAT [7]. AMAT, as we formulate it, including the effects of the private caches, NOC, LLC and off-chip memory, reflects network load and contention inherently, providing an approximation of the latency seen by typical core memory references. Therefore, we think it captures a more global system effects than a purely network-based metric such as Liang and Jantsch's approach [28]. Our AMAT metric, however, does not truly separate out the LLC and NOC utility to the core from the effects of the off-chip memory. Thus, applications which frequently miss in the LLC, causing off-chip memory accesses, will lead to high AMAT values, and thus high LLC and NOC frequencies, despite the LLC utility being low in this case. Later in this work, we will improve it and propose another more accurate metric called critical latency. Chapter 3's DVFS policy uses a PI controller, subsuming that of Liang and Jantsch. This work also describes implementation techniques on how to monitor system metric of multicore with low overhead.

In this work we assume a low-overhead, in-network monitoring scheme is used for the transmission system status information to the power controller.

Both Liang and Jantsch's [28] suffer from another weakness, they have no system-

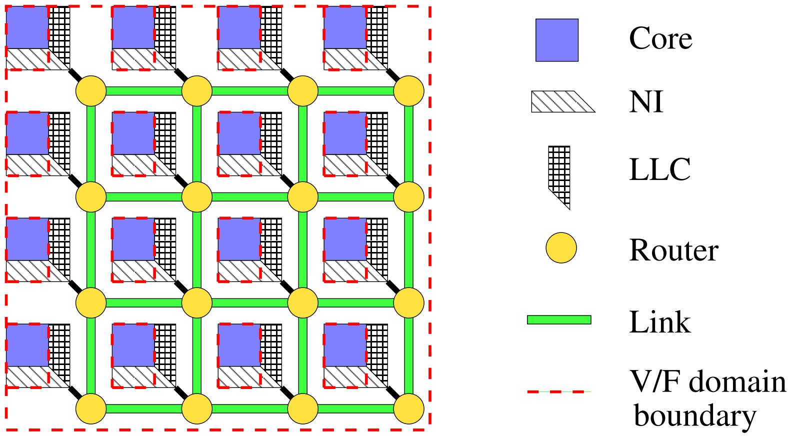


Figure 3.2: A multicore processor design where the uncore (NOC+LLC) forms a single V/F domain.

atic approach to decide the reference point for their controllers. The reference point is decided empirically according to offline simulations. However, it is very difficult, if not impossible, for a fixed reference point to be appropriate for different kinds of applications. Moreover, neither work provides performance results from full-system simulation to validate their approach.

3.2.2 Problem Description

We consider a common case in multicore processor design where the entire chip is composed of an array of tiles. Each tile contains a processor core and private caches. The communication fabric is a 2D mesh NOC with one router residing in each tile. There is a shared LLC partitioned into slices and distributed uniformly among tiles. The NOC and the LLC together are referred to as the *uncore* system in this work. The system is illustrated in Figure 3.2 where NI denotes Network Interface with cores.

The problem we attempt to solve is formulated as follows.

Uncore dynamic voltage and frequency scaling: *within a set time window, find the voltage/frequency level for the uncore such that the uncore energy dissipation is* *minimized while the chip performance, in terms of total application runtime, has negligible or user-specified degradation.*

Please note that our problem formulation has a key difference from previous works on NOC DVFS, which try to optimize the performance of NOC itself, not considering directly its utility to the system. In contrast, we have the more challenging goal of optimizing uncore energy under the constraint of entire system performance. We present the first work we are aware of in this area with such formulation. The uncore energy we try to minimize includes both dynamic and leakage energy and their models are well-known.

3.2.3 CMP Uncore Basics

Typical CMPs are composed of a set of cores, consisting of the processor and private lower level caches (level-1 and sometimes level-2), along with an "uncore". The uncore portion of the die refers to all the integrated subsystems on the chip except the cores. More precisely, the LLC, the routers and links of NoC, integrated memory controller, integrated I/O controller etc. constitute the uncore. In other words, the uncore enables communication between the processing cores, and with the LLC, off-chip memory, I/O devices, graphics core and accelerators, if any. Therefore, any miss in the local caches of a core will result in an "uncore request". Finding the location of the requested cache line, transferring the cache line to the core or the memory controller as well as controlling the global state of the cache line are all managed by the uncore. In modern LLCs, the banks of the LLC are partitioned and distributed such that a portion of the LLC is co-located with each core. This LLC arrangement has the advantage of improving performance over the prior, monolithic cache designs.

In our baseline design, we assume coherence between the private caches in the cores is maintained via a distributed directory cache in the uncore. The NoC interconnecting the uncore and the cores primarily carries memory system traffic. Particularly, the NoC carries lower-level cache spills and fills, lower-level cache coherence messages, and LLC cache spills and fills. We assume that LLC cache set indices are spread about the partitions of the LLC in a round robin fashion, to ensure that each partition receives approximately the same amount of traffic and no single partition becomes a hotspot.

3.2.4 Uncore Power and Performance Implications

The uncore consumes a significant fraction of the whole chip power due to the relatively large proportion of the chip area it consumes. In particular, the uncore power can be broken down into static and dynamic power components, corresponding the constant power drain due to leakage, and the usage dependent power consumption due to gate switching respectively.

Static power in CMOS circuits is mainly caused by subthreshold leakage, gate leakage and junction leakage current, and can be expressed by

$$P_{static} = V_{dd} \cdot I_{sub} + V_{dd} \cdot I_{gate} + V_{dd} \cdot I_{junc} \tag{3.1}$$

The supply voltage affects static power not only through the multiplication factors above but also through the leakage current. For example, leakage current is described by [21]

$$I_{sub} = \mu C_{ox} V_{th}^2 \frac{W}{L} \cdot e^{\frac{V_{GS}-V_T}{n \cdot V_{th}}} \tag{3.2}$$

where μ, C_{ox} and n are technology-dependent parameters, W and L are transistor dimensions, and V_{th} is the thermal voltage. Since the gate-source voltage V_{GS} is decided by the supply voltage, the leakage current is directly affected by the supply voltage. Overall, voltage scaling can help to reduce static power, taken together the relationship between static power and V_{dd} is roughly linear.

Dynamic power dissipation for CMOS circuits is given by

$$P = \alpha \cdot C \cdot V^2 \cdot f \tag{3.3}$$

Although the activity factor (α) for the uncore is not necessarily high, its total area and capacitance (C) can be large. Similarly the leakage power, a growing problem in future VLSI process technologies, is also proportional to the area of uncore. Equation (3.3) also includes two interrelated components – the voltage squared (V^2) and frequency (f). For a given design, increasing the voltage makes transistors to switch faster, allowing the chip to operate at a higher frequency. Conversely, lowering the voltage forces a decrease of the clock frequency to meet timing constraints. *Dynamic voltage and frequency scaling* (DVFS), is a well-known technique which leverages this relationship to lower dynamic power consumption. Lowering the voltage has a quadratic effect on the dynamic power of the circuit being lowered, though this comes at the cost of some performance due to the required decrease in frequency. By lowering the voltage and frequency to match but not exceed the demands of the application, a good DVFS policy can achieve substantial power savings.

Achieving power savings through DVFS in the cores is a comparatively simple problem, considering that the information needed to select an appropriate voltage and frequency are available in a localized place, the CPU core itself (e.g. from performance counters within that core). Determining an appropriate DVFS state for the uncore is a significantly more difficult problem. Because the uncore consists of the LLC and network, the relative criticality of the uncore's performance to the performance of the system is highly dependent on the application's demand for LLC data and inter-thread communication. Applications which are mostly L1 cache resident place little performance pressure on the uncore and the uncore can safely run at a relatively low frequency, while those with frequent L2 cache misses place high demands on the uncore and require the uncore to run at a high frequency.

For purposes of studying uncore DVFS polices, we decompose the problem into three major components. First, because the uncore consists of two very different components, the LLC and NoC, it is unclear what performance metrics are appropriate as an input to the DVFS policy. Second, because the uncore is distributed across the chip, a mechanism must be developed to monitor the status of the uncore performance metrics and inform the DVFS policy. Finally third, once the inputs have been defined and arrive at the DVFS controller, an appropriate policy must be developed. We explore these components in the remainder of this section.

3.2.5 Options for DVFS Policy

Broadly speaking, there are two categories of approaches for DVFS: open-loop control and closed-loop control. Open-loop control decides control variables based on the current system state and a system model obtained either theoretically or through machine learning. The behavior of a multicore system is typically very complex. Even if a decent model is available, its behavior depends on environmental parameters that are highly dynamic and thus are very difficult to reliably predict.

Closed-loop control adjusts control variables with consideration of observed output. It includes several options: rule-based, PID (Proportional-Integral-Differential) control, state-space model-based control and optimal control. In a rule-based approach, a set of *ad hoc* rules are determined, such as simply increase (decrease) uncore V/F level if the network performance is poor (good) according to given metric. Based on the observed output error, PID control adjusts the system to track a target output. State-space model-based control formally synthesizes a control policy [40]. Optimal control [5] decides system operations by solving an optimization problem and sometimes can be applied with a state-space model.

We adopt PID control due to its simplicity, flexibility, low implementation overhead. In a discrete-time system, where the output is a time-varying function y_i and the control variable is u_i for control interval i, the error function is defined by $e_i = y_{ref} - y_i$, where y_{ref} is the reference point or target output. Since differential control is sensitive to data noise, we drop the differential term. A PI controller can be described by:

$$u_i = u_{i-1} + K_I \cdot e_i + K_P \cdot (e_i - e_{i-1}) \tag{3.4}$$

where K_P and K_I are constant parameters. Please note that the problem described in Section 3.2.2 is an optimization problem, which implies a dynamic goal instead of a steady target in typical PI controls. To bridge this gap, it is very important to use a dynamic reference point, as shown in subsequent sections, instead of fixed reference as in [7].

In this paper, we address DVFS for the uncore as a whole. We will demonstrate that there is a large opportunity for uncore power saving with an accurate but low cost monitoring technique to fetch network information, meanwhile using a simple but effective control algorithm to adjust uncore voltage and frequency based on the monitoring result.

3.3 Uncore Performance Metrics

Monitoring is a very important part in the power management process. A good monitoring technique can not only minimize the hardware overhead but also mitigate the system power consumption. On the contrary, the inappropriate monitoring technique may bring extra burden to the system. In this work, we use critical latency(CL) as the system metric, piggy-back as the information propagation technique and extrapolation technique to compute collected data.

3.3.1 Previous Uncore Performance Metrics

Network communication traffic load and performance can be described via several potential networks metrics. In this section we explore some potentially suitable indicators that may predict network load.

Queue occupancy and crossbar demand: Routers typically have queues, or input FIFOs for temporary storage of data. The occupancy level of a queue naturally indicates local congestion. For example, an emptying queue may suggest the receiver processing speed is too fast compared to the sender. While a filled queue may suggest the router is congested and that the processing speed of the receiver is not great enough. A stable queue occupancy would probably suggest a perfect match between receiver speed and sender speed. Hence, queue occupancy has been widely used in adaptive routing [11, 22] and DVFS for individual links [46, 40, 43]. The overall congestion status of a network can be estimated by monitoring queue occupancy of all routers. We investigate the queue occupancy during the simulation, the result is showed in Figure 3.3. From the figure, we can see that the middle four have the highest demand rate, which is what we expected because statistically, if in a uniform traffic, the middle four routers where crossbars locate experience the most conjected traffic.

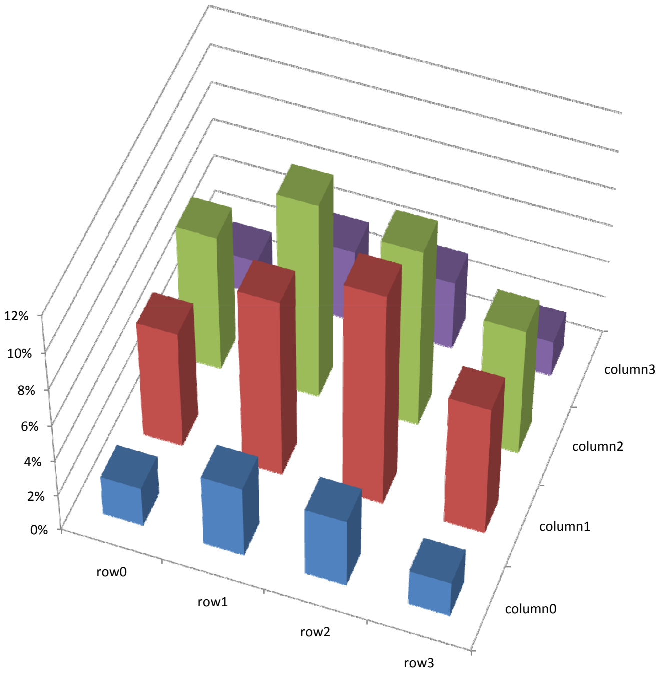


Figure 3.3: Crossbar demand of Blackscholes.

A different and slightly overlapping metric for congestion is crossbar demand, which is the number of active requests to an output of a router [11]. Many requests to an output imply a convergent traffic patten, which is likely to become a bottleneck. While queue occupancy measures the degree of downstream congestion, crossbar demand indicates local congestion. In either case, estimating the performance of the network as a whole requires monitoring many if not all routers.

