Submission under review, please do not distribute! Design and Scaling of Multi-Core Microprocessors

Weiping Liao, Changbo Long, Lucanus J. Simonson and Lei He

Electrical Engineering Department
University of California, Los Angeles, CA 90095
[*]

Abstract

In this paper we study the design and scaling of multi-core microprocessors, modeled as Chip-level Multi-Processors (CMP) with a shared bus structure. We develop analytical models to avoid time-consuming cycle-accurate simulation, and perform quick yet accurate multi-core design space exploration. Experiments show that our models achieve high fidelity and an average error of 9% compared to cycle accurate simulation, and our optimized CMP design with a same area has 23% higher throughput compared to a published dual-core system based on IBM POWER5-like cores. In addition, we study the impact of technology and area scaling and scalability of the shared bus structure. We show that technology scaling has a diminishing or saturated throughput increase for single-core design, but multi-core design with a same area improve throughput by $5.58X$ when scaling the technology from 65nm to 32nm. Furthermore, as chip area increases, the area efficiency defined as throughput per unit area decreases, and so does the area efficiency gap between different technology generations. Both are due to the growing limitation of the shared bus structure.

1. INTRODUCTION

Multi-core design has become a new trend for the high-performance microprocessor. Compared to traditional single-core design, which can only leverage Instruction-Level Parallelism (ILP), multi-core processor design can explore both ILP and Thread-Level Parallelism (TLP), and therefore provide higher system throughput. A number of multi-core microprocessors have been available [1, 2]. It is expected that multi-core microprocessors will become the mainstream in the near future.

Existing multi-core designs [1, 2] only integrate two cores into one chip using existing single-core designs. Such approach is simple. However, there is no guarantee of optimality in terms of total system throughput. Co-optimization of organization between cores and microarchitecture inside cores may improve the chip-level throughput. However, given the large amount of choices for different computing core architectures (issue width, number of functional units, and etc.) and local cache hierarchies, it is not efficient, if not totally infeasible to to explore the design space by traditional, time-consuming cycle-accurate simulations. Instead, an efficient yet accurate automatic methodology from the CAD point of view is necessary. Such a requirement places a daunting challenge on CAD researchers.

[*]This paper is partially supported by NSF CAREER award CCR-0401682, SRC grant 1116, a UC MICRO grant sponsored by Mindspeed, and a Faculty Partner Award by IBM. We used computers donated by Intel and SUN Microsystems. Address comments to lhe@ee.ucla.edu.

In the literature, multi-core microprocessors modeled as Chip MultiProcessing (CMP) architecture have been studied. Most of these studies focus on the comparison among CMP, Simultaneous MultiThreading (SMT), and aggressively wide-issued SuperScalar architecture [3, 4]. Similarly, [5] studies the performance, energy and thermal considerations for SMT and CMP architectures based on IBM POWER4/POWER5-like cores. All these studies assumes fixed CMP architecture and no automatic method for CMP design exploration has been proposed.

Besides the throughput-optimal design, the impact of technology scaling is always an interesting problem for semiconductor technology. Performance impact of technology scaling for single-core SuperScalar architecture has been studied [6]. However, there is no similar study for multi-core microprocessor design.

In this paper, we study the automatic exploration of area-constrained throughput-optimal multi-core microprocessors. We model the multi-core microprocessor as Chip MultiProcessing (CMP) architecture with shared bus structure. We develop quick yet accurate high-level analytical models to avoid time-consuming cycle-accurate simulation, and perform quick yet accurate multi-core design space exploration. Experiments show that our models achieve high fidelity and an average error of 9% compared to cycle accurate simulation, and our optimized CMP design with a same area has 23% higher throughput compared to a published dual-core system based on IBM POWER5-like cores.

Leveraging our multi-core design methodology, we further study the impact of technology and area scaling as well as the scalability of shared bus architecture. We show that technology scaling has a diminishing or saturated throughput increase for single-core design, but multi-core design with a same area improve throughput by $5.58X$ when scaling the technology from 65nm to 32nm. Furthermore, as chip area increases, the area efficiency defined as throughput per unit area decreases, and so does the area efficiency gap between different technology generations. Both are due to the growing limitation of the shared bus structure. To the best of our knowledge, this paper is the first in-depth study on automatic exploration multi-core microprocessors and scaling of area and technology generations.

The rest of this paper is organized as follows. Section 2 presents system architecture and problem formulation. Section 3 introduces our methodology. Section 4 presents the experimental setting and model validation. Section 5 further discusses the impact of area and technology scaling, and shared bus structure. We conclude in Section 6.

2. SYSTEM ARCHITECTURE AND PROBLEM FORMULATION

2.1 System Architecture

In this paper we model the multi-core microprocessor as Chip MultiProcessing (CMP) architecture with shared bus structure. The overall structure of our CMP is shown in Figure 1. There are multiple uniprocessor cores on the same chip, each of which is called a Processing Element (PE). Each PE is a fully functional microprocessor with local caches. Shared bus is the on-chip communication mechanism connecting the memory controller and PEs. The memory controller is in charge of off-chip memory accesses. It receives the memory requests from PEs, performs reads or writes to off-chip main memory, and returns the data from main memory to PEs.

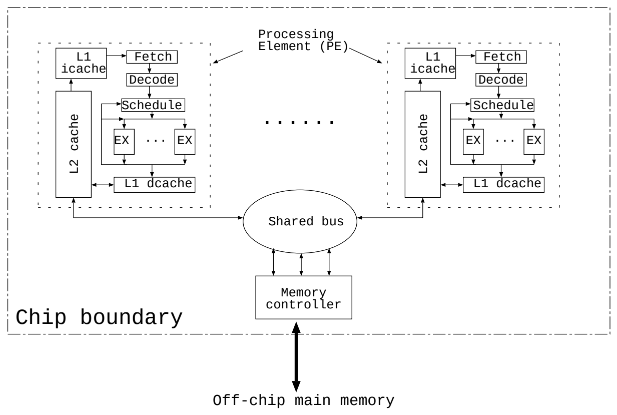


Figure 1: CMP architecture with shared bus.

