Fundamental Conditions on Bottleneck Resource Identification for Multithreaded Processors

Abstract

This paper studies general properties for a class of queuing network models that at the thread level, characterize a class of single-core multithreaded processors with various kind of thread scheduling disciplines and parallel resources. In particular, we find, with mathematical rigor, the general conditions under which the bottleneck resources appear. Our simulation testing demonstrates that these general conditions remain to be accurate when some key simplistic assumptions made in the queuing network models are removed. The test of these conditions for a specific workload and processor in this class only involves some statistic parameter estimation of the workload, making it possible to quickly identify bottleneck resources without actually running the program in the processor. We also demonstrate how these general conditions, combined with any given caching model, can lead to useful guidelines and algorithms for thread and cache resource provisioning that maximizes the throughput performance. Finally, we provide insights on how these conditions may be generalized to the case of many-core processors.

Keywords: Multi-threading, Queuing network

1 Introduction

As the gap between processor speed and resource (e.g., memory and I/O) access speeds ever widens, the role of multithreading and caching as the two major resource access latency hiding and reduction techniques

has become increasingly important. However, it is a challenge as to how to provision thread and cache resources to achieve high throughput performance, which is determined by multiple intertwined factors, including workload characteristics, thread scheduling discipline, resource access latencies, and resource access mechanisms. The thread and cache resource provisioning must take all these factors into account to be effective. In fact, adding more threads and caching more data with respect to a given resource do not always help hide more resource access latency from the CPU, if the resource access itself does not contribute to longer CPU idle time. To be effective, the thread and cache resource provisioning must aim at removing resource bottlenecks that lead to longer CPU idle time and hence, reduced overall throughput performance.

Hence, for thread and cache provisioning, a key issue to be addressed is how to identify bottleneck resources that constrain the overall throughput performance. Benchmark testing is the widely adopted approach to address this issue. However, this approach has limited predictive power for future workloads and is time consuming. The existing design space exploration techniques aiming at finding general properties of the design space requires intelligent interpolation of sampled benchmarks and hence can still be too time consuming and heavy-duty from a programmer's point of view. A more practical approach is to be able to identify bottleneck resources by simply performing a test of certain conditions involving all the above mentioned intertwined factors. In this paper, the aim is to find such general conditions for a large class of single-core processors. The generalization of the results in this paper to the case of many-core processors will be reported elsewhere. In Section 8, however, the insights on how we may extend the current results to the many-core case will be briefly discussed.

Given the intricate interactions among many factors, it would be very difficult, if ever possible, to derive general conditions from benchmark or simulation based approaches. Ideally, these general conditions need to be presented in a mathematical expression involving all the interacting factors. To this end, an analytical modeling technique that can capture all the major interacting factors and be solved analytically should to be sought. In this paper, we show that such general conditions can be derived for a class of closed queuing

networks that models a class of single-core multithreaded processors at the thread level, overlooking micro-architectural details. As a first order approximation, this class of processor models allows simultaneous multithreaded (SMT), fine-grained, and coarse-grained CPU and sequential, pipelined, and parallel resource access mechanisms. We also demonstrate that these general conditions remain to be accurate, when some key simplistic assumptions made in this class of models are removed. Moreover, our testing of a thread-level processor simulation tool against cycle-accurate simulation indicated that overlooking micro-architectural details, such as instruction-level-pipeline (ILP) aborts, introduces $5-15\%$ inaccuracy to the thread-level tool [24]. Hence, we believe that these general conditions can be used in practice to aid the initial configuration of the thread and cache resources.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes the modeling technique used in this paper. Section 4 presents the main results, i.e., the general conditions, and highlights the main ideas by means of an example. Section 5 gives the theoretical proof of the general conditions. Based on these general conditions, Section 6 develops a generic algorithm that serves as a useful guideline for the development of algorithms for effective thread and cache resource provisioning. Section 7 tests the accuracy of the general conditions with some key assumptions in the modeling technique removed. Finally, Section 8 concludes the paper and presents a brief discussion on how we may generalize the general conditions to the case of many-core processors.

2 related work

For a given processor and workload, the bottleneck resources can be identified through simulation and/or benchmark testing (e.g., [5, 7, 8, 12, 14]). Clearly, general conditions cannot be drawn from such approaches. In contrast, the existing design space exploration techniques (e.g., [6, 10, 11, 18]) can identify bottleneck resources over a large design space. However, since they are all based on intelligent search algorithms guided by the benchmark samples, it is computationally too expensive, if ever possible, for the

existing techniques to construct the general conditions for bottleneck resources to appear, expressible in terms of the functional relationships among various system and workload parameters covering the entire design space. More importantly, it is overkill for a programmer to employ such techniques to aid the programming.

Compared with the above approaches, analytical modeling techniques are more viable in dealing with the problem in question. We can roughly classify the existing analytical processor models into two categories, i.e., processor specific and generic. The processor specific models attempt to model in substantial detail of specific processor architectures and to accurately characterize the performance of the processors, e.g., [4, 19]. It is obvious that by design, processor specific models are not suitable for the study of the general properties pertaining to a large design space.

On the other hand, the generic models attempt to capture the general properties shared by many processors of the same kind, overlooking micro-architectural details. The most notable models of this kind is Amdahl's [20] and its variations (e.g., [21]) that capture the scaling properties for multiprocessors without taking into account of the memory resource access mechanisms, multithreading disciplines, and so on. In [9], a mean value analysis of a generic multithreaded multicore processor model of similar nature is performed. The performance results reveal that there is a performance valley to be avoided as the number of threads increases, a phenomenon similar to the one found in [15, 1] in the context of single multithreaded processors. Again, this model does not take into account of the processor specific features into account, such as memory access mechanisms and thread scheduling disciplines. Although the generic queuing models presented in [15, 1, 13] do capture somewhat more processor specific features, which, however, are restricted to a single processor model, i.e., an M/M/1 queuing model, modeling both coarse-grained CPUs and memory resources with a FCFS access mechanism. Moreover, none of the existing models attempts to model the workload in a large workload space. If we draw a three dimensional design space in a cone shape, in terms of resource access mechanisms, thread scheduling disciplines, and workloads as shown in Fig. 1,

the existing analytical models only cover a small cone on the left.

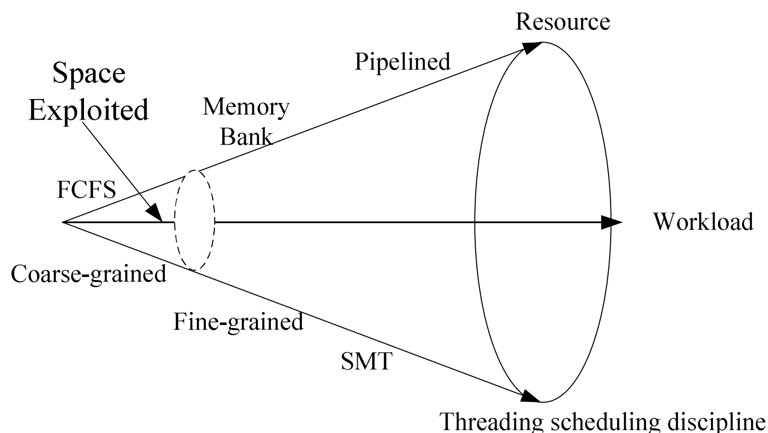

Figure 1: Design Space

The approach taken in this paper differs from the above existing ones in the sense that it directly studies a class of processor models, which inherently accounts for all possible component models defined in the design space in Fig. 1, including SMT, coarse and fine grained thread scheduling disciplines for a multi-threaded CPU and both sequential and parallel/pipelined access mechanisms for other resources, and the entire workload dimension. As a result, the general properties obtained for this class of processor models apply to the entire design space. Even though the work presented in this paper is restricted to a class of single-core processors, the approach taken can be generalized to the case of many-core processors.

