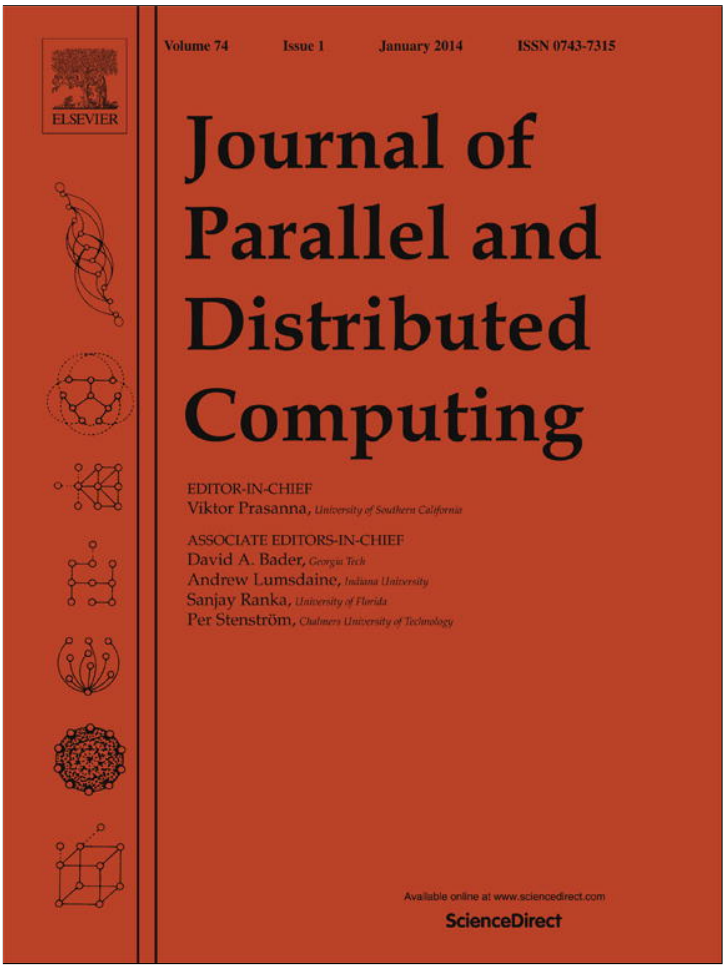

J. Parallel Distrib. Comput. 74 (2014) 1872–1890

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# A queueing theoretic approach for performance evaluation of low-power multi-core embedded systems

Arslan Munir [a,*], Ann Gordon-Ross [b,c], Sanjay Ranka [d], Farinaz Koushanfar [a]

[a] Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA
[b] Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA
[c] NSF Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida, USA
[d] Department of Computer and Information Science and Engineering at the University of Florida, Gainesville, FL, USA

## HIGHLIGHTS

- Queueing theory-based modeling technique for evaluating multi-core architectures.
- Enables quick and inexpensive architectural evaluation.
- Architectural evaluation for workloads with any computing requirements.
- Can be used for performance per watt & performance per unit area characterizations.
- Provides insights about shared last-level caches (LLCs) orchestration.

## ARTICLE INFO

*Article history:*
Received 18 June 2012
Received in revised form
6 July 2013
Accepted 26 July 2013
Available online 11 August 2013

*Keywords:*
Multi-core
Low-power
Embedded systems
Queueing theory
Performance evaluation

## ABSTRACT

With Moore's law supplying billions of transistors on-chip, embedded systems are undergoing a transition from single-core to multi-core to exploit this high transistor density for high performance. However, the optimal layout of these multiple cores along with the memory subsystem (caches and main memory) to satisfy power, area, and stringent real-time constraints is a challenging design endeavor. The short *time-to-market* constraint of embedded systems exacerbates this design challenge and necessitates the architectural modeling of embedded systems to reduce the time-to-market by expediting target applications to device/architecture mapping. In this paper, we present a queueing theoretic approach for modeling multi-core embedded systems that provides a quick and inexpensive performance evaluation both in terms of time and resources as compared to the development of multi-core simulators and running benchmarks on these simulators. We verify our queueing theoretic modeling approach by running SPLASH-2 benchmarks on the SuperESCalar simulator (SESC). Results reveal that our queueing theoretic model qualitatively evaluates multi-core architectures accurately with an average difference of 5.6% as compared to the architectures' evaluations from the SESC simulator. Our modeling approach can be used for performance per watt and performance per unit area characterizations of multi-core embedded architectures, with varying number of processor cores and cache configurations, to provide a comparative analysis.



## 1. Introduction and motivation

With Moore's law supplying billions of transistors on-chip, embedded systems are undergoing a paradigm shift from single-core to multi-core to exploit this high transistor density for high performance. This paradigm shift has led to the emergence of diverse multi-core embedded systems in a plethora of application domains (e.g., high-performance computing, dependable computing, mobile computing, etc.). Many modern embedded systems integrate multiple cores (whether homogeneous or heterogeneous) on-chip to satisfy computing demand while maintaining design constraints (e.g., energy, power, performance, etc.). For example, a 3G mobile handset's signal processing requires 35–40 Giga operations per second (GOPS). Considering the limited energy of a mobile handset battery, these performance levels must be met with a power dissipation budget of approximately 1W, which translates to a performance efficiency of 25 mW/GOP or 25 pJ/operation for the 3G receiver [4]. These demanding and competing power–performance requirements make modern embedded system design challenging.

* Corresponding author.
*E-mail addresses:* arslan@rice.edu, arslanmn@gmail.com (A. Munir), ann@ece.ufl.edu (A. Gordon-Ross), ranka@cise.ufl.edu (S. Ranka), farinaz@rice.edu (F. Koushanfar).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jpdc.2013.07.003

Increasing customer expectations/demands for embedded system functionality has led to an exponential increase in design complexity. While industry focuses on increasing the number of on-chip processor cores to meet the customer performance demands, embedded system designers face the new challenge of optimal layout of these processor cores along with the memory subsystem (caches and main memory) to satisfy power, area, and stringent real-time constraints. The short *time-to-market* (time from product conception to market release) of embedded systems further exacerbates design challenges. Architectural modeling of embedded systems helps in reducing the time-to-market by enabling fast application-to-device mapping since identifying an appropriate architecture for a set of target applications significantly reduces the design time of an embedded system. To ensure the timely completion of an embedded system's design with sufficient confidence in the product's market release, design engineers must make tradeoffs between the abstraction level of the system's architecture model and the attainable accuracy.

Modern multi-core embedded systems allow processor cores to share hardware structures, such as last-level caches (LLCs) (e.g., level two (L2) or level three (L3) cache), memory controllers, and interconnection networks [12]. Since the LLC's configuration (e.g., size, line size, associativity) and the layout of the processor cores (on-chip location) has a significant impact on a multi-core embedded system's performance and energy, our work focuses on performance and energy characterization of embedded architectures based on different LLC configurations and layout of the processor cores. Though there is a general consensus on using private level one (L1) instruction (L1-I) and data (L1-D) caches in embedded systems, there has been no dominant architectural paradigm for private or shared LLCs. Since many embedded systems contain an L2 cache as the LLC, we focus on the L2 cache, however, our study can easily be extended for L3 caches and beyond as LLCs.

Since multi-core benchmark simulation requires significant simulation time and resources, a lightweight modeling technique for multi-core architecture evaluation is crucial [11]. Furthermore, simulation-driven architectural evaluation is based on specific benchmarks and consequently only provides performance information for programs similar to the benchmarks. A well-devised modeling technique can model diverse workloads and thus enables performance evaluation for workloads with any computing requirements. Previous work presents various multi-core system models, however, these models become increasingly complex with varying degrees of cache sharing [33]. Many of the previous models assumed that sharing amongst processor cores occurred at either the main memory level or the processor cores all shared the same cache hierarchy, however, multi-core embedded systems can have an L2 cache shared by a subset of cores (e.g., Intel's six-core Dunnington processor has L2 caches shared by two processor cores). We leverage for the first time, to the best of our knowledge, queueing network theory as an alternative approach for modeling multi-core embedded systems for performance analysis (though queueing network models have been studied in the context of traditional computer systems [31]). Our queueing network model approach allows modeling the layout of processor cores (processor cores can be either homogeneous or heterogeneous) with caches of different capacities and configurations at different cache levels. Our modeling technique only requires a high-level workload characterization of an application (i.e., whether the application is processor-bound (requiring high processing resources), memory-bound (requiring a large number of memory accesses), or mixed).

Our main contributions in this paper are:

- we present a novel, queueing theory-based modeling technique for evaluating multi-core embedded architectures that does not require architectural-level benchmark simulation. This modeling technique enables quick and inexpensive architectural evaluation, with respect to design time and resources, as compared to developing and/or using the existing multi-core simulators and running benchmarks on these simulators. Based on a preliminary evaluation using our models, architecture designers can run targeted benchmarks to further verify the performance characteristics of selected multi-core architectures (i.e., our queueing theory-based models facilitate early design space pruning).
- our queueing theoretic approach enables the architectural evaluation for synthetic workloads with any computing requirements characterized probabilistically. We also propose a method to quantify computing requirements of real benchmarks probabilistically. Hence, our modeling technique can provide performance evaluation for workloads with any computing requirements as opposed to simulation-driven architectural evaluation that can only provide performance results for specific benchmarks.
- our queueing theoretic modeling approach can be used for performance per watt and performance per unit area characterizations of multi-core embedded architectures, with varying number of processor cores and cache configurations, to provide a comparative analysis. For performance per watt and performance per unit area computations, we calculate chip area and power consumption for different multi-core embedded architectures with a varying number of processor cores and cache configurations.

We point out that although queueing theory has been used in the literature for performance analysis of multi-disk and pipelined systems [13,18,31], we for the first time, to the best of our knowledge, apply queueing theory-based modeling and performance analysis techniques to multi-core embedded systems. Furthermore, we for the first time develop a methodology to synthesize workloads/benchmarks on our queueing theoretic multi-core models based on probabilities that are assigned according to workload characteristics (e.g., processor-bound, memory-bound, or mixed) and cache miss rates. We verify our queueing theoretic modeling approach by running SPLASH-2 multi-threaded benchmarks on the SuperESCalar simulator (SESC). Results reveal that our queueing theoretic model qualitatively evaluates multi-core architectures accurately with an average difference of 5.6% as compared to the architectures' evaluations from the SESC simulator. The SESC simulation results validate our queueing theoretic modeling approach as a quick and inexpensive architectural evaluation method.

Our queueing theoretic approach can be leveraged for *early design space pruning* by eliminating infeasible architectures in very early design stages, which reduces the number of lengthy architectural evaluations when running targeted benchmarks in later design stages. Specifically, our approach focuses on the qualitative comparison of architectures in the early design stage and not the quantitative comparison of architectures for different benchmarks. Our model is designed to operate using *synthetic workloads* that a designer can categorize for an expected behavior, such as processor or memory-bound workloads, along with an estimate of the expected cache miss rates. The synthetic workloads preclude the need to obtain benchmark-specific statistics from an architecture-level simulator. Furthermore, the cache miss rates are estimates, and thus are not required to be the exact miss rates for any specific benchmark. Our discussions in Section 4.2 regarding statistics obtained from an architecture-level simulator only explains how a real workload can be represented with our queueing theoretic model and is not required for synthetic workloads.

Our investigation of performance and energy for different cache miss rates and workloads is significant because cache miss rates and workloads can significantly impact the performance and energy of an embedded architecture. Furthermore, cache miss rates

also give an indication of the degree of cache contention between different threads' working sets. Our performance, power, and performance per watt results indicate that multi-core embedded architectures that leverage shared LLCs are scalable and provide the best LLC performance per watt. However, shared LLC architectures may introduce main memory response time and throughput bottlenecks for high cache miss rates. The architectures that leverage a hybrid of private and shared LLCs are scalable and alleviate main memory bottlenecks at the expense of reduced performance per watt. The architectures with private LLCs exhibit less scalability but do not introduce main memory bottlenecks at the expense of reduced performance per watt.

## 2. Related work

Several previous works investigated analytical modeling techniques for performance evaluation. Sorin et al. [37] developed an analytical model for evaluating shared memory systems with processors that aggressively exploited instruction-level parallelism (ILP). The application's ILP and interaction with the memory subsystem were characterized by the model's input parameters. Ipek et al. [16] used predictive modeling based on artificial neural networks to efficiently explore the architectural design spaces. Simulation of sampled points in the design space helped the model to generate functions that described the relationship between design parameters, which were then used to estimate performance for other points in the design space. Chandra et al. [7] investigated the performance models that predicted the impact of cache sharing on co-scheduled threads for chip multi-processors (CMPs). The authors observed that cache contention could significantly increase a thread's cache miss rate depending on the co-scheduled threads' memory requirements. Chen et al. [8] proposed an analytical model for predicting the impact of cache contention on cache miss rates for multi-programmed workloads running on multi-threaded many-core architectures.

Queueing theory has been used in the literature for the performance analysis of computer networks and other computer systems. Samari et al. [32] used queueing theory for the analysis of distributed computer networks. The authors proposed a correction factor in the analytical model and compared these results with the analytical model without this correction factor [21] and with simulation results to verify the correctness of the proposed model. Mainkar et al. [24] used queueing theory-based models for the performance evaluation of computer systems with a central processing unit (CPU) and disk drives. Willick et al. [39] used queueing theory to model packet-switched multistage interconnection networks. The analytical model permitted general interconnection network topologies (including arbitrary switch sizes) and arbitrary memory reference patterns. Our work differs from the previous work on queueing theory-based models for computer systems in that our work applies queueing theory for performance evaluation of multi-core embedded systems with different cache subsystems, which have not been investigated using queueing theory. Furthermore, our work introduces a novel way of representing workloads with different computing requirements probabilistically in a queueing theory based model.

Characterization of workloads for analytical models has been explored in the literature. Nussbaum et al. [27] developed an analytical performance model for superscalar processors based on statistical simulation. The model depended on detailed simulation to gather statistical information that was used to generate a synthetic instruction trace that was then fed as input to the model along with cache miss rate and branch prediction statistics. Karkhanis et al. [20] developed an analytical performance model for superscalar processors. The model used trace-derived data dependence information, data and instruction cache miss rates, and branch miss-prediction rates as input. Wunderlich et al. [41] proposed a framework for statistical sampling that enabled sampling of a minimal subset of a benchmark's instruction execution stream to estimate the performance of the complete benchmark. Our work differs from the previous work on workload characterization for analytical models in that we characterize workloads probabilistically, which provides an alternative to statistical sampling.

The previous work presents evaluation and modeling techniques for multi-core embedded architectures for different applications and varying workload characteristics. Savage et al. [33] proposed a unified memory hierarchy model for multi-core architectures that captured varying degrees of cache sharing at different cache levels. However, the model was only applicable to straight-line computations that could be represented by directed acyclic graphs (DAGs) (e.g., matrix multiplication, fast Fourier transform (FFT)). Our queueing theoretic models are applicable to virtually any type of workload with any computing requirements. Fedorova et al. [12] studied contention-aware task scheduling for multi-core architectures with shared resources (caches, memory controllers, and interconnection networks). The authors modeled the contention-aware task scheduler and investigated the scheduler's impact on application performance for multi-core architectures. In contrast to these prior works, our queueing theoretic models permit a wide range of scheduling disciplines based on the workload requirements (e.g., first-come–first-served (FCFS), priority, round robin (RR), etc.).