One PE is consisted of two major structures. The first is the computing core, which is referred to as the pipeline of control logic and datapath. The computing core fetchs, decodes, schedules(issues) and executes instructions. We focus on SuperScalar architecture with multi-issue and out-of-order execution, and choose the following parameters to specify one computing core: issue width (IW), issue window (WIN) size, reorder buffer (ROB) size, load/store queue (LSQ) size, and number of functional units that include integer ALU (IALU), integer multiplier (IMULT), floating-point ALU (FPALU) and floating-point multiplier (FPMULT). The second major structure is the cache hierarchy. In this paper we assume two levels of cache hierarchy with level-one instruction cache (IL1), level-one data cache (DL1), and level-two unified cache (L2) for both instruction and data.

In this paper we assume homogeneous core design, which means all PEs have the same configuration of computing core and caches hierarchy. Although heterogeneous core design has been proposed in the literature [7], it is beyond the scope of this paper.

2.2 Problem Formulation

We study the throughput-optimal CMP design problem under given chip area constraint. We focus on instruction throughput, which is equal to the product of instruction-per-cycle (IPC) and the clock frequency. The total CMP system throughput is the sum of throughput of all PEs considering the impact of shared bus, which is modeled as increased memory latency in the calculation of a PE's throughput.

We formulate our CMP optimization problem as follows:

FORMULATION 1. **CMP optimization problem***: Given a total chip area constraint TA for a homogeneous CMP system with shared bus, decide the number of PEs and the configuration of each PE's computing core and cache hierarchy, to maximize CMP system throughput.*

In this problem formulation, we define the the configuration of the computing core as a data set $CORE$ that include data of IW, WIN size, ROB size, LSQ size, and the numbers of each type of functional units. The configuration of the cache hierarchy is another data set $CACHE$ including the design of IL1, DL1 and L2. Each cache design is identified by three parameter: C for cache size in kilobyte, B for block size in byte, and A for associativity. Such terminologies are similar to those in [8].

For each $PE_i, 1 \leq i \leq n$, we can select its computing core from totally T types of different computing core, $CORE_1, ..., CORE_T$, each of which is associated with a clock frequency $CK_m, 1 \leq m \leq T$ and area $CR_m, 1 \leq m \leq T$. The choices of cache hierarchy $CACHE_i$ in $PE_i, 1 \leq i \leq n$ include totally I types of level-one instruction cache $IL1_1$, ..., $IL1_I$, D types of level-one data cache $DL1_1$, ..., $DL1_D$, and U types of level-two cache $L2_1$, ..., $DL1_U$. Each type of cache is uniquely identified by their parameters of C, B and A. Each cache hierarchy has an area CHR_{jkl}, where $1 \leq j \leq I, 1 \leq k \leq D$, and $1 \leq l \leq U$.

For a given workload, we assume there is one benchmark running on each PE. The details of workload construction is discussed in Section 4.2. We target the multi-programming environment and assume all PEs are always busy during the process of a given workload. The total throughput of the CMP system is computed as (1):

$$TP \;=\; \sum_{i=1}^{N} (IPC(CORE_i, CACHE_i) \cdot CK_i) \qquad (1)$$

where IPC is the function to obtain the IPC for given PE considering the impact of shared bus. The chip area constraint is represented as (2):

$$\sum_{i=1}^{N} (CR_i + CHR_i) + bus\_area < TA \qquad (2)$$

where bus_area is the chip area dedicated to shared bus and memory controller.

Traditionally in microprocessor design, cycle-accurate simulation is the major method to obtain the IPC. However, cycle-accurate simulation is extremely time-consuming. In our CMP optimization problem, the possible solution space is a five-dimensional space on number of PEs, total types of computing cores, total types of IL1 total types of DL1 and total types of L2. Such space increases exponentially as we increase any design choices. Exploration of such a large solution space requires large number of iterations. It is not efficient, if not totally infeasible to to explore the design space by traditional, time-consuming cycle-accurate simulations. Instead, efficient yet accurate analytical performance models are necessary for our CMP optimization problem. In Section 3 we will discuss our systematic approach to solve the CMP optimization problem with novel analytical performance models.

3. METHODOLOGY

In this section we present our methodology for obtaining throughput-optimal CMP designs. We first develop analytic performance models for SuperScalar architectures and shared bus structure, based on the understanding of pipeline execution in the computing core, cache miss events and queuing theory. These models are then integrated into a traditional floorplanning optimization engine. This combination of expertize from two sides of both micro-architecture and CAD create a powerful methodology to effectively search the huge solution space of CMP designs, as well as to reveal the big picture regarding throughput of CMP systems from the perspective of area and technology scaling.

3.1 Overall CMP Optimization Flow

Our CMP optimization method include three components: (1) an analytical IPC model for SuperScalar architecture named *SS model*; (2) an analytical model for memory access latency of shared bus architectures called *bus model*; and (3) the Simulated Annealing *SA* engine to explore the large design space.

In our method, we use our bus model to decide the average memory access latency, given the number of PEs and the workload characteristics on each PE in current CMP design. Such memory access latency is fed into the SS model along with other off-line profiling information to calculate the IPC of one PE. The overall system throughput is then obtained by multiplying such IPC by the clock frequency and the total number of PEs. The SA engine takes the throughput results along with the number of PEs and configuration of each PE, to perform optimization.