3 Model

The approach taken in this paper finds a middle ground between generic modeling and processor-specific modeling approaches. To scale to a large design space, the approach focuses on modeling thread-level activities, overlooking microarchitectural details. In the meantime, it takes into account processor specific features that have strong effects on the thread level performance. Note that based on our testing results in [24], overlooking micro-architectural details, such as instruction-level-pipeline (ILP) aborts, only introduces $5-15\%$ inaccuracy to the thread-level models/simulators. So we can expect that the results obtained in this

paper could be used in practice for quantitative resource allocations, as well as providing significant insights on understanding the processor behaviors.

In this section, we describe our model, including processor model organization, processor component models, and the workload model.

Processor Model Organization: Like the existing queuing network models for multithreaded processors (e.g., [15, 1, 13]), our model works at the thread level, in the sense that it only captures the events that have major impacts on the thread level performance. In other words, the instruction level and microarchitectural details are overlooked, unless they trigger events at the thread level, such as an instruction for memory access that causes the thread to stall or an instruction to access a critical region that causes serialization effect at the thread level. Correspondingly all the components including CPU, cache/memory, and interconnect network are modeled at a highly abstract level, overlooking microarchitectual details, just enough to capture the thread level activities. More specifically, each component is modeled as a queuing server with a set of possible queuing disciplines, modeling the thread scheduling disciplines or resource access mechanisms, as given in Fig. 1. These queuing servers are interconnected to form a queuing network. In this queuing network, a number of jobs or threads move from one server to another with given routing probabilities (note that this view is purely conceptual and in a real processor, it is not the threads that move from one component to another but the tasks that the threads handle).

In this paper, we consider a class of closed queuing network models given in Fig. 2. This class of models characterizes a class of processors with a single CPU, a cache, and an arbitrary number of parallel resources, denoted as mq, (e.g., a main memory, a coprocessor, an I/O device, or even a critical region). In this model, we assume that the interconnection network has sufficient bandwidth to transfer data between CPU and the parallel resources without creating a bottleneck and hence it is not explicitly modeled. Upon exiting the CPU server, a thread has probability p_{0i} to visit parallel resource i, where $i = 1, 2, \cdots, mq$. In the case of memory resource access, the thread will first check if the requested data is available in the cache.

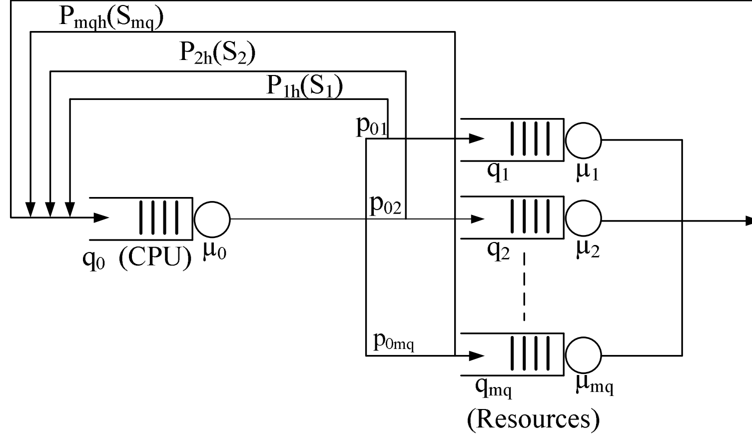

Figure 2: queuing network model

In our model, no details of cache access mechanisms are modeled, except a cache hit probability $P_{ih}(S_i)$ that causes the thread to immediately loop back to the CPU and a cache miss probability $(1 - P_{ih}(S_i))$ that causes the thread to access the ith resource. Here S_i is the size of the cache memory block allocated to the cached data from memory resource i. All the other components are modeled as queuing servers. The CPU queuing server mimics the thread scheduling disciplines listed in the thread scheduling discipline dimension in Fig. 1 and all other parallel queuing servers mimic the resource access mechanisms in the resource dimension in Fig. 1. An arbitrary number of threads, M_t, circulates in the closed queuing network, with routing probability p_{ij} to visit server j upon exiting server i, mimicking the stochastic behaviors of a long-running parallelizable program or multiple programs handled by threads in parallel (such as parallel packet processing in a network processor or parallel queries at a server processor).

Component Models: Without resorting to any approximation techniques, the existing queuing network modeling techniques will allow both resource and thread scheduling discipline dimensions in the design space (see Fig. 1) to be exploited analytically. As a first order approximation, any instance in these two dimensions can be modeled using a queuing model that has local balance equations (i.e., it leads to solutions of product form or closed form). More specifically, Table 1 shows how these two dimensions can be approximately modeled by only three queuing models with local balance equations, including $M/G/\infty$; M/M/m FCFS

Component \ Queue Model	$M/G/\infty$	M/M/m FCFS	M/G/1 PS	M/M/1
SMT	✓	✓		
Fine-Grained Thread scheduling			✓	
Coarse-Grained Thread scheduling scheduling				✓
FCFS shared Memory, Cache, Interconnection Network,or Critical Region				✓
FCFS Memory with with Popelined Access		✓		

Table 1: Component modeling using queuing models with local balance equations

(including M/M/1); and M/G/1 PS (processor sharing).

Note that for all the multithread scheduling disciplines, the service time distribution in a queuing model models the time distribution for a thread to be serviced at the corresponding queuing server. With this in mind, the following explains the rationales behind the mappings in Table 1:

- SMT: It allows multiple issues in one clock cycle from independent threads, creating multiple virtual CPUs. If the number of threads in use is no greater than the number of issues in one clock cycle, the CPU can be approximately modeled as an $M/G/\infty$ queue, mimicking multiple CPUs handling all the threads in parallel, otherwise, it can be approximately modeled as an M/M/m queue, i.e., not enough virtual CPUs to handle all the threads and some may have to be queued.

- Fine-grained thread scheduling discipline: All the threads access the CPU resource will share the CPU resource at the finest granularity, i.e., one instruction per thread in a round-robin fashion. This discipline can be approximately modeled as an M/G/1 PS queue, i.e., all the threads share equal amount of the total CPU resource in parallel.
- Coarse-Grained thread scheduling discipline: All the threads access the CPU resource will be serviced in a round-robin fashion and the context is switched only when the thread is stalled, waiting for the return of other resource accesses. This can be approximately modeled as a FCFS queue, e.g., an M/M/1 queue.
- FCFS Shared Memory, Cache, Interconnect Network, or Critical Region: This kind of resources can be modeled as an M/M/1 queue.
- FCFS Memory with Pipelined Access: The pipeline depth determines how many threads can be serviced simultaneous in the M/M/m FCFS queue.

Note that in the case of memory structure with multiple memory banks, each memory bank needs to be modeled as a separate queuing server.

Workload Model: A workload is generally defined as the collection of data processing requirements presented to the system during a specific period of time [17]. In the context of our model, *a workload is a mapping of a program to the model organization and a model configuration*.