Some previous works investigated the performance and energy aspects for multi-core systems. Kumar et al. [22] studied power, throughput, and response time metrics for heterogeneous CMPs. The authors observed that heterogeneous CMPs could improve energy per instruction by 4–6× and throughput by 63% over an equivalent-area homogeneous CMP because of closer adaptation to the resource requirements of different application phases. The authors used a multi-core simulator for performance analysis, however, our queueing theoretic models can be used as a quick and inexpensive alternative for investigating the performance aspects of heterogeneous CMPs. Sabry et al. [30] investigated performance, energy, and area tradeoffs for private and shared L2 caches for multi-core embedded systems. The authors proposed a SystemC-based platform that could model private, shared, and hybrid L2 cache architectures. A hybrid L2 cache architecture contains several private L2 caches, each containing only private data, and a unified shared L2 cache that stores only shared data. However, the SystemC-based model required integration with other simulators, such as MPARM, to obtain the performance results. Our queueing theoretic models do not require integration with other multi-core simulators to obtain performance results and serve as an independent comparative performance analysis approach for multi-core architecture evaluation.

Benítez et al. [5] proposed an adaptive L2 cache architecture that adapted to fit the code and data during runtime using partial cache array shut down. The adaptive cache could be configured in four modes that prioritized either instructions per cycle (IPC), processor power dissipation, processor energy consumption, or processor power$^2\times$ delay product. Experiments revealed that CMPs with 2 MB of private adaptive L2 cache provided 14.2%, 44.3%, 18.1%, and 29.4% improvement in IPC, power dissipation, energy consumption, and power$^2\times$ delay, respectively, over a 4 MB shared L2 cache. Our work does not consider the adaptive private caches but compares private, shared, and hybrid caches on an equal area basis to provide a fair comparison between different LLCs.

There exists work in the literature related to the memory subsystem latency and throughput analysis. Ruggiero [29] investigated cache latency (at all cache levels), memory latency, cache bandwidth/throughput (at all cache levels), and memory bandwidth for multi-core embedded systems using LMBench, an open

**Table 1**
Multi-core embedded architectures with varying processor cores and cache configurations (P denotes a processor core, M main memory, and integer constants in front of P, L1ID (L1 instruction and data cache), L2, and M denotes the number of these architectural components in the embedded architecture).

| Architecture | Description |
|---|---|
| 2P-2L1ID-2L2-1M | Multi-core embedded architecture with 2 processor cores, private L1 I/D caches, private L2 caches, and a shared M |
| 2P-2L1ID-1L2-1M | Multi-core embedded architecture with 2 processor cores, private L1 I/D caches, a shared L2 cache, and a shared M |
| 4P-4L1ID-4L2-1M | Multi-core embedded architecture with 4 processor cores, private L1 I/D caches, private L2 caches, and a shared M |
| 4P-4L1ID-1L2-1M | Multi-core embedded architecture with 4 processor cores, private L1 I/D caches, a shared L2 cache, and a shared M |
| 4P-4L1ID-2L2-1M | Multi-core embedded architecture with 4 processor cores, private L1 I/D caches, 2 shared L2 caches, and a shared M |

source benchmark suite, on Intel processors. Our models enable measurement of cache latency and throughput of modeled architectures in an early design phase when the fabricated architectures are not available.

Although there exist previous works on performance evaluation, our work is novel because we for the first time develop queueing network models for various multi-core embedded architectures. Some previous works present benchmark-driven evaluation for specific embedded architectures, however, multi-core embedded architecture evaluation considering different workload characteristics, cache configurations (private, shared, or hybrid), and miss rates with comparative analysis has not been addressed.

## 3. Queueing network modeling of multi-core embedded architectures

Although queueing networks are widely used in computer networks, the interpretation of queueing network terminology for multi-core embedded architectures is a pre-requisite for queueing theory-based modeling. Furthermore, because of the diversity of queueing network models, the specific approach taken to model multi-core embedded architectures is the most critical aspect of queueing theory-based modeling. Finally, the queueing theory-based model relies on some assumptions under which the model is a valid representation of the multi-core embedded architectures. This section discusses the queueing network terminology in the context of multi-core embedded architectures, our modeling approach, and the underlying simplifying assumptions.

### 3.1. Queueing network terminology

A *queueing network* consists of *service centers* (e.g., processor core, L1-I cache, L1-D cache, L2 cache, and main memory (MM)) and *customers* (e.g., jobs/tasks). A service center consists of one or more queues to hold jobs waiting for service. We use the term *jobs* instead of *tasks* (decomposed workload resulting from parallelizing a job) to be consistent with general queueing network terminology. Our modeling approach is broadly applicable to *multi-programmed workloads* where multiple jobs run on the multi-core embedded architecture as well as for *parallelized applications/jobs* that run different *tasks* on the multi-core architectures. Arriving jobs enter the service center's queue and a *scheduling/queueing discipline* (e.g., first-come–first-served (FCFS), priority, round robin (RR), processor sharing (PS), etc.) selects the next job to be served when a service center becomes idle. The queueing discipline is *preemptive* if an arriving higher priority job can suspend the service/execution of a lower priority job, otherwise the queueing discipline is *non-preemptive*. FCFS is a non-preemptive queueing discipline that serves the waiting jobs in the order in which the jobs enter the queue. Priority-based queueing disciplines can be preemptive or non-preemptive and serve the jobs based on an assigned job priority. In the RR queueing discipline, a job receives a service time quantum (slot). If the job does not complete during the service time quantum, the job is placed at the end of the queue to resume during a subsequent service time quantum. In the PS queueing discipline, all jobs at a service center are serviced simultaneously (and hence there is no queue) with the service center's speed equally divided across all of the jobs. After being serviced, a job either moves to another service center or leaves the network.

A queueing network is *open* if jobs arrive from an external source, spend time in the network, and then depart. A queueing network is *closed* if there is no external source and no departures (i.e., a fixed number of jobs circulate indefinitely among the service centers). A queueing network is a *single-chain* queueing network if all jobs possess the same characteristics (e.g., arrival rates, required service rates, and routing probabilities for various service centers) and are serviced by the same service centers in the same order. If different jobs can belong to different chains, the network is a *multi-chain* queueing network. An important class of queueing networks is *product-form* where the joint probability of the queue sizes in the network is a product of the probabilities for the individual service centers' queue sizes.

The queueing network performance metrics include response time, throughput, and utilization. The *response time* is the amount of time a job spends at the service center including the queueing delay (the amount of time a job waits in the queue) and the service time. The service time of a job depends on the amount of work (e.g., number of instructions) needed by that job. The *throughput* is defined as the number of jobs served per unit of time. In our multi-core embedded architecture context, throughput measures the number of instructions/data (bits) processed by the architectural element (processor, cache, MM) per second. *Utilization* measures the fraction of time that a service center (processor, cache, MM) is busy. Little's law governs the relationship between the number of jobs in the queueing network $N$ and response time $tr$ (i.e., $N = \kappa \cdot tr$ where $\kappa$ denotes the average arrival rate of jobs admitted to the queueing network [25]).

### 3.2. Modeling approach

We consider the closed product-form queueing network for modeling multi-core embedded architectures because the closed product-form queueing network enables unequivocal modeling of workloads. A typical embedded system executes a fixed number of jobs (e.g., a mobile phone has only a few applications to run, such as instant messaging, audio coding/decoding, calculator, graphics interface, etc.). We point out that additional applications can be added/updated in an embedded system (e.g., a smartphone) over time, however, these additional applications can be represented as synthetic workloads in our queueing-theoretic model. Furthermore, closed product-form queueing networks assume that a job leaving the network is replaced instantaneously by a statistically identical new job [31]. Table 1 describes the multi-core embedded architectures that we evaluate in this paper. We focus on embedded architectures ranging from 2 (2P) to 4 (4P) processor cores to reflect current architectures [15], however, our model is applicable to any number of cores. Our modeled embedded architectures contain processor cores, L1-I and L1-D private caches, L2 caches (private or shared), and MM (embedded systems are typically equipped with DRAM/NAND/NOR Flash memory [23,36]).

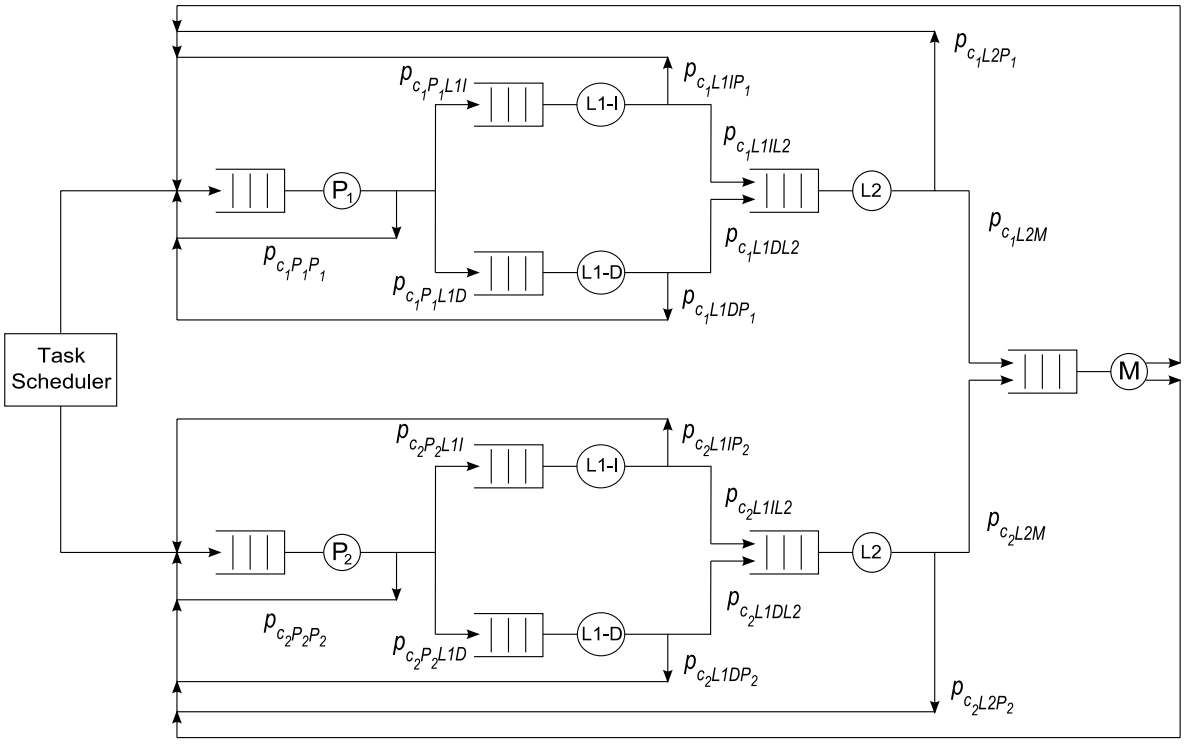


**Fig. 1.** Queueing network model for the 2P-2L1ID-2L2-1M multi-core embedded architecture.

We consider a closed product-form queueing network with $I$ service centers where each service center $i \in I$ has a service rate $\mu_i$. Let $p_{ij}$ be the probability of a job leaving service center $i$ and entering another service center $j$. The relative visit count $\vartheta_j$ to service center $j$ is:

$$\vartheta_j = \sum_{i=1}^{I} \vartheta_i p_{ij}. \tag{1}$$

The performance metrics (e.g., throughput, response time, etc.) for a closed product-form queueing network can be calculated using a *mean value analysis* (MVA) iterative algorithm [28]. The basis of MVA is a theorem stating that when a job arrives at a service center in a closed network with $N$ jobs, the distribution of the number of jobs already queued is the same as the steady state distribution of $N-1$ jobs in the queue [35]. Solving (1) using MVA recursively gives the following performance metric values: the mean response time $r_i(k)$ at service center $i$ ($k$ denotes the number of jobs in the network), mean network response time $R(k)$, mean queueing network throughput $T(k)$, the mean throughput of jobs $\lambda_i(k)$ at service center $i$, and the mean queue length $l_i(k)$ at service center $i$ when there are $k$ jobs in the network. The initial recursive conditions are $k = 0$ such that $r_i(0) = R(0) = T(0) = t_i(0) = l_i(0) = 0$. The values for these performance metrics can be calculated for $k$ jobs based on the computed values for $k-1$ jobs as [31]:

$$r_i(k) = \frac{1}{\mu_i}(1 + l_i(k-1)) \tag{2}$$

$$R(k) = \sum_{i=1}^{I} \vartheta_i \cdot r_i(k) \tag{3}$$

$$T(k) = \frac{k}{R(k)} \tag{4}$$

$$\lambda_i(k) = \vartheta_i \cdot T(k) \tag{5}$$

$$l_i(k) = \lambda_i(k) \cdot r_i(k) \tag{6}$$

To explain our modeling approach for multi-core embedded architectures, we describe a sample queueing model for the 2P-2L1ID-2L2-1M architecture in detail (other architecture models follow a similar explanation). Fig. 1 depicts the queueing network model for 2P-2L1ID-2L2-1M. The *task scheduler* schedules the tasks/jobs on the two processor cores $P_1$ and $P_2$. We assume that the task scheduler is contention-aware and schedules tasks with minimal or no contention on cores sharing LLCs [12]. The queueing network consists of two chains: chain one corresponds to processor core $P_1$ and chain two corresponds to processor core $P_2$. The jobs serviced by $P_1$ either reenter $P_1$ with probability $p_{c_1P_1P_1}$ ($c_1$ in the subscript denotes chain one) or enter the L1-I cache with probability $p_{c_1P_1L1I}$ or the L1-D cache with probability $p_{c_1P_1L1D}$. The job arrival probabilities into the service centers (processor core, L1-I, L1-D, L2, or MM) depend on the workload characteristics (i.e., processor-bound, memory-bound, or mixed). The data from the L1-I cache and the L1-D cache return to $P_1$ with probabilities $p_{c_1L1IP_1}$ and $p_{c_1L1DP_1}$, respectively, after L1-I and L1-D cache hits. The requests from the L1-I cache and the L1-D cache are directed to the L2 cache with probabilities $p_{c_1L1IL2}$ and $p_{c_1L1DL2}$, respectively, after L1-I and L1-D cache misses. The probability of requests entering $P_1$ or the L2 cache from the L1-I and L1-D cache depends on the miss rates of the L1-I and L1-D caches. After an L2 cache hit, the requested data is transferred to $P_1$ with probability $p_{c_1L2P_1}$ or enters MM with probability $p_{c_1L2M}$ after an L2 cache miss. The requests from MM always return to $P_1$ with probability $p_{c_1MP_1} = 1$. The queueing network chain and path for chain two corresponding to $P_2$ follow the same pattern as chain one corresponding to $P_1$. For example, requests from the L2 cache in chain two either return to $P_2$ with probability $p_{c_2L2P_2}$ after an L2 cache hit or enter MM with probability $p_{c_2L2M}$ after an L2 cache miss ($c_2$ in the subscript denotes chain two).