3.2 Analytical Performance Model for SuperScalar Architecture

[9] proposed an analytical performance model for multi-issue out-of-order SuperScalar architectures. Hoever, the model in [9] assumes an unbounded number of arithmetic units and ignore the variation in pipeline depth introduced by pipelined interconnects. Extended from that in [9], our SS model achieves these additional requirements by decoupling all non-ideal aspects of the micro-architecture. We use off-line profiling to measure the impact of each independent source of performance loss and additively construct the performance estimation for a given processor configuration. Our SS model is suitable for iterative optimization methodologies such as simulated annealing because generating a performance estimation is a constant time operation. The full derivation and experimental validation of our analytical model is presented in [10].

Like [9] our model is formulated as the sum of the cycles per instruction, CPI, of an ideal processor and the CPI overhead introduced by each non-ideal aspect of the micro-architecture. Unlike [9] our model includes many additional sources of overlapping performance loss. Branch misprediction penalty, for instance, can be masked by performance loss due to data dependency stalls in the back end.

$$CPI = CPI_{ideal} + CPI_{IL1} + CPI_{L2} + CPI_{Front} + CPI_{Back} \tag{3}$$

The basic form of our model is shown in (3). Micro-architecture CPI is calculated by adding the CPI of an ideal micro-architecture (unit latency to memory, perfect branch prediction and minimal pipeline depth) to a set of sources of performance loss. These include the performance loss incurred due to instruction level one cache, CPI_{IL1}, and level two cache, CPI_{L2}. Also included in the model is the performance loss in the frontend and backend pipeline. CPI_{Front} includes branch misprediction and prefetch penalties that vary as a function of branch predictor setting and pipeline depth. CPI_{Back} includes performance loss due to data dependency stalls for integer and floating point operations that vary as a function of datapath latency. CPI_{Back} also includes $DL1$ cache performance loss that varies as a function of the access latency $DL1$ and $L2$ cache to determine the penalty of $DL1$ cache hits and misses.

We assume that CPI_{IL1} and CPI_{L2} do not overlap with the other sources of performance loss. To compensate for the overlap of branch prediction penalty, prefetch penalty and data dependencies, we assume that the timing of these events are independent random variables and construct a probabilistic overlap model for these performance loss events where the number and duration of each individual source of performance loss determines the probability that it overlaps with any other. The result is a ratio between (1) cycles during which the processor is considered to be performing useful work, and performance will suffer if a performance loss event were to occur; and (2) cycles during which the processor is considered to be incurring overhead due to some performance loss event that has already been counted by the model. This ratio is then used to scale the impact of performance loss in the front end to model the overlap of the various sources of performance loss.

3.3 Analytical Performance Model for Shared Bus Structure

We develop a new bus model based on the finite source queuing theory. We assume the CMP has one shared bus memory controller. There is only one shared memory module off-chip, but the memory module can accommodate multiple requests at the same time due to internal pipelining and subbanking.

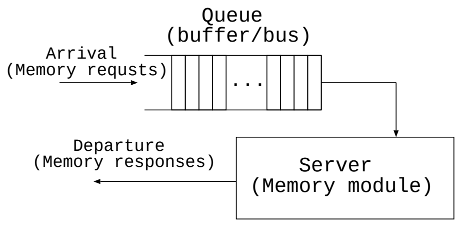


Figure 2: Queuing model for bus and memory structures. The queue includes the bus and the memory request buffer inside the memory controller. The server models the memory module.

The bus, memory controller, and memory structures can be modeled as a queuing system as shown in Figure 2, where bus and memory request buffer in the memory controller are modeled as the queue and the off-chip main memory module are modeled as the server. The overall system latency is the average memory latency T_a observed by each PE. For high-performance microprocessors running at a few GHz clock frequency, the average memory latency should be on the order of hundreds of cycles, given the latency of main memory module in tens of nanoseconds. Such long latency cannot be covered by out-of-order execution. Therefore, whenever there is a memory access due to L2 cache miss in any PE, that PE will eventually halt before the data comes back from the main memory. In other words, no more memory access can be generated from that PE. For this reason, we model the queueing system modeled as a finite source queue such as that in the machine repair problem [11] instead of a general M/D/1 queue[1]. Furthermore, if the main memory can handle c memory accesses simultaneously (due to internal pipelining and subbanking), we apply total c servers in our queueing system.

For each PE, the average memory access rate $\lambda = \frac{M}{C_p}$, where M is the total number of L2 misses and C_p is the number of execution cycles in the CPI_{ideal} case defined in the SS model in Subsection 3.2. The average response rate of one server is $\mu = 1/TM$, where TM is the main memory access latency. For a CMP with n PEs, assuming all PE runs identical benchmark (such assumption will be removed later), the utilization rate of the queueing system is $r = \lambda/\mu$, and the probability for exactly i memory requests to reside in the system is given in [11] as

$$p_i = \begin{cases} \frac{n!/(n-i)!}{i!} r^i p_0 & (1 \le i < c) \\ \frac{n!/(n-i)!}{c^{i-c}c!} r^i p_0 & (c \le i \le n) \end{cases}$$

[1]Infinite number of sources are assumed in general M/D/1 queues.

where p_0 is the probability of the case when no memory request is in the queueing system, given as (4):

$$p_0 = \frac{1}{1 + \sum_{i=1}^{c-1} \frac{n!/(n-i)!}{i!} r^i + \sum_{i=c} n \frac{n!/(n-i)!}{c^{i-c} c!} r^i} \quad (4)$$

The average number of memory requests in the system L is given by $\sum_{k=1}^{n} (k * p_k)$. According to Little's formula [11], the total system latency, i.e., the total memory access latency T_a, can be calculated as

$$T_a = \frac{L}{\lambda * (n - L)} \quad (5)$$

After the derivation, we remove the restriction that all PE run identical benchmarks. For different benchmarks, the memory access rates may be different. In this case, we make two changes: (1) assuming each PE has an access rate λ_i, we use the maximum rate $\lambda_{max} = max_{i=1}^{n}(\lambda_i)$ to replace access ratio λ; and (2) we use the *equivalent total PE number* $n_{eq} = \frac{\sum_{i=1}^{n} \lambda_i}{R_{max}}$ to replace the total PE number n. After that, T_a can be calculated following the same approach as discussed above.