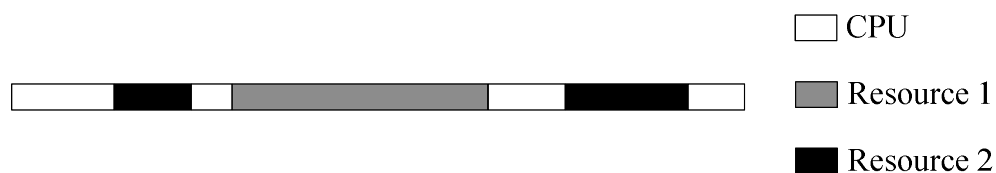

Figure 3: code path

To be more specific, consider an example code path at the thread level, depicted in Fig. 3. This code path is a mapping of a program or a program task to a thread in the model organization in Fig. 2. It is

composed of different colored segments, each corresponding to the number of CPU cycles the thread is processed at the CPU or a resource. Obviously, the length of each segment is determined by both program or program task characteristics and the processing speed at the corresponding CPU or resource. It represents the actual service time the thread is being serviced at a queuing server (not including the queue waiting time). Therefore ideally, the service time distribution for a queuing server must be equal to the segment length distribution for all the segments from all the threads processed at that server. This can be done for $M/G/\infty$ and $M/G/1$ PS queuing models in Table 1. However, for the M/M/1 and M/M/m queuing models in Table 1, the service time distributions are a given, i.e., exponentially distributed. This service time distribution is uniquely determined by the average service rate μ_i for queuing server i. In this case, μ_i should be calculated as the inverse of the average segment length for all the segments from all the threads processed at server i. This is obviously an inaccurate characterization of the workload. In spite of this, in Section 7, we shall demonstrate that the general conditions derived from this simple model turns out to be accurate. In what follows, we use μ_i to collectively represent the service time distributions or configuration parameters that need to be configured for server i. Obviously, the routing probability p_{ij}, in the model in Fig. 2, should be determined by the i-to-j segment transition statistics collectable from the code paths handled by all the threads.

In summary, a workload for the model in Fig. 2 is uniquely defined by a given model configuration in terms of $\{\mu_i\}$ that determine the service time distributions for all the queuing servers and routing probabilities $\{p_{ij}\}$, hereafter denoted as ($\{\mu_i\}$, $\{p_{ij}\}$). As a result, ($\{\mu_i\}$, $\{p_{ij}\}$) defines the entire workload space. Our work focuses on the study of the entire workload space, i.e., the space spanned by the parameters ($\{\mu_i\}$, $\{p_{ij}\}$).

4 Main Results

To limit the exposure, for the time being, we assume that there is no cache in the model in Fig. 2, i.e., $P_{ih}(S_i) = 0$. The results with caching will be given at the end of this section. Define $P(m)$ as the probability that there are m threads at the CPU server. We focus on the performance measure $P_I = P(0)$, i.e., the probability that the CPU server is in the idle state. In particular, we are interested in its asymptotic behavior, i.e., whether $\lim_{M_t \to \infty} P_I = 0$. It tells us whether or not multithreading can completely hide the resource access latencies from the CPU, provided that the thread resource is abundant, and hence, whether the multithreading can help achieve the maximum throughput performance. This performance measure will lead to the identification of the general conditions under which the bottleneck resources are bound to appear, regardless how many threads are used.

Define $f_i(k_i)$ as the steady state probability that there are k_i threads at queuing server i, for $i = 0, 1, \cdots, mq$, where $\sum_{i=0}^{mq} k_i = M_t$. Let q_i represent queuing server i (see Fig. 2), for $i = 0, 1, \cdots, mq$. Following the convolution algorithm [2], we have,

$$P_I = \frac{f_0(0) q_{1*2*\cdots*mq}(M_t)}{q_{0*1*2*\cdots*mq}(M_t)} \tag{1}$$

where

$$q_{0*1*2*\cdots*mq}(M_t) = \sum_{m+n=M_t} f_0(m) q_{1*2*\cdots*mq}(n) \tag{2}$$

and

$$q_{1*2*\cdots*mq}(n) = \sum_{k_1+\cdots+k_{mq}=n} \prod_{i=1}^{mq} f_i(k_i) \tag{3}$$

Eq. (1) holds true for any queuing servers listed in Table 1 and any model parameters $(\{\mu_i\}, \{p_{ij}\})$. In other words, P_I is a performance measure for a class of processor models defined in the entire design space

in Fig. 1.

According to Table 1, both $M/G/\infty$ and $M/M/m$ queuing models can be used to model SMT-based CPU server and the $M/G/1$ PS queuing model is used to model fine-grained CPU server in Fig. 2. All the resource-related servers can be modeled using $M/M/1$ and $M/M/m$ queuing models. To simplify the design, in this paper, we adopt $M/M/m$ for SMT and only consider a special case of $M/G/1$ PS, i.e., $M/M/1$ PS for the fine-grained CPU server. So the queuing model for CPU server in Fig. 2 is $M/M/m_0$, and queuing model for the resource-related servers are $M/M/m_i$ $i = 1, \cdots, mq$. With these simplifications, the service time distributions for all the queuing servers are then exponentially distributed and uniquely determined by their average service rates μ_i for $i = 0, 1, \cdots, mq$ and $f_i(k_i)$ can be generally express as follows:

$$f_i(k_i) = \frac{\alpha_i^{k_i}}{\beta(k_i)} \tag{4}$$

where α_i is the relative utilization of q_i. $\alpha_i = \frac{p_{0i} e_i}{\mu_i}$ for $i = 1, 2, \cdots mq$ and $\alpha_0 = \frac{e_0}{\mu_0}$ and e_i is the relative thread arrival rate at q_i. In this system, $e_0 = \sum_{i=1:mq} e_i$. $\beta(x)$ is define as follows:

$$\beta(x) = \begin{cases} 1 & M/M/1 \quad \& \quad M/M/1\, PS \\ x! & M/M/\infty \\ x! & M/M/m \quad FCFS \quad x \leq m \\ m!m^{(x-m)} & M/M/m \quad FCFS \quad x > m \end{cases} \tag{5}$$

Substituting Eq.(4) into Eq. (1), we have,

$$P_I = \frac{\sum_{k_1+\cdots+k_{mq}=M_t} \prod_{i=1}^{mq} \frac{a_i^{k_i}}{\beta_i(k_i)}}{\sum_{n=0}^{M_t} \frac{1}{\beta_0(M_t-n)} \sum_{k_1+\cdots+k_{mq}=n} \prod_{i=1}^{mq} \frac{a_i^{k_i}}{\beta_i(k_i)}} \tag{6}$$

where $a_i = \frac{p_{0i}\mu_0}{\mu_i}$.

We now have the following general result:

Theorem : $\lim_{M_t \to \infty} P_I = 0$ if and only if $\frac{m_0 a_i}{m_i} < 1, \forall i = 1, 2, \cdots, mq$.

The theorem simply states that when $\frac{m_0 a_i}{m_i} > 1$, adding threads cannot completely hide the ith resource access latency from the CPU, i.e., resource i presents a performance bottleneck. This condition is surprisingly simple, while being very general, applicable to any processor architectures that can be modeled by the class of processor models defined by the queuing network in Fig. 2 and the design space in Fig. 1.