To further elaborate on our modeling approach, Fig. 2 depicts the queueing model for 2P-2L1ID-1L2-1M, which is similar to the model for 2P-2L1ID-2L2-1M (Fig. 1) except that this queueing model contains a shared L2 cache (L2s denotes the shared L2 cache in Fig. 2) for the two processor cores $P_1$ and $P_2$ instead of private L2

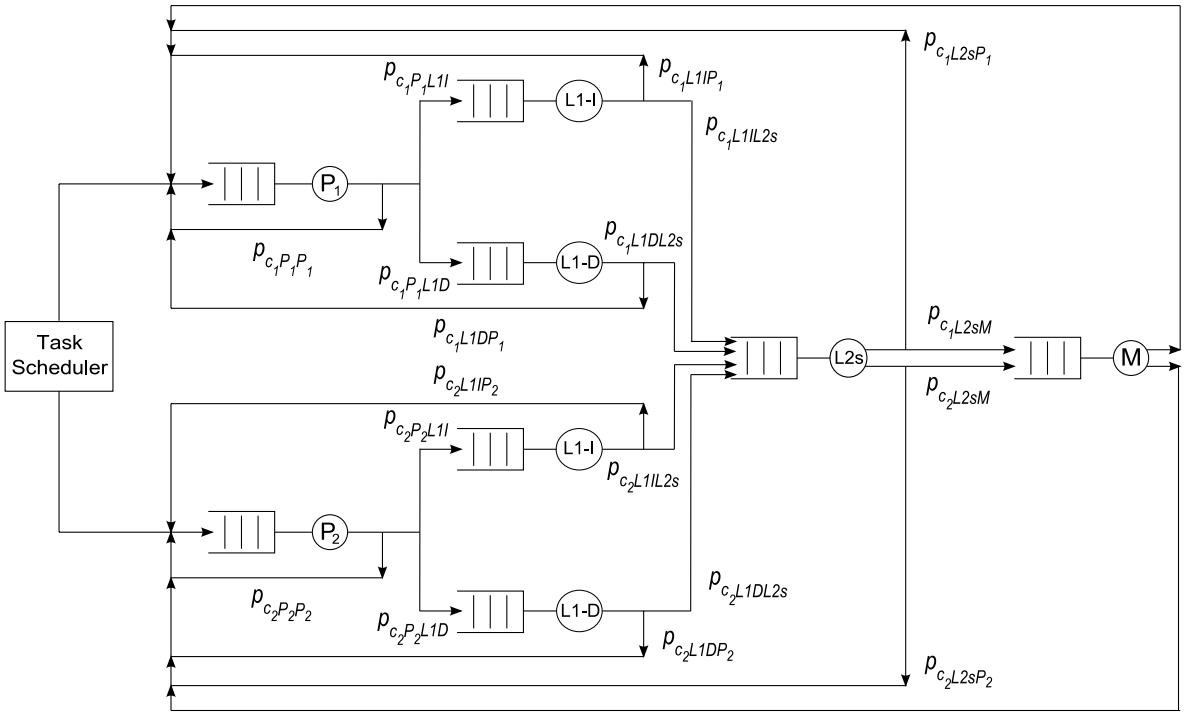


**Fig. 2.** Queueing network model for the 2P-2L1ID-1L2-1M multi-core embedded architecture.

caches. The queueing network consists of two chains: chain one corresponds to processor core $P_1$ and chain two corresponds to processor core $P_2$. The requests from the L1-I cache and the L1-D cache for chain one go to the shared L2 cache (L2s) with probability $p_{c_1 L1IL2s}$ and $p_{c_1 L1DL2s}$, respectively, on L1-I and L1-D cache misses. The requested data is transferred from the L2 cache to $P_1$ with probability $p_{c_1 L2sP_1}$ on an L2 cache hit whereas the data request goes to MM with probability $p_{c_1 L2sM}$ on an L2 cache miss. The requests from the L1-I cache and the L1-D cache for chain two go to the shared L2 cache (L2s) with probability $p_{c_2 L1IL2s}$ and $p_{c_2 L1DL2s}$, respectively, on L1-I and L1-D cache misses. The requested data from the L2 cache is transferred to $P_2$ with probability $p_{c_2 L2sP_2}$ on an L2 cache hit whereas the data request goes to MM with probability $p_{c_2 L2sM}$ on an L2 cache miss.

The probabilities assignment in our queueing network models to represent a synthetic workload (or to emulate a real workload) on a multi-core architecture is critical to our modeling approach and the fidelity of our evaluations. Our models can be leveraged for studying workloads based on an overall workload behavior, where processor-to-processor probability $p_{PP}$ and processor-to-memory probability $p_{PM}$ remain uniform throughout the workload, or for a more detailed study of workloads with different phases, where a different $p_{PP}$ and $p_{PM}$ would be assigned for each phase. These probabilities can be determined based on processor and memory statistics for a given workload using one of two methods: actual statistics for real workloads can be gathered using a functional simulator if these statistics are not available from prior research; or synthetic workloads can be leveraged wherein the designer would assign these statistics based on the expected workload behavior. For our models, $p_{PP}$ can be estimated as:

$$p_{PP} = O_P / O_T \tag{7}$$

where $O_P$ and $O_T$ denote the number of processor operations and total operations (processor and memory) in a benchmark, respectively. Processor operations refer to arithmetic and logic unit (ALU) micro-operations, such as add and subtract, and memory operations refer to micro-operations involving memory accesses, such as load and store. For floating point benchmarks, $O_P$ can be assumed to be equal to the number of floating point operations. $O_T$ can be estimated as:

$$O_T = O_P + O_M \tag{8}$$

where $O_M$ denotes the number of memory operations, which is the sum of the total read and total write operations. $p_{PM}$ can be estimated as:

$$p_{PM} = 1 - p_{PP}. \tag{9}$$

The probability of requests going from the processor core $P_i$ in chain $i$ to the L1-I and L1-D caches can be estimated from the L1-I and L1-D cache access ratios (defined below), which require the number of L1-I and L1-D accesses. The number of L1-I accesses can be calculated as (assuming there is no self-modifying code):

$$\text{L1-I accesses} = \text{L1-I read misses} + \text{L1-I read hits}. \tag{10}$$

The number of L1-D accesses can be calculated as:

$$\text{L1-D accesses} = \text{L1-D read misses} + \text{L1-D read hits} + \text{L1-D write misses} + \text{L1-D write hits}. \tag{11}$$

Total number of L1 cache accesses can be calculated as:

$$\text{Total L1 accesses} = \text{L1-D accesses} + \text{L1-I accesses}. \tag{12}$$

The L1-I access ratio can be calculated as:

$$\text{L1-I Access Ratio} = \frac{\text{L1-I accesses}}{\text{Total L1 accesses}} \tag{13}$$

The L1-D access ratio can be calculated as

$$\text{L1-D Access Ratio} = 1 - \text{L1-I Access Ratio} \tag{14}$$

The probability of requests going from the processor core $P_i$ in chain $i$ to the L1-I cache $p_{c_i P_i L1I}$ can be calculated as:

$$p_{c_i P_i L1I} = p_{PM} \times \text{L1-I Access Ratio} \tag{15}$$

Similarly, the probability of requests going from the processor core $P_i$ in chain $i$ to the L1-D cache $p_{c_i P_i L1D}$ can be calculated as:

$$p_{c_i P_i L1D} = p_{PM} \times \text{L1-D Access Ratio} \tag{16}$$

The probabilities of requests going from the L1-I and L1-D caches to the L2 cache and from the L2 cache to MM can be calculated directly from the cache miss rate statistics and are omitted for brevity.

We exemplify the assignment of probabilities for our queueing network models using the 2P-2L1ID-2L2-1M multi-core architecture for memory-bound workloads given the following statistics: $p_{PP} = 0.1$; $p_{PM} = 0.9$ assuming that the L1-I, L1-D, and L2 cache miss rates are 25%, 50%, and 30%, respectively; and L1-I and L1-D access ratios are 0.8 and 0.2, respectively. The probabilities are set as: $p_{c_1P_1P_1} = 0.1$, $p_{c_1P_1L1I} = 0.72$, $p_{c_1P_1L1D} = 0.18$, $p_{c_1L1IP_1} = 0.75$, $p_{c_1L1DP_1} = 0.5$, $p_{c_1L1IL2} = 0.25$, $p_{c_1L1DL2} = 0.5$, $p_{c_1L2P_1} = 0.7$, $p_{c_1L2M} = 0.3$, $p_{c_1MP_1} = 1$ (different probabilities can be assigned for processor-bound or mixed workloads).

Our queueing theoretic models determine the performance metrics of component-level architectural elements (e.g., processor cores, L1-1, L1-D, etc.), however, embedded system designers are often also interested in system-wide performance metrics. For example, system-wide response time of an architecture is an important metric for real-time embedded applications. Our queueing theoretic models enable calculations of system-wide performance metrics. Based on our queueing theoretic models, we can calculate the system-wide response time $\bar{R}$ of a multi-core embedded architecture as:

$$\bar{R} = \max_{\forall\, i=1,\ldots,N_P} \big((p_{c_iP_iP_i} \times r_{P_i}) + (p_{c_iP_iL1I} \times r_{iL1I}) + (p_{c_iP_iL1D} \times r_{iL1D}) + ((p_{c_iL1IL2} + p_{c_iL1DL2}) \times r_{iL2}) + (p_{c_iL2M} \times r_M)\big) \quad (17)$$

where $N_P$ denotes the total number of processor cores in the multi-core embedded architecture. $r_{P_i}$, $r_{iL1I}$, $r_{iL1D}$, $r_{iL2}$, and $r_M$ denote the response times for processor core $P_i$, L1-I, L1-D, and L2 corresponding to chain $i$, and MM, respectively. $p_{c_iP_iP_i}$ denotes the probability of requests looping back from processor core $P_i$ to processor core $P_i$ in the queueing network chain $i$ (the total number of chains in the queueing network is equal to $N_P$). $p_{c_iP_iL1I}$ and $p_{c_iP_iL1D}$ denote the probability of requests going from processor core $P_i$ in chain $i$ to the L1-I cache and the L1-D cache, respectively. $p_{c_iL1IL2}$ and $p_{c_iL1DL2}$ denote the probability of requests going from the L1-I cache and the L1-D cache in chain $i$ to the L2 cache, respectively. $p_{c_iL2M}$ denotes the probability of requests going from the L2 cache in chain $i$ to MM. We point out that (17) is an extension of (3) for a multi-chain queueing network and also simplifies the calculation by using probabilities between architectural elements directly instead of relative visit counts as in (3). Since processor cores in a multi-core embedded architecture operate in parallel, the effective response time of a multi-core architecture is the maximum response time out of all of the chains in the queueing network operating in parallel, as given in (17). In the context of queueing networks for parallel computing, the overall response time is determined by the slowest chain since the other chains must wait idle for the slowest chain to complete (load balancing deals with making the processor cores' response times close to each other to minimize the idle waiting). The response time of a single chain in the queueing network is the sum of the processing times of the architectural elements in a chain (e.g., processor core, L1-1, L1-D, L2, and MM) multiplied by the associated service probabilities of these architectural elements. System-wide throughput can be given similar to (17).

Our queueing network modeling provides a faster alternative for performance evaluation of multi-core architectures as compared to running complete benchmarks on multi-core simulators (and/or trace simulators) though at the expense of accuracy. Our queueing network models require simulating only a subset of the benchmark's instructions (specified implicitly by the service rates of the architectural components, such as processor cores and caches) that are necessary to reach a steady state/equilibrium in the queueing network with the workload behavioral characteristics captured by the processor-to-processor and processor-to-memory probabilities (as shown in Fig. 1). Since the service centers (processors, caches, memory) have been modeled explicitly in our queueing theoretic models, the effect of these service centers on the overall system performance is captured by our model [39].

### 3.3. Assumptions

Our queueing theoretic models make some simplifying assumptions, which do not affect the general applicability of our approach. Our queueing network models assume cycle-level assignments of tokens (service time slices) for a given workload/job such that in each cycle, the tokens receive service from a particular service center with a given probability. For example, a job leaving the processor core either returns to the processor core's queue to wait for another time slice or goes to either the L1-I or L1-D cache for an instruction or data fetch, respectively [31]. Completed jobs are replaced immediately by a statistically identical job, an assumption for closed product-form queueing networks, which holds true for embedded systems [31]. Our queueing network modeling approach can be extended to instruction-level tokens if desired (although not required) for a given workload where an instruction can go to multiple service centers simultaneously (e.g., an instruction going to the L1-I and L1-D cache simultaneously to obtain the instruction opcodes and data, respectively). For these cases, a compound service center can be added to our queueing network model (e.g., L1-I + L1-D) that represents multiple service centers. The probability of a job going to a compound service center would be the sum of the probabilities of the individual service centers represented by that compound service center.

Our models are well suited for multi-programmed workloads that are common in the embedded systems, however, our approach is also applicable to multi-threaded workloads with some simplifying assumptions. We assume that the LLCs' sizes are large enough to hold the working sets of the executing threads, which alleviates performance problems, such as sub-optimal throughput arising due to cache thrashing and thread starvation [7]. We assume that the appropriate inter-thread cache partitioning schemes alleviate the performance issues due to cache sharing and is not the focus of our model. Furthermore, since cache contention impacts cache miss rates, our model allows specification of appropriate cache miss rates for investigating the impact of workloads with cache contention. For example, workloads that are likely to cause cache contention can be assigned higher cache miss rates in our models to incorporate the cache contention effects. Additionally, shared LLCs and MM in our queueing network model account for the contention of resources between the processor cores and/or executing threads [31].

Although our current models do not explicitly model the critical sections and coherence in multi-threaded programs as stated in our modeling assumptions, our models can capture the critical sections in a workload. We point out that a critical section is a piece of code that accesses a shared resource (e.g., a data structure) that must not be concurrently accessed by more than one thread of execution. Since critical sections are effectively serialized, the response time of the workload containing critical sections will increase depending on the number of critical sections and the number of instructions in each critical section. Hence, additional time for executing critical sections can be calculated by the number of critical sections and the number of instructions in each critical section and added to the response time of the workload. The coherence is implicitly modeled in our queueing theory models since coherence issues will cause coherence misses and these misses can be incorporated into our cache miss rate specifications for a workload as our models enable specification of any cache miss rates for

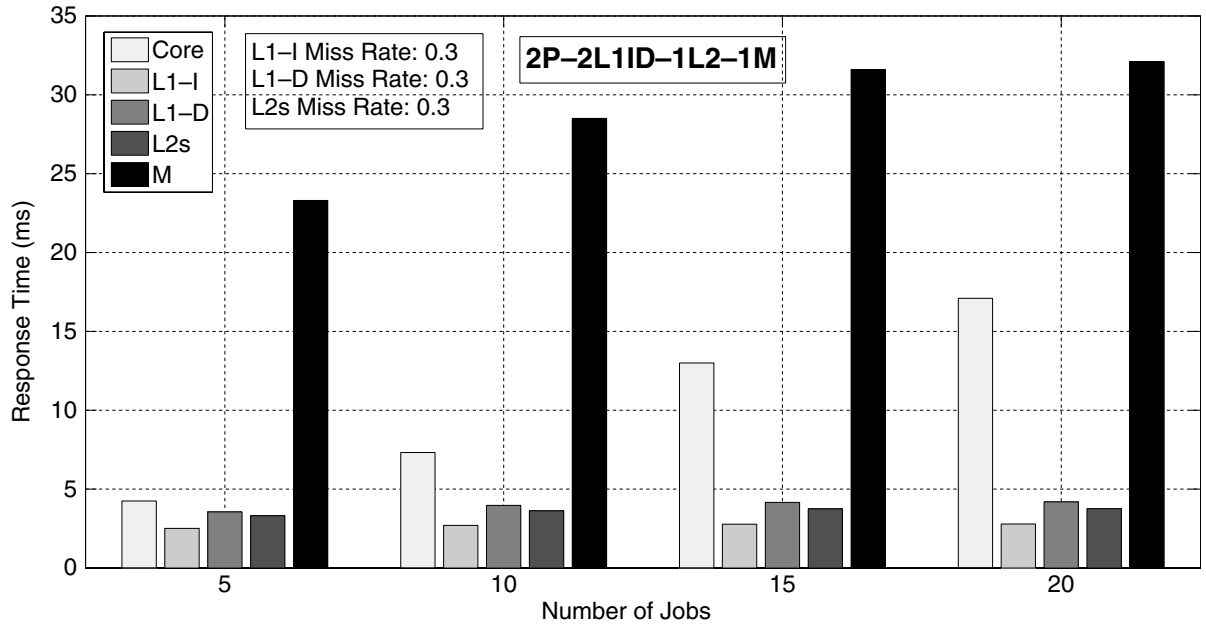


**Fig. 3.** Queueing network model validation of the response time in ms for mixed workloads for 2P-2L1ID-1L2-1M for a varying number of jobs $N$.

our synthesized workloads. The effects of different cache miss rates for a workload on performance are discussed in Appendix B.