Because of out-of-order execution, each PE may have multiple L2 cache misses and send out multiple memory requests before it halts due to the pending memory requests. Such phenomenon is called the *overlap of L2 cache misses* in our SS model. For benchmarks with a large number of L2 cache misses, such overlap can not be ignored and will significantly increase the average memory access latency. To consider this effect we append an empirical term to the latency calculated by our model. Finally, the average memory access latency T_{lat} is shown in (6):

$$T_{lat} = T_a + \alpha^{\frac{n-1}{n}} \quad (6)$$

where T_a is calculated from (5), n is the total number of PEs, and α is an empirical constant obtained by profiling.

3.4 Simulated Annealing Engine

To achieve optimal throughput for CMP systems, SA starts with a random configuration at a high temperature, and gradually decreases the temperature while changing the configuration by moves. The moves involving micro-architecture configurations include 1) choosing different types of computing cores, 2) choosing a different IL1, 3) choosing a different DL1, and 4) choosing a different L2. Throughput is re-evaluated after moves. Moves to improve throughput are accepted with no conditions. Otherwise, it can also be acceptable by a probability decreasing with the temperature.

The throughput of a CMP system depends on the configuration of each core as well as the number of cores in the system. Given a total area constraint, a compact floorplan could contain more cores and have a higher throughput. In this paper, we estimate the area of a CMP system by the bounding box containing all cores and buses, and obtain a compact floorplan of the system by employing moves in the SA engine to change the placement of components. These moves include changing the position, orientation and aspect ratio of a component, and are described in detail in the literature[12, 13, 14]. To maximize throughput and reduce the running time, moves for configuration and floorplanning are alternatively used with a ratio of around 1:5. As reported in [15, 16], interconnect latencies significantly impact performance. We extract interconnect latencies from the floorplanning and feed them into the analytical model.

4. EXPERIMENTAL SETTING AND MODEL VALIDATION

4.1 Experimental Settings

We collect totally 7 types of computing cores, as shown in Table 1. These types cover three levels of parallelism: 2-issue architecture exploring small ILP, 4-issue architecture for medium ILP, and 8-issue architecture for aggressive ILP. The numbers of functional units in each type are also listed in Table 1, where the functional unit is presented as a vector containing number of IALU, number of IMULT, number of FPALU, and number of FPMULT, respectively.

Clock frequency for each type of computing core is determined by the following approach: first we identify the integer ALUs and their bypass interconnects as the critical path. We require the execution of one integer ALU operation and the bypass of such data should be finished within one clock cycle. Therefore, the clock cycle time should be no shorter than the total latency of the integer ALU and associated bypass interconnects. We estimate the latency of one integer adder as $(log_2(bits) + 1)$ FO4, where bits is the bitwidth of data path and FO4 is the Fan-Out-4 delay. We assume 64-bit datapath and 65nm technology. The FO4 delay is obtained by SPICE simulation for the 65nm technology. For data bypass interconnects, we calculate the total interconnect length based on the estimation in [17]. By multiplying this interconnect length with the interconnect delay per unit length (67.4 ps for the 65 nm technology according to [18]), we obtain the total delay on the bypass interconnects. Finally the clock period is equal to the sum of ALU delay and the bypass interconnect delay. The results of clock frequency for each computing core is shown in Table 1.

IW	WIN size	ROB size	LSQ size	Functional units	Clock (GHz)
2	32	32	16	(2, 1, 1, 1)	4.68
4	64	64	32	(2, 1, 1, 1)	4.31
4	64	64	32	(4, 1, 2, 1)	4.31
8	128	128	64	(6, 1, 2, 1)	3.99
8	128	128	64	(8, 2, 4, 1)	3.72
8	256	256	128	(6, 1, 2, 1)	3.99
8	256	256	128	(8, 2, 4, 1)	3.72

Table 1: Choices of computing core. The functional unit is presented as a vector containing number of IALU, number of IMULT, number of FPALU, and number of FPMULT, respectively.

The area of issue window, ROB and LSQ are estimated by CACTI 3.0 toolset [8]. The area of functional units is based on the area of those in DEC ALPHA 21264 [19], and scale from 350nm technology down to 65nm technology. The detailed area results are omitted due to page limit but will be presented in our technical report.

We explore totally 21 types of IL1, 21 types of DL1, and 19 types of L2. The configuration of all these types are listed in Table 2, where each cache configuration is present in the format $(C/B/A)$. The area and access latency (in ns) of these configurations are obtained using the CACTI 3.0 toolset. During SA optimization, such access latency is converted to clock cycles, depending on the clock frequency of the individual computing core. The detailed area results are omitted due to page limit but will be presented in our technical report.