Average per-request latency (APRL): While queue occupancy and crossbar demand reflect network congestion, packet latency is a direct measure of network performance. Packet latency is defined by the time span from a packet request being made to the arrival to its destination. Although low packet latency is obviously preferred, it is not directly obvious what should be the absolute goal in packet latency. Different types of workload can be expected to produce different average hop-counts and average packet lengths and hence different nominal packet latencies. Figure 3.4

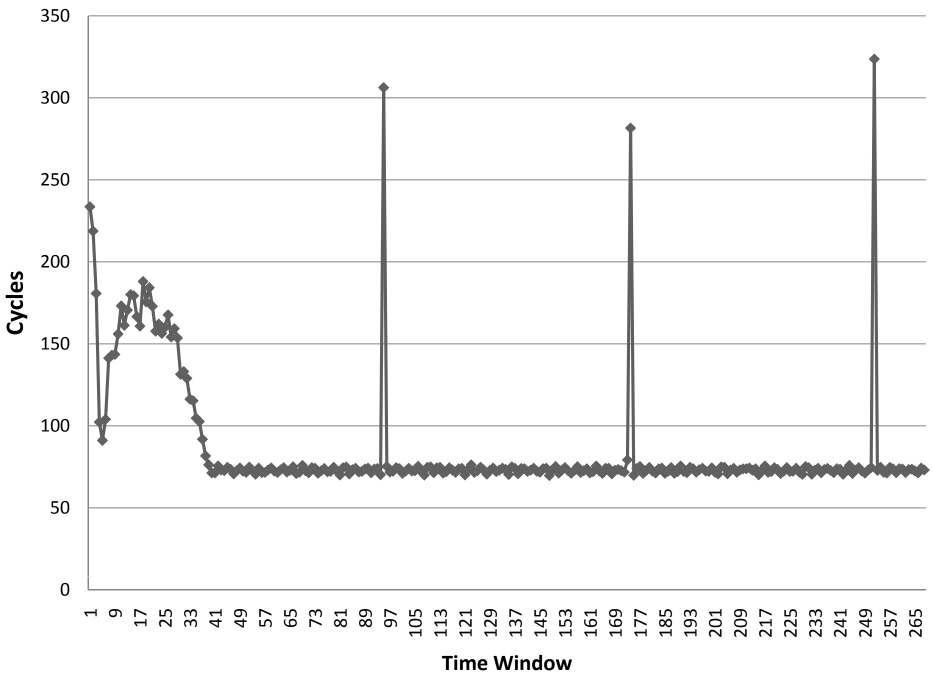


Figure 3.4: Average per-request latency of Blackscholes.

is an example of Blackscholes benchmark.

Average per-hop latency (APHL): Average Per-Request Latency can roughly reflect the network traffic condition. However it suffers the following drawbacks: 1) It ignores the difference of the number of hops between each request. 2) The packet size of different request also varies. For the convenience of a normalized comparison, we suggest average per-hop latency (APHL), which is the average latency a flit incurs as it traverses each router along its path through the NoC. For any specific system, there is an unique minimum latency for one hop which is determined by the link delay plus the router pipeline latency. By removing distance traveled and serialization latency, APHL gives a traffic pattern independent metric of network load as Figure 3.5.

As such, APHL's deviance from a given NoC's inherent minimum per-hop latency serves as a clear target for the DVFS tuning. Figure 3.6 is an example of APHL.

Figure 3.7 compares APRL and APHL and we can see that after ruling out the

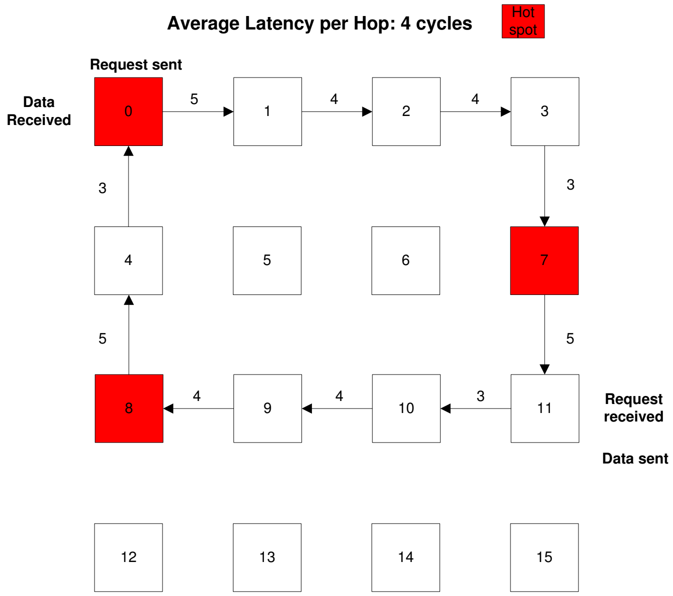


Figure 3.5: Average per-hop latency working principle.

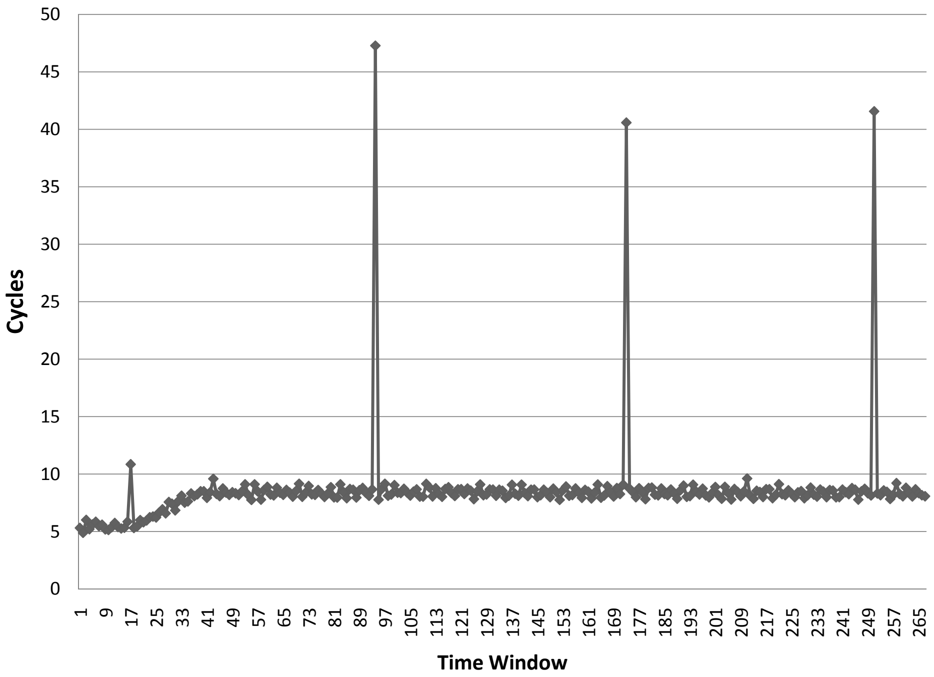


Figure 3.6: Average per-hop latency of Blackscholes.

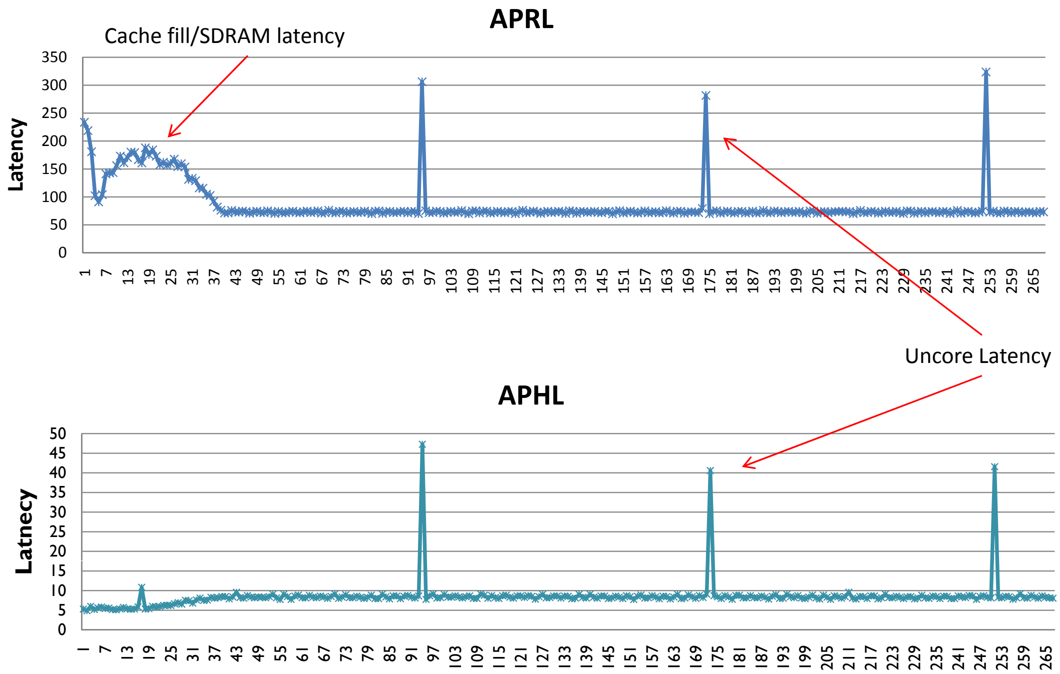


Figure 3.7: Comparison of APHL and APRL.

drawbacks of APRL mentioned above, ALPH can better reflect the network traffic condition.

3.3.2 Average Memory Access Time – AMAT

While the previous metrics merely provide the information of current network congestion, we also propose average memory access time (AMAT) as a metric which provides not only the current network status but also the demand for its performance. When a cache miss occurs on the private caches, the NoC is used to fetch the missing cache line from LLC. Thus, NoC performance translates to memory operation latency. Experimentally we determined that for small AMAT increases, IPC (instructions per cycle) of the cores decreases approximately linearly with AMAT with a slope of .5. Consequently, AMAT provides a good index of required uncore

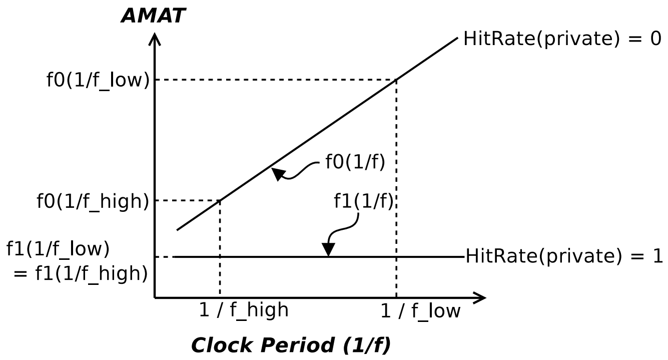


Figure 3.8: AMAT with respect to uncore clock period.

performance. Equation (3.5) shows a simplified uncore AMAT formula:

$$\begin{aligned} AMAT = {} & HitRate(private) \times Latency(private) \\ & + (1 - HitRate(private)) \times Latency(uncore) \end{aligned} \tag{3.5}$$

where $HitRate(private)$ is the aggregate hit rate on the private caches (L1 and L2) and $Latency(private)$ is the average access time on those caches. $Latency(uncore)$ is the average access time to the shared LLC (ie. access time to the L3 slice on a remote tile), plus the latency of the memory controller if the required cache block is missing in LLC. For simplicity we assume $Latency(uncore)$ linear with the clock period $(1/f)$ of the uncore, such that Figure 3.8 shows a representation of AMAT.

Figure 3.8 shows two extreme cases. f_0 depicts the AMAT with respect to the clock period when $HitRate(private) = 0$, while f_1 shows the AMAT when $HitRate(private) = 1$. f_0 represents the case the most of the memory access results in the private cache miss such that the missing block must be transferred over the network. In this case, the AMAT is highly dependent on the uncore performance, hence decreasing the uncore frequency via DVFS has a strong negative impact on system performance and should be avoided. On the other hand, f_1 represents the case where all memory accesses are served by the private caches. In such a case, decreasing uncore frequency has no impact on system behavior, hence should be done to achieve power savings with little impact on performance. Thus, it is desirable for the DVFS controller to account for AMAT in deciding whether increasing the frequency is worth the increased power or not.

3.3.3 Composite Uncore Metric – Critical Latency

The throughput-driven DVFS described in the work [7] largely solves the dynamic reference problem. Although the metric used by the technique captures the performance of the uncore well, it is still oblivious of the overall chip performance. Ideally, the metric should reflect both the uncore performance and its criticality to the overall chip performance. We therefore define a new metric - *critical latency*, expressed by

$$\Gamma = \eta \cdot \lambda_U \tag{3.6}$$

where η is the criticality factor and λ_U is the uncore latency.

The uncore latency should account for the latency in the network and LLC. Subtracting the reply return from request inject time, one can obtain the packet latency; however, this latency may contain off-chip memory latency in the event of an LLC miss. This memory latency should be excluded from consideration since it is not affected by the uncore DVFS. In certain cases, the LLC miss can be very high and the overall data latency is dominated by the memory latency. In this case, increasing uncore V/F does not help to improve chip performance while causes more power dissipation [†]. The uncore latency can be described by

$$\lambda_U = \frac{(\sum_{j=1}^{N_{packets}} \lambda_{packet,j}) - \lambda_{Mem} \cdot N_{LLC\_Misses}}{N_{packets}} \tag{3.7}$$

[†] We intentionally do not differentiate between packets which lead to coherence traffic and those which do not. Coherence related traffic can increase LLC latency and is sensitive to uncore V/F state.

where $\lambda_{packet,j}$ is the total round-trip latency for packet j, λ_{Mem} is the memory access latency, N_{LLC_Misses} is the number of LLC misses in a control interval and $N_{packets}$ is the number of packets in the same interval.