To illustrate the power of the above result, we give a special case here. Consider the case when $mq = 1$, i.e., the model in Fig. 2 only has two queuing servers, a CPU server and a resource server. We have,

As M_t goes to infinity,

$$\lim_{M_t \to \infty} P_I = \begin{cases} \frac{1}{\frac{m_0^{m_0}}{m_0!} \frac{A^{-m_0+1}}{A-1} + \sum_{k'=0}^{m_0-1} \left(\frac{a_1}{m_1}\right)^{-k'} / k'!} & A > 1 \\ 0 & A < 1 \end{cases} \tag{7}$$

This gives the general condition under which the resource becomes a bottleneck when $A > 1$, where $A = \frac{a_1 m_0}{m_1} = \frac{\mu_0 m_0}{\mu_1 m_1}$, agreeing with the general result in Theorem. This condition tells us that if the average service rate times the level of parallelism (i.e., m_1) at the resource is slower than the service rate times the level of parallelism (i.e., m_0) at the CPU server, the resource becomes a bottleneck that throttles the overall throughput.

Finally, it is interesting to note that when $m_0 = m_1 = 1$ (i.e., the CPU is coarse-grained and resource access mechanism is FCFS), we have,

$$\lim_{M_t \to \infty} P_I = \begin{cases} \frac{a_1 - 1}{a_1} & a_1 > 1 \\ 0 & a_1 < 1 \end{cases} \tag{8}$$

A deterministic version of this result was derived in [3] and later on a result identical to the one in Eq. (8) was derived and studied in [22]. Clearly, the result given in Theorem is far more general than the results given in [3] and [22].

So far, we have assumed that there is no cache effect, i.e., $P_{ih}(S_i) = 0$, for $i = 1, 2, \cdots, mq$. Since

S_i is the cache resource allocated to accommodate the cached data from resource i, we must have $S \geq \sum_{i=1:mq} S_i$, where S is the total cache size. Now, we take into account of the caching effects for the model in Fig. 2, i.e., $P_{ih}(S_i) \geq 0$. To simplify the discussion, we assume that all the resources are memory resources, so that caching can help reduce the resource access latencies for all the resources.

Assuming there is no correlation among consecutive cache hits, our cache model only amounts to the change of p_{0i} to $(1 - P_{ih}(S_i))p_{0i}$ and consequently, a_i changes to $(1 - P_{ih}(S_i))a_i$. The product-form property of the model is preserved. Hence, we have the following corollary in parallel to Theorem,

Corollary : $\lim_{M_t \to \infty} P_I = 0$ if and only if $\frac{m_0(1-P_{ih}(S_i))a_i}{m_i} \leq 1, \forall i = 1, 2, \cdots, mq$.

5 Proof of Theorem

This section provides a detailed proof of the theorem given in the previous section.

Proof of Theorem: The theorem can be decomposed into two parts:

(1) if $\frac{m_0 a_i}{m_i} \leq 1, \forall i = 1, 2, \cdots, m_q$, then $\lim_{M_t \to \infty} P_I = 0$.

(2) if there is at least one term in $\{m_0 a_i / m_i\}_{i=1:m_q}$ larger than 1, then $\lim_{M_t \to \infty} P_I > 0$.

In what follows, we prove these two parts separately.

First, we prove the first part. Without loss of generality, assume $\frac{a_{m_q}}{m_{m_q}} = \max\left\{\frac{a_i}{m_i}\right\}$ and $\frac{a_1}{m_1} = \min\left\{\frac{a_i}{m_i}\right\}$, for $i \in [1, m_q]$. Dividing both the numerator and denominator of Eq. (6) by $\left(\frac{a_{m_q}}{m_{m_q}}\right)^{M_t}$, we have,

$$P_I = \frac{\sum_{k_1+\cdots+k_{m_q}=M_t} \prod_{i=1}^{m_q} \left(\frac{a_i m_{m_q}}{m_i a_{m_q}}\right)^{k_i} \frac{m_i^{k_i}}{\beta_i(k_i)}}{\sum_{n=0}^{M_t} \frac{\left(\frac{a_{m_q}}{m_{m_q}}\right)^{n-M_t}}{\beta_0(M_t-n)} \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \left(\frac{a_i m_{m_q}}{m_i a_{m_q}}\right)^{k_i} \frac{m_i^{k_i}}{\beta_i(k_i)}} \tag{9}$$

Since $\frac{a_1}{m_1} \leq \frac{a_i}{m_i} \leq \frac{a_{m_q}}{m_{m_q}}$, for $i = 2, \cdots, m_q - 1$, we have,

$$\left(\frac{a_1 m_{m_q}}{m_1 a_{m_q}}\right) \leq \left(\frac{a_i m_{m_q}}{m_i a_{m_q}}\right) \leq \left(\frac{a_{m_q} m_{m_q}}{m_{m_q} a_{m_q}}\right) = 1 \tag{10}$$

Hence,

$$P_I \le \frac{\sum_{k_1+\cdots+k_{m_q}=M_t} \prod_{i=1}^{m_q} \frac{m_i^{k_i}}{\beta_i(k_i)}}{\sum_{n=0}^{M_t} \frac{\left(\frac{a_{m_q}}{m_{m_q}}\right)^{n-M_T}}{\beta_0(M_t-n)} \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \left(\frac{a_1 m_{m_q}}{m_1 a_{m_q}}\right)^{k_i} \frac{m_i^{k_i}}{\beta_i(k_i)}} \tag{11}$$

The queuing server q_i can be generally viewed as an M/M/m_i queue, $i \in (0, \cdots, m_q)$. Since for $\forall k_i$, $\beta_i(k_i) \ge m_i! m_i^{k_i - m_i}$ and $\beta_i(k) < m_i! m_i^{k_i}$ and noticing that there are $\frac{(M_t+m_q-1)!}{M_t!(m_q-1)!}$ elements in $\sum_{k_1+\cdots+k_{m_q}}$, we have:

$$\begin{aligned}
P_I &< \frac{\sum_{k_1+\cdots+k_{m_q}=M_t} \prod_{i=1}^{m_q} \frac{m_i^{m_i}}{m_i!}}{\sum_{n=0}^{M_t} \frac{\left(\frac{a_{m_q}}{m_{m_q}}\right)^{n-M_T}}{\beta_0(M_t-n)} \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \left(\frac{a_1 m_{m_q}}{m_1 a_{m_q}}\right)^{k_i} \frac{1}{m_i!}} \\
&= \frac{\frac{(M_t+m_q-1)!}{M_t!(m_q-1)!} \prod_{i=1}^{m_q} \frac{m_i^{m_i}}{m_i!}}{\sum_{n=0}^{M_t} \left[\frac{\left(\frac{m_{m_q}}{a_{m_q}}\right)^{M_T-n}}{\beta_0(M_t-n)} \cdot \frac{(n+m_q-1)!}{n!(m_q-1)!} \cdot \left(\frac{a_1 m_{m_q}}{m_1 a_{m_q}}\right)^n \cdot \prod_{i=1}^{m_q} \frac{1}{m_i!} \right]} \\
&< \frac{1}{\left(\frac{M_{m_q}}{a_{m_q}}\right)^{M_t} \cdot \sum_{n=0}^{M_t} \frac{1}{\beta_0(M_t-n)} \left(\frac{a_1}{m_1}\right)^n \cdot \prod_{i=1}^{m_q} {m_i}^{m_i}} \\
&= \frac{1}{\left(\frac{M_{m_q}}{m_0 a_{m_q}}\right)^{M_t} \prod_{i=1}^{m_q} {m_i}^{m_i} \sum_{n=0}^{M_t} \frac{m_0^{M_t-n}}{\beta_0(M_t-n)} \left(\frac{a_1 m_0}{m_1}\right)^n} \\
&\le \frac{1}{\left(\frac{M_{m_q}}{m_0 a_{m_q}}\right)^{M_t} \prod_{i=1}^{m_q} {m_i}^{m_i} \sum_{n=0}^{M_t} \frac{m_0^{M_t-n}}{m_0! m_0^{M_t-n}} \left(\frac{a_1 m_0}{m_1}\right)^n} \\
&= \frac{1}{\left(\frac{1}{m_0!} \prod_{i=1}^{m_q} {m_i}^{m_i}\right) \left(\frac{M_{m_q}}{m_0 a_{m_q}}\right)^{M_t} \cdot \frac{1-\left(\frac{a_1 m_0}{m_1}\right)^{M_t+1}}{1-\frac{a_1 m_0}{m_1}}}
\end{aligned} \tag{12}$$

Denote the last expression in Eq. (12) as $= P_I'$. Since $\frac{a_1 m_0}{m_1} \le \frac{a_{m_q} m_0}{m_{m_q}} \le 1$, $\lim_{M_t \to \infty} P_I' = 0$. From Eq. (12), we have $P_I' > P_I$. Note that $P_I \ge 0$. Hence $\lim_{M_t \to \infty} P_I = 0$.