We assume the main memory latency is 40ns, which is converted to clock cycles during SA optimization. The main memory can simultaneously handle up to four requests due to internal pipelining and subbanking. The area for shared bus is determined by (7):

$$bus\_area = TA * (0.0001 * n^2 + 0.0021 * n + 0.0613) \quad (7)$$

IL1 or DL1		L2	
8/16/1	16/16/1	128/64/4	256/64/4
32/16/1	8/32/1	512/64/8	1024/64/8
16/32/1	32/32/1	2048/64/8	512/128/8
8/16/2	16/16/2	1024/128/8	2048/128/8
32/16/2	64/16/2	4096/128/8	512/256/8
16/32/2	32/32/2	1024/256/8	2048/256/8
64/32/2	32/64/2	4096/256/8	1024/128/16
64/64/2	32/16/4	2048/128/16	4096/128/16
64/16/4	32/32/4	1024/256/16	2048/256/16
64/32/4	32/64/4	4096/256/16	
64/64/4			

Table 2: Design freedom for caches. each cache configuration is present in the format $C/B/A$, where C is the cache size in kilobyte, B is the block size in byte, and A is the associativity.

where TA is the total chip area and n is the total number of PEs. This formula is obtained by curve fitting based on the results presented in [20].

We use SimpleScalar 3.0 [21] toolset for ALPHA ISA as the PE simulator, and develop additional programs to simulate the bus, memory controller, and memory module. During simulation, we create multiple instances of PE simulators and all of them communicate with the bus-memory simulator via UNIX interprocedure calls.

We implement our SA engine based on the *parquet* package [14]. We start the SA optimization engine with a high temperature of 50,000 and end it at a low temperature of 0.05. As observed in our experiment, the ratio of accepted moves decrease from over 90.0% at the high temperature to below 10.0%[2] at the low temperature, indicating that the solution space is freely accessed at the high temperature and the solution is frozen at the low temperature.

4.2 Workload Construction

We choose eight benchmarks from the SPEC 2000 benchmark set. Five of them are integer benchmarks: *bzip2*, *gcc*, *gzip*, *mcf*, and *parser*. The rest are floating-point benchmarks: *art*, *equake*, and *mesa*. For each benchmark, we first fastforward 400 million instructions and then simulate for 200 million instructions. We assume each benchmark has independent address space when running on each individual cores and there is no direct communication between different cores.

For any CMP with n cores we use *workload* to represent the group of n benchmarks running on it simultaneously. We construct the workload as follows: first, we consider workloads containing only identical benchmarks. Since we have eight benchmarks, we have eight such workloads. Then, we generate a benchmark list where each item on the list is a random selection of one of the eight benchmarks. After that, we use a sliding window (with wraparound) of size n to select n benchmarks on the list, and compose one workload. We compose multiple workloads by repeatedly shifting the window right by one. This method of workload construction with sliding window is similar to that used in [7]. In our experiment, we construct eight workloads by sliding window. Therefore, for any CMP system, we use sixteen workloads to evaluate its throughput. The final CMP throughput metric is the average throughput over all sixteen workloads. Although this number of workloads may not cover all possible workloads on the CMP system, we believe it is sufficient to statistically represent the average performance in a realistic computing environment.

4.3 Model Validation

We validate our model by cycle-accurate simulation. We evaluate three different situations: (1) the single core configuration is fixed and the number of cores is varied. All cores run identical benchmarks. In this case, no total area constraint is enforced. (2) the total area is fixed, but the single core configuration is varied (the core number changes accordingly). All cores still run identical benchmarks. And (3) the same setting as the second case, except that different benchmarks are applied on different cores. The first situation validates the model fidelity as core number changes. The second and third situations approximately cover most of realistic situations during our optimization.

IW	WIN size	ROB size	LSQ size	Functional units	Clock (GHz)
4	64	64	32	(4, 1, 2, 1)	4.31

IL1	DL1	L2
32/16/2	32/16/2	1024/128/8

Table 3: PE configuration for our first validation case.

Table 3 lists the configuration of PEs used for the first case. We adjust the core number from one to eight. Figure 3 plots the curves for performance with different number of cores. The results from two benchmarks *art* and *mcf* are shown. For benchmark *art*, the configuration of single core results in a large number of L2 cache misses, which put a heavy burden on the shared bus. Therefore, the performance is not linearly scalable with respect to the number of cores. Instead, it tends to saturate as core number approaches to eight. On the contrary, benchmark *mcf* with such configuration of single core presents little memory-boundness due to a small number of L2 cache misses. Therefore, the performance scales almost linearly when the core number increases.

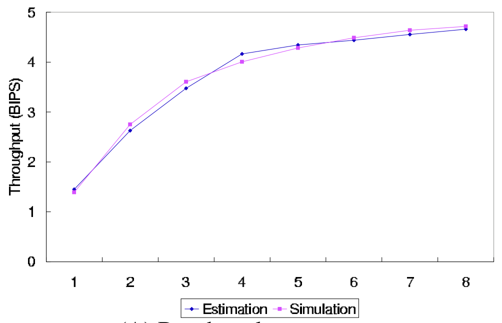
From Figure 3 we can see that results from our analytical model closely match the performance change, and correctly present the two different performance trends. For *mcf*, although the gap between estimation and simulation increases as we increase the core number, the relative error is constant at around 10%. These results clearly domenstrate the high fidelity of our analytical model.