We note that even though some of these assumptions may violate practical scenarios, such violations would not significantly impact the insights obtained from our queueing theoretic models because our models measure performance trends and focus on the *relative* performance of architectures for different benchmarks rather than the absolute performance.

## 4. Queueing network model validation

In this section, we validate our queueing network models to assure that our models' results conform with expected queueing theoretic results. We further validate our queueing network models with a multi-core simulator running multi-threaded benchmarks. We also calculate the speedups attained by our queueing network models as compared to a multi-core simulator for architectural performance evaluation.

### 4.1. Theoretical validation

We analyzed our queueing network models for different cache miss rates and workloads and find that the model's simulation results conform with expected queueing theoretical results. For example, Fig. 3 depicts the response time for mixed workloads ($p_{PP} = 0.5$, $p_{PM} = 0.5$) for 2P-2L1ID-1L2-1M as the number of jobs/tasks $N$ varies. The figure shows that as $N$ increases, the response time for the processor core, L1-I, L1-D, L2, and MM increases for all of the cache miss rates. We point out that cache miss rates could increase as $N$ increases due to inter-task address conflicts and increasing cache pressure (increased number of working sets in the cache), but we assume that the cache sizes are sufficiently large enough so that capacity misses remain the same for the considered number of jobs. We present the average response time individually for the processor cores and the L1-I, L1-D, and L2 caches. For smaller L1-I, L1-D, and L2 cache miss rates, the processor core response time increases drastically as $N$ increases because most of the time jobs are serviced by the processor core whereas for larger L1-I, L1-D, and L2 cache miss rates, the MM response time increases drastically because of a large number of MM accesses. These results along with our other observed results conform with the expected queueing theoretical results and validate our queueing network models for multi-core architectures.

### 4.2. Validation with a multi-core simulator

We further validate our queueing theoretic approach for modeling multi-core architectures using multi-threaded benchmarks executing on a multi-core simulator. We choose kernels/applications from the SPLASH-2 benchmark suite, which represent a range of computations in the scientific, engineering, and graphics domains. Our selected kernels/applications from the SPLASH-2 benchmark suite include fast Fourier transform (FFT), LU decomposition, radix, raytrace, and water-spatial [40].

Tables 2 and 3 depict the queueing network model probabilities for SPLASH-2 benchmarks for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively (Section 3.2). These probabilities are obtained using Eqs. (7)–(16) where the statistics required by these equations are acquired from an architecture-level simulator (SESC in this case) for an accurate representation of the benchmarks in our queueing network model. From our probabilistic characterization of the workloads, FFT, LU, radix, and water-spatial can be classified as processor-bound workloads and raytrace can be classified as a mixed workload. We reemphasize that this statistics gathering is only required for the accurate representation of real benchmarks and is not required for synthetic workloads.

We simulate the architectures in Table 1 using SESC [34]. To accurately capture our modeled architectures with SESC, our queueing theoretic models use the same processor and cache parameters (e.g., processor operating frequency, cache sizes and associativity, etc.) for the architectures as specified in the SESC configuration files. We consider single-issue processors with five pipeline stages and a 45 nm process technology. The execution times for the benchmarks on SESC are calculated from the number of cycles required to execute those benchmarks (i.e., execution time = (Number of cycles) × (cycle time)). For example, FFT requires 964,057 cycles to execute on 4P-4L1ID-4L2-1M at 16.8 MHz (59.524 ns), which gives an execution time of 57.38 ms.

To verify that the insights obtained from our queueing theoretic models regarding architectural evaluation are the same as

**Table 2**
Queueing network model probabilities for SPLASH-2 benchmarks for 2P-2L1ID-2L2-1M.

| Benchmark | Chain $i$ | $p_{c_iP_iP_i}$ | $p_{c_iP_iL1I}$ | $p_{c_iP_iL1D}$ | $p_{c_iL1IP_i}$ | $p_{c_iL1DP_i}$ | $p_{c_iL1IL2}$ | $p_{c_iL1DL2}$ | $p_{c_iL2P_i}$ | $p_{c_iL2M}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| FFT | $i=1$ | 0.68 | 0.26 | 0.06 | 0.99 | 0.947 | 0.00962 | 0.053 | 0.97 | 0.0258 |
| | $i=2$ | 0.7 | 0.249 | 0.051 | 0.9996 | 0.837 | 0.000414 | 0.163 | 0.956 | 0.044 |
| LU | $i=1$ | 0.68 | 0.2598 | 0.0602 | 0.99979 | 0.852 | 0.00021 | 0.148 | 0.9858 | 0.0142 |
| | $i=2$ | 0.76 | 0.2016 | 0.0384 | 0.9999 | 0.87 | 0.0000314 | 0.13 | 0.988 | 0.012 |
| Radix | $i=1$ | 0.781 | 0.184 | 0.035 | 0.9999 | 0.932 | 0.000051 | 0.068 | 0.894 | 0.106 |
| | $i=2$ | 0.78 | 0.1848 | 0.0352 | 0.99998 | 0.932 | 0.0000151 | 0.068 | 0.894 | 0.106 |
| Raytrace | $i=1$ | 0.58 | 0.3158 | 0.1042 | 0.9788 | 0.9382 | 0.0212 | 0.0618 | 0.961 | 0.039 |
| | $i=2$ | 0.584 | 0.312 | 0.104 | 0.9862 | 0.9146 | 0.0138 | 0.0854 | 0.9463 | 0.0537 |
| Water-spatial | $i=1$ | 0.68 | 0.2544 | 0.0656 | 0.99717 | 0.982 | 0.00283 | 0.018 | 0.9956 | 0.00442 |
| | $i=2$ | 0.68 | 0.2547 | 0.0653 | 0.9991 | 0.9814 | 0.000882 | 0.0186 | 0.9969 | 0.00307 |

**Table 3**
Queueing network model probabilities for SPLASH-2 benchmarks for 2P-2L1ID-1L2-1M.

| Benchmark | Chain $i$ | $p_{c_iP_iP_i}$ | $p_{c_iP_iL1I}$ | $p_{c_iP_iL1D}$ | $p_{c_iL1IP_i}$ | $p_{c_iL1DP_i}$ | $p_{c_iL1IL2}$ | $p_{c_iL1DL2}$ | $p_{c_iL2P_i}$ | $p_{c_iL2M}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| FFT | $i=1$ | 0.68 | 0.26 | 0.06 | 0.9905 | 0.947 | 0.0095 | 0.053 | 0.982 | 0.018 |
| | $i=2$ | 0.7 | 0.249 | 0.051 | 0.9996 | 0.84 | 0.0004 | 0.16 | 0.982 | 0.018 |
| LU | $i=1$ | 0.68 | 0.2624 | 0.0576 | 0.9998 | 0.9655 | 0.000166 | 0.0345 | 0.9987 | 0.0013 |
| | $i=2$ | 0.76 | 0.202 | 0.036 | 0.9999 | 0.86 | 0.000011 | 0.14 | 0.9987 | 0.0013 |
| Radix | $i=1$ | 0.781 | 0.184 | 0.035 | 0.99996 | 0.9331 | 0.000036 | 0.0669 | 0.888 | 0.112 |
| | $i=2$ | 0.78 | 0.1848 | 0.0352 | 0.999998 | 0.9335 | 0.0000022 | 0.0665 | 0.888 | 0.112 |
| Raytrace | $i=1$ | 0.582 | 0.314 | 0.1037 | 0.9791 | 0.752 | 0.0209 | 0.248 | 0.9881 | 0.0119 |
| | $i=2$ | 0.584 | 0.312 | 0.104 | 0.9867 | 0.9285 | 0.0133 | 0.0715 | 0.9881 | 0.0119 |
| Water-spatial | $i=1$ | 0.679 | 0.256 | 0.0655 | 0.9972 | 0.9821 | 0.00283 | 0.0179 | 0.99819 | 0.00181 |
| | $i=2$ | 0.679 | 0.2558 | 0.0652 | 0.99915 | 0.9814 | 0.000854 | 0.0186 | 0.99819 | 0.00181 |

obtained from a multi-core simulator, we compare the execution time results for the FFT, LU (non-contiguous), Raytrace, radix (an example for memory-bound workloads), and water-spatial (an example for mixed workloads) benchmarks on SESC and our queueing theoretic models (the benchmarks are represented probabilistically for our queueing theoretic models (as shown in Tables 2 and 3 for the dual-core architectures). System-wide response time is obtained from our queueing theoretic models for these benchmarks using (17). We compare the execution time trends for the FFT, LU, raytrace, water-spatial, and radix benchmarks on SESC and the modeled benchmarks for our queueing theoretic models using the SHARPE modeling tool/simulator [31].

Table 4 summarizes the performance (execution time) results on SESC for the FFT, LU (non-contiguous), radix, raytrace, and water-spatial benchmarks for both the two core and four core processor architectures. The results from SESC and our queueing theoretic models provide similar insights and show that the multi-core architectures with shared LLCs provide better performance than the architectures with private and hybrid LLCs for these benchmarks. Furthermore, the architectures with hybrid LLCs exhibit superior performance than the architectures with private LLCs. Since our queueing theoretic models provide relative performance measures for different architectures and benchmarks by simulating a minimum number of the benchmarks' representative instructions, the results show a difference in the absolute execution times obtained from SESC and our queueing theoretic models. Furthermore, the execution time of the benchmarks on SESC depends on the input sizes for the benchmarks and varies for different input sizes but normally retains similar trends across different architectures. Our queueing theoretic models capture the performance trends, which are important for relative comparison of different architectures, and these trends match the performance trends obtained from SESC.

In a high-level qualitative comparison of architectures, designers are typically interested in the relative performance measures for various benchmarks on the evaluated architectures. To verify that our queueing network models can capture the differences in execution times for various benchmarks, Table 5 summarizes the evaluation results comparing different dual-core architectures (2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M) on SESC and our queueing theoretic model. Results indicate that our queueing theoretic model qualitatively evaluates the two architectures accurately with an average difference of 5.6% as compared to the SESC simulator evaluation of the architectures. We observed similar architectural evaluation trends for four-core architectures and omit these details for brevity.

The SESC simulation results on multi-threaded benchmarks verify the results and insights obtained from our queueing theoretic models for mixed and processor-bound workloads (Section 5.2). From the probabilistic characterization of the SPLASH-2 benchmarks, we ascertain that it is difficult to find benchmarks

**Table 4**
Execution time comparison of the SPLASH-2 benchmarks on SESC for multi-core architectures.

| Architecture | FFT (ms) | LU (ms) | Radix (s) | Raytrace (s) | Water-spatial (s) |
|---|---|---|---|---|---|
| 2P-2L1ID-2L2-1M | 65.06 | 518.52 | 4.24 | 26.32 | 24.47 |
| 2P-2L1ID-1L2-1M | 60.23 | 480.85 | 4.16 | 23.5 | 24.22 |
| 4P-4L1ID-4L2-1M | 57.38 | 362.13 | 2.24 | 17.84 | 13.06 |
| 4P-4L1ID-1L2-1M | 52.77 | 336.54 | 2.19 | 15.91 | 14.08 |
| 4P-4L1ID-2L2-1M | 52.81 | 337.3 | 2.38 | 16.52 | 14.16 |

**Table 5**
Dual-core architecture evaluation ($T_y/T_z$) on SESC and $QT$ (our queueing theoretic model) based on the SPLASH-2 benchmarks. $T_y$ and $T_z$ denote the time to execute a benchmark on architecture $y$ = 2P-2L1ID-2L2-1M and $z$ = 2P-2L1ID-1L2-1M, respectively.

| Evaluation | FFT ($T_y/T_z$) | LU ($T_y/T_z$) | Radix ($T_y/T_z$) | Raytrace ($T_y/T_z$) | Water-spatial ($T_y/T_z$) |
|---|---|---|---|---|---|
| SESC | 1.08× | 1.08× | 0.935× | 1.166× | 1.01× |
| QT | 1.16× | 1.13× | 0.99× | 1.07× | 1.02× |
| % Difference | 7.4 % | 4.63 % | 5.88 % | 8.97 % | 0.99 % |

**Table 6**
Execution time and speedup comparison of our queueing theoretic models versus SESC. $T_Y^{x\text{-core}}$ denotes the execution time required for simulating an $x$-core architecture using $Y$ where $Y = \{SESC, QT\}$ ($QT$ denotes our queueing theoretic model).

| Benchmark | $T_{SESC}^{2\text{-core}}$ (s) | $T_{SESC}^{2\text{-core}}/T_{QT}^{2\text{-core}}$ | $T_{SESC}^{4\text{-core}}$ (s) | $T_{SESC}^{4\text{-core}}/T_{QT}^{4\text{-core}}$ |
|---|---|---|---|---|
| FFT | 1.7 | 885 | 1.4 | 168 |
| LU | 21 | 10,938 | 26.1 | 3,133 |
| Radix | 72.1 | 37,552 | 77.3 | 9,280 |
| Raytrace | 772.7 | 402,448 | 780.3 | 93,673 |
| Water-spatial | 927.35 | 482,995 | 998.4 | 119,856 |

in a benchmark suite that cover the entire range of processor-to-processor and processor-to-memory probabilities (e.g., $p_{PP}$ ranging from 0.05 for some benchmarks to 0.95 for others). The lack of computationally diverse benchmarks in terms of processor and memory requirements in a benchmark suite makes our queueing theoretic modeling approach an attractive solution for rigorous architectural evaluation because our modeling approach enables the architectural evaluation via synthetic benchmarks with virtually any computing requirements characterized probabilistically.

### 4.3. Speedup

To verify that our queueing theoretic modeling approach provides a quick architectural evaluation as compared to executing benchmarks on a multi-core simulator, we compare the execution time required to evaluate multi-core embedded architectures on SESC to our queueing theoretic model's execution time. The execution times for the SPLASH-2 benchmarks on SESC were measured using the Linux `time` command on an Intel Xeon E5430 processor running at 2.66 GHz. The execution times for our queueing theoretic models were measured using the Linux `time` command on an AMD Opteron 246 processor running at 2 GHz and these results were scaled to 2.66 GHz to provide a fair comparison. The SHARPE execution time was measured on the AMD Opteron processor due to site-specific server installations [10]. The results reveal that our queueing theoretic models require 1.92 ms and 8.33 ms on average for multi-core embedded architectures with two and four processor cores, respectively. We note that the execution times of our queueing theoretic models do not include the time taken to obtain processor, memory, and cache statistics (either from any prior work in the literature or running benchmarks on a multi-core simulator or a functional simulator if these statistics are not available in any prior work) as the time to gather these statistics can vary for different benchmarks and depends on the existing work in the literature and available functional simulators for different benchmarks. Table 6 depicts the average execution time for the SPLASH-2 benchmarks on SESC for multi-core embedded architectures with two and four processor cores and the ratio of the SESC execution times compared to our queueing theoretic models' execution times. Results reveal that our queueing theoretic models can provide architectural evaluation results 482, 995× faster as compared to executing benchmarks on SESC. Therefore, our queueing theoretic modeling approach can be used for quick architectural evaluation for multi-core embedded systems for virtually any set of workloads.

## 5. Queueing theoretic model insights

In this section, we present insights obtained from our queueing theoretic models regarding performance, performance per watt, and performance per unit area for the five different multi-core embedded architectures depicted in Table 1. Furthermore, we verify the insights/trends for various workloads with different computational requirements that cannot be captured by a subset of benchmarks in a benchmarks suite and corroborate these results with our presented trends (for brevity, we present a subset of the results, however, our analysis and derived conclusions are based on our complete set of experimental results).