Now we prove the second part. To facilitate the proof, the dependency of P_I on m_q is explicitly included in P_I as a superscript, i.e., $P_I^{(m_q)}$. Futhermore, since there is at least one term in $\{m_0 a_i / m_i\}_{i=1:m_q}$ larger than 1, we assume $\frac{m_0 a_1}{m_1} > 1$. For $m_q = 1$ (i.e., there is only one resource), from Eq. (6), we have

$$P_I^{(1)} = \frac{\frac{a_1^{M_t}}{\beta_1(M_t)}}{\sum_{k_1=0}^{M_t} \frac{a_1^{k_1}}{\beta_0(M_t-k_1)\beta_1(k_1)}} \tag{13}$$

According to the assumption in part (2), $\frac{m_0 a_1}{m_1} > 1$; from Eq. (7) we have,

$$\lim_{M_t \to \infty} P_I^{(1)} = \frac{1}{\frac{m_0^{m_0}}{m_0!} \cdot \frac{\left(\frac{a_1 m_0}{m_1}\right)^{-m_0+1}}{\frac{a_1 m_0}{m_1} - 1} + \sum_{k_1=0}^{m_0-1} \frac{\left(\frac{a_1}{m_1}\right)^{-k_1}}{k_1!}} > 0 \tag{14}$$

This means that for the single resource case, the second part of the theorem holds true. To prove the theorem holds true in general, we need to show that it holds true for $\forall m_q$. Now if $P_I^{(m_q+1)} \geq P_I^{(m_q)}$ for $\forall m_q$, the second part of the theorem will hold true for $\forall m_q$. In the following, we show this is indeed the case.

For $\forall m_q$, Let

$$P_I^{(m_q)} = \frac{\sum_{k_1+\cdots+k_{m_q}=M_t} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}}{\sum_{n=0}^{M_t} \frac{1}{\beta_0(M_t-n)} \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}} \tag{15}$$

Notice that Eq. (15) is the same as Eq. (6). For $m_q + 1$, we have,

$$P_I^{(m_q+1)} = \frac{\sum_{k_1+\cdots+k_{m_q}+k_{m_q+1}=M_t} \prod_{i=1}^{m_q+1} \frac{a_i^{k_i}}{\beta_i(k_i)}}{\sum_{n=0}^{M_t} \frac{1}{\beta_0(M_t-n)} \sum_{k_1+\cdots+k_{m_q}+k_{m_q+1}=n} \prod_{i=1}^{m_q+1} \frac{a_i^{k_i}}{\beta_i(k_i)}} \tag{16}$$

or

$$P_I^{(m_q+1)} = \frac{\sum_{k_1+\cdots+k_{m_q}=M_t} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} + \sum_{k_{m_q+1}=1}^{M_t} \frac{a_{m_q+1}^{k_{m_q+1}}}{\beta_{m_q+1}(k_{m_q+1})} \sum_{k_1+\cdots+k_{m_q}=M_t-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}}{\sum_{n=0}^{M_t} \frac{1}{\beta_0(M_t-n)} \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} + \sum_{n=1}^{M_t} \frac{1}{\beta_0(M_t-n)} \sum_{k_{m_q+1}=1}^{n} \frac{a_{m_q+1}^{k_{m_q+1}}}{\beta_{m_q+1}(k_{m_q+1})} \sum_{k_1+\cdots+k_{m_q}=n-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}} \tag{17}$$

Eq. (17) is written in such a form that the first terms in both the numerator and the denominator are the same as the numerator and denominator in Eq. (16), respectively. Now, let L and R be the second term in the denominator multiplied by the first term in the numerator, and the first term in the denominator multiplied by the second term in the numerator, respectively, as given below:

$$L = \{\sum_{n=1}^{M_t} \frac{1}{\beta_0(M_t-n)} \cdot \sum_{k_{m_q+1}=1}^{n} \frac{a_{m_q+1}^{k_{m_q+1}}}{\beta_{m_q+1}(k_{m_q+1})} \cdot \sum_{k_1+\cdots+k_{m_q}=n-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}\} \cdot \{\sum_{k_1+\cdots+k_{m_q}=M_t} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}\} \tag{18}$$

$$R = \{\sum_{n=0}^{M_t} \frac{1}{\beta_0(M_t - n)} \cdot \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}\} \cdot \{\sum_{k_{m_q+1}=1}^{M_t} \frac{a_{m_q+1}^{k_{m_q+1}}}{\beta_{m_q+1}(k_{m_q+1})} \cdot \sum_{k_1+\cdots+k_{m_q}=M_t-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)}\} \tag{19}$$

Obviously, $P_I^{(m_q+1)} \geq P_I^{(m_q)}$ if and only if $R \geq L$. To show $R \geq L$, we construct another quantity L' as follows:

$$L' = \sum_{n=1}^{M_t} \frac{1}{\beta_0(M_t - n)} \cdot \sum_{k_{m_q+1}=1}^{n} \frac{a_{m_q+1}^{k_{m_q+1}}}{\beta_{m_q+1}(k_{m_q+1})} \cdot \sum_{k_1+\cdots+k_{m_q}=M_t-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \cdot \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \tag{20}$$

To prove $R \geq L$, we first show that $L' \geq L$ and then $R \geq L'$.

First, we note that the first two sums in Eq. (18) are the same as the first two sums in Eq. (20), except in Eq. (20), there is an extra term at $n = 0$. Clearly, if we could show that for any given $n > 0$, the last two sums in Eq. (20) is no less than the last two sums in Eq. (18), we have $L' \geq L$. In other words, we want to show $E' \geq E$, where,

$$E = \sum_{k_1+\cdots+k_{m_q}=n-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \cdot \sum_{k_1+\cdots+k_{m_q}=M_t} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \tag{21}$$

and

$$E' = \sum_{k_1+\cdots+k_{m_q}=M_t-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \cdot \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \tag{22}$$

According to **Corrolary A** given in Appendix, both E and E' have the same Z value, i.e., $Z = M_t + n - k_{m_q+1}$. In E, let A be the smaller one of M_t and $n - k_{m_q+1}$ and in E', A' be the smaller one of n and $M_t - k_{m_q+1}$. Note that if $A' \geq A$ then $E' \geq E$, since, according to **Corrolary A**, E and E' are monotonously increasing function when $A', A \in [1, \lfloor \frac{Z}{2} \rfloor]$. As a result, to prove $L' \geq L$, we need to show that $A' \geq A$.