In microprocessors, both *store* and *load* data induce network traffic and potentially LLC access. These two types of packet requests have different impact on the overall chip performance. Often, a long latency *load* can block the execution of instructions that need the data, and therefore is performance critical. In contrast, a *store* operation can often run in parallel with subsequent instructions and is rarely critical. Thus, the criticality factor of uncore performance includes $Loads\_Fraction$, which is the number of *load* instructions per cycle. We scale $Loads\_Fraction$ by the L1 miss rate, assuming L1 is the only level of private cache, because loads which hit in the private caches never enter the uncore and are not affected by uncore performance. Therefore, we have

$$\eta = L1\_Miss \times Loads\_Fraction \tag{3.8}$$

3.4 Uncore Monitoring Technique

In the prior work, network status information has been predominately used locally, particularly for adaptive routing [22, 53, 48, 47], and for localized DVFS policies [46, 40, 43]. Gratz et al. [11] and Ma et al. [34] propose small, light-weight status information networks to provide deeper visibility into the NoC and hence enable better adaptive routing decisions. In these cases, the performance metric obtained from the local router is assumed to provide enough information to enable local decisions, such as which output port to send a packet to, or what DVFS setting for a given router. These techniques, however, provide a limited view of the status of the network as a whole, either decreasing exponentially with distance [11] or limited to one bit of data per node in only the orthogonal directions [34]. There are some other monitoring techniques to inform DVFS policy in NoCs. Yin et al. [58] propose dedicated links and virtual channels for collecting and monitoring system status information; this approach is expensive in terms of design time to create a dedicated interface, power for these links and area for the logic associated. Rahimi et al. [43] propose to monitor network performance based on link utilization; this approach is reasonable for fine-grained, local link DVFS control but does not scale to a full-chip shared resource.

3.4.1 Uncore Status Monitoring

Ideally for global decisions, one would like continuous monitoring of uncore status, whether per-packet statistics to calculate APHL, or per-core memory statistics to calculate average AMAT. Such complete, active monitoring entails a calculation overhead at every tile, which includes counting starting/arrival time of packets, calculating the average, etc. Moreover, the status obtained at each tile must be regularly sent to the central power controller to set DVFS policy, consequently causing increased traffic and congestion.

To avoid increasing traffic and congestion within the primary NoC we propose to leverage unused space in the header flits of existing packets to “piggyback” network status information. This approach has the advantage of being scalable to larger networks because no extra networks or links must be designed. Furthermore, it is passive, ie. it does not perturb the network with extra status packets. A potential disadvantage to this approach is the non-determinism of status message delivery. As we will show, however, this does not significantly degrade the accuracy of this approach.

We propose to estimate current AMAT to provide feedback to the DVFS controller. Each tile keeps track of its status information and a monitor collects the information for AMAT computation. To minimize hardware overhead, we use the existing NoC infrastructure to convey the information, to avoid increasing traffic and congestion, we leverage unused space in the header flits and employ passive monitoring rather than any active system. We propose a single monitor for our 4×4 network; experimental results show this is sufficient for this size network. We assume the uncore contains a Power Control Unit (PCU), which is a dedicated small processor for chip power management as in Intel’s Nehalem architecture [26].

3.4.2 Data Collection

A holistic view of the full system is largely missing from much of the prior work in DVFS policies for the on-chip interconnect. In interconnect research, researchers typically focus on the network, trying to minimize its power consumption or increase its performance. This approach misses the strong connection between processing elements and the interconnect, wherein small changes in the network may have a strong impact on the whole system. For example, when executing a given application,

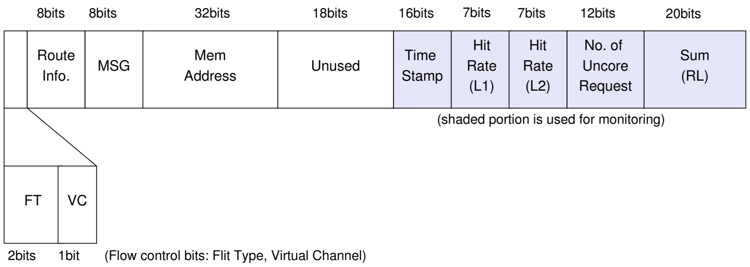


Figure 3.9: Header flit bit fields.

the cores may be memory bound, requiring data fetched from the memory through the network to make progress. If the frequency of the network is lowered even slightly, the processing will slow down accordingly, leading to wasted time and energy while the processors idle. Alternately, however, if the program is computation centric or if the data required fits within the upper-level, core-private caches, requested data is not as urgently important to the processing unit. Therefore, slowing down the network frequency will not significantly affect processing performance much, meanwhile achieving a lot of energy saving. Thus, it is critically important to view interconnect performance demands within the larger context of system performance.

In our proposed technique, each tile maintains statistics on its private, lower-level cache's behavior necessary to compute AMAT. These statistics must be sent to a central, monitoring node for overall AMAT estimation and DVFS policy generation. Rather than injecting more packets into the system, increasing traffic and congestion, each tile "piggybacks" its status information into every packet injected into the network. Under the assumption of 128-bit wide links, and 64-byte cache blocks, we find there are ~ 64 unused bits in the header flit or single flit packets. We leverage these unused bits to encode the status information as shown in Figure 3.9. Here, *Time Stamp* denotes the time the packet is generated. $HitRate(L1)$ and $HitRate(L2)$ show the hit rate of L1 and L2 caches during the time window. $No.ofUncoreRequest$ and

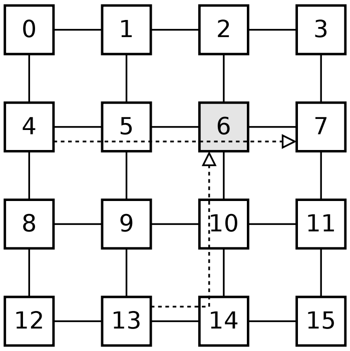


Figure 3.10: NoC layout; the monitor resides at tile 6

$Sum(RL)$ correspond to the number of L2 requests sent into the Uncore and the sum request latency during the time window, respectively. The $Sum(RL)$ is the accumulated time span required for L2 requests into the uncore, and includes the access time of the L3, cache coherence resolution time and main memory access time on L3 misses. The additional hardware cost to maintain this status information is discussed in Section 3.6.1.

Figure 3.10 shows an NoC layout illustrating our monitoring technique. Instead of adding monitors to each tile, we employ a single monitor that collects the data from the whole network. We chose tile 6 as the location of the monitor intuitively because a central location provides the best vantage point to passively collect the desired statistics (an assumption experimentally verified). In addition, the monitor tile should be near the PCU so that the overhead of interconnect between the monitor tile and the PCU is negligible. The monitor grabs status information from all passing packets (e.g. a packet traveling from tile 4 to tile 7) or packets bound to tile 6 (e.g. a packet from tile 13 to tile 6). Data collection is passive, a potential downside of this approach is non-determinism of message delivery, as we will discuss in the next section.

3.4.3 Overall Metrics Computation

Once a packet arrives at the monitor, the source node's status information, encoded in the packet's header, is handed over to the PCU (Power Control Unit). The PCU is in charge of computing collected data on the packet's arrival.

This propagation technique is applicable to both AMAT and critical latency or some other potential metrics that require core information. Here we use AMAT as an example to elaborate the propagation process. Critical latency can use a similar method.

The AMAT for each tile is computed as per Equation 3.9 based on the information gathered.

$$\begin{aligned} AMAT &= \mathit{HitRate}(L1) \times \mathit{AccessTime}(L1) \\ &\quad + (1 - \mathit{HitRate}(L1)) \times \mathit{Latency}(L2) \\ \mathit{Latency}(L2) &= \mathit{HitRate}(L2) \times \mathit{AccessTime}(L2) \\ &\quad + (1 - \mathit{HitRate}(L2)) \times \mathit{Latency}(\mathit{uncore}) \\ \mathit{Latency}(\mathit{uncore}) &= (\mathit{Sum}(RL))/(\text{No. of Uncore Request}) \end{aligned} \tag{3.9}$$

Note that Eq. 3.9 is a more detailed version of Eq. 3.5, where $\mathit{HitRate}(\mathit{private})$ and $\mathit{Latency}(\mathit{private})$ are decomposed into their constituent L1 and L2 components. The $\mathit{AccessTime}$ values are constants for a given L1 and L2 cache design and hence need not be sent in the header. Also note, the $\mathit{Sum}(RL)$ the sum of round trip time for Uncore requests, including NoC latency, L3 access time, cache coherency latency, as well as main memory latency when L3 misses occur.

Our DVFS policy requires the overall, chip-wide AMAT as an input. This overall AMAT is computed for every time window of 50000 core cycles. To calculate overall AMAT, we use a weighted average of the per-node AMAT, with respect to the number of memory instructions issued by that tile during the window. Upon packet arrival or traversal of the monitor tile, the PCU calculates the AMAT for the source tile of the packet and stores it in a table with 16 entries; one for each tile. Memory instruction count is also stored in the table. At the end of each time window, the PCU, using the memory instruction counts as weights, computes the weighted average of AMAT across all tiles as the overall AMAT. At the end of every time window, all entries are reset to 0; entries not updated are excluded from overall AMAT computation. Figure 3.11 depicts this process. The graph shows the AMATs of $Tile_i$ and $Tile_j$, in a two tile system. The cross marks on the time line denote packet arrivals, and the circles are the AMAT for the corresponding tile to be stored in the table. The numbers above the circles denote the memory instruction count. The triangles are the final AMAT values used in overall AMAT calculation. In "Window 1", $Tile_i$ sends two packets, but the data from the first packet is discarded as the second one overrides it. At the end of this window, the PCU first calculates the per-tile AMAT for $Tile_i$ and $Tile_j$ of 80 and 65 respectively from the status information in the final packet from each. It then determines there are 120 memory instructions in $Tile_i$ and 60 in $Tile_j$, and these values are used in the weighted average overall AMAT thus $(80 \times 120 + 65 \times 60)/(120 + 60)$. In "Window 2", the PCU receives no packet from $Tile_i$ thus the overall AMAT is computed only from $AMAT_j$ and it is 120. The system clock is also reset at the end of time window, thus if a packet's time stamp is later than the current system time, the packet is discarded as it is from the previous window.

We refer the above method as "Naïve" as it calculates AMAT without accounting for packet arrival time. Unfortunately, this passive monitoring method does not guarantee that each tile's statistics are current. As the final value is evaluated at the end of each time window, it is better to have a packet from a tile near the end

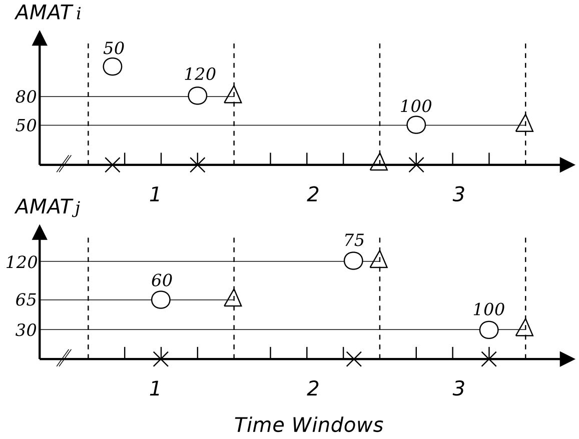
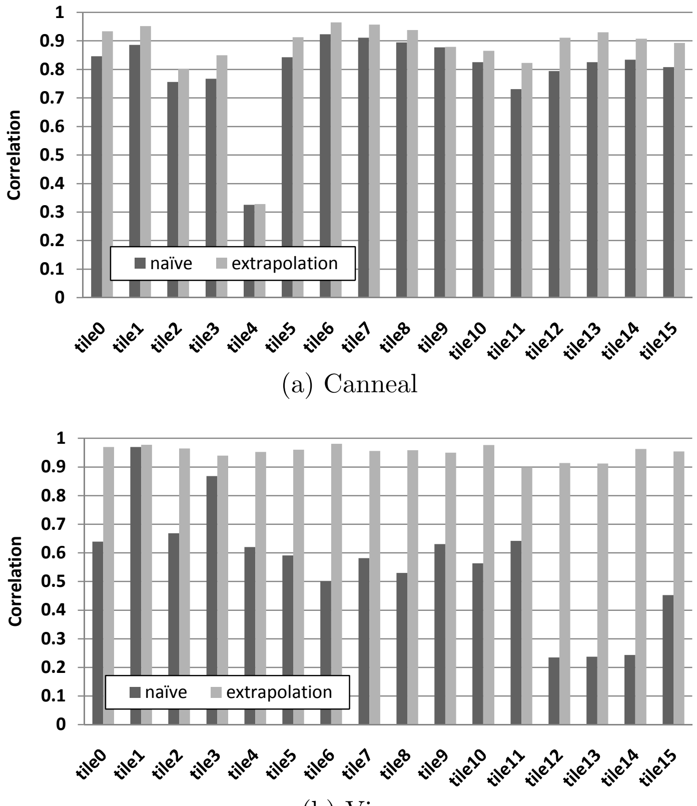


Figure 3.11: Overall AMAT computation

of the window. For example, at “Window 3” in Figure 3.11, $AMAT_i$ and $AMAT_j$ have the same weight as the numbers of memory instructions carried over are the same. However, in the end of the time window, it is very likely that $Tile_i$ eventually has more memory instructions than $Tile_j$ since the count had been determined much earlier. Thus, we introduce a method of linear extrapolation to correct this bias. We assume that the number of memory instructions linearly increases in a time window. With the fact that the memory instruction count is 0 at the beginning of a time window, we can estimate the count in the end of it using a sampled count at any location of the window. We use the “Time Stamp” in the packet to define the relative location within the time window, and that needs to be stored in the table. By this “extrapolation”, in “Window 3”, the effective memory instruction count of $Tile_i$ becomes 400 while that of $Tile_j$ becomes 133. Finally, the overall AMAT becomes 45 while in “Naïve” it is 40.