	IW	WIN size	ROB size	LSQ size	Functional units	Clock (GHz)
CMP_1 (8 PEs)	2	32	32	16	(2, 1, 1, 1)	4.68
	IL1		DL1		L2	
	16/16/1		16/16/1		512/64/8	
	IW	WIN size	ROB size	LSQ size	Functional units	Clock (GHz)
CMP_2 (4 PEs)	4	64	64	32	(4, 1, 2, 1)	4.31
	IL1		DL1		L2	
	32/16/2		32/16/2		1024/128/8	
	IW	WIN size	ROB size	LSQ size	Functional units	Clock (GHz)
CMP_3 (2 PEs)	8	128	128	64	(8, 2, 4, 1)	3.72
	IL1		DL1		L2	
	64/32/2		64/32/2		2048/128/8	

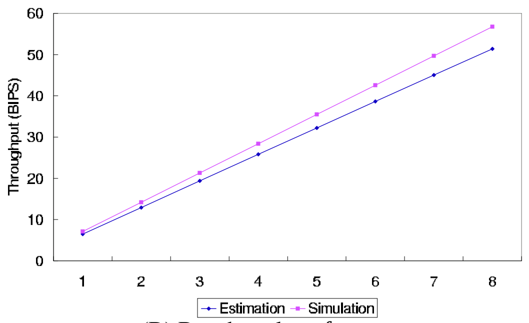
Table 4: Three CMPs with PE number and per-PE setting in our second validation case.

For the second case, we fix the total chip area at 250 mm^2, and

[2]includes moves keep throughput the same.

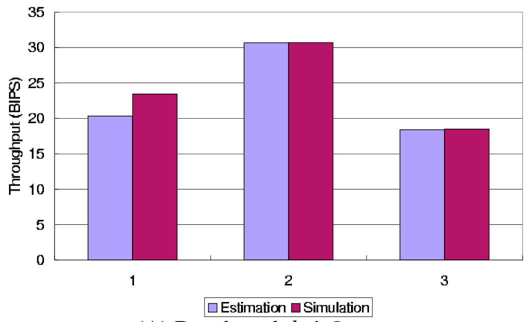


(A) Benchmark *art*

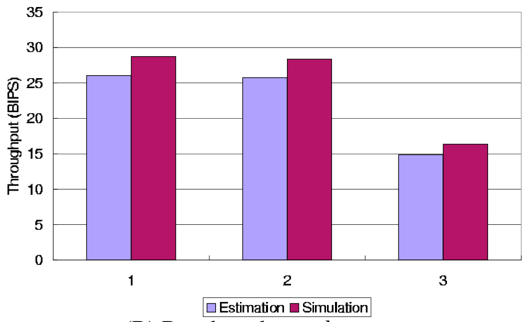


(B) Benchmark *mcf*.

Figure 3: CMP performance with identical benchmarks. All cores have the same configuration.



(A) Benchmark *bzip2*



(B) Benchmark *equake*.

Figure 4: CMP performance with identical benchmarks. All three configurations have the same total area constraint.

select three different CMP systems as listed in Table 4. Figure 4 compares the results from cycle-accurate simulation and our analytical model. The results from two benchmarks *bzip2* and *equake* are shown. Again, our analytical method provides high Fidelity and accurate estimation compared to cycle-accurate simulation.

In the third case, we keep all the three CMP systems in Table 4 unchanged, but apply workload with mixed benchmarks. For CMP_1, we apply all eight benchmarks we choose. For CMP_2, we apply these four benchmarks: *gcc*, *gzip*, *equake*, and *mesa*. For CMP_3, we apply the two benchmarks: *gzip* and *mesa*. All the mixes of benchmark include both integer and floating-point benchmarks. Figure 5 compares the results from our analytical estimation against those from cycle-accurate simulation. Overall, our analytical estimation closely keeps track of the performance change for different CMP design with high fidelity. The average error is 9%.

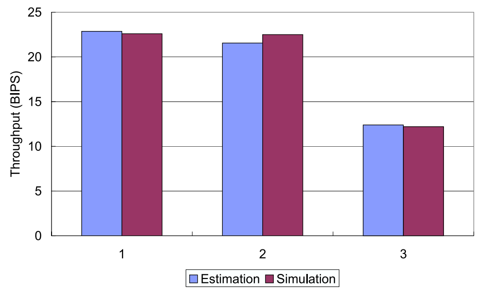


Figure 5: CMP performance with mix benchmarks.

4.4 Throughput Comparison between Optimal CMP and Existing Design

We demonstrate the effectiveness of our CMP optimization methodology by comparing the throughput of our optimal CMP design against the dual-core system based on an IBM POWER5-like core used in [5], under the same area constraint. According to [5], we configure the POWER5-like core to be a 5-issue out-of-order SuperScalar processor with 2 IALUs, 1 IMULTs, 2 FPALUs, 1 FPMULTs, 32/128/2 IL1, 64/128/2 DL1. In [5] the two POWER5-like cores share a 2MB L2 cache. We do not consider shared cache, instead, we apply one 1MB L2 cache to each core and make the total L2 cache for the whole dual-core system match the setting in [5].

We calculate the total area of the POWER5-like dual-core system according to our method, and design the throughput-optimal CMP by our optimization methodology with the same chip area constraint, which we named *optimal design*. Figure 6 compares the throughput of POWER5-like dual-core system and the optimal design. Approximately the optimal design achieves 23% higher throughput than the POWER5-like dual-core system. Given the significant throughput difference between the optimal design and

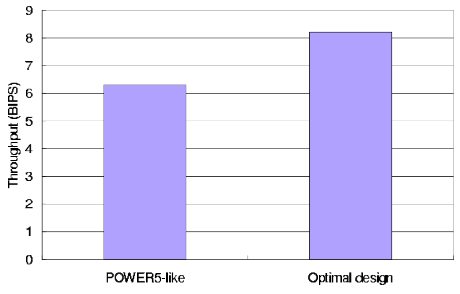


Figure 6: CMP throughput comparison. The optimal design is the CMP designed by our throughput-optimal methodology.