Since $M_t \geq n \geq k_{m_q+1} \geq 1$, $M_t > n - k_{m_q+1}$, $A = n - k_{m_q+1}$. In E', since both n and $M_t - k_{m_q+1}$ can be smaller than $\lfloor \frac{Z}{2} \rfloor$, we have,

$$A' - A = \begin{cases} M_t - n & n > \lfloor \frac{Z}{2} \rfloor \\ k_{m_q+1} & n \leq \lfloor \frac{Z}{2} \rfloor \end{cases}$$

Again, since $M_t \geq n \geq k_{m_q+1} \geq 1$, $A' - A \geq 0$ always holds. So we have $E' \geq E$, and therefore, $L' \geq L$.

Finally, we show that $R \geq L'$. We first rewrite Eq. (20) as follows:

$$L' = \sum_{n=1}^{M_t} \frac{1}{\beta_0(M_t - n)} \cdot \sum_{k_1+\cdots+k_{m_q}=n} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \cdot \sum_{k_{m_q+1}=1}^{n} \frac{a_{m_q+1}^{k_{m_q+1}}}{\beta_{m_q+1}(k_{m_q+1})} \cdot \sum_{k_1+\cdots+k_{m_q}=M_t-k_{m_q+1}} \prod_{i=1}^{m_q} \frac{a_i^{k_i}}{\beta_i(k_i)} \tag{23}$$

We note that at any given n, the part with the last two sums in Eq. (19) is no less than the part with the last two sums in Eq. (23), because $M_t \geq n$ for $\forall n$. Furthermore, the parts involving the first two sums in Eq. (19) and Eq. (23) are the same, except that the part in Eq. (19) has an extra term at $n = 0$. Hence, we have $R \geq L'$. Since we have shown that $L' \geq L$, $R \geq L$ and therefore $P_I^{(m_q+1)} \geq P_I^{(m_q)} \geq P_I^{(1)} > 0$.

6 Thread and Cache Resource Provisioning

It is clear that cache is needed in addition to multithreading to remove the bottleneck resource i if $\frac{m_0 a_i}{m_i} > 1$, according to Theorem. Now according to Corollary, the minimum amount of cache resource S_i that is needed to remove the bottleneck resource i must satisfy the following equation:

$$\frac{m_0(1 - P_{ih}(s_i))a_i}{m_i} = 1 \tag{24}$$

from Eq. (24) we have,

$$S_i = P_{ih}^{-1}(1 - \frac{m_i}{m_0 a_i}) \tag{25}$$

where P_{ih}^{-1} is the inverse function of P_{ih}. Clearly, $S_i = 0$ if resource i is not a bottleneck resource. The condition that $S \geq \sum_{i=1:mq} S_i$ gives us a good idea as to how much total cache resource is needed

to maximize the throughput performance. If S is a given, this condition determines whether maximum throughput performance can be achieved or not, with maximal thread and cache resource provisioned.

In summary, we have the following algorithm for effective thread and cache resource provisioning:

- if $\frac{m_0 a_i}{m_i} < 1$, for all $i = 1, 2, \cdots, mq$, the maximum throughput performance can be achieved by adding sufficient number of threads and cache is not needed
- else if (without loss of generality) $\frac{m_0 a_1}{m_1} \leq \frac{m_0 a_2}{m_2} \leq \cdots \leq \frac{m_0 a_{k-1}}{m_{k-1}} \leq 1 < \frac{m_0 a_k}{m_k} \leq \cdots \leq \frac{m_0 a_{mq}}{m_{mq}}$, calculate S_i for $i = k, \cdots, mq$ from Eq. (25). If $S \geq \sum_{i=k:mq} S_i$, output S_i for cache resource provisioning; else output S_i and request for additional $(\sum_{i=k:mq} S_i - S)$ cache memory.

To make the above discussion generally applicable to any detailed caching models, so far we have not mentioned what $P_{ih}(S_i)$ should look like. In practice, $P_{ih}(S_i)$ is a complicated function of not only S_i, but also data request patterns, thread scheduling discipline, cache replacement algorithm, etc. A widely adopted analytical model is: $P_{ih}(S_i) = 1 - \left(\frac{S_i}{\delta} + 1\right)^{-(\epsilon-1)}$, as discussed in [23]. Since how to model the cache hit probability is not the focus of this paper, we shall not discuss this issue further in this paper.

7 Testing of General Conditions

The general conditions given in this paper are derived based on the queuing network models with closed-form solutions. This raises the concern whether these general conditions are accurate enough when applied to real multithreaded processors.

To address the above concern, we first note that when modeling the multithreaded processors using queuing network models, there are two major areas that may introduce inaccuracies. The first area is the modeling of various types of resources, e.g., SMT, coarse-grained CPU, and pipelined memory accesses. The second area is the modeling of the stochastic nature of the workload, e.g., exponentially distributed CPU service time and unloaded memory access latency. The first area is generally concerned with the lack

of modeling of micro-architectural and the instruction-level details. Our study in [24], however, indicated that this may introduce $5 - 15\%$ inaccuracies to the performance data. This should be tolerable given that these general conditions are generally used in the initial programming phase for resource provisioning. In general, program fine tuning may be done in later phases to further optimize the performance. On the other hand, the second area needs more careful justification. The exponential service time assumption made in M/M/1 and M/M/m models are far from accurate enough to characterize the stochastic nature of the program execution. Hence, in this section, we focus on testing the accuracy of the general conditions by removing the assumptions made in the second area.

To test the accuracy of the general conditions in rather extreme conditions, we consider service time distributions with long tails for both CPU and memory components. More specifically, the Pareto distributions are used to characterize the service times. Pareto distributions account for a wide range of code segment sizes, or equivalently, the thread service times at the CPU, and large variations of memory access latencies. The aim is to test whether such significant deviations from the exponential distributions would (a) shift the appearance of a bottleneck resource away from the point in the parameter space identified by the general conditions; and (b) significantly blur the boundaries between the bottleneck and non-bottleneck regions. We use the simulation results of the original queuing network models as benchmarks for the testing.

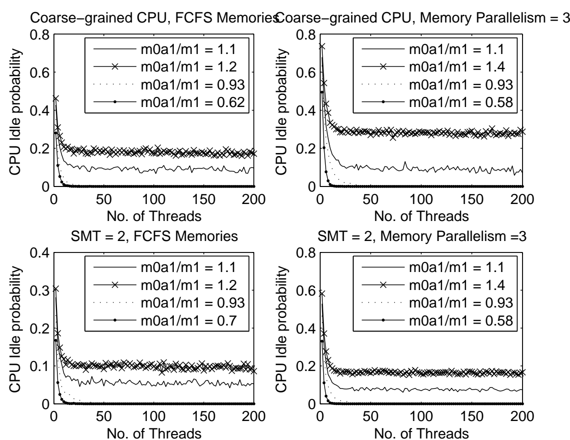

Figure 4: Exponential Distribution

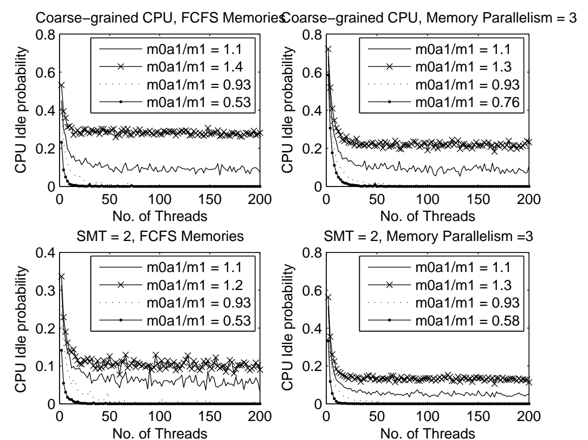

Figure 5: Pareto Distribution

We consider the processor model with four memory resources. We run simulation for both the original queuing network models (whose service times are Exponential) and the queuing network models with Pareto service times. The results in term of CPU idle probability versus the number of threads are presented in Fig. 4 and Fig. 5. For each of these two cases, four scenarios are studied: (a) coarse-grained CPU and FCFS memories; (b) coarse-grained CPU and memories with pipelined accesses; (c) SMT and FCFS memories; and (d) SMT and memories with pipelined accesses. The result for the four scenarios are presented in the four subplots in Fig. 4 and Fig. 5. In each subplot, four curves are given. Of which two correspond to the cases where one bottleneck resource is identified according to the general conditions ($\frac{m_0 a_1}{m_1} > 1$), whereas the other two do not involve bottleneck resource according to the general conditions.