Figure 3.12 compares the performances of the “Naïve” and “Extrapolation” methods. The figure shows the correlation between actual and computed overall AMATs for monitor tiles, tile0-tile15 (tiles numbered as shown in Figure 3.10). Generally,



(a) Canneal

(b) Vips

Figure 3.12: Overall AMAT from each tile's perspective

higher correlations across all tiles indicates a given method provides a better estimate of overall AMAT. For *Canneal* we see that "extrapolation" generally results in a more accurate overall AMAT than "Naïve." For *Vips*, much higher correlation is achieved by "extrapolation" method. *Vips*'s traffic pattern is more highly skewed and hence requires "extrapolation" to produce reasonable results. Note that in both cases tile6 shows the highest correlation among the tiles, thus we select tile6 as our monitor tile which also matches our assumption that central location provides better visibility.

The composition of critical latency(CL) is similar to that of the AMAT, so the computation procedure of AMAT can also be applied to critical latency.

3.5 Control Policy for the Uncore System

3.5.1 PID-Based DVFS Policy

We choose to implement a DVFS control scheme based on PID (Proportional-Integral-Derivative) control. Compared to rule-based approaches [17, 45], PID control can easily adapt to various application scenarios. Its computation cost is significantly less than learning-based methods [18]. It has been applied for DVFS in processor cores [55].

As the broad application of the propagation technique in Section 3.4.3, the PID control system is also capable of taking different metrics as system inputs and references. We take AMAT as an input example here, but critical latency can also be used in the control system.

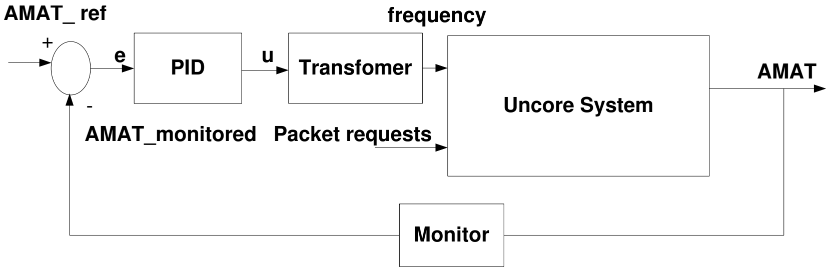


Figure 3.13: PID system diagram with AMAT as system metric.

The block diagram of PID control system is shown in Figure 3.13. The controller takes two inputs: the reference and monitored AMAT. The reference AMAT is the control target and can be obtained from empirical data. The control output is updated once per control interval monitoring). The difference between the two inputs is the error function $e_j = AMAT_{ref} - AMAT_j$ where $AMAT_j$ is the AMAT observed at control interval j. The controller calculates the control output u according to

$$u_j = u_{j-1} + K_I \cdot e_j + K_P \cdot (e_j - e_{j-1}) \tag{3.10}$$

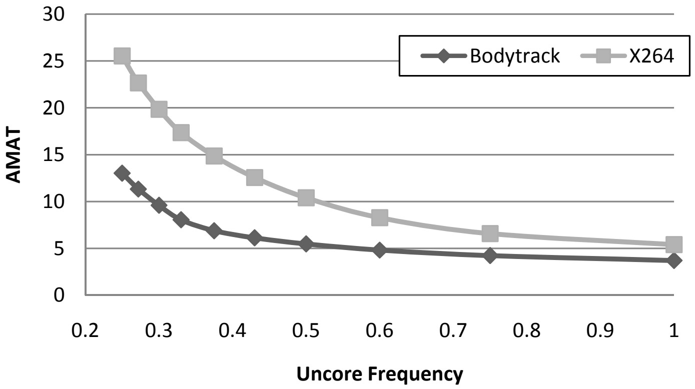


Figure 3.14: AMAT versus uncore frequency.

where K_I and K_P are constant coefficients. Note we only implement the Proportional (P) and Integral (I) terms in our controller, as this is simpler and often more robust than including the Derivative (D) term [55]. Control output u is converted to a V/F setting for the uncore system. In general, AMAT is a nonlinear function with respect to uncore frequency f. We perform a transformation of $u = 1/f$ such that AMAT is approximately a linear function of u.

3.5.2 Latency-Based PI Control with Dynamic Reference

The model-assisted controller is conceptually superior to that proposed by in [8], where one fixed reference point is used. The improvement, however, can be limited as the model-based prediction may be inaccurate, as the number of optional reference points is still limited. If we add many more options, the model would manifest a fine-grained guidance to the PI controller. In fact, it virtually replaces the PI controller. Such model based approach heavily relies on the assumption that the system characteristics of two adjacent control intervals are highly correlated, which is not always true. Hence, using too many optional references is risky as the prediction is likely to be wrong. To fix this problem, we examine an alternate PI controller that allows a truly dynamic reference point. The reference point should be the target for the control. For the uncore latency λ_U, defined in Equation (3.7), we can determine our desired target. The λ_U mainly consists of the propagation latency in the network, the serialization latency, the queuing latency in the network and the LLC access latency. When there is no congestion, the queuing latency should approach zero. Ideally, we want to keep the network lightly loaded. Thus, we can define a reference uncore latency as

$$\lambda_{ref} = (1 + \rho) \cdot (2 \cdot (\lambda_{hop} \cdot N_{hops} + L_{packet}) + \lambda_{LLC}) \tag{3.11}$$

where ρ is a constant selected to be 0.1, λ_{hop} is the propagation (without queuing) latency per hop, N_{hops} is the average number of hops for packets in a uniformly random traffic, L_{packet} is the average packet length and λ_{LLC} is the LLC access latency. The packet length L_{packet} is in terms of flits and to account for the serialization latency. The coefficient 2 in Equation (3.11) is to cover the round-trip. The underlying reason for including the ρ is to avoid the situation that when the reference uncore latency is equal to the real uncore latency, the ρ allows the queuing latency to be slightly above zero, which is also equivalent to a very small network congestion. Please note that λ_{ref} is a constant and can be pre-characterized offline. Overall, the reference for the critical latency becomes:

$$\Gamma_{ref,i} = \eta_i \cdot \lambda_{ref} \tag{3.12}$$

As the criticality factor η_i varies from interval to interval, this reference is dynamic with respect to packets at runtime.

3.6 Design Implementation

The implementation of the proposed DVFS methods mainly include: (1) information collection at each tile; (2) a central controller that aggregates the collected information and performs control policy computation; (3) information transportation from tiles to the central controller.

3.6.1 AMAT Implementation Overhead

The DVFS controller needs a modest amount of hardware support. At each tile, one counter is needed to frame the control interval, in our case 50000 cycles, so 16 bits is sufficient. To track each tile's memory operation status the following additional registers are required by each tile: One 20-bit register for L1 hit count, one 12-bit register for L2 hit count, one 12-bit register is required to count the number of L2 misses and one 20-bit register to sum up all of the L2 miss latencies. These registers are updated at the completion of memory instructions, and reset as the time windows end. These registers are used to compute $HitRate(L1)$ and $HitRate(L2)$ prior to encoding in the header flit, this latency is hidden by the packet generation delay. Both AMAT computation and the PID algorithm are composed of a few simple arithmetic calculations, which can be easily handled by the PCU. The PCU has sufficient storage to accommodate the data collected. We assume the PCU is co-located with the monitor tile, if this is not the case the monitor will need additional registers for the temporary storage of collected data prior to sending it to the PCU. Therefore, the overall hardware overhead is 80 bits per tile plus a possible 64 bits at the monitor tile.

3.6.2 Critical Latency Implementation Overhead

For the critical latency-based DVFS controls, registers are required to save relevant information at each tile. The bit-width of each register is decided by the data to be saved. Here we show a design setting. We use three 16-bit registers to save the numbers of *load* instructions, private cache hits and private cache misses, respectively. Additionally, there is a 20-bit register for accumulating the total request latency. Another 12-bit register is used to count the number of LLC misses. A 16-bit register is needed to save the uncore request count. Last, there is a 16-bit counter to keep track of control interval. Overall, 112 bits of registers are required for each tile.

In modern multicore processor designs, e.g., Intel's Nehalem architecture [26], there exists a Power Control Unit (PCU)dedicated to chip power management. Thus, the control of our DVFS can be implemented using PCU without additional hardware. The PCU retains a lookup table with each entry containing the data for a given tile and it also needs to keep the reference points and parameters for the controller. At the end of each control interval, the PCU computes the metric and uncore V/F level. The computations are several arithmetic operations and thus can be carried out quickly. For the transmission of status information from the cores to the PCU, we use a method similar to Section 3.6.1. When a packet is sent out from a tile, the 96 bits information (by excluding the control interval counter) is scaled to 64 bits and is embedded in the header flit. If the packet reaches or passes by the tile where the PCU resides, the data is scaled back to 96 bits and downloaded to the lookup table. Within each control interval, later data from a tile overwrites the old data from the same interval. Since we do not use additional network or send dedicated packets, the data transportation overhead is fairly low. In Section 3.4.1, it will show that a single monitor tile can obtain sufficient sample data in a control interval.

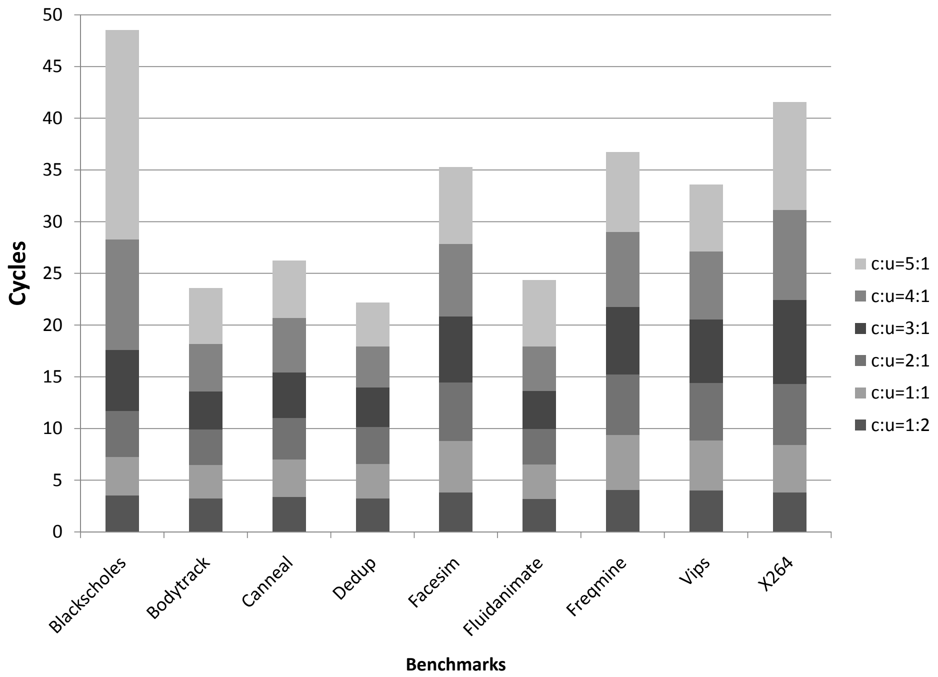


Figure 3.15: Oracle APHL simulation results of PARSEC.

3.7 Evaluation

In this section we first discuss our methodology and then compare the performance of our proposed technique and several variation versus baseline.

3.7.1 DVFS Oracle Simulation

In order to investigate how uncore system will affect the APHL, we try to change the injection ratio of the core and uncore here. For example:

core:Uncore = 3:1

which meas uncore clock speed is 3 times slower than core part. In Figure 3.15, the APHL under different ratio is clearly showed and we can tell that with ratio increaing, the APHL also increases.