POWER5-like dual-core system, we believe the assumption to use split L2 cache instead of shared L2 cache is unlikely to affect the conclusion that our methodology can achieve better CMP design than simply putting existing single-core designs together. This result clearly illustrates the effectiveness of our CMP optimization methodology.

5. SCALING OF CMP DESIGN

In this section, we leverage our CMP optimization method to study the impact of area and technology scaling, and shared bus structure in the current state-of-the-art CMP design. All CMPs in this section refer to the throughput optimal CMP obtained by our methodology.

5.1 Impact of Area and Technology Scaling

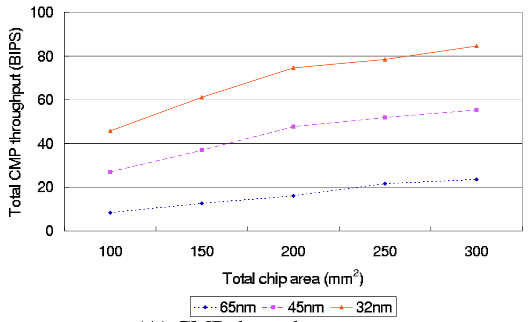
We study the impact of technology scaling from 65nm down to 45nm and 32nm. For 45nm technology, we modify our experimental settings for area and clock frequency, following the same methodology as in 65nm technology, and repeat the CMP optimization. For 32nm technology, we are not able to obtain SPICE model file to determine the FO4 delay. Alternatively, we make two assumptions: (1) the FO4 delay scales down linearly with respective to the feature size. Such relationship matches our results when scaling down from 65nm to 45nm technology. And (2) wire delay per unit length for the bypass interconnects stays constant. This assumption is based on the results of wire delay per unit length in 65nm and 45nm technologies presented in [18]. With these two assumptions we can estimate the clock frequencies for all types of computing cores in 32nm technology. The clock frequencies in different technology generations are shown in Table 5.

	IALU number	2	4	6	8
Clock frequency (GHz)	65nm technology	4.68	4.31	3.99	3.72
	45nm technology	7.13	6.54	6.04	5.61
	32nm technology	10.16	9.31	8.59	7.97

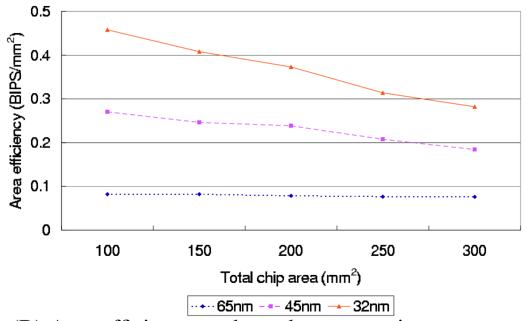
Table 5: Clock frequencies in GHz for different generations and types of computing cores. Note we estimate the clock frequency based on the latency of execution and bypass interconnects. So the clock frequency only depends on the number of integer ALU.

We scale the total chip area constraint from $100mm^2$ to 300 mm^2 with step $50mm^2$. Such range is within the reasonable chip area specified by International Technology Roadmap for Semiconductors (ITRS) [22]. Figure 7 shows the CMP throughput and area efficiency defined as throughput per unit area for different total area constraints in 65, 45 and 32nm technologies. For all three technologies, similar trends are observed for throughput and area efficiency with respect to total chip area. Overall, CMP throughput increases with larger total area or more advanced technology. From Figure 7 (A) we can see the throughput difference between technology generations increases when the chip area increases from $100mm^2$ to $300mm^2$. Such results indicate that technology scaling has stronger impact on CMP with larger total area. The reason is that in more advanced technology, we can explore more design choices with larger total area. As shown in 7 (B). different from CMP throughput, the area efficiency is larger with more advanced technology, but decreases as the total area increases. Similarly, the area efficiency gap between different tecnologies decreases as total area increases. More importantly, it is easy to observe that in more advanced technologies, the area efficiency decreases more quickly as we increase total area. In other words, the area efficiency advantage at small chip area is more significant in more advanced technology. The reason will be discuss in Subsection 5.2. Clearly, as semiconductor technology keeps scaling down, small chip area may become more and more preferred in terms of area efficiency.

From Figure 7 we can also see that there are significant gaps between the throughput of different technology generations. This result indicates that technology scaling can be every effective in increasing CMP throughput. For better illustration, Figure 8 shows the normalized throughput of CMP and single-core design for the three technologies. It is easy to see that CMP throughput scales linearly with respect to technology generation, but the throughput from single-core design tends to saturate after 45nm technology.



(A) CMP throughput.



(B) Area efficiency as throughput per unit area.

Figure 7: Impact of technology scaling.

Clearly, unlike the single-core design where technology scaling has been rewarded with diminishing performance increases, CMP design can significantly benefit from technology scaling. By scaling from 65nm to 32nm technology, we can increase the CMP throughput by $5.58X$, under the same chip area constraint.

Figure 8 also shows that the throughput gap between CMP and single-core design keeps increasing. Such trends convincingly point out that CMP will become the prevailing architecture in the future high-performance microprocessor design.

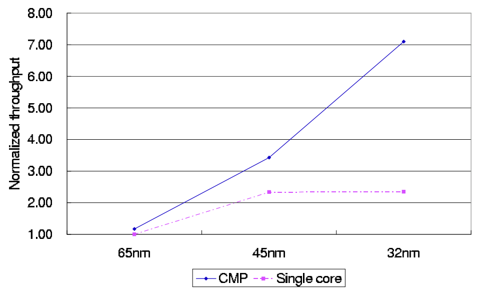


Figure 8: Normalized throughput for 65, 45, and 32nm technologies. The chip area is limited to 100mm^2.