As one can see, for both Fig. 4 and Fig. 5, there is a clean division between the two sets of curves for all the subplots. Namely, as the number of threads increases, the two curves corresponding to the cases without bottleneck resource identified converge to zero, whereas the other two level out at some nonzero values.

The above results clearly indicate that the general conditions derived in this paper are insensitive to the actual service time distributions of the processor components, even though they are obtained based on the exponential service time distributions. As a result, the general conditions may be used as power means to help quickly identify the bottleneck resources by performing a simple statistic estimation of a few parameters.

8 Conclusions and Future Work

In this paper, the fundamental conditions for multithreaded processor bottleneck resource identification are derived for a class of processor models based on queuing network techniques. These conditions are general and applicable to a large design space. Based on these conditions, we arrive at a generic algorithm for thread and cache resource provisioning. This algorithm can serve as guidelines for the design of practically useful algorithms for thread and cache resource provisioning.

The results presented in this paper can be generalized to the case where there are multiple cores in the system, which is under way and will be reported elsewhere. Here we provide some intuitions how the results presented in this paper may be generalized to the multicore case. For multicore processors, we are concerned with the potential bottleneck resources shared by multiple cores. With a class of multicore processor models, quite similar to the one presented in this paper, one can show that the aggregated thread arrival process at a shared resource can be derived from the thread arrival processes coming from individual cores. This will lead to general conditions that apply to virtually unlimited number of cores, i.e., many-core processors.

References

[1] A. Agarwal, Performance Tradeoffs in Multithreaded Processors, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 5, pp. 525-539, Sept. 1992.

[2] G. Bolch, S. Greiner, H. Meer, K. S. Trivedi, *Queuing Networks and Markov Chains, 2nd Edition*, A John Wiley & SONS, Inc., 2006.

[3] H. Che, C. Kumar, and B. Menasinahal, Fundamental Network Processor Performance Bounds, in *Proc. of NCA 2005*, Aug. 2005.

[4] X. E. Chen and T. M. Aamodt, A First-Order Fine-Grained Multithreaded Throughput Model, in *Proc. of HPCA 2009*, Feb. 2009.

[5] M. Curtis-Maury, X. Ding, C.D. Antopnopoulos, and D. S. Nikolopoulos, An Evaluation of OpenMP on Current and Emerging Multithreaded/Multicore Processors, in *Proc. of IWOMP 2005*, Jun. 2005.

[6] C. Dubach, T. M. Jones, and M. F. P. O Boyle, Microarchitectural Design Space Exploration Using an Architecture-Centric Approach, in *Proc. of Micro 2007*, Dec. 2007.

[7] R. E. Grant and A. Afsahi, A Comprehensive Analysis of OpenMP Applications on Dual-Core Intel Xeon SMPs, in *Proc. of IPDPS 2007*, Mar. 2007

[8] R. E. Grant and A. Afsahi, Characterization of Multithreaded Scientific Workloads on Simultaneous Multithreading Intel Processors,in *Proc. of IOSCA 2005*, Aug. 2005.

[9] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser, Many-Core vs. Many-Tthread Machines: Stay Away From the Valley, *IEEE Computer Architecture Letter*, vol. 8, no. 1, pp. 25-28, Jan. 2009.

[10] E. Ipek, S. A. McKee, K. Singh, and R. Caruana, Efficient Architectural Design Space Exploration via Predictive Modeling, *ACM Transactions on Architecture and Code Optimization*, Vol. 4, No. 4, Article No. 1, Jan. 2008.

[11] B. C. Lee and D. M. Brooks, Illustrative Design Space Studies with Microarchitectural Regression Models, in *Proc. of HPCA 2007*, Feb. 2007.

[12] C. Liao, Z. Liu, L. Huang, and B. Chapman, Evaluating OpenMP on Chip Multithreading Platforms, in *Proc. of IWOMP 2005*, Jun. 2005.

[13] S. S. Nemawarkar, R. Govindarajan, G. R. Gao, and V. K. Agarwal, Analysis of Multithreaded Multiprocessors with Distributed Shared Memory, in *Proc. of the 5th IEEE Symposium on Parallel and Distributed Processing*, Dec. 1993.

[14] L. Peng, J. Peri, T. K. Prakaksh, C. Staelin, Y. K. Chen, and D. Koppelman, Memory Hierarchy Performance Measurement of Commerical Dual-Core Desktop Processorrs, *Journal of Systems Architecture: the EUROMICRO Journal*, Vol. 54, No. 8, pp. 816-828, Aug. 2008.

[15] R. Saavedra-Barrera, D. Culler, and T. von Eicken, Analysis of Multithreaded Architectures for Parallel Computing, in *Proc. of SPAA 1990*, pp. 169-178, 1990.

[16] S. Subramaniam, M. Prvulovic, and G. H. Loh, PEEP: Exploiting Predictability of Memory Dependences in SMT Processors, in *Proc. of HPCA 2008*, Feb. 2008.

[17] M. Woodbury and K. G. Shin, Performance Modeling and Measurement of Real-Time Multiprocessors with Time-Shared Buses, *IEEE Transactions on Computers*, Vol. 37, No. 2, pp.214-224, Feb. 1988.

[18] Y. Li, B. Lee, D. Brooks, Z. Hu, K. Skadron, CMP Design Space Exploration Subject to Physical Constraints, in *Proc. of HPCA 2006*, pp. 17-28, Feb. 2006

[19] A. Navarro, R. Asenjo, S. Tabik, and C. Cascaval, Analytical Modeling of Pipeline Parallelism, in *Proc. of PACT 2009*, pp.281-290, Sept. 2009

[20] G. Amdahl, Validity of the Single-processor Approach to Achieving Large Scale Computing Capabilities, *AFIPS Conference Proceedings*, Vol. 30, pp. 483-485, 1967.

[21] M. Hill and M. Marty, Amdahls Law in the Multicore Era, *Computer*, vol. 41, no. 7, pp. 33-38, July. 2008

[22] Y. N. Lin, Y. D. Lin, Y. C. Lai, Thread Allocation in Chip Multiprocessor Based Multithreaded Network Processors, in *Proc. of AINA 2008*, pp.718-725, Mar. 2008.

[23] C. K. Chow. Determination of Cache's Capacity and its Matching Storage Hierarchy, *IEEE Transactions on Computers*, c-25, 157 - 164, 1976. 811 - 825, 1992.

[24] H. Jung, M. Ju, H. Che, Z. Wang, A Fast Performance Analysis Tool for Multicore, Multithreaded communication Processors, in *Proc. of HASE 2008*, Nov. 2008.

