Journal of Parallel and Distributed Computing 144 (2020) 172–188

J. Parallel Distrib. Comput.

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

Evaluation of memory performance in NUMA architectures using Stochastic Reward Nets

Reza Entezari-Maleki [a], Younghyun Cho [b], Bernhard Egger [b,*]

[a] School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
[b] Department of Computer Science and Engineering, Seoul National University, Seoul, Korea

ARTICLE INFO

Article history:
Received 13 March 2019
Received in revised form 3 April 2020
Accepted 28 May 2020
Available online 10 June 2020

Keywords:
NUMA architectures
Performance modeling
Stochastic Reward Nets
Approximate models

ABSTRACT

Understanding memory performance in multi-core platforms is a prerequisite to perform optimizations. To this end, this paper presents analytical models based on Stochastic Reward Nets (SRNs) to model and evaluate the memory performance of Non-Uniform Memory Access (NUMA) multi-core architectures. The approach considers the details of the architecture and first proposes a monolithic SRN model that evaluates the memory performance in terms of the mean memory response time. Since the monolithic model incurs a state space explosion with an increasing number of cores and memory controllers, two approximate models are presented that are able to evaluate large-scale NUMA architectures. The SRNs are validated through measurements on two NUMA multi-core platforms, a 64-core AMD Opteron server and a 72-core Intel system. The results demonstrate the ability of the proposed models to accurately compute the mean memory response time on NUMA architectures. The results also provide valuable information that runtime systems and application designers can use to optimize execution of parallel applications on such architectures.

1. Introduction

To accommodate the ever increasing need for computation power and large data workloads, multi-socket multi-core systems comprising several multi-core processors and memory controllers are being built. A processor interconnect enables the cores in different sockets, called CPU nodes, to access the different memory controllers, named memory nodes. Because of the varying access latency from an individual core to the different memory nodes, such systems are called Non-Uniform Memory Accesses (NUMA) architectures.

Performance analysis of memory accesses is of utmost importance in managing a large number of cores in multi-core systems because the memory system provides limited bandwidth. Based on information about memory performance, runtime systems constrain application parallelism [14,44] or manage memory bandwidth [27]. Although several models have been proposed to predict memory performance on Symmetric Multi-Processing (SMP) systems [24,45], NUMA architectures pose additional challenges since the performance of memory accesses is strongly affected by the asymmetric memory system. In such architectures contention can occur at multiple locations, which makes modeling NUMA systems a challenging task. Although analytical models based on queueing systems have been proposed to evaluate the performance of memory accesses in NUMA architectures [10,43], these models often fail to achieve a high prediction accuracy due to the simplifications made when modeling the relation between the interconnection network and the memory controllers.

This paper presents Stochastic Reward Nets (SRNs) [12,32] that model the memory system of NUMA architectures in detail. The models are able to evaluate and predict the memory performance of NUMA architectures for a varying number of cores and memory nodes. SRNs are extensions of Stochastic Petri Nets (SPNs) with the advantage that real systems can be specified and evaluated in an intuitive way. Known as a powerful modeling paradigm for performance, availability, and reliability analysis of computing systems, SRNs allow to automatically generate and solve large Markov reward models (MRMs). By applying SRNs to model memory accesses in NUMA architectures, the access latency of different CPU nodes to different memory controllers can be graphically modeled and analytically evaluated. SRNs and the reduction techniques defined on them enable modeling and evaluation of more complex architectures that cannot be easily analyzed with existing state–space models. In this work, we first present a monolithic model for common NUMA architectures. This monolithic model extends primitives that model memory accesses from one core of a CPU node to one memory node into a full system model. Although the monolithic model is helpful to understand memory performance of NUMA architectures, it suffers from a state space explosion at a relatively low number

* Corresponding author.
E-mail addresses: entezari@iust.ac.ir (R. Entezari-Maleki), younghyun@csap.snu.ac.kr (Y. Cho), bernhard@csap.snu.ac.kr (B. Egger).

https://doi.org/10.1016/j.jpdc.2020.05.022
Preprint

of CPU and memory nodes which makes it difficult to apply the model to existing or future architectures. To overcome this limitation, we present two approximate models with significantly less complexity based on the folding method [20] and the fixed-point iteration technique [40]. The folded model can be applied to existing NUMA machines comprising several CPU and memory nodes but still experiences a state space explosion for larger systems. The fixed-point model, finally, is able to analyze large-scale NUMA machines at the expense of a small reduction in accuracy.

The presented SRN models are validated through experiments on a 64-core AMD Opteron server with eight memory nodes. Machine-dependent model parameters, such as the data transfer rate of the interconnection links or the service rate of the memory controllers, are determined offline by executing synthetic workloads. The memory access pattern and the last-level cache (LLC) miss rate of an application are extracted by executing the application in isolation. The accuracy of the models is tested by comparing the computed mean memory response time with measurements on the AMD server and a state-of-the-art model based on queueing systems [10]. The results demonstrate that the presented SRNs are good candidates for practical performance evaluation of large-scale NUMA multi-core systems.