### 5.1. Model setup

We consider the ARM7TDMI processor core, which is a 32-bit low-power processor with 32-bit instruction and data bus widths [1,2]. We consider the following cache parameters [38]: *cache sizes* of 8 KB, 8 KB, and 64 KB for the L1-I, L1-D, and L2 caches, respectively; *associativities* of direct-mapped, 2-way, and 2-way for the L1-I, L1-D, and L2 caches, respectively; and *block (line) sizes* of 64 B, 16 B, and 64 B for the L1-I, L1-D, and L2 caches, respectively. We assume a 32 MB MM for all architectures, which is typical for mobile embedded systems (e.g., Sharp Zaurus SL-5600 personal digital assistant (PDA)) [42]. To provide a fair comparison between architectures, we ensure that the total L2 cache size for the shared L2 cache architectures and the private L2 cache architectures is equal. Furthermore, the shared bus bandwidth for the shared LLC (L2 in our experiments) architectures is $n$ times the bandwidth of the private LLC architectures where $n$ is the number of cores sharing an LLC cache.

We implement our queueing network models of the multi-core embedded architectures using the SHARPE modeling tool/simulator [31]. Fig. 4 depicts the flow chart for our queueing network model setup in SHARPE. To set up our queueing network model simulation in SHARPE, we first specify the probabilities for a synthesized workload for a given multi-core architecture (Section 3). We then calculate the service rates for the service centers used in our multi-core queueing models. We assume that the processor core delivers 15 MIPS @ 16.8 MHz [1] (cycle time $= 1/(16.8 \times 10^6) =$ 59.524 ns), which for 32-bit instructions corresponds to a service rate of 480 Mbps. We assume L1-I, L1-D, and L2 cache, and MM access latencies of 2, 2, 10, and 100 cycles, respectively [1,14]. With an L1-I cache line size of 64 B, an access latency of 2 cycles, and a 32-bit (4 B) bus, transferring 64 B requires $64/4 = 16$ cycles, which results in a total L1-I time (cycles) = access time + transfer time $= 2 + 16 = 18$ cycles, with a corresponding L1-I service rate $= (64 \times 8)/(18 \times 59.524 \times 10^{-9}) = 477.86$ Mbps. With an L1-D cache line size of 16 B, the transfer time $= 16/4 = 4$ cycles, and the total L1-D time $= 2 + 4 = 6$ cycles, with a corresponding L1-D service rate $= (16 \times 8)/(6 \times 59.524 \times 10^{-9}) = 358.4$ Mbps. With an L2 cache line size of 64 B, the transfer time $= 64/4 = 16$ cycles, which gives the total L2 time $= 10 + 16 = 26$ cycles, with a corresponding L2 service rate $= (64 \times 8)/(26 \times 59.524 \times 10^{-9}) =$ 330.83 Mbps. With an MM line size of 64 B, the transfer time $= 64/4 = 16$ cycles, which gives a total MM time $= 100 + 16 =$ 116 cycles, with a corresponding service rate $= (64 \times 8)/(116 \times 59.524 \times 10^{-9}) = 74.15$ Mbps. We assume that each individual job/task requires processing 1 Mb of instruction and data, which is implicit in our queueing models via service rate specifications (ensures steady state/equilibrium behavior of the queueing network for our simulated workloads).

After service rate assignments for the architectural elements, we create a multi-chain product form the queueing network that outlines the architectural elements in each chain along with the associated transition probabilities between the architectural elements. Next, we specify the appropriate scheduling disciplines for the architectural elements (Section 3), such as FCFS scheduling for the processor core, L1-I, L1-D, and L2, and PS for MM. We specify the number of jobs for each chain in the queueing network depending on the workloads (parallelized tasks in a workload or multi-programmed workloads). To simulate a particular (single) benchmark in SHARPE, we set the number of jobs equal to 1 in our queueing network models. Finally, we compute statistics

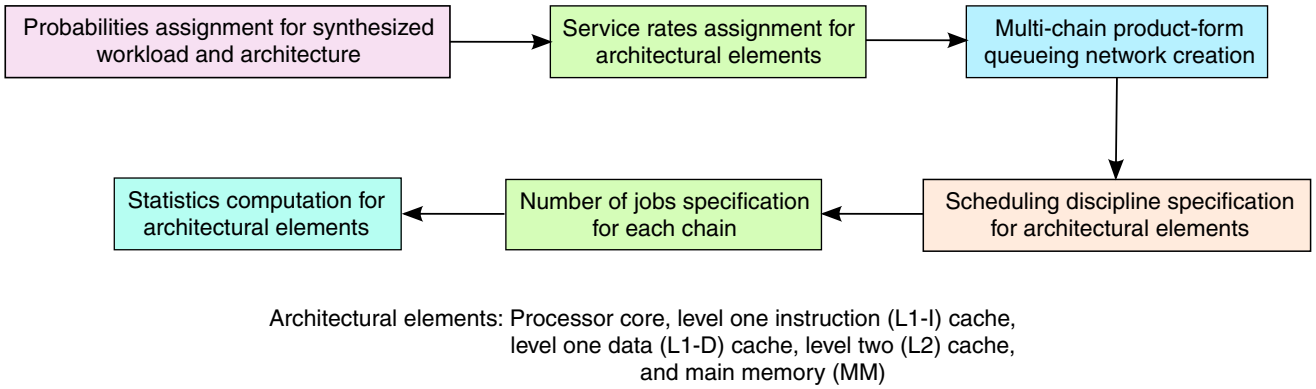


**Fig. 4.** Flow chart for our queueing network model setup in SHARPE.

(e.g., queue length, throughput, utilization, response time) from the queueing network models for a given number of jobs for the architectural elements. The code snippets for implementing the flow chart for our queueing network model in SHARPE are presented in Appendix A.

### 5.2. The effects of workloads on performance

In this subsection, we present the results describing the effects of different workloads on the response time and throughput performance metrics when the L1-1, L1-D, and L2 cache miss rates are held constant. We discuss the effects of varying the *computing requirements* of these workloads. The computing requirement of a workload signifies the workload's demand for processor resources, which depends on the percentage of arithmetic, logic, and control instructions in the workload relative to the load and store instructions. The computing requirements of the workloads are captured by $p_{PP}$ and $p_{PM}$ in our models.

The results show that the response time for the processor core, L1-I, and L1-D for 2P-2L1ID-1L2-1M is very close (within 7%) to 2P-2L1ID-2L2-1M as the computing requirements of the processor-bound workload vary. However, 2P-2L1ID-1L2-1M provides a 21.5% improvement in L2 response time and a 12.3% improvement in MM response time as compared to 2P-2L1ID-2L2-1M when $p_{PP} = 0.7$ and a 23.6% improvement in L2 response time and a 1.4% improvement in MM response time when $p_{PP} = 0.95$ and $N = 5$. 4P-4L1ID-2L2-1M provides a 22.3% improvement in L2 response time and a 13% improvement in MM response time over 4P-4L1ID-4L2-1M when $p_{PP} = 0.7$ and $N = 5$. 4P-4L1ID-2L2-1M provides a 22.3% improvement in L2 response time and a 3% improvement in MM response time as compared to 4P-4L1ID-4L2-1M when $p_{PP} = 0.95$ because a larger $p_{PP}$ results in fewer MM references. 4P-4L1ID-1L2-1M provides a 7.4% improvement in L2 response time with a 5.2% degradation in MM response time as compared to 4P-4L1ID-2L2-1M when $p_{PP} = 0.7$ and $N = 5$. 4P-4L1ID-1L2-1M provides a 12.4% improvement in L2 response time with no degradation in MM response time over 4P-4L1ID-2L2-1M when $p_{PP} = 0.95$ and $N = 5$. These results indicate that the shared LLCs provide more improvement in L2 response time as compared to the hybrid and private LLCs for more compute-intensive processor-bound workloads. However, the hybrid LLCs provide better MM response time than shared LLCs for more compute-intensive processor-bound workloads. These results suggest that hybrid LLCs may be more suitable than shared LLCs in terms of scalability and overall response time for comparatively less compute-intensive processor-bound workloads.

Response time and throughput results reveal that memory subsystems for an architecture with private, shared, or hybrid LLCs have a profound impact on the response time of the architecture and throughputs for the L2 and MM with relatively little impact on throughput for the processor cores, L1-I, and L1-D.

**Table 7**
Area and power consumption of architectural elements for two-core embedded architectures.

| Element | 2P-2L1ID-2L2-1M | | 2P-2L1ID-1L2-1M | |
|---|---|---|---|---|
| | Area ($mm^2$) | Power (mW) | Area ($mm^2$) | Power (mW) |
| Core | 0.065 | 2.016 | 0.065 | 2.016 |
| L1-I | 0.11 | 135.44 | 0.11 | 135.44 |
| L1-D | 0.0998 | 79.76 | 0.0998 | 79.76 |
| L2 | 0.578 | 307.68 | 0.5075 | 253.283 |
| MM | 34.22 | 3174.12 | 34.22 | 3174.12 |

The effects of different cache miss rates on performance for synthesized workloads are presented in Appendix B. Further insights obtained from our queueing theoretic models on the effects of workloads on response time and throughput are presented in Appendix C.

### 5.3. Performance per watt and performance per unit area computations

In this subsection, we compute performance per watt and performance per unit area for the multi-core embedded architectures using our queueing theoretic models. The performance per unit area is an important metric for embedded systems where the entire system is constrained to a limited space, however, performance per unit area is less important for desktop and supercomputing. Our performance per watt and performance per unit area computations assist in relative comparisons between different multi-core embedded architectures. For these computations, we first need to calculate the area and worst-case (peak) power consumption for different multi-core embedded architectures, which we obtain using CACTI 6.5 [6], International Technology Roadmap for Semiconductors (ITRS) specifications [17], and datasheets for multi-core embedded architectures.

Tables 7 and 8 show the area and peak power consumption for the processor cores, L1-I, L1-D, L2, and MM for two-core and four-core embedded architectures, respectively, assuming a 45 nm process. The core areas are calculated using Moore's law and the ITRS specifications [17] (i.e., the chip area required for the same number of transistors reduces by approximately $1/2\times$ every technology node (process) generation). For example, ARM7TDMI core area is 0.26 $mm^2$ at 130 nm process [3], the core area at 45 nm process (after three technology node generations, i.e., 130 nm, 90 nm, 65 nm, 45 nm) is approximately $(1/2)^3 \times 0.26 = 0.0325$ $mm^2$.

To illustrate our area and power calculation procedure that can be combined with the results obtained from our queueing theoretic models to obtain the performance per unit area and the

**Table 8**
Area and power consumption of architectural elements for four-core embedded architectures.

| Element | 4P-4L1ID-4L2-1M | | 4P-4L1ID-1L2-1M | | 4P-4L1ID-2L2-1M | |
|---|---|---|---|---|---|---|
| | Area ($mm^2$) | Power (mW) | Area ($mm^2$) | Power (mW) | Area ($mm^2$) | Power (mW) |
| Core | 0.13 | 4.032 | 0.13 | 4.032 | 0.13 | 4.032 |
| L1-I | 0.2212 | 270.88 | 0.2212 | 270.88 | 0.2212 | 270.88 |
| L1-D | 0.1996 | 159.52 | 0.1996 | 159.52 | 0.1996 | 159.52 |
| L2 | 1.1556 | 615.36 | 0.9366 | 354.04 | 1.015 | 506.8 |
| MM | 34.22 | 3174.12 | 34.22 | 3174.12 | 34.22 | 3174.12 |

performance per watt, we provide the example area and power calculations for 2P-2L1ID-2L2-1M. The area calculations for 2P-2L1ID-2L2-1M are: total processor core area $= 2 \times 0.0325 = 0.065\ mm^2$; total L1-I cache area $= 2 \times (0.281878 \times 0.19619) = 2 \times 0.0553 = 0.11\ mm^2$ (CACTI provides cache height $\times$ width for individual caches (e.g., $0.281878 \times 0.19619$ for the L1-I cache)); total L1-D cache area $= 2 \times (0.209723 \times 0.23785) = 2 \times 0.0499 = 0.0998\ mm^2$; total L2 cache area $= 2 \times (0.45166 \times 0.639594) = 2 \times 0.2889 = 0.578\ mm^2$; total MM area $= 8.38777 \times 4.08034 = 34.22\ mm^2$. The power consumption of the caches and MM is the sum of the dynamic power and the leakage power. Since CACTI gives dynamic energy and access time, dynamic power is calculated as the ratio of the dynamic energy to the access time. For example, for the L1-I cache, the dynamic power $= 0.020362$ (nJ)/$0.358448$ (ns) $= 56.8$ mW and the leakage power $= 10.9229$ mW, which gives the L1-I power consumption $= 56.8 + 10.9229 = 67.72$ mW. For the MM dynamic power calculation, we calculate the average dynamic energy per read and write access $= (1.27955+1.26155)/2 = 1.27055$ mJ, which gives the dynamic power $= 1.27055$ (nJ)/$5.45309$ (ns) $= 233$ mW. The total power consumption for the MM $= 233 + 2941.12 = 3174.12$mW (2941.12 mW is the leakage power for the MM). The power consumption calculations for 2P-2L1ID-2L2-1M are: total processor core power consumption $= 2 \times 1.008 = 2.016$ mW; total L1-I cache power consumption $= 2 \times 67.72 = 135.44$ mW; total L1-D cache power consumption $= 2 \times 39.88 = 79.76$ mW; total L2 cache power consumption $= 2 \times 153.84 = 307.68$ mW.

The area and power results for multi-core embedded architectures show that the MM consumes the most area and power consumption followed by L2, L1-I, L1-D, and the processor core. We observe that the shared L2 caches for 2P-2L1ID-1L2-1M and 4P-4L1ID-1L2-1M require 14% and 24% less area and consume 21.5% and 74% less power as compared to the private L2 caches for 2P-2L1ID-2L2-1M and 4P-4L1ID-4L2-1M, respectively. The hybrid L2 caches for 4P-4L1ID-2L2-1M require 14% less area and consume 21.4% less power as compared to the private L2 caches for 4P-4L1ID-4L2-1M whereas the shared L2 cache for 4P-4L1ID-1L2-1M requires 8.7% less area and consumes 43% less power as compared to the hybrid L2 caches for 4P-4L1ID-2L2-1M. These results indicate that the power-efficiency of shared LLCs improves as the number of cores increases.

Table 9 shows the area and peak power consumption for different multi-core embedded architectures. Table 9 does not include the MM area and power consumption, which allows the results to isolate the area and peak power consumption of the processor cores and caches. This MM isolation from Table 9 enables deeper insights and a fair comparison for the embedded architectures since we assume an off-chip MM that has the same size and characteristics for all evaluated architectures. To illustrate the area and power calculations for multi-core embedded architectures, we provide area and power consumption calculations for 2P-2L1ID-2L2-1M as an example. We point out that these area and power consumption calculations use constituent area and power consumption calculations for the architectural elements in a multi-core embedded architecture. For 2P-2L1ID-2L2-1M, the total cache area $= 0.11 + 0.0998 + 0.578 = 0.7878\ mm^2$, which

**Table 9**
Area and power consumption for multi-core architectures.

| Architecture | Area ($mm^2$) | Power (mW) |
|---|---|---|
| 2P-2L1ID-2L2-1M | 0.8528 | 524.896 |
| 2P-2L1ID-1L2-1M | 0.7823 | 470.5 |
| 4P-4L1ID-4L2-1M | 1.7064 | 1049.79 |
| 4P-4L1ID-1L2-1M | 1.4874 | 788.472 |
| 4P-4L1ID-2L2-1M | 1.5658 | 941.232 |

gives an overall area (excluding the MM) $= 0.065 + 0.7878 = 0.8528\ mm^2$ ($0.065\ mm^2$ is the area for the processor cores as calculated above). For 2P-2L1ID-2L2-1M, the total cache power consumption $= 135.44 + 79.76 + 307.68 = 522.88$ mW, which gives an overall power consumption (excluding the MM) $= 2.016 + 522.88 = 524.896$ mW (2.016 mW is the power consumption for the processor cores).