A Appendix

Theorem A: There are N boxes ($N \geq 2$). The capacity of the ith box ($i = 1, 2, \cdots, N$) is k_i, and $\sum_{i=1}^{N} k_i = M$. $S_N(y)$ is the number of different ways to put y identical balls into these N boxes. Then $S_N(y)$ is a monotonously increasing function of y, when $y \in [1, \lfloor \frac{M}{2} \rfloor]$, and $S_N(y)$ reaches its maximal value when $y = \lfloor \frac{M}{2} \rfloor$.

Proof: We prove it by induction.

Basis step ($N = 2$): we define a step function $u(x)$ as follows:

$$u(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{26}$$

$S_2(y)$ can be calculated as follows: first, assume that there is no capacity constraints for both boxes (i.e. $k_1 = \infty, k_2 = \infty$). Then there are $\frac{(y+1)!}{y! \cdot 1!}$ different ways to put y balls into these two boxes. However, since the size of the boxes is not infinity, we can only put at most k_1 balls in box 1, so the number of ways by which we can put more than k_1 balls in the 1st box must be excluded, which is $\sum_{j_1=1}^{y} u(j_1 - k_1)$. Similarly, for the 2nd box, there are $\sum_{j_2=1}^{y} u(j_2 - k_2)$ number of different ways needs to be excluded. Therefore, for $y, y+1 \leq \lfloor \frac{M}{2} \rfloor$, we have,

$$S_2(y) = \frac{(y+1)!}{y! \cdot 1!} - \sum_{j_1=1}^{y} u(j_1 - k_1) - \sum_{j_2=1}^{y} u(j_2 - k_2) \tag{27}$$

$$\begin{aligned} S_2(y+1) &= \frac{(y+2)!}{(y+1)! \cdot 1!} - \sum_{j_1=1}^{y+1} u(j_1 - k_1) - \sum_{j_2=1}^{y+1} u(j_2 - k_2) \\ &= \frac{(y+2)!}{(y+1)!} - \sum_{j_1=1}^{y} u(j_1 - k_1) - \sum_{j_2=1}^{y} u(j_2 - k_2) - u(y+1-k_1) - u(y+1-k_2) \end{aligned} \tag{28}$$

We further have,

$$\begin{aligned} S_2(y+1) - S_2(y) &= \frac{1}{y+1} \cdot \frac{(y+1)!}{y!} - [u(y+1-k_1) + u(y+1-k_2)] \\ &= 1 - [u(y+1-k_1) + u(y+1-k_2)] \end{aligned} \tag{29}$$

We want to show $S_2(y+1) - S_2(y) \geq 0$ or equivalently, $[u(y+1-k_1) + u(y+1-k_2)] \leq 1$.

For $u(y+1-k_1)$

If $(y+1-k_1) \leq 0$ then $u(y+1-k_1) = 0$, $[u(y+1-k_1) + u(y+1-k_2)] = u(y+1-k_2) \leq 1$, and

$S_2(y+1) - S_2(y) \geq 0$.

else if $(y+1-k_1) > 0$ then $u(y+1-k_1) = 1$ and $y+1 > k_1$

$u(y+1-k_2) = u(y+1-M+k_1)$, because $k_2 = M - k_1$.

$\because (y+1-k_1) > 0$,

$\therefore y+1 > k_1$. $\because y+1 \leq \lfloor \frac{M}{2} \rfloor$,

$\therefore k_1 < y+1 \leq \lfloor \frac{M}{2} \rfloor$

$\therefore (y+1-M+k_1) < 0$, $\therefore u(y+1-k_2) = u(y+1-M+k_1) = 0$ and $[u(y+1-k_1) + u(y+1-k_2)] \leq 1$.

$\therefore S_2(y+1) - S_2(y) \geq 0$, i.e., the theorem holds true for $N = 2$.

Induction hypothesis ($N \geq 2$): $S_N(y+1) \geq S_N(y)$, for $y, y+1 \leq \lfloor \frac{M}{2} \rfloor$

Induction step: now consider $N+1$ boxes and y balls ($y, y+1 \leq \lfloor \frac{M}{2} \rfloor$). Here we consider the first N boxes as one group and the $(N+1)$th box as the other group. Consider putting j balls into the $(N+1)$th box, and the rest $(y-j)$ balls into the first N boxes. The different ways to put $(y-j)$ balls into the first N boxes is given by $S_N(y-j)$. We may put j ($j \in [0, k_{N+1}]$) balls into the $(N+1)$th box. We have,

$$S_{N+1}(y) = \sum_{j=0}^{y} u(k_{N+1} - j) S_N(y-j) \tag{30}$$

Similarly, we have,

$$S_{N+1}(y+1) = \sum_{j=0}^{y+1} u(k_{N+1} - j) S_N(y+1-j) \tag{31}$$

Every term in Eq. (31), i.e., $u(k_{N+1} - j)S_N(y+1-j)$, is no less than the term in Eq. (30), since $S_N(y+1-j) \geq S_N(y-j)$ (induction hypothesis). Furthermore, Eq. (31) has one more term $u(k_{N+1} - (y+1))S_N(y+1-(y+1))$, which is non-negtive. Therefore, $S_{N+1}(y+1) \geq S_{N+1}(y)$, for $y, y+1 \leq \lfloor \frac{M}{2} \rfloor$

Corollary A For polynomial function $F = \sum_{k_1+\cdots+k_n=A} \prod_{i=1}^{n} a_i^{k_i^1} \sum_{k_1+\cdots+k_n=B} \prod_{i=1}^{n} a_i^{k_i^2}$, and $A + B = Z$, Z is a constant. Assume $A \leq B$. Then F reaches its maximal value when $A = \lfloor \frac{Z}{2} \rfloor$, and F is a monotonously increasing function of A, for $A \in [1, \lfloor \frac{Z}{2} \rfloor]$

Proof: We define:

$$\begin{aligned} F &= \sum_{k_1+\cdots+k_n=A} \prod_{i=1}^{n} a_i^{k_i^1} \sum_{k_1+\cdots+k_n=B} \prod_{i=1}^{n} a_i^{k_i^2} \\ &= \sum_{k_1^0+\cdots+k_n^0=Z} C(k_1^0, \cdots, k_n^0) \cdot \prod_{i=1}^{n} a_i^{k_i^0} \end{aligned} \tag{32}$$

We have $A + B = Z$, and $k_i^1 + k_i^2 = k_i^0$. $C(k_1^0, \cdots, k_n^0)$ (denoted by C for convenience) is the coefficient for each term in Eq. (32). For given A and B, the value of F is determined by C. The question of how to calculate C can be mapped to a combinatorial problem below:

Suppose we have n boxes. The capacity of the ith box is k_i^0 ($\sum_{i=1}^{n} k_i^0 = Z$). It turns out that C is the number of ways to put A identical balls into these boxes. This combinatorial problem is addressed in **Theorem A**. Since $A + B = Z$, and Z is a constant, we just have one variable. For convenience, we assume that A is the smaller than B. According to **Theorem A**, C reaches its maximal value when $A = \lfloor \frac{Z}{2} \rfloor$, and C is a monotonously function of A, for $A \in [1, \lfloor \frac{Z}{2} \rfloor]$. Therefore, F reaches its maximal value when $A = \lfloor \frac{Z}{2} \rfloor$, and F is a monotonously function of A, for $A \in [1, \lfloor \frac{Z}{2} \rfloor]$.