Knee point: We define the knee point as the point at which the increase in latency is a certain amount over the latency of 1:1 The more of the rate, the more

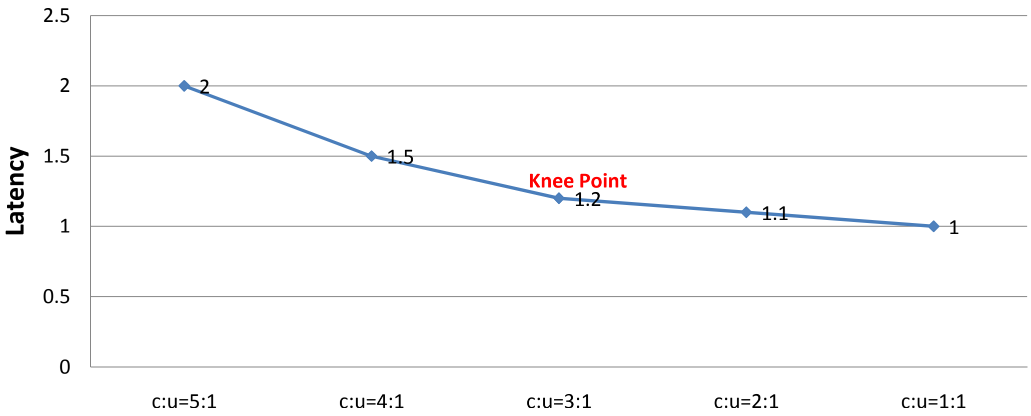
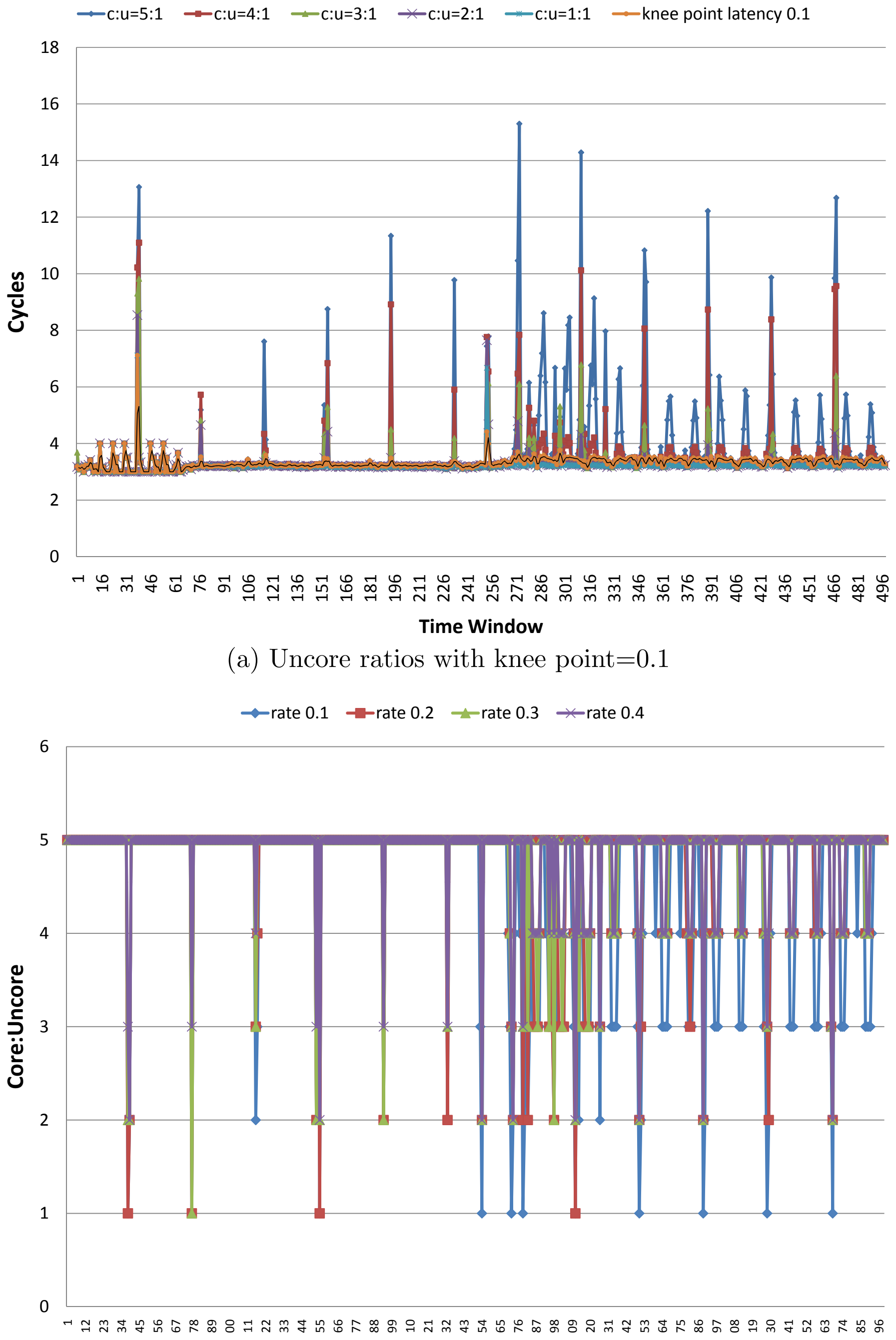


Figure 3.16: Knee point example, ratio=20%.

aggressive the algorithm will be. Meanwhile the more power it will save. Figure 3.16 is an example when raito factor is set to 0.2(or 20%).

When it applies knee point to the oracle simulation, we can get an ideal result that in each time window, it will always achieve the best frequency level. Figure 3.17a is the knee point choice when ratio is set to 0.1 for Bodytrack benchmark. When knee point goes high, the uncore will run slower as in Figure 3.17b.



(a) Uncore ratios with knee point=0.1

(b) Different knee point choice

Figure 3.17: Oracle simulations with various knee points of Bodytrack.

3.7.2 Experiment Setup for AMAT-Based PI Controller

The testbed architecture in our experiment is a 16-tile CMP as shown in Figure 3.2(b). Each tile is composed of a processing core with 2-levels of private cache, a network interface (NI) and a partition of the shared L3 cache (LLC). Table 3.1 summarizes our experimental configurations and parameters. For shared-memory, multi-processor application results, we use gem5 full system simulator to generate PARSEC shared-memory multi-processor benchmark memory system traces [3]. For multi-application workloads, gem5-generated traces of randomly selected applications from the SPEC CPU2006 benchmark suite were chosen to run simultaneously. Each trace contains up to 250 million memory operations. These traces are run through a memory hierarchy (L1-L2-LLC+directory) and network simulator based upon Ocin-tsim [42]. Although trace-driven, open-loop, NoC simulation can introduce error, we expect that for the small changes in AMAT experienced, these errors are minimal. Uncore DVFS is emulated by varying packet injection rate (e.g. slowing down uncore frequency by a half emulates doubling the injection rate). Please note that uncore DVFS does not affect off-chip memory access time. We assume that memory access time is constant with respect to core frequency. We explored several options for the DVFS control interval between 10000 clock cycles to 100000 clock cycles. In this work present results assuming an interval of 50000 clocks cycles as it provides more than sufficient time to collect traffic information as well as being short enough to capture fine-grain program phase behavior.

In the experiment, both dynamic and static power are considered. The NOC and LLC power models are based on ORION 2.0 [20] and CACTI 6.0 [37] of 65nm technology, respectively.

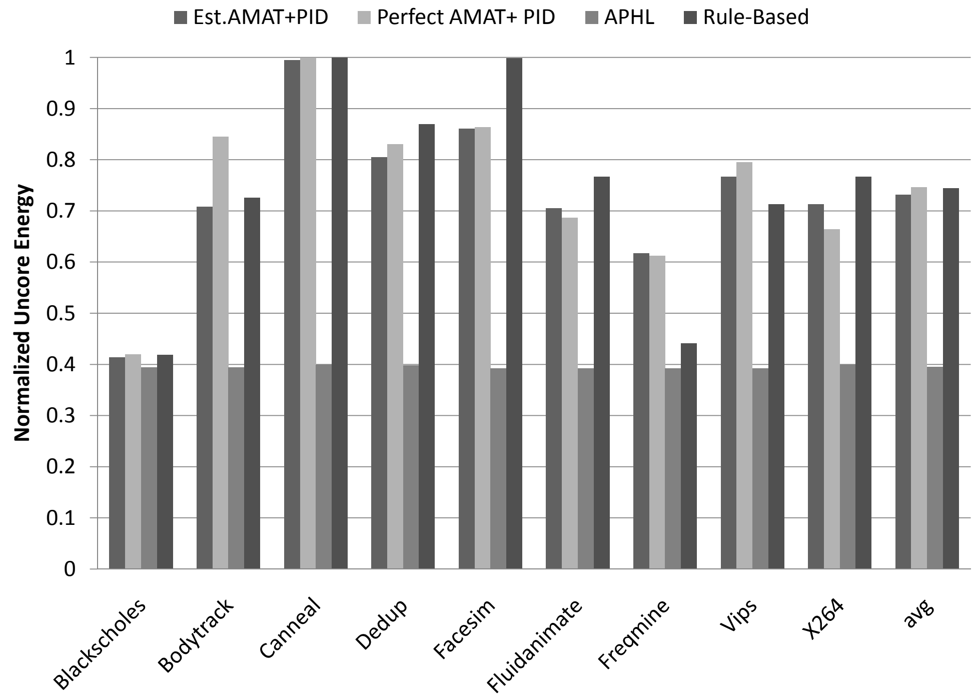
Table 3.1: AMAT simulation setup

Parameter	Values
Core Frequency	1GHz
#processing cores	16
L1 data cache	2-way 32Kb, 1 core cycle latency
L2 cache	8-way 256Kb, 13 core cycle latency
L3 cache (LLC)	16-way, 2MB/bank, 32MB/total, 15 uncore cycle latency
Directory cache	MESI, 4 uncore cycle latency
Memory access latency	100 core cycles
NoC	4×4 2D mesh, X-Y DOR, 2VCs/port 4flits deep
Voltage/Frequency	10 levels, voltage: 0.5V–1V, frequency: 250MHz–1GHz
V/F response time	500 core cycles

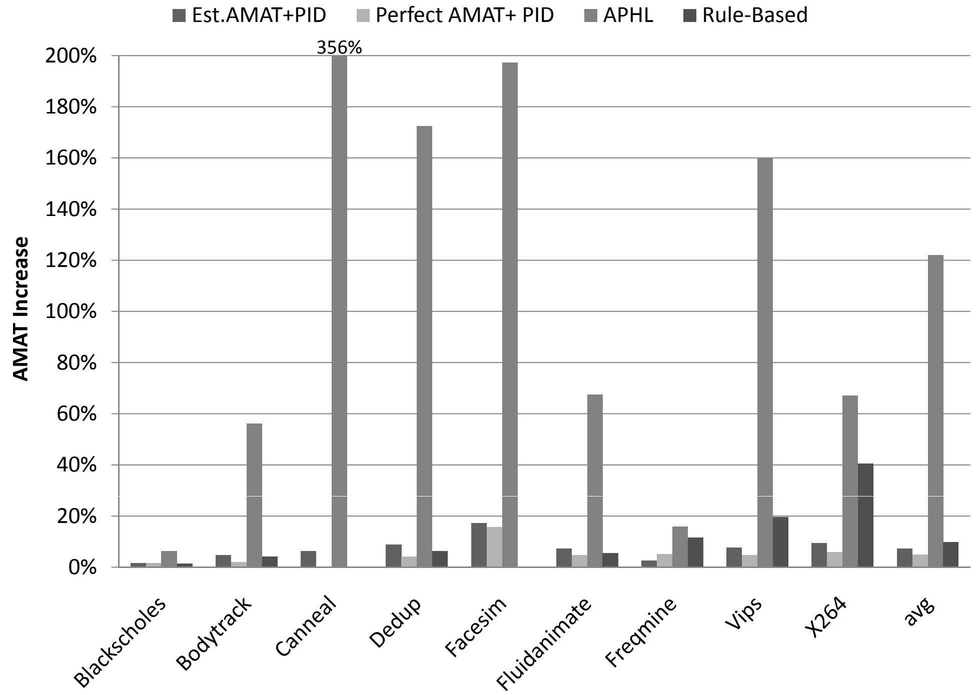
3.7.3 Simulation Results

First in this section we examine the results of our technique versus baseline and other competing techniques on shared-memory, multi-processor applications. We then examine its performance on multi-application workloads.

Figure 3.18 compares the total energy (including both dynamic and static energy) savings and performance degradation of our best proposed approach (labeled *Est. AMAT+PID*) versus baseline without DVFS, along with three variations which we discuss in the following subsections. The figure shows our PID technique, informed by estimated AMAT provides an average savings of 27% while reducing the performance of the uncore (measured by impact on absolute AMAT) by 7%. We empirically determined that AMAT increases of 7% tend to decrease processor performance (IPC) by $< 3.5\%$ (see Section 3.7.4). *Blackscholes* shows the best benefit, with a 59% reduction in energy, while *Canneal* shows the least reduction in energy at 1%. Both benchmarks show negligible performance impact. In *Blackscholes* the uncore is not critical to performance, because the L1 and L2 hit rates are high, so voltage and frequency (V/F) can be dropped without impacting performance. *Canneal*,



(a) Normalized energy consumption



(b) Performance degradation

Figure 3.18: Energy and performance impact for PARSEC benchmarks. Our best proposed method is “Est. AMAT + PID”.