The overall area and power consumption results for different multi-core embedded architectures (Table 9) show that 2P-2L1ID-2L2-1M requires 8.3% more on-chip area and consumes 10.4% more power as compared to 2P-2L1ID-1L2-1M. 4P-4L1ID-4L2-1M requires 8.2% and 12.8% more on-chip area and consumes 10.3% and 24.9% more power as compared to 4P-4L1ID-2L2-1M and 4P-4L1ID-1L2-1M, respectively. These results reveal that the architectures with shared LLCs become more area and power efficient as compared to the architectures with private or hybrid LLCs as the number of cores in the architecture increases.

We discuss performance per watt and performance per unit area results for multi-core embedded architectures assuming 64-bit floating point operations. We observe that the performance per watt and performance per unit area delivered by the processor cores and the L1-I and L1-D caches for these architectures are very close (within 7%), however, the L2 cache presents interesting results. Although the MM performance per watt for these architectures also differs, this difference does not provide meaningful insights for the following two reasons: (1) the MM is typically off-chip and the performance per watt is more critical for on-chip architectural components than the off-chip components, and (2) if more requests are satisfied by the LLC, then fewer requests are deferred to the MM, which decreases the MM throughput and hence the performance per watt. Therefore, we mainly focus on the performance per watt and performance per unit area calculations for the LLCs for our studied architectures.

We calculate the performance per watt results for memory-bound workloads when the L1-I, L1-D, and L2 cache miss rates are 0.01, 0.13, and 0.3, respectively. The performance per watt values for the L2 caches are 2.42 MFLOPS/W and 3.1 MFLOPS/W and the performance per watt for the MM is 0.164 MFLOPS/W and 0.074 MFLOPS/W for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively, when $p_{PM} = 0.95$ and $N = 5$. Our performance per watt calculations for 2P-2L1ID-2L2-1M incorporate the aggregate throughput for the L2 cache, which is the sum of the throughputs for the two private L2 caches in 2P-2L1ID-2L2-1M. The performance per watt for the L2 caches drops to 2.02 MFLOPS/W and 2.53 MFLOPS/W whereas the performance per watt for the MM drops to 0.137 MFLOPS/W and 0.06 MFLOPS/W for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively, when $p_{PM} = 0.7$

and $N = 5$. The performance per watt values for the L2 caches are 2.21 MFLOPS/W, 4.77 MFLOPS/W, and 3.08 MFLOPS/W for 4P-4L1ID-4L2-1M, 4P-4L1ID-1L2-1M, and 4P-4L1ID-2L2-1M, respectively, when $p_{PM} = 0.95$ and $N = 10$. We observe similar trends for mixed workloads and processor-bound workloads but with comparatively lower performance per watt for the LLC caches because these workloads have comparatively lower $p_{PM}$ as compared to memory-bound workloads. The performance per watt for caches drops as $p_{PM}$ decreases because fewer requests are directed to the LLC caches for a low $p_{PM}$, which decreases the throughput and hence the performance per watt. These results indicate that the architectures with shared LLCs provide the highest LLC performance per watt followed by architectures with hybrid LLCs and private LLCs. The difference in performance per watt for these multi-core architectures is mainly due to the difference in the LLC power consumption as there is a relatively small difference in the throughput delivered by these architectures for the same workloads with identical cache miss rates.

Based on our experimental results for different cache miss rates and workloads, we determine the peak performance per watt for the LLCs for our studied multi-core embedded architectures. The peak performance per watt values for the L2 caches are 11.8 MFLOPS/W and 14.3 MFLOPS/W for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively, when the L1-I, L1-D, and L2 cache miss rates are all equal to 0.3, $p_{PM} = 0.9$, and $N = 20$. The peak performance per watt values for the L2 caches are 7.6 MFLOPS/W, 9.2 MFLOPS/W, and 13.8 MFLOPS/W for 4P-4L1ID-4L2-1M, 4P-4L1ID-2L2-1M, and 4P-4L1ID-1L2-1M, respectively, when the L1-I, L1-D, and L2 cache miss rates are all equal to 0.2, $p_{PM} = 0.9$, and $N = 20$. Results reveal that these architectures deliver peak LLC performance per watt for workloads with mid-range cache miss rates (e.g., miss rates of 0.2 or 0.3) because at higher cache miss rates, a larger number of requests are directed towards the LLCs, which causes the LLCs utilization to be close to one, which results in an increased response time and decreased throughput.

To investigate the effects of cache miss rates on the performance per watt of the LLCs, we calculate the performance for watt for memory-bound workloads ($p_{PM} = 0.9$) at high cache miss rates: L1-1 = 0.5, L1-D = 0.7, and L2 = 0.7, and when $N = 10$. The performance per watt values for the L2 caches are 5.4 MFLOPS/W and 6.55 MFLOPS/W for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively, whereas the performance per watt values are 2.7 MFLOPS/W, 3.28 MFLOPS/W, and 4.69 MFLOPS/W for 4P-4L1ID-4L2-1M, 4P-4L1ID-2L2-1M, and 4P-4L1ID-2L2-1M, respectively. The results reveal that at high cache miss rates, the performance per watt of the LLCs increases because relatively more requests are directed to the LLCs at higher cache miss rates than lower cache miss rates, which increases the throughput, and hence the performance per watt.

We calculate the performance per unit area results for memory-bound workloads when the L1-I, L1-D, and L2 cache miss rates are 0.01, 0.13, and 0.3, respectively. The performance per unit area values for the L2 caches are 1.29 MFLOPS/mm$^2$ and 1.54 MFLOPS/mm$^2$ and the performance per unit area values for the MM are 15.25 KFLOPS/mm$^2$ and 6.9 KFLOPS/mm$^2$ for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively, when $p_{PM} = 0.95$ and $N = 5$. The performance per unit area values for the L2 caches drop to 1.08 MFLOPS/mm$^2$ and 1.26 MFLOPS/mm$^2$ whereas the performance per unit area values for the MM drop to 12.68 KFLOPS/mm$^2$ and 5.6 KFLOPS/mm$^2$ for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively, when $p_{PM} = 0.7$ and $N = 5$. The performance per unit area values for the L2 caches are 1.18 MFLOPS/mm$^2$, 1.8 MFLOPS/mm$^2$, and 1.54 MFLOPS/mm$^2$ for 4P-4L1ID-4L2-1M, 4P-4L1ID-1L2-1M, and 4P-4L1ID-2L2-1M, respectively, when $p_{PM} = 0.95$ and $N = 10$. We observe similar trends for performance per unit area for mixed and processor-bound workloads as for the performance per watt trends explained above. These results indicate that the architectures with shared LLCs provide the highest LLC performance per unit area followed by architectures with hybrid LLCs and private LLCs. The difference in performance per unit area for these multi-core architectures is mainly due to the difference in the LLC throughput as we ensure that the total LLC area occupied by a multi-core embedded architecture with a given number of cores remains close enough (a minor difference in the occupied area of the LLCs occurs for different multi-core embedded architectures due to practical implementation and fabrication constraints as determined by CACTI) to provide a fair comparison.

Based on our queueing theoretic models' results and area calculations, we determine the peak performance per unit area for the LLCs for our studied multi-core embedded architectures. The peak performance per unit area values for the L2 caches are 6.27 MFLOPS/mm$^2$ and 7.14 MFLOPS/mm$^2$ for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M, respectively, when the L1-I, L1-D, and L2 cache miss rates are all equal to 0.3, $p_{PM} = 0.9$, and $N = 20$. The peak performance per unit area values for the L2 caches are 4.04 MFLOPS/mm$^2$, 4.6 MFLOPS/mm$^2$, and 5.22 MFLOPS/mm$^2$ for 4P-4L1ID-4L2-1M, 4P-4L1ID-2L2-1M, and 4P-4L1ID-1L2-1M, respectively, when the L1-I, L1-D, and L2 cache miss rates are all equal to 0.2, $p_{PM} = 0.9$, and $N = 20$. Other results for peak performance per unit area reveal similar trends as for the peak performance per watt trends and therefore omit these discussions for brevity.

## 6. Conclusions and future work

In this paper, we developed closed product-form queueing network models for performance evaluation of multi-core embedded architectures for different workload characteristics. The simulation results for the SPLASH-2 benchmarks executing on the SESC simulator (an architecture-level cycle-accurate simulator) verified the architectural evaluation insights obtained from our queueing theoretic models. Results revealed that our queueing theoretic model qualitatively evaluated multi-core architectures accurately with an average difference of 5.6% as compared to the architectures' evaluations from the SESC simulator. The performance evaluation results indicated that the architectures with shared LLCs provided better cache response time and MFLOPS/W than the private LLCs for all cache miss rates especially as the number of cores increases. The results also revealed the disadvantage of shared LLCs indicating that the shared LLCs are more likely to cause a main memory response time bottleneck for larger cache miss rates as compared to the private LLCs. The memory bottleneck caused by shared LLCs may lead to increased response time for processor cores because of stalling or idle waiting. However, the results indicated that the main memory bottleneck created by shared LLCs can be mitigated by using a hybrid of private and shared LLCs (i.e., sharing LLCs by a fewer number of cores) though hybrid LLCs consume more power than the shared LLCs and deliver comparatively less MFLOPS/W. The performance per watt and performance per unit area results for the multi-core embedded architectures revealed that the multi-core architectures with shared LLCs become more area and power efficient as compared to the architectures with private LLCs as the number of processor cores in the architecture increases.

In our future work, we plan to enhance our queueing theoretic models for performance evaluation of heterogeneous multi-core embedded architectures.

## Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the National Science Foundation (NSF) (CNS-0953447 and CNS-0905308), and the Office

of Naval Research (ONR R16480). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSERC, the NSF, and the ONR. Furthermore, the views expressed are those of the author(s) and do not reflect the official policy or position of the Department of Defense or the US Government.

## Appendix A. Queueing network model setup in SHARPE

Our queueing network model setup in SHARPE requires several steps, as depicted in the flow chart (Fig. 4). This section presents code snippets for implementing the flow chart for our queueing network model in SHARPE. To set up our queueing network model simulation in SHARPE, we first specify the probabilities for a synthesized workload for a given multi-core architecture. For example, for 2P-2L1ID-2L2-1M for memory-bound workloads (processor-to-processor probability $p_{PP} = 0.1$, processor-to-memory probability $p_{PM} = 0.9$) assuming that the L1-I, L1-D, and L2 cache miss rates are 25%, 50%, and 30%, respectively, and L1-I and L1-D access ratios are 0.8 and 0.2, respectively, the probabilities are set as: $p_{c_1P_1P_1} = 0.1$, $p_{c_1P_1L1I} = 0.72$, $p_{c_1P_1L1D} = 0.18$, $p_{c_1L1IP_1} = 0.75$, $p_{c_1L1DP_1} = 0.5$, $p_{c_1L1IL2} = 0.25$, $p_{c_1L1DL2} = 0.5$, $p_{c_1L2P_1} = 0.7$, $p_{c_1L2M} = 0.3$, $p_{c_1MP_1} = 1$ (different probabilities can be assigned for processor-bound or mixed workloads). The code snippet for assigning these probabilities in SHARPE for the two chains is:

```
bind
pr1P1P1 0.1
pr1P1L1I 0.72
pr1L1IP1 0.75
pr1L1IL2 0.25
pr1P1L1D 0.18
pr1L1DP1 0.5
pr1L1DL2 0.5
pr1L2P1 0.7
pr1L2M 0.3

pr2P2P2 0.1
pr2P2L1I 0.72
pr2L1IP2 0.75
pr2L1IL2 0.25
pr2P2L1D 0.18
pr2L1DP2 0.5
pr2L1DL2 0.5
pr2L2P2 0.7
pr2L2M 0.7
```

Next, we assign the service rates for the architectural elements (processor cores, caches, and MM) in Mbps.

```
P1rate 480
P2rate 480
L1Irate 477.86
L1Drate 358.4
L2rate 330.83
// Main memory service rate for chain 1
sMrate1 74.15
// Main memory service rate for chain 2
sMrate2 74.15
```

After assigning the service rates, we specify the architectural elements in each chain of the multi-chain product form queueing network with associated transition probabilities between the architectural elements. The queueing network performs calculations for a given number of jobs `NumJobs`.

```
mpfqn embarch1(NumJobs)

chain 1
P1 1L1I pr1P1L1I
1L1I 1L2 pr1L1IL2
1L1I P1 pr1L1IP1
P1 1L1D pr1P1L1D
1L1D 1L2 pr1L1DL2
1L1D P1 pr1L1DP1
1L2 P1 pr1L2P1
1L2 M pr1L2M
M P1 1
end

chain 2
P2 2L1I pr2P2L1I
2L1I 2L2 pr2L1IL2
2L1I P2 pr2L1IP2
P2 2L1D pr2P2L1D
2L1D 2L2 pr2L1DL2
2L1D P2 pr2L1DP2
2L2 P2 pr2L2P2
2L2 M pr2L2M
M P2 1
end

end
```

In the next step, the scheduling discipline is specified for the architectural elements along with the associated service rates for these elements. For example, FCFS scheduling for the processor core, L1-I, L1-D, and L2, and PS for MM.

```
P1 fcfs P1rate
end
P2 fcfs P2rate
end

1L1I fcfs L1Irate
end
1L1D fcfs L1Drate
end
2L1I fcfs L1Irate
end
2L1D fcfs L1Drate
end
1L2 fcfs L2rate
end
2L2 fcfs L2rate
end

M ps
1 sMrate1
2 sMrate2
end
```

Next we specify the number of jobs for each chain. For a balanced workload, the number of jobs should be divided equally between the two chains.

```
1 NumJobs/2
2 NumJobs/2
end
```

Finally, we compute statistics (e.g., queue length, throughput, utilization, response time) from the queueing network model for a given number of jobs for the architectural elements. The SHARPE code snippet below calculates these statistics for the processor core in chain 1. The code loops through the number of jobs varying from five to twenty, incrementing by five jobs in each iteration.

```
loop NumJobs,5,20,5
   expr mqlength(embarch1,P1;NumJobs)
   expr mtput(embarch1,P1;NumJobs)
   expr mutil(embarch1,P1;NumJobs)
   expr mrtime(embarch1,P1;NumJobs)
end
```

## Appendix B. The effects of cache miss rates on performance

In this subsection, we present the results describing the effects of different L1-I, L1-D, and L2 cache miss rates on the architecture response time and throughput performance metrics for mixed, processor-bound, and memory-bound workloads. Considering the effects of different cache miss rates is an important aspect of performance evaluation of multi-core embedded architectures with shared resources because cache miss rates give an indication whether the threads (corresponding to tasks) are likely to experience cache contention. Threads with higher LLC miss rates are more likely to have large working sets since each miss results in the allocation of a new cache line. These working sets may suffer from contention because threads may repeatedly evict the other threads' data (i.e., *cache thrashing*) [12]. We obtained the results for cache miss rates of 0.0001, 0.05, and 0.2, up to 0.5, 0.7, and 0.7 for the L1-I, L1-D, and L2 caches, respectively. These cache miss rate ranges represent typical multi-core embedded systems for a wide diversity of workloads [26,9,19].