An interesting observation by comparing Figures 4 and Figure 7 (A) is that the maximum CMP throughput with 250mm^2 chip area shown in Figure 7 (A) for 65nm technology is smaller than the throughput from CMP_2 configuration with the same chip area in our validation experiment shown in Figure 4. The reason is that in validation (Figure 4) the results are only for one individual benchmark, while the results from optimization (Figure 7 (A) for 65nm technology) are based on mixed-benchmark workloads. This results indicate the importance of our workload construction.

5.2 Impact of Shared Bus Structure

In this paper we assume shared bus structure for CMP design due to its simplicity and low design cost. However, such structure may lack of scalability when the core number is large. We believe such in-scalability is part of the reason for the fact that, as chip area increases, both area efficiency and the area efficiency gap between different technology generations decrease. Table 6 summarizes the core number for each optimal CMP design under different area constraints in all three technologies we study. From Table 6 we can see the number of cores in a CMP increases by $2X$ when the total area increases from 100mm^2 to 300mm^2. Large numbers of cores increase the bus contention and reduce the overall area efficiency of our CMP system. Furthermore, the impact of shared bus may be relieved with SPEC2000 benchmark due to its lack of memory-bound benchmarks [5], and is expected to be more severe in workloads with large numbers of memory-bound benchmarks. Above all, we believe that as multi-core design becomes aggressive (i.e., with cores numbering thirty or more), alternative on-chip communication mechanisms such as Network-on-Chip (NoC), may be a better design choice for area efficiency.

6. CONCLUSION

We have studied the design and scaling of multi-core microprocessor modeled as Chip-level Multi-Processors (CMP) with a shared bus structure. We develop analytical models to avoid time-consuming cycle-accurate simulation, and perform quick yet accurate multi-core design space exploration. Experiments show that

	Area (mm^2)				
Technology	100	150	200	250	300
65 nm	4	6	8	7	12
45nm	9	13	13	21	24
32nm	17	25	22	27	33

Table 6: Core number in optimal CMP design.

our models achieve high fidelity and an average error of 9% compared to cycle accurate simulation, and our optimized CMP design with a same area has 23% higher throughput compared to a published dual-core system based on IBM POWER5-like cores. In addition, we study the impact of technology and area scaling and scalability of the shared bus structure. We show that technology scaling has a diminishing or saturated throughput increase for single-core design, but multi-core design with the same area improve throughput by $5.58X$ when scaling the technology from 65nm to 32nm. Furthermore, as chip area increases, the area efficiency defined as throughput per unit area decreases, and so does the area efficiency gap between different technology generations. Both are due to the growing limitation of the shared bus structure.

7. REFERENCES

[1] J. Clabes and et al, "Design and implementation of the POWER5 microprocessor," in *ISSCC*, 2004.

[2] T. Takayanagi, J. L. Shin, B. Petrick, J. Su, and A. S. Leon, "A dual core 64b UltraSPARC microprocessor for dense server applications," in *DAC*, 2004.

[3] L. Hammond, B. A. Nayfeh, and K. Olukotun, "A single-chip multiprocessor," *IEEE Computer*, vol. 30, pp. 79–85, Sept. 1997.

[4] J. Burns and J.-L. Gaudiot, "Area and system clock effects on SMT/CMP throughput," *IEEE Transactions on Computers*, vol. 54, pp. 141–152, Feb. 2005.

[5] Y. Li and et al, "Performance, energy, and thermal considerations for SMT and CMP architectures," in *HPCA*, 2005.

[6] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus ipc: The end of the road for conventional microarchitectures," in *ISCA*, 2000.

[7] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of 31st Annual International Symposium on Computer Architecture*, Jun 2004.

[8] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model," in *WRL Research Report 2001/2*, 2001.

[9] T. Karkhanis and J. Smith, "A first-order superscalar processor model," in *ISCA*, 2004.

[10] L. J. Simonson and L. He, "Micro-architecture performance estimation by formula," in *Embedded Computer Systems: Architectures, MOdeling, and Simulation*, 2005.

[11] D. Gross and C. M. Harris, *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc, 1998.

[12] N. Sherwani, *Algorithms For VLSI Design Automation*. Kluwer, 3rd ed., 1999.

[13] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence pair," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1518–1524, 1996.

[14] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proc. IEEE Int. Conf. on Computer Design*, pp. 328–334, 2001.

[15] C. Long, L. Simonson, W. Liao, and L. He, "Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects," in *Proc. Design Automation Conf*, pp. 640 – 645, June 2004.

[16] M. Ekpanyapong, J. R. Minz, T. Watewai, H.-H. S. Lee, and S. K. Lim, "Profile-guided microarchitectural floorplanning for deep

submicron processor design," in *Proc. Design Automation Conf*, pp. 634 – 639, June 2004.

[17] N. S. Palacharla and J.E.Smith, "Quantifying the complexity of superscalar processors.," in *Technology Report CSTR-96-1328, Univ. of Wisconsin-Madison*, November 1996.

[18] K. Banerjee and A. Mehrotra, "Power dissipation issues in interconnect performance optimization for sub-180 nm designs," in *Proceedings of 2002 Symposium on VLSI Circitus*, 2002.

[19] B. A. Gieseke and et al, "A 600MHz superscalar RISC microprocessor with out-of-order execution," in *Proc. IEEE Int. Solid-State Circuits Conf.*, pp. 176–177, 1997.

[20] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," in *(to appear) Proceedings of 32^{n}d Annual International Symposium on Computer Architecture*, June 2005.

[21] S. LLC, "The simplescalar tool set," in *http://www.simplescalar.com*.

[22] *The International Technology Roadmap for Semiconductors*. http://public.itrs.net/, 2004.