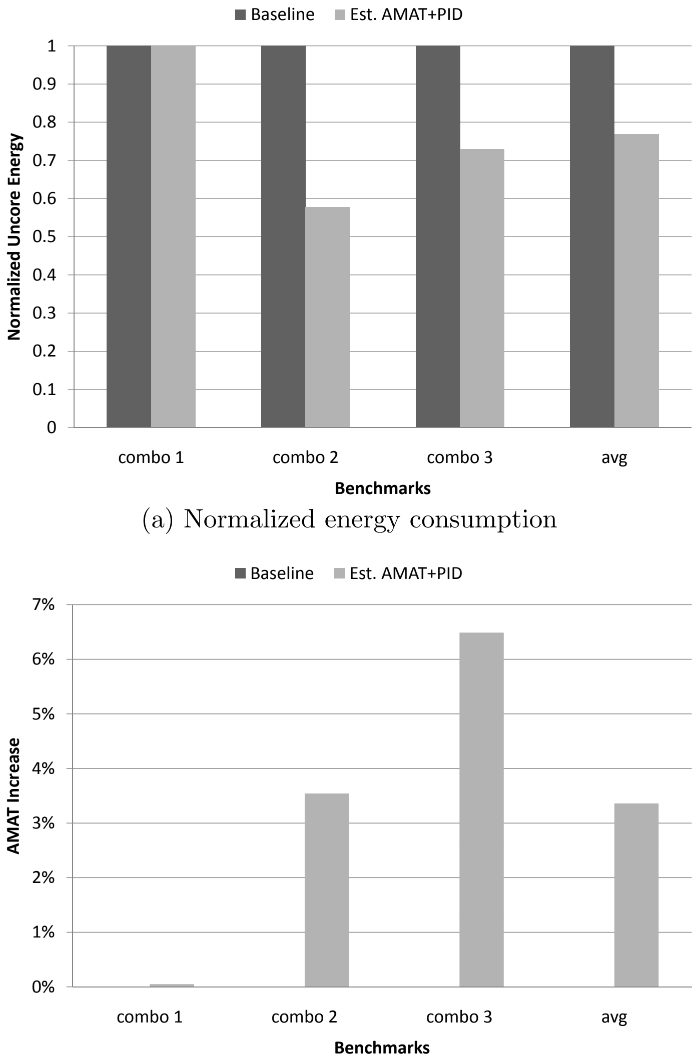
however, has a relatively low L1 and L2 hit rate and thus the uncore's performance is more critical to application performance. In both cases our algorithm preserves our performance goal of <5% IPC loss.

PID-based vs. Rule-based DVFS: Figure 3.18 also shows the results for a simple, naïve, rule based approach for DVFS policy (labeled *Rule-Based*), also informed by our proposed monitoring technique. In this approach, V/Fs are associated with specific ranges of AMAT (e.g. if monitored AMAT is a then uncore frequency level is set to b). The advantage to this technique would be that it eschews the overheads of the PID controller, however, its static nature ignores time-varying dynamics of the system. While rule-based DVFS obtains similar energy reduction as PID-based DVFS, it is unable to adapt as well and performance decreases by an additional 37% relative PID-based DVFS with an AMAT degradation of 10%.

Monitoring AMAT vs. APHL: APHL (Average Per-Hop Latency) is a simple, direct measure of network performance discussed in Section 3.3.1. Figure 3.18 also shows results for PID informed by the APHL metric. In the figure we see that APHL is generally a poor metric. Due to relatively low congestion in the interconnect APHL ends up reducing uncore V/F and saving energy significantly, however, this results comes at a high price in terms of impact on performance as it appears largely oblivious to the demands of the applications. This leads to extremely large performance losses, up to 350% in the case of *Canneal*. This phenomenon is due to the static APHL target of three *uncore* cycles. When traffic load is not high, lowering uncore frequency may still maintain an APHL close to 3 uncore cycles, however, the network latency in core cycles increases due to the frequency ratio change. This dramatically affects performance when L1 or L2 miss rates are high. Monitoring APHL in core cycles, however, is impractical because of the difficulty in finding appropriate PID reference levels without knowing the dynamically changing network latency relative to system performance.

Sampled AMAT vs. Perfect knowledge of AMAT: The AMAT monitoring technique we propose uses a single tile to collect information from its own packets and passing-by traffic. Compared to monitoring all tiles for complete knowledge of AMAT, our technique has much lower overhead, however, this comes at the potential cost of some inaccuracy in AMAT estimation due to incomplete knowledge. Figure 3.18 also shows results for PID DVFS based upon perfect knowledge of the system AMAT each time window (labeled *Perfect AMAT+PID*). The figure shows estimating AMAT produces results quite close to perfect knowledge. Overall, the difference in energy savings and performance is $\sim 1 - 2\%$.

In order to further test our monitoring technique and control algorithm, examine a use case in which multiple single threaded applications are run simultaneously on the system. The results from a set this experiment are shown in Figure 3.19. As the number of possible combinations of applications is extremely large, we show the results from three representative combinations to demonstrate how our technique adapts to the demands of the applications being run. The groupings of applications shown are as follows: **Combo 1**: cactusADM + gromacs + GemsFDTD + milc, **Combo 2**: cactusADM + sphinx3 + namd + sjeng + GemsFDTD, and **Combo 3**: sphinx3 + zeusmp + wupwise + sjeng. In the event that the combination of applications places a high demand on uncore performance ($comb1$ from the figure), the uncore remains at a high frequency and no performance is lost (though no energy is saved). In the event that there is uncore performance to spare, the *Est. AMAT+PID* lowers the V/F accordingly and saves energy with a minimal impact on performance. Generally across all the combination used, the performance and energy savings seen match those found in the shared-memory applications.



(a) Normalized energy consumption

(b) Performance degradation

Figure 3.19: Energy and performance impact for multi-application benchmarks.

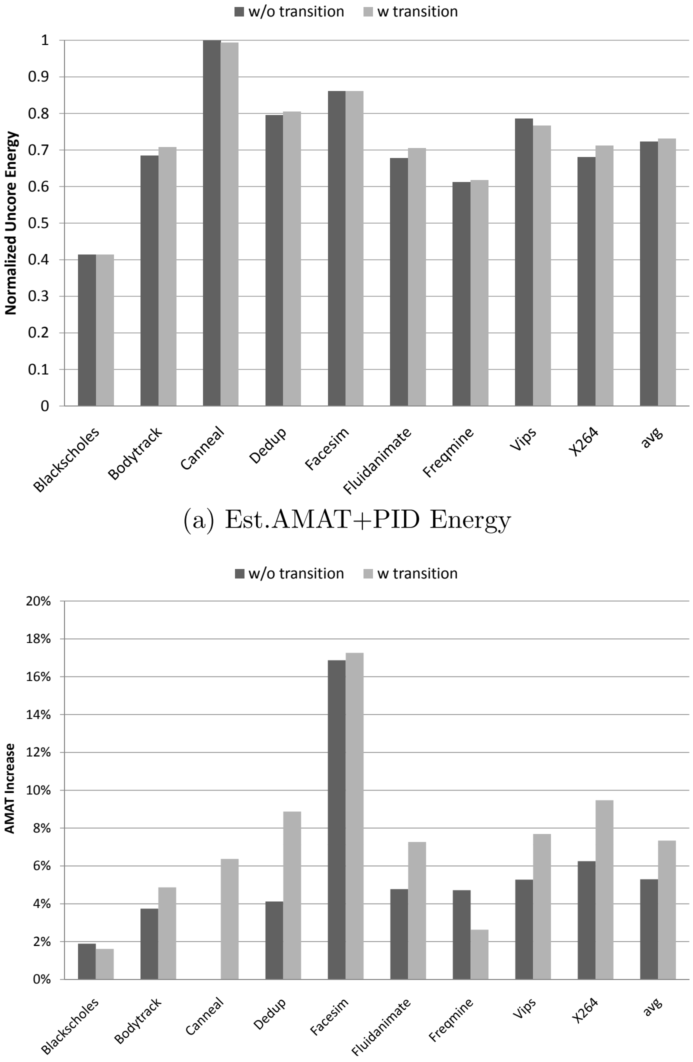
3.7.4 Analysis

In this section we explore how varying configurations affect the results, examine how IPC and AMAT are related, and examine how well our proposed controller tracks the ideal V/F point on a given application.

Impact of V/F transition time: In real systems, it takes certain amount of time to make V/F level changes. In the prior results, we assume the V/F transition time of 500 core clock cycles is spent for each V/F level change, during which time the entire network is stalled. This transition time value is based on previous literature [55] with anticipation of technology progress in recent years. Figure 3.20 examines how the results are effected by this transition time. The figure shows that the transition time incurs a relatively small, 1-2% penalty on AMAT performance with negligible effect on energy savings.

Impact of PID control parameters: A given power management policy should be able to achieve different energy-performance tradeoffs. In our proposed controller, this tradeoff can be managed by varying the parameters of the PID controller. In the *Est. AMAT+PID*, we set $AMAT_{ref}$ to 2. We also run the simulations with $AMAT_{ref}$ of 4.2 to achieve a more aggressive power savings. The results in Figure 3.21 indicate that an additional 6% energy savings may be achieved through the aggressive settings, however this comes at the cost of doubling the performance penalty at 12%.

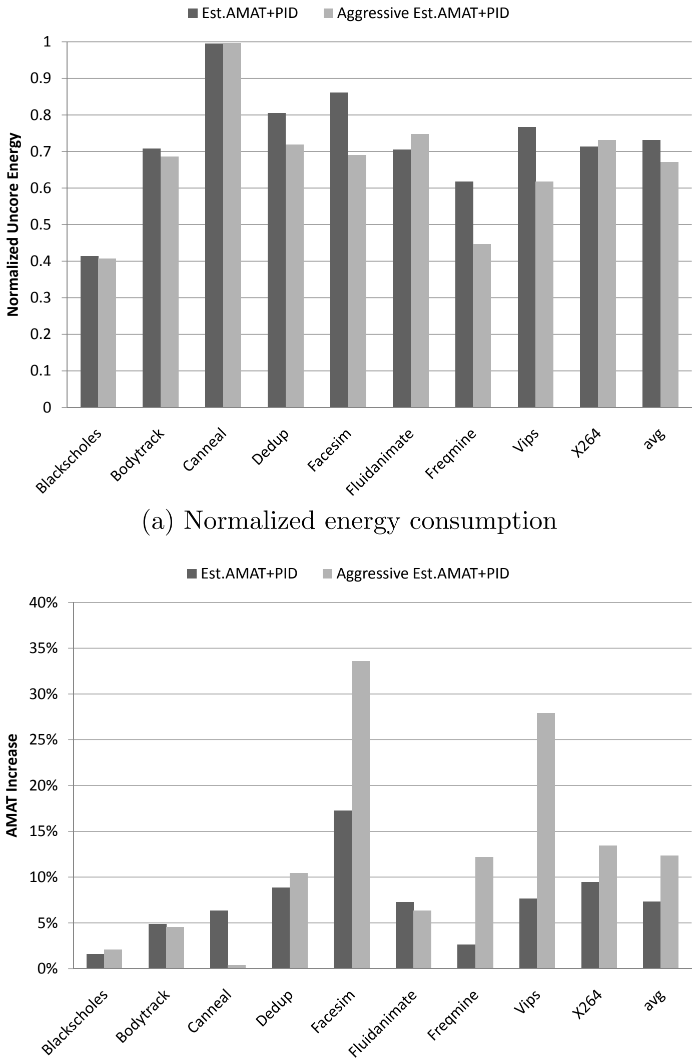
Relationship between IPC and AMAT: While AMAT is an important metric of memory system performance, ultimately application performance is more typically described in terms of instructions per clock (IPC). In order to find the relationship between IPC and AMAT, we derive the cache latency that allows the increase of the AMAT to 10%,20% and so on. The Simulation are done on Gem5 detailed mode with



(a) Est.AMAT+PID Energy

(b) Est.AMAT+PID Performance

Figure 3.20: Impact of V/F transition time.



(a) Normalized energy consumption

(b) Performance degradation

Figure 3.21: Impact of different PID control parameters.

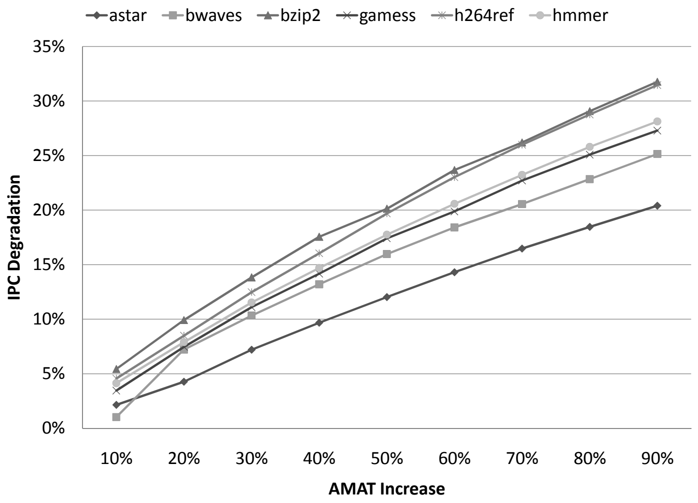


Figure 3.22: Relationship between IPC and AMAT.

SPEC 2006 benchmarks. From the Figure 3.22, we can roughly draw an conclusion that IPC degradation : AMAT Increase = 1 : 2, thus an increase in AMAT of 7% should be expected to lead to a decrease in performance of ∼3.5%.