Fig. 5 depicts the effects of cache miss rate on response time for mixed workloads ($p_{PP} = 0.5$, $p_{PM} = 0.5$) for 2P-2L1ID-2L2-1M as the number of jobs $N$ varies. We observe that the MM response time increases by 314% as miss rates for the L1-1, L1-D, and L2 caches increases from 0.0001, 0.05, and 0.2, to 0.5, 0.7, and 0.7, respectively, when $N = 5$. This response time increase is explained by the fact that as the cache miss rate increases, the number of accesses to MM increases, which increases the queue length and MM utilization, which causes an increase in the MM response time. The L1-I and L1-D response times decrease by 14% and 18%, respectively, as the miss rates for the L1-1 and L1-D caches increase from 0.0001 and 0.05, respectively, to 0.5 and 0.7, respectively, when $N = 5$. This decrease in response time occurs because increased miss rates decrease the L1-1 and L1-D queue lengths and utilizations. The L2 cache response time increases by 12% as the miss rates for the L1-1 and L1-D caches increase from 0.0001 and 0.05, respectively, to 0.5 and 0.7, respectively, when $N = 5$ (even though the L2 cache miss rate also increases from 0.2 to 0.7 but increased L1-I and L1-D miss rates effectively increase the number of L2 cache references, which increases the L2 cache queue length and utilization and thus L2 cache response time).

We observed that for mixed workloads ($p_{PP} = 0.5$, $p_{PM} = 0.5$), the response times for the processor core, L1-I, L1-D, and MM for 2P-2L1ID-1L2-1M are very close to the response times for 2P-2L1ID-2L2-1M, however, the L2 response time presents interesting differences. The L2 response time for 2P-2L1ID-1L2-1M is 22.3% less than the L2 response time for 2P-2L1ID-2L2-1M when the L1-1, L1-D, and L2 cache miss rates are 0.0001, 0.05, and 0.2, respectively, and $N = 5$ (similar percentage differences were observed for other values of $N$) whereas the L2 response time for 2P-2L1ID-1L2-1M is only 6.5% less than the L2 response time when the L1-1, L1-D, and L2 cache miss rates are 0.5, 0.7, and 0.7, respectively. This result shows that the shared L2 cache (of comparable area as the sum of the private L2 caches) performs better than the private L2 caches in terms of response time for small cache miss rates, however, the performance improvement decreases as the cache miss rate increases. Similar trends were observed for processor-bound ($p_{PP} = 0.9$, $p_{PM} = 0.1$) and memory-bound workloads ($p_{PP} = 0.1$, $p_{PM} = 0.9$).

For mixed workloads, the response time for the processor core, L1-I, L1-D, and MM for 4P-4L1ID-1L2-1M is 1.2×, 1×, 1.1×, and 2.4× greater than the corresponding architectural elements for 4P-4L1ID-4L2-1M whereas the L2 response time for 4P-4L1ID-1L2-1M is 1.1× less than the L2 response time for 4P-4L1ID-4L2-1M when the L1-1, L1-D, and L2 cache miss rates are 0.5, 0.7, and 0.7, respectively, and $N = 5$. This observation, in conjunction with our other experiments' results, reveals that the architectures with private LLCs provide improved response time for processor cores and L1 caches as compared to the architectures with shared LLCs, however, the response time of the LLC alone can be slightly better for architectures with shared LLCs because of the larger effective size for each core. The results also indicate that the MM response time could become a bottleneck for architectures with shared LLCs, especially when the cache miss rates become high. Another interesting observation is that shared LLCs could lead to increased response time for processor cores as compared to the private LLCs because of stalling or idle waiting of processor cores for bottlenecks caused by MM. Similar trends were observed for processor- and memory-bound workloads.

For mixed workloads, the L2 response time for 4P-4L1ID-2L2-1M is 1.2× less than 4P-4L1ID-4L2-1M and 1.1× greater than 4P-4L1ID-1L2-1M when the L1-1, L1-D, and L2 cache miss rates are 0.0001, 0.05, and 0.2, respectively, and $N = 5$. MM response time for 4P-4L1ID-2L2-1M is 2.3× less than 4P-4L1ID-1L2-1M whereas MM response time for 4P-4L1ID-2L2-1M and 4P-4L1ID-4L2-1M is the same when the L1-1, L1-D, and L2 cache miss rates are 0.5, 0.7, and 0.7, respectively, and $N = 5$. The response times for the processor core and L1-I/D are comparable for the three architectures (4P-4L1ID-4L2-1M, 4P-4L1ID-2L2-1M, and 4P-4L1ID-1L2-1M). These results, in conjunction with our other experiments' results, show that having LLCs shared by fewer cores (e.g., the L2 cache shared by two cores in our considered architecture) do not introduce MM as a response time bottleneck whereas the MM becomes the bottleneck as more cores share the LLCs, especially for large cache miss rates. Similar trends were observed for the processor- and memory-bound workloads.

We observe the effects of cache miss rates on throughput for various multi-core embedded architectures. For mixed workloads, the throughput for the processor core, L1-I, L1-D, and MM for 2P-2L1ID-1L2-1M is very close to the throughput for 2P-2L1ID-2L2-1M, however, L2 throughput for 2P-2L1ID-1L2-1M is 100% greater on average than the L2 throughput for 2P-2L1ID-2L2-1M for different miss rates for the L1-1, L1-D, and L2 and $N = 5$. However, the combined throughput of the two private L2 caches in 2P-2L1ID-2L2-1M is comparable to the L2 throughput for 2P-2L1ID-1L2-1M. This shows that the shared and private L2 caches provide comparable net throughputs for the two architectures. The throughput for the processor core, L1-I, L1-D, and L2 for 4P-4L1ID-4L2-1M is 2.1× less on average than the corresponding 2P-2L1ID-2L2-1M architectural elements whereas the throughput for the processor core, L1-1, L1-D, and L2 for the two architectures is the same when the miss rates for the L1-1, L1-D, and L2 caches are 0.5, 0.7, and 0.7, respectively, and $N = 5$. This indicates that the throughput for the individual architectural elements (except MM since MM is shared for both the architectures) decreases for the architecture with more cores since the workload remains the same. The throughput for the processor core, L1-I, L1-D, L2, and MM for 4P-4L1ID-2L2-1M is 1.5×, 1.5×, 1.5×, 2.5×, and 1.3× less than the throughput for the corresponding 4P-4L1ID-1L2-1M architectural elements when the miss rates for the L1-1, L1-D, and L2 caches are 0.0001, 0.05, and 0.2, respectively, and $N = 5$. These observations reveal that changing the L2 cache from private to shared can also impact the throughput for other architectural elements because of the interactions between these elements.

We evaluate the effects of cache miss rates on throughput for processor-bound workloads ($p_{PP} = 0.9$, $p_{PM} = 0.1$) for 2P-2L1ID-2L2-1M as $N$ varies. Results reveal that there is no appreciable increase in the processor core throughput as $N$ increases from 5 to 20 because the processors continue to operate at utilization close to 1 when the L1-1, L1-D, and L2 cache miss rates are 0.3, 0.3, and 0.3, respectively (similar trends were observed for other cache miss rates). The MM throughput increases by 4.67% (4.67% − 1.64% = 3.03% greater than the mixed workloads) as $N$ increases from 5 to 20 when L1-1, L1-D, and L2 cache miss rates are 0.5, 0.7, and

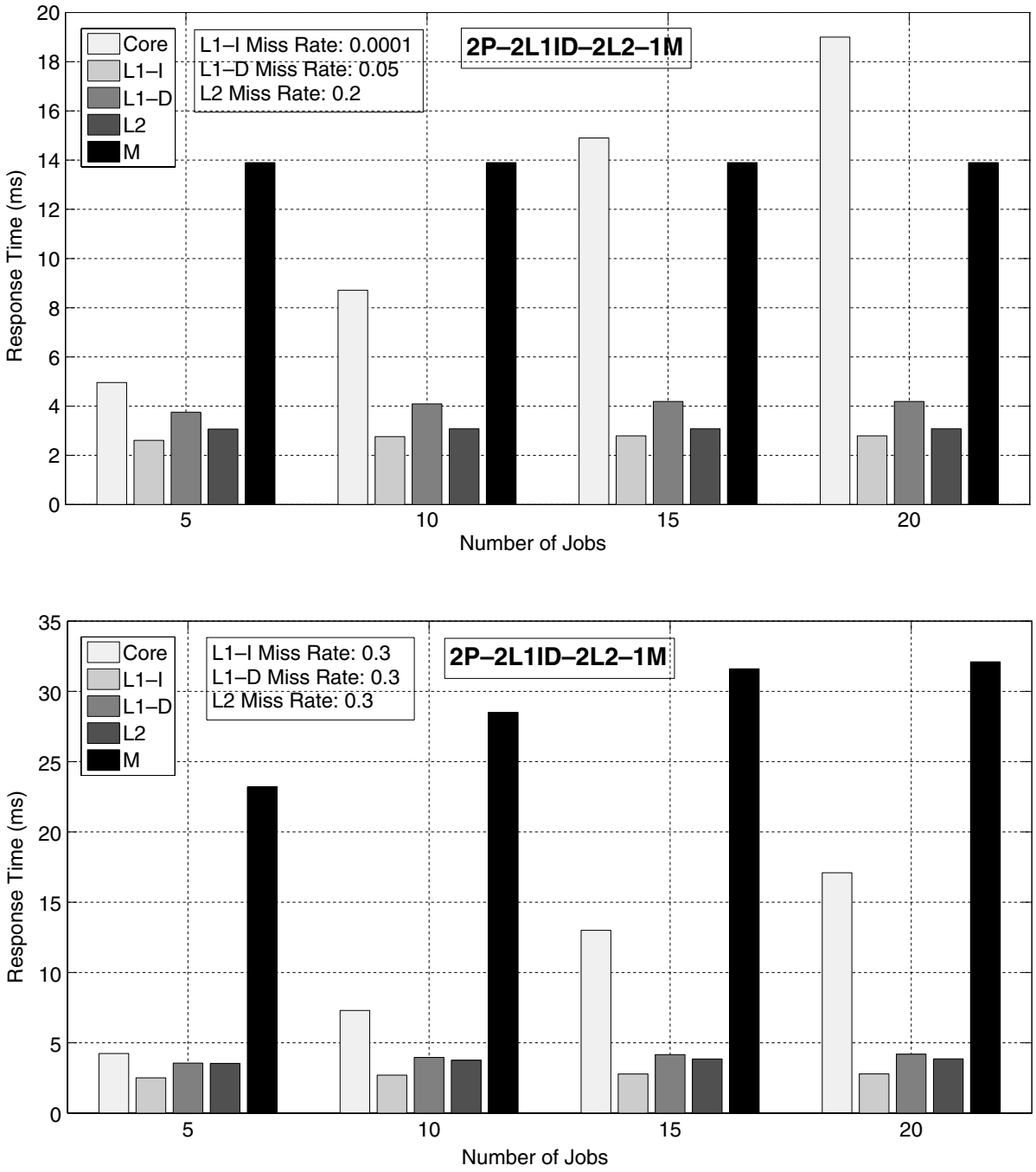


**Fig. 5.** The effects of cache miss rate on response time (ms) for mixed workloads for 2P-2L1ID-2L2-1M for a varying number of jobs $N$.

0.7, respectively. In this case, the MM percentage throughput increase is greater for processor-bound workloads as compared to mixed workloads because the MM is underutilized for processor-bound workloads (e.g., a utilization of 0.519 for processor-bound workloads as compared to a utilization of 0.985 for mixed workloads when $N = 5$). However, the MM absolute throughput for processor-bound workloads is less than the mixed workloads (e.g., an MM throughput of 38.5 Mbps for processor-bound workloads as compared to an MM throughput of 73 Mbps for mixed workloads when $N = 5$). For the processor-bound workloads, the throughput for the processor core, L1-I, L1-D, and MM for 2P-2L1ID-1L2-1M is similar to the throughput for 2P-2L1ID-2L2-1M, however, the L2 throughput for 2P-2L1ID-1L2-1M is 100% greater than the L2 throughput for 2P-2L1ID-2L2-1M for all cache miss rates on average and $N = 5$. Similar trends were observed for the memory-bound and mixed workloads for the architectures with two or four cores with private and shared LLCs (these throughput trends would continue as the number of cores increases).

## Appendix C. The effects of workloads on performance

Fig. 6 depicts the effects of varying computing requirements for processor-bound workloads on response time for 2P-2L1ID-2L2-1M as $N$ varies where the L1-I, L1-D, and L2 cache miss rates are 0.01, 0.13, and 0.3, respectively. The figure depicts that as $N$ increases, the response time for the processor core, L1-I, L1-D, L2, and MM increases for all values of $p_{pp}$ and $p_{PM}$. The figure shows that as $p_{pp}$ increases, the response time of the processor increases whereas the response times of L1-I, L1-D, L2, and MM show negligible effects due to the processor-bound nature of the

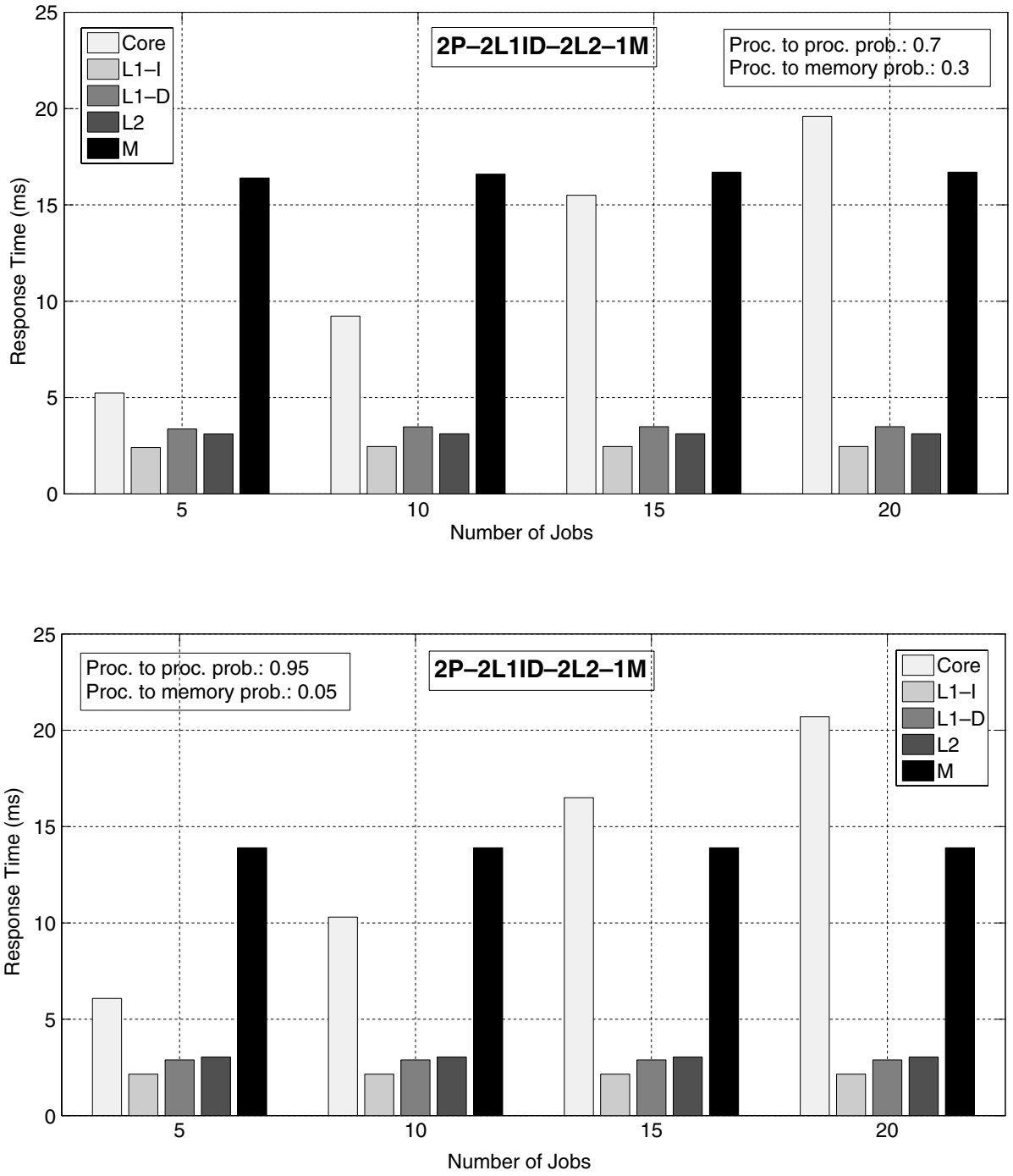


**Fig. 6.** The effects of processor-bound workloads on response time (ms) for 2P-2L1ID-2L2-1M for a varying number of jobs $N$ for cache miss rates: L1-I = 0.01, L1-D = 0.13, and L2 = 0.3.

workloads. For example, the processor response time increases by 19.8% as $p_{PP}$ increases from 0.7 to 0.95 when $N = 5$. The response times of L1-I, L1-D, L2, and MM decrease by 10.8%, 14.2%, 2.2%, and 15.2%, respectively, as $p_{PP}$ increases from 0.7 to 0.95 when $N = 5$ because an increase in $p_{PP}$ results in a decrease in memory requests, which decreases the response time for the caches and MM.

For memory-bound workloads, 2P-2L1ID-1L2-1M provides a 16.7% improvement in L2 response time and a 31.5% improvement in MM response time as compared to 2P-2L1ID-2L2-1M when $p_{PM} = 0.95$ and $N = 5$. 2P-2L1ID-1L2-1M provides an 18.2% improvement in L2 response time and a 25.8% improvement in MM response time over 2P-2L1ID-2L2-1M when $p_{PM} = 0.7$ and $N = 5$. 4P-4L1ID-2L2-1M provides a 19.8% improvement in L2 response time and a 20.2% improvement in MM response time on average over 4P-4L1ID-4L2-1M for both $p_{PM} = 0.95$ and $p_{PM} = 0.7$ and $N = 5$. 4P-4L1ID-1L2-1M provides a 2.4% improvement in L2 response time with a 15% degradation in MM response time as compared to 4P-4L1ID-2L2-1M when $p_{PM} = 0.95$ and $N = 5$. 4P-4L1ID-1L2-1M provides no improvement in L2 response time, with an 11.5% degradation in MM response time as compared to 4P-4L1ID-2L2-1M when $p_{PM} = 0.7$ and $N = 5$. These results indicate that the shared LLCs provide a larger improvement in L2 and MM response time as compared to private LLCs for memory-bound workloads. Furthermore, the hybrid LLCs are more amenable in terms of response time as compared to the shared and private LLCs for memory-bound workloads. Similar trends were observed for

the mixed workloads for architectures with two or four cores containing private, shared, or hybrid LLCs.

We observe the effects of varying computing requirements for processor-bound workloads on throughput for 2P-2L1ID-2L2-1M as $N$ varies. As $N$ increases, the throughputs for the processor core, L1-I, L1-D, L2, and MM increase for all values of $p_{PP}$ and $p_{PM}$. Furthermore, as $p_{PP}$ increases, the throughput of the processor core increases whereas the throughputs of L1-I, L1-D, L2, and MM decrease because of relatively fewer memory requests. For memory-bound workloads, L1-I and L1-D throughputs for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M are comparable, however, 2P-2L1ID-1L2-1M improves the L2 throughput by 106.5% and 111% (due to larger combined L2 cache) whereas the MM throughput decreases by 126% and 121.2% when $p_{PM}$ is 0.7 and 0.95, respectively. For memory-bound workloads, 2P-2L1ID-1L2-1M provides a 5.3% and 3.4% improvement in the processor core throughput over 2P-2L1ID-2L2-1M when $p_{PM} = 0.95$ and $p_{PM} = 0.7$, respectively, and $N = 5$. For processor-bound workloads, the processor core throughputs for 2P-2L1ID-2L2-1M and 2P-2L1ID-1L2-1M are comparable. Similar trends were observed for the architectures with four cores containing private, shared, or hybrid LLCs since the processor cores operate close to saturation (at high utilization) for processor-bound workloads, and memory stalls due to memory subsystem response time have a negligible effect on the processor core performance as memory accesses are completely overlapped with computation.

## References


[1] ARM7TDMI, ATMEL Embedded RISC Microcontroller Core: ARM7TDMI, November 2010. [Online]. Available: http://www.atmel.com/.

[2] ARM7TDMI, ARM7TDMI Data Sheet, November 2010. [Online]. Available: http://www.atmel.com/.

[3] ARM, ARM7 Thumb Family, January 2011. [Online]. Available: http://saluc.engr.uconn.edu/refs/processors/arm/arm7_family.pdf.

[4] J. Balfour, Efficient embedded computing, Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, May 2010.

[5] D. Benítez, J. Moure, D. Rexachs, E. Luque, Adaptive L2 cache for chip multiprocessors, in: Proc. of ACM International European Conference on Parallel and Distributed Computing, Euro-Par, Rennes, France, August 2007.

[6] CACTI, An Integrated Cache and Memory Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model, November 2010. [Online]. Available: http://www.hpl.hp.com/research/cacti/.

[7] D. Chandra, F. Guo, S. Kim, Y. Solihin, Predicting inter-thread cache contention on a chip multi-processor architecture, in: Proc. of the 11th International Symposium on High-Performance Computer Architecture, HPCA-11, San Francisco, California, February 2005.

[8] X.E. Chen, T.M. Aamodt, Modeling cache contention and throughput of multiprogrammed manycore processors, IEEE Trans. Comput. (99) (2011).

[9] Y. Chen, E. Li, J. Li, Y. Zhang, Accelerating video feature extractions in CBVIR on multi-core systems, Intel Technol. J. 11 (4) (2007) 349–360.

[10] CHREC, NSF Center for High-Performance Reconfigurable Computing, September 2011. [Online]. Available: http://www.chrec.org/.

[11] D. Culler, J. Singh, A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publishers, Inc., 1999.

[12] A. Fedorova, S. Blagodurov, S. Zhuravlev, Managing contention for shared resources on multicore processors, Commun. ACM 53 (2) (2010) 49–57.

[13] M.J. Flynn, Computer Architecture: Pipelined and Parallel Processor Design, Jones & Bartlett Learning, 1995.

[14] Freescale, Cache Latencies of the PowerPC MPC7451, January 2011. [Online]. Available: http://cache.freescale.com/files/32bit/doc/app_note/AN2180.pdf.

[15] Intel, Dual-Core Intel Xeon Processors LV and ULV for Embedded Computing, March 2011. [Online]. Available: ftp://download.intel.com/design/intarch/prodbref/31578602.pdf.

[16] E. Ïpek, S.A. McKee, B. Supinski, M. Schulz, R. Caruana, Efficiently exploring architectural design spaces via predictive modeling, in: Proc. of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS-XII, San Jose, California, October 2006.

[17] ITRS, International Technology Roadmap for Semiconductors, January 2011. [Online]. Available: http://www.itrs.net/.

[18] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, 1991.

[19] P. Jain, Software-assisted cache mechanisms for embedded systems, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February 2008.

[20] T.S. Karkhanis, J.E. Smith, A first-order superscalar processor model, in: Proc. of the 31st Annual International Symposium on Computer Architecture, ISCA'04, Munchen, Germany, June 2004.

[21] L. Kleinrock, Queueing Systems, Volume II: Computer Applications, Wiley-Interscience, 1976.

[22] R. Kumar, D. Tullsen, N. Jouppi, P. Ranganathan, Heterogeneous chip multiprocessors, IEEE Comput. 38 (11) (2005) 32–38.

[23] O. Kwon, H. Bahn, K. Koh, FARS: a page replacement algorithm for NAND flash memory based embedded systems, in: Proc. of IEEE CIT, Sydney, Australia, July 2008.

[24] V. Mainkar, K. Trivedi, Performance modeling using SHARPE, in: Proc. of the 8th Symposium on Reliability in Electronics, RELECTRONIC, Budapest, Hungary, August 1991.

[25] J. Medhi, Stochastic Models in Queueing Theory, Academic Press, 2003. An imprint of Elsevier Science.

[26] R. Min, W.-B. Jone, Y. Hu, Location cache: a low-power L2 cache system, in: Proc. of ACM International Symposium on Low Power Electronics and Design (ISLPED), Newport Beach, California, August 2004.

[27] S. Nussbaum, J.E. Smith, Modeling superscalar processors via statistical simulation, in: Proc. of the 2001 International Conference on Parallel Architectures and Compilation Techniques, PACT, Barcelona, Spain, September 2001.

[28] M. Reiser, S. Lavenberg, Mean value analysis of closed multi-chain queueing networks, J. ACM 27 (2) (1980) 313–322.

[29] J. Ruggiero, Measuring cache and memory latency and CPU to memory bandwidth, Intel White Paper, December 2008, pp. 1–14.

[30] M. Sabry, M. Ruggiero, P. Valle, Performance and energy trade-offs analysis of L2 On-chip cache architectures for embedded MPSoCs, in: Proc. of IEEE/ACM Great Lakes Symposium on VLSI, GLSVLSI, Providence, Rhode Island, USA, May 2010.

[31] R. Sahner, K. Trivedi, A. Puliafito, Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package, Kluwer Academic Publishers, 1996.

[32] N. Samari, G. Schneider, A queueing theory-based analytic model of a distributed computer network, IEEE Trans. Comput. C-29 (11) (1980) 994–1001.

[33] J. Savage, M. Zubair, A unified model for multicore architectures, in: Proc. of ACM International Forum on Next-generation Multicore/Manycore Technologies, IFMT, Cairo, Egypt, November 2008.

[34] SESC, SESC: SuperESCalar Simulator, September 2011. [Online]. Available: http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/.

[35] K. Sevcik, I. Mitrani, The distribution of queueing network states at input and output instants, J. ACM 28 (2) (1981) 358–371.

[36] L. Shi, et al. Write activity reduction on flash main memory via smart victim cache, in: Proc. of ACM GLSVLSI, Providence, Rhode Island, USA, May 2010.

[37] D.J. Sorin, V.S. Pai, S.V. Adve, M.K. Vernon, D.A. Wood, Analytic evaluation of shared-memory systems with ILP processors, in: Proc. of the 25th Annual International Symposium on Computer Architecture, ISCA'98, Barcelona, Spain, June 1998.

[38] TILERA, Tile Processor Architecture Overview, in: TILERA Official Documentation, Copyright 2006–2009 Tilera Corporation, November 2009.

[39] D.L. Willick, D.L. Eager, An analytic model of multistage interconnection networks, in: Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Boulder, Colorado, May 1990.

[40] S. Woo, M. Ohara, E. Torrie, J. Singh, A. Gupta, The SPLASH-2 programs: characterization and methodological considerations, in: Proc. of ACM ISCA, Santa Margherita Ligure, Italy, June 1995.

[41] R.E. Wunderlich, T.F. Wenisch, B. Falsafi, J.C. Hoe, SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling, in: Proc. of the 30th Annual International Symposium on Computer Architecture, ISCA, San Diego, California, June 2003.

[42] L. Yang, R. Dick, H. Lekatsas, S. Chakradhar, Online memory compression for embedded systems, ACM Trans. Embedded Comput. Syst. (TECS) 9 (3) (2010) 27:1–27:30.


**Arslan Munir** received his B.S. in Electrical Engineering from the University of Engineering and Technology (UET), Lahore, Pakistan, in 2004, and his M.A.Sc. degree in Electrical and Computer Engineering (ECE) from the University of British Columbia (UBC), Vancouver, Canada, in 2007. He received his Ph.D. degree in ECE from the University of Florida (UF), Gainesville, Florida, USA, in 2012. He is currently a postdoctoral research associate in the ECE department at Rice University, Houston, Texas, USA. From 2007 to 2008, he worked as a software development engineer at Mentor Graphics in the Embedded Systems Division. He was the recipient of many academic awards including the Gold Medals for the best performance in Electrical Engineering, academic Roll of Honor, and doctoral fellowship from Natural Sciences and Engineering Research Council of Canada (NSERC). He received a Best Paper award at the IARIA International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM) in 2010. His current research interests include embedded systems, cyber-physical/transportation systems, low-power design, computer architecture, multi-core platforms, parallel computing, dynamic optimizations, fault-tolerance, and computer networks.

**Ann Gordon-Ross** received her B.S and Ph.D. degrees in Computer Science and Engineering from the University of California, Riverside (USA) in 2000 and 2007, respectively. She is currently an Assistant Professor of Electrical and Computer Engineering at the University of Florida (USA) and is a member of the NSF Center for High Performance Reconfigurable Computing (CHREC) at the University of Florida. She is also the faculty advisor for the Women in Electrical and Computer Engineering (WECE) and the Phi Sigma Rho National Society for Women in Engineering and Engineering Technology. She received her CAREER award from the National Science Foundation in 2010 and Best Paper awards at the Great Lakes Symposium on VLSI (GLSVLSI) in 2010 and the IARIA International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM) in 2010. Her research interests include embedded systems, computer architecture, low-power design, reconfigurable computing, dynamic optimizations, hardware design, real-time systems, and multi-core platforms.

**Sanjay Ranka** is a Professor in the Department of Computer and Information Science and Engineering at the University of Florida, Gainesville, Florida, USA. His current research interests are energy-efficient computing, high-performance computing, data mining and informatics. Most recently he was the Chief Technology Officer at Paramark where he developed real-time optimization software for optimizing marketing campaigns. Sanjay has also held positions as a tenured faculty positions at Syracuse University and as a researcher/visitor at IBM T.J. Watson Research Labs and Hitachi America Limited.

Sanjay earned his Ph.D. (Computer Science) from the University of Minnesota and a B. Tech. in Computer Science from IIT, Kanpur, India. He has coauthored two books: Elements of Neural Networks (MIT Press) and Hypercube Algorithms (Springer Verlag), 75 journal articles and 125 refereed conference articles. His recent work has received a student best paper award at ACM-BCB 2010, best paper runner up award at KDD-2009, a nomination for the Robbins Prize for the best paper in the journal of Physics in Medicine and Biology for 2008, and a best paper award at ICN 2007.

He is a fellow of the IEEE and AAAS, and a member of IFIP Committee on System Modeling and Optimization. He is the associate Editor-in-Chief of the Journal of Parallel and Distributed Computing and an associate editor for IEEE Transactions on Parallel and Distributed Computing, IEEE Transactions on Computers, Sustainable Computing: Systems and Informatics, Knowledge and Information Systems, and International Journal of Computing.

**Farinaz Koushanfar** (S'99M'06) received the Ph.D. degree in electrical engineering and computer science and the M.A. degree in statistics, both from University of California Berkeley, in 2005, and the M.S. degree in electrical engineering from the University of California Los Angeles. She is currently an Associate Professor with the Department of Electrical and Computer Engineering, Rice University, Houston, TX, where she directs the Texas Instruments DSP Leadership University Program. Her research interests include adaptive and low power embedded systems design, hardware security, and design intellectual property protection.

Prof. Koushanfar is a recipient of the Presidential Early Career Award for Scientists and Engineers (PECASE), the ACM SIGDA Outstanding New Faculty Award, the National Academy of Science Kavli Foundation fellowship, the Army Research Office (ARO) Young Investigator Program Award, the Office of Naval Research (ONR) Young Investigator Program Award, the Defense Advanced Project Research Agency (DARPA) Young Faculty Award, the National Science Foundation CAREER Award, MIT Technology Review TR-35, an Intel Open Collaborative Research fellowship, and a best paper award at Mobicom.