The remainder of this work is organized as follows. After a discussion of related work, the architecture of NUMA multi-core systems and its implications on performance modeling are given in Sections 2 and 3. Section 4 introduces Petri nets and the SRN formalism. Section 5 presents the stochastic models used to evaluate memory performance. Section 6 explains the application of the SRN models to a 64-core NUMA multi-core system, and Section 7 analyzes the results of this study. Section 8 applies the models to large-scale NUMA architectures and illustrates future research directions. Section 9, finally, concludes the paper.

2. Related work

A significant portion of research has focused on modeling memory performance of multi/ many-core systems. Kim [24] and Wang [45] present memory performance prediction techniques for SMP architectures. Hong [19] proposes an analytical performance model for GPUs that considers memory hierarchies in GPU architectures. Dao [16] uses a machine-learning based approach for GPU performance modeling. These proposals model memory performance for multi/many-cores by taking the hardware architecture and the performance characteristics of the memory systems into account, but do not consider NUMA architectures comprising multiple memory controllers and the processor interconnection network.

A number of approaches have been presented for the purpose of modeling and the evaluation of the performance of memory accesses in NUMA architectures. Several works apply queueing theory to model the performance of memory accesses in multi-processor systems [42]. Jonkers [22,23] proposes performance modeling methodologies for parallel applications. Based on a simulation of the hardware architecture, the presented approaches explore the effectiveness of queueing systems for multi-processors consisting of multiple memory servers and an interconnection network. Although such systems did not exist at the time of the publication, the simulated multi-processor systems exhibit almost the same behavior as recent NUMA multi-core architectures. Tudor [43] models memory contention in the memory controller of an SMP system using an M/M/1 queueing model. The memory performance for a varying number of CPU and memory nodes is predicted using linear regression. The approach provides a simple method for modeling memory performance, but does not properly model the processor interconnection network. Cho [10,11] presents online performance models for NUMA architectures. Separate M/M/1/N/N queueing systems are applied to each memory controller and interconnection link. Although the models can quickly predict the memory performance while applications are running, the separated queueing systems do not properly consider the relationship between the processor interconnection network and memory controllers. A comparison of the presented approach over [10] is given in Section 7.

Curtis-Maury [14] presents a runtime model for power and performance optimization of parallel applications in NUMA architectures. To predict both power and performance of parallel applications in dependence of the number of cores, the performance model applies linear regression using hardware performance counters queried at runtime. Lastovetsky [26] applies functional performance models to formulate the performance and energy consumption of parallel applications in NUMA architectures. Wang [44] proposes an Integer Programming-based method to model core allocation of parallel applications in NUMA architectures with the goal of maximizing memory bandwidth usage by assigning the appropriate number of cores. Cho [9] manages the thread count of co-located parallel applications based on CPU and memory bandwidth utilization predicted by an online performance model [10]. The SRN models presented in this work are orthogonal to previous NUMA performance optimizations and can also be applied to runtime systems to provide an accurate and efficient memory performance model.

3. System description

Fig. 1 depicts a generalized architecture of a typical NUMA multi-socket multi-core system. The architecture consists of several NUMA nodes. Each node contains a CPU node with CPU resources (CPU cores and their private/shared caches) and a memory node comprising a memory controller and memory. The nodes are connected by an interconnection network that enables each CPU core to access every memory node. Interconnection links can exhibit different service times depending on the topology of the interconnect. In Fig. 1, all nodes are connected with a dedicated link (full crossbar). This general architecture can represent direct interconnects, such as Intel's QuickPath Interconnect [33], and indirect interconnects with irregular latencies as used in AMD's HyperTransport [37].

The SRN models are tailored to modeling memory performance of scientific memory-intensive workloads. Such workloads have been shown to generate a steady and exponentially distributed memory requests with only few access bursts [10,43]. These characteristics allow for a number of simplifications in the models without sacrificing accuracy. First, the models assume that a core executes a single thread that is stalled for LLC read misses, and memory requests are processed in First-In-First-Out (FIFO) order. While modern processors support out-of-order execution to avoid stalling for memory requests [25] and memory controllers reorder requests [36], the analytical model is able to predict the mean memory request time for a multitude of memory requests in the steady-state. Second, memory requests that leave the LLC are served by two shared resources: the interconnection link and the memory controller. The presented models consider these two serialization points and compute a mean memory response time that includes the time taken by both the interconnection link and the memory controller.

In summary, the key assumptions of this work regarding memory accesses of the generalized architecture are as follows:

- A thread issuing memory accesses is stalled when a read access misses in the LLC.LLC misses trigger a number of memory operations on a memory node.