Performance relative an omniscient controller: Here we provide some simulation details to aid developing an intuition on the behavior of our approach. In Figure 3.23, we show a snapshot of the uncore frequency over time of our *Est. AMAT+PID* system versus an "*Ideal*" DVFS policy. The *Ideal* policy is an unrealistic case where every benchmark is simulated once for each V/F setting and the lowest V/F are chosen for each time window which meet the performance goal of < 5% performance loss. Generally *Est. AMAT+PID* follows the *Ideal* policy very closely. Where variance occurs, the estimated method is generally more conservative. Initially, both frequencies are high due to high cache miss rate during initialization. After two millions of clock cycles, the lower level caches are filled and the frequency is lowered, reflecting the lower performance criticality of the uncore. There are some frequency spikes arising from occasional traffic spikes due to application phase changes. Surprisingly, the *Est. AMAT+PID* policy is able to track the

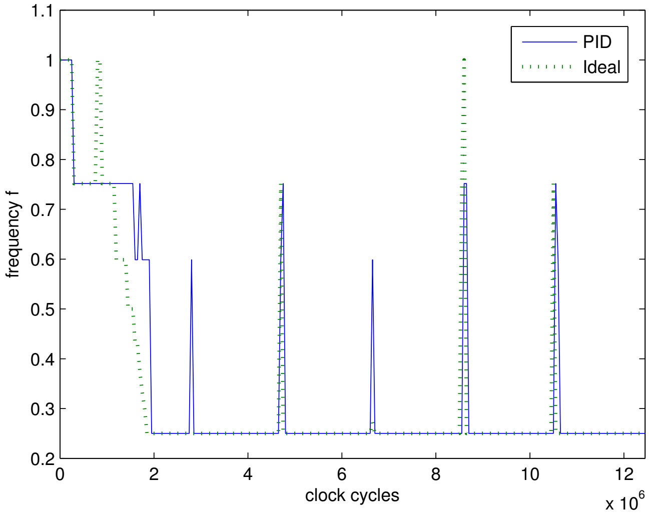


Figure 3.23: Simulation snapshot of *Blackscholes* showing DVFS frequency over time, Est. AMAT+PID vs. Ideal.

performance needs of the uncore quite well even during these spikes.

3.7.5 Experiment Setup for Critical Latency Based PI Controller

The baseline architecture in our experiments is a 16-tile chip multiprocessor. Each tile is composed of a processing core with 1-level of private cache, a network interface, an NOC router and a partition of the shared L2 cache (LLC) and directory.

Parameter	Values
Core Frequency	1GHz
#processing cores	16
L1 data cache	2-way 256Kb, 2 core cycle latency
L2 cache (LLC)	16-way, 2MB/bank, 32MB/total, 10 uncore cycle latency
Directory cache	MESI, 4 uncore cycle latency
NoC	4 × 4 2D mesh, X-Y DOR, 4 flits depth/VC
Voltage/Frequency	10 levels, voltage: 1V–2V, frequency: 250MHz–1GHz

Table 3.2: Critical latency simulation setup.

Each core is an in-order Alpha ISA processor. Table 3.2 summarizes our experimental configurations and parameters. We use Gem5 [4] full system simulator with PARSEC shared-memory multi-processor benchmarks [3]. For each benchmark, the entire application is run in full-system mode; the results obtained are based upon statistics from the region of interest (ROI), which is usually hundreds of millions of cycles long. Gem5 "Ruby" memory hierarchy (L1-LLC+directory) and "Garnet" network simulator are used for all results. Frequency scaling for uncore is emulated by changing the latency of each uncore component. For instance, the latencies of each router pipeline stage, link traversal time and the LLC access time are doubled when the uncore frequency is reduced to 50%. As off-chip memory is not affected by uncore DVFS, its access latency is assumed to be a constant. We expect the proposed techniques should work well with core DVFS as this would be expressed through decreased L1 demand, decreasing the utility of the uncore. The control interval for the uncore DVFS is 50K core clock cycles. According to our experience and existing literature, such interval size allows sufficient time for the uncore V/F change to settle, and is sufficiently small to capture fine-grain program phase behavior. Both dynamic and leakage power are considered in the experiment. We use ORION 2.0 [20] and CACTI 6.0 [37] based on 65nm technology for the power models of NOC and LLC, respectively. The overall performance is evaluated as the execution time for the ROI of each application.

In this work, we compare the following methods:

Baseline constantly high uncore V/F level.

AMAT+PI chapter 3's method [7].

CL+PI PI control based on the critical latency described in Section 3.3.3.

CL+ModelAssist the critical latency-driven, model assisted PI control described

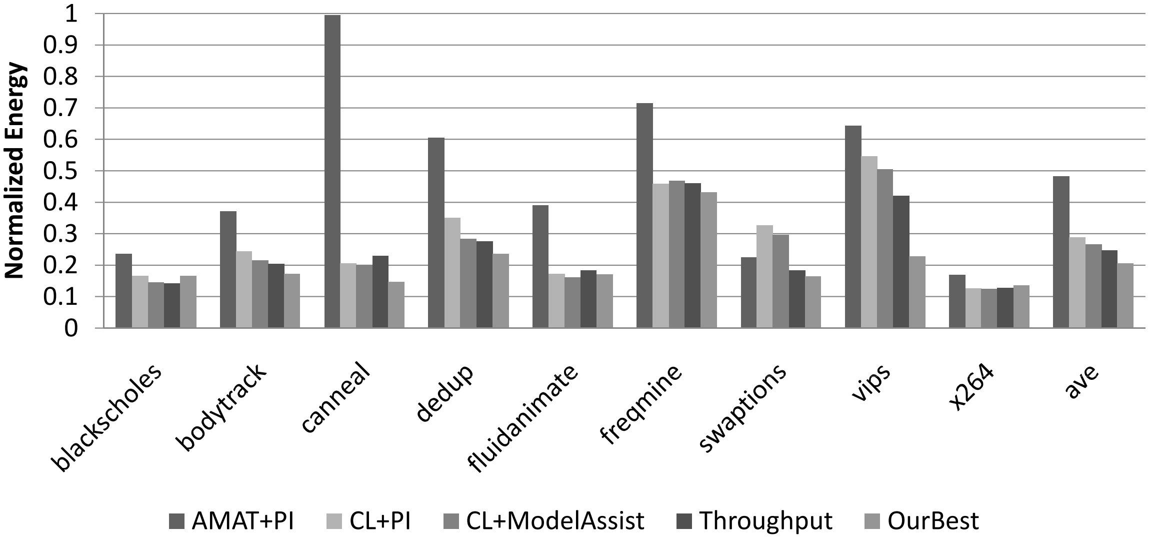


Figure 3.24: Normalized energy (with the baseline result as 1) for PARSEC benchmarks.

in [8].

Throughput the throughput driven PI control with dynamic reference method described in [8].

OurBest the critical latency-driven, PI control with dynamic reference described in Section 3.5.2.

3.7.6 Energy and Performance Comparisons

Figure 3.24 shows the energy comparison for the different methods. The figure shows, the critical latency defined by Equation (3.6) leads to significantly more power savings than that from AMAT [7]. This is especially obvious for the case *canneal*. The model-assist technique can further reduce power dissipation. Our best method can provide additional 50% power reduction over prior techniques [7].

Figure 3.25 provides a comparison of the performance impact of the different methods.

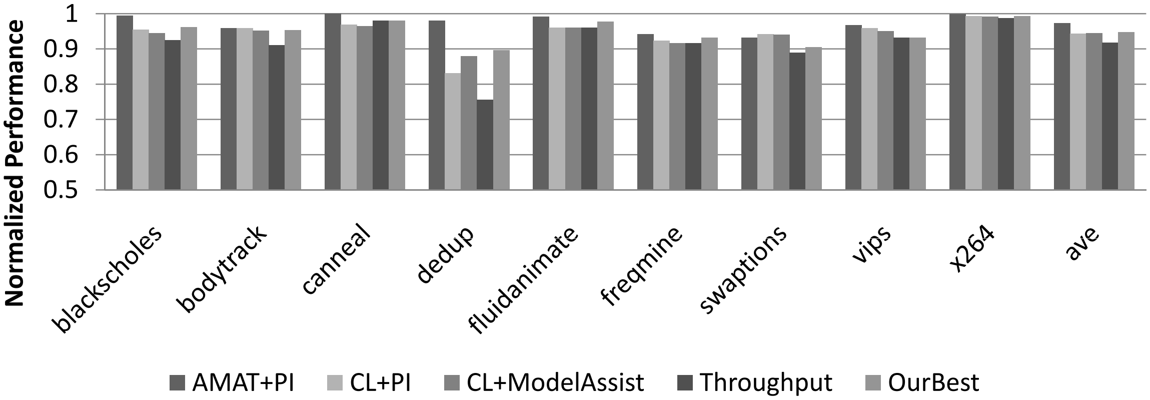


Figure 3.25: Normalized system performance (with the baseline result as 1) for PARSEC benchmarks.

Method	$Energy \times Delay$	$Energy \times Delay^2$
Baseline	1.0	1.0
AMAT+PI	0.5	0.51
CL+PI	0.31	0.33
CL+ModelAssist	0.28	0.30
Throughput	0.27	0.29
OurBest	0.22	0.23

Table 3.3: Normalized energy-delay product on average for all PARSEC cases.

Except *dedup*, the performance degradation from all of our techniques is quite limited. In the worst case of *dedup*, model assist and our best control progressively improve the performance. Overall, the performance loss from all our methods is around only 5%. In Table 3.3, the normalized energy-delay product among all cases are listed. The progressive improvement from the new metric (critical latency), model assist and dynamic reference can be observed. Compared to AMAT+PI [7], our best method reduces the energy-delay product by 56%.

To analyze controller sensitivity to parameters, we varied the parameters to obtain different solutions for the throughput-based and our best methods. The results are plotted in Figure 3.26 together with those from AMAT+PI and CL+ModelAssist. The solutions at lower-left envelope for each technique are Pareto optimal as they imply either high performance or low energy consumption. One can see that OurBest, the latency-based PI control with dynamic reference, achieves the best performance-energy tradeoff regardless of parameter set point. On the other hand, please note that the throughput-driven DVFS has much lower implementation overhead as shown in Section 3.6.

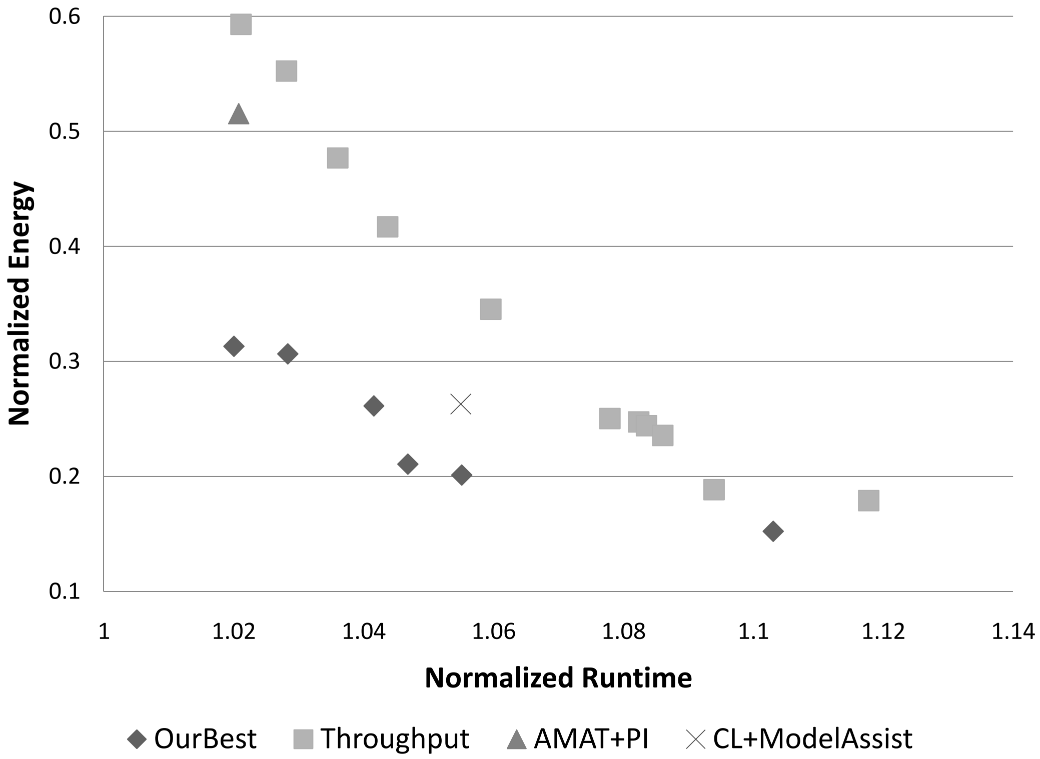


Figure 3.26: Energy-runtime tradeoff comparison.

3.8 Additional Analysis

In this section we provide some additional analysis, exploring how the control algorithm works and its sensitivity to control interval size.

In the DVFS control, one practical issue is how to decide the control interval size for the control. We use $50K$ core clock cycles, which is long enough for V/F change and short enough for fine-grained power control. We performed simulations on two PARSEC benchmarks with different control interval sizes and the results are summarized in Table 3.4. One can see that the results are not sensitive to moderate interval size changes.

Interval size	$Energy \times Delay$	
(# cycles)	freqmine	x264
12K	0.468	0.151
25K	0.470	0.145
50K	0.462	0.136
75K	0.479	0.147
100K	0.500	0.145

Table 3.4: Impact of different control interval sizes. The $Energy \times Delay$ values are normalized with the baseline result as 1.

For the PARSEC benchmark *fluidanimate*, we simulate with different constant uncore V/F levels throughout the entire ROI. The average critical latency versus uncore clock period results are plotted in Figure 3.27. The results confirm that critical latency has approximately linear dependence on the uncore period.

In Section 3.5.2, we argue that it is very hard to accurately predict the system behaviors. Figure 3.28, 3.29 and 3.30 show some simulation results for a segment of *x264*, an application from the PARSEC benchmarks. Its horizontal axis is in terms of control interval of $50K$ clock cycles, i.e., each data point is an average over $50K$ cycles. Despite such smoothing, the data still show drastic changes from interval

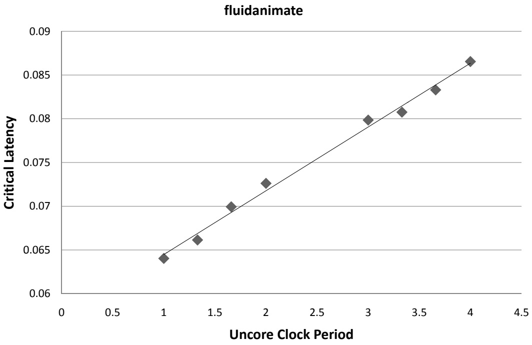


Figure 3.27: Critical latency vs. uncore clock period.

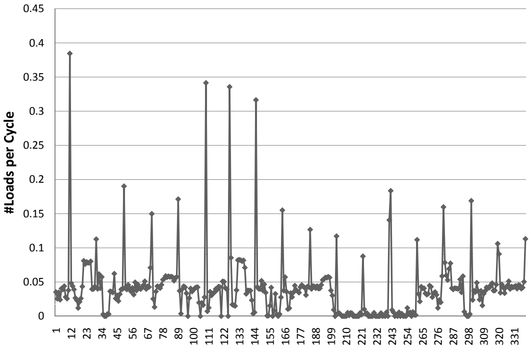


Figure 3.28: Loads fraction over control intervals.

to interval. These results confirm that it is very difficult to accurately predict the behavior of a multicore system.

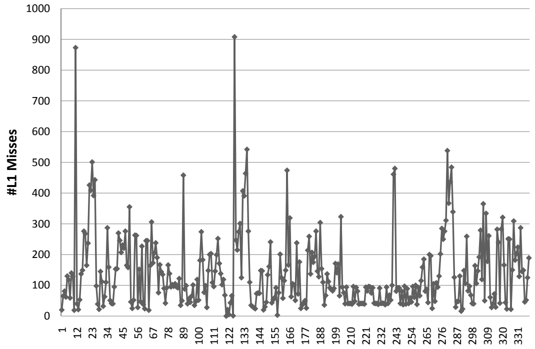


Figure 3.29: The number of L1 misses over control intervals.

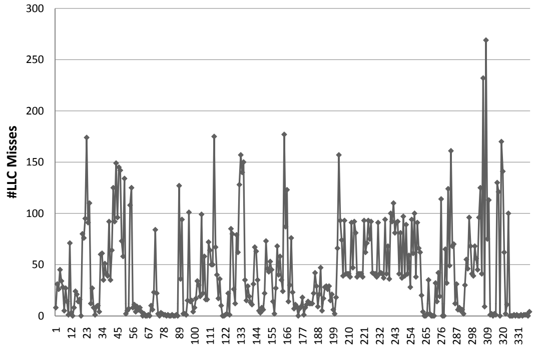


Figure 3.30: The number of LLC misses over control intervals.

4. CONCLUSION

In this work, a floorplanning approach for multi-core processors where cores and memory cores are required to be placed under regularity constraint is proposed. This is the first work on applying regularity constraint floorplanning to multi-processor. Experimental results indicate that the proposed approach significantly outperforms a naive semi-automatic method. In future research, further study on the multi-core processor floorplanning using other representations like TCG [29] will be conducted.

We also propose a DVFS policy for a CMP's uncore in this work. This policy leverages an uncore performance monitoring based upon AMAT estimation. This technique is integrated with a PID-based DVFS control system. In simulation on PARSEC benchmarks, the proposed approach achieves 27% NoC and LLC energy reduction, with only 7.3% degradation on AMAT, correlated to a $\sim$ 3.7% system performance drop. We show that the technique has low computation and communication overheads and is practical to implement. Besides, we investigated DVFS for shared resources (NOC/LLC) in multicore processor designs. Several metrics and policies are developed. The proposed techniques are evaluated on public domain architecture benchmarks with full-system simulation. The results show quite large energy savings and improvement over recent previous work.

REFERENCES

[1] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 338–347, 2010.

[2] F. Balasa and K. Lampaert. Symmetry within the sequence-pair representation in the context of placement for analog design. *IEEE Transactions on CAD*, 19(7):721–731, July 2000.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of ACM/IEEE International Conference on Parallel Architecture and Compilation Techniques*, pages 72–81, 2008.

[4] Nathan L. Binkert, Ronald G. Dreslinski, Lisa R. Hsu, Kevin T. Lim, Ali G. Saidi, and Steven K. Reinhardt. The M5 simulator: modeling networked systems. *IEEE Micro*, 26:52–60, 2006.

[5] P. Bodgan, R. Marculescu, S. Jain, and T. R. Gavila. An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads. In *Proceedings of IEEE/ACM International Symposium on Networks on Chip*, pages 35–42, 2012.

[6] Y. C. Chang, Y. W. Chang, G. M. Wu, and S. W. Wu. B*-trees: A new representation for non-slicing floorplans. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 458–463, 2000.

[7] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras. In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches. In *Proceedings of IEEE/ACM International Symposium on Networks on Chip*, pages 43–50, 2012.

[8] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, U. Ogras, and Raid Ayoub. Dynamic voltage and frequency scaling for shared resources in multi-core processor designs. In *Proceedings of the ACM/IEEE Design Automation Conference*, page to appear, 2013.

[9] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley. *Analog Device-Level Layout Automation*. Kluwer Academic Pub., the Netherlands, first edition, 1994.

[10] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 148–157, 2002.

[11] P. Gratz, B. Grot, and S. W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture*, pages 203–215, 2008.

[12] P. Gratz and S. W. Keckler. Realistic workload characterization and analysis for networks-on-chip design. In *The 4th Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, pages 1–10, 2010.

[13] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger. On-chip interconnection networks of the TRIPS chip. *IEEE Micro*, 27:41–50, 2007.

[14] L. Guang, E. Nigussie, L. Koskinen, and H. Tenhunen. Autonomous DVFS on supply islands for energy-constrained NoC communication. *Lecture Notes in Computer Science: Architecture of Computing Systems*, 5455/2009:183–194, 2009.

[15] P. N. Guo, C. K. Cheng, and T. Yoshimura. An o-tree representation of non-slicing floorplan and its applications. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 268–273, 1999.

[16] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 8–12, 2000.

[17] A. Iyer and D. Marculescu. Power efficiency of voltage scaling in multiple clock multiple voltage cores. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 379–386, 2002.

[18] H.-S. Jung and M. Pedram. Supervised learning based power management for multicore processors. *IEEE Transactions on Computer-Aided Design*, 29(9):1395–1408, September 2010.

[19] A. B. Kahng. Classical floorplanning harmful. In *Proceedings of ACM International Symposium of Physical Design*, pages 207–213, 2000.

[20] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: a power-area simulator for interconnection networks. *IEEE Transactions on VLSI Systems*, 20(1):191–196, January 2012.

[21] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi. *Low Power Methodology Manual*. Synopsys, Mountain View, California, first edition, 2007.

[22] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das. A low latency router supporting adaptivity for on-chip interconnects. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 559–564, 2005.

[23] T. Konstantakopoulos, J. Eastep, J. Psota, and A. Agarwal. Energy scalability of on-chip interconnection networks in multicore architectures. Technical report, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, Massachusetts, November 2007.

[24] S. Kouda, C. Kodama, and K. Fujiyoshi. Improved method of cell placement with symmetry constraints for analog ic layout design. In *Proceedings of ACM International Symposium of Physical Design*, pages 192–199, 2006.

[25] Cyril Kowaliski. Gelsinger reveals details of Nehalem, Larrabee, Dunnington. `http://techreport.com/news/14361/gelsinger-reveals-details-of-nehalem-larrabee-dunnington`, 2008.

[26] R. Kumar and G. Hinton. A family of 45nm IA processors. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 58–59, 2009.

[27] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *IEEE Computer*, 38(11):32–38, Nov 2005.

[28] G. Liang and A. Jantsch. Adaptive power management for the on-chip communication network. In *Proceedings of EUROMICRO Conference on Digital System Design*, pages 649–656, 2006.

[29] J. M. Lin and Y. W. Chang. Tcg: a transitive closure graph based representation for general floorplans. *IEEE Transactions on VLSI Systems*, 13(2):288–292, Feb 2005.

[30] J. M. Lin, G. M. Wu, Y. W. Chang, and J. H. Chuang. Placement with symmetry constraints for analog layout design using tcg-s. In *Proceedings of ACM/IEEE Asia South Pacific Design Automation Conference*, pages 1135–1138, 2005.

[31] P. H. Lin and S. C. Lin. Analog placement based on novel symmetry-island formulation. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 464–467, 2007.

[32] J. Y. Liu, S. Q. Dong, X. L. Hong, Y. B. Wang, O. He, and S. Goto. Symmetry constraint based on mismatch analysis for analog layout is soi technology. In *Proceedings of ACM/IEEE Asia South Pacific Design Automation Conference*, pages 772–775, 2008.

[33] Q. Ma, L. Xiao, Y. C. Tam, and E. F. Y. Young. Simultaneous handling of symmetry, common centroid, and general placement constraints. *IEEE Transactions on CAD*, 30(1):85–95, Jan 2011.

[34] S. Ma, N. E. Jerger, and Z. Wang. Dbar: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 413–424, 2011.

[35] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote. Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design*, 28(1):3–21, January 2009.

[36] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das. A case for dynamic frequency tuning in on-chip networks. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 292–303, 2009.

[37] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: a tool to model large caches. Technical report, HP Laboratories, Palo Alto, California, 2009.

[38] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi block placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on CAD*, 15(12):1518–1524, 1996 December.

[39] S. Nakatake. Structured placement with topological regularity evaluation. In *Proceedings of ACM/IEEE Asia South Pacific Design Automation Conference*, pages 215–220, 2007.

[40] U. Y. Ogras, R. Marculescu, and D. Marculescu. Variation-adaptive feedback control for networks-on-chip with multiple clock domains. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 614–619, 2008.

[41] Y. Pang, F. Balasa, L. Lampaert, and C. K. Cheng. Block placement with symmetry constraints based on the o-tree non-slicing representation. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 464–467, 2007.

[42] S. Prabhu, B. Grot, P. V. Gratz, and J. Hu. Ocin_tsim: DVFS aware simulator for NoCs. In *The 1st Workshop on SoC Architecture, Accelerators and Workloads (SAW)*, pages 1–8, 2010.

[43] A. Rahimi, M. E. Salehi, S. Mohammadi, and S. M. Fakhraie. Low-energy GALS NoC with FIFO-monitoring dynamic voltage scaling. *Microelectronics Journal*, 42(6):889–896, June 2011.

[44] F. Rotem, A. Mendelson, R. Ginosar, and U. Weiser. Multiple clock and voltage domains for chip multi processors. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 459–468, 2009.

[45] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 356–367, 2002.

[46] L. Shang, L. Peh, and N. K. Jha. Power-efficient interconnection networks: dynamic voltage scaling with links. *IEEE Computer Architecture Letters*, 1(1), 2002.

[47] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. GOAL: a load-balanced adaptive routing algorithm for Torus networks. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 194–205, 2003.

[48] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta. Globally adaptive load-balanced routing on Tori. *IEEE Computer Architecture Letters*, 3(1):2, 2004.

[49] S. W. Son, K. Malkowski, G. Chen, M. Kandemir, and P. Raghavan. Integrated link/CPU voltage scaling for reducing energy consumption of parallel sparse matrix applications. In *Proceedings of ACM/IEEE International Parallel and Distributed Processing Symposium*, pages 8–15, 2006.

[50] Y. C. Tam, E. F. Y. Young, and C. C. N. Chu. Analog placement with symmetry and other placement constraints. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 349–354, 2006.

[51] X. Tang, R. Tian, and D. F. Wong. Fast evaluation of sequence-pair in block placement by longest common subsequence computation. In *Proceedings of IEEE Design Automation and Test in Europe*, pages 106–111, 2000.

[52] X. Tang and D. F. Wong. Floorplanning with alignment and performance constraints. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 848–853, 2002.

[53] B. Towles, W. J. Dally, and S. Boyd. Throughput-centric routing algorithm design. In *Symposium on Parallel Algorithms and Architectures*, pages 200–209, 2003.

[54] H. Wang, L.-S. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 105–116, 2003.

[55] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 248–259, 2004.

[56] L. Xiao and E. F. Y. Young. Analog placement with common centroid and 1-d symmetry constraints. In *Proceedings of ACM/IEEE Asia South Pacific Design Automation Conference*, pages 353–360, 2009.

[57] J. Z. Yan and C. Chu. Defer: Deferred decision making enabled fixed-outline floorplanning algorithm. *IEEE Transactions on CAD*, 39(3):367–381, March 2010.

[58] A. W. Yin, L. Guang, P. Liljeberg, P. Rantala, E. Nigussie, J. Isoaho, and H. Tenhunen. Hierarchical agent architecture for scalable NoC design with online monitoring services. In *International Workshop on Network on Chip Architecture*, pages 58–63, 2008.

[59] E. F. Y. Young, C. C. N. Chu, and M. L. Ho. Placement constraints in floorplan design. *IEEE Transactions on VLSI Systems*, 12(7):735–745, July 2004.

[60] L. Zhang, C. J. Shi, and Y. Jiang. Symmetry-aware placement with transitive closure graphs for analog layout design. In *Proceedings of ACM/IEEE Asia South Pacific Design Automation Conference*, pages 180–185, 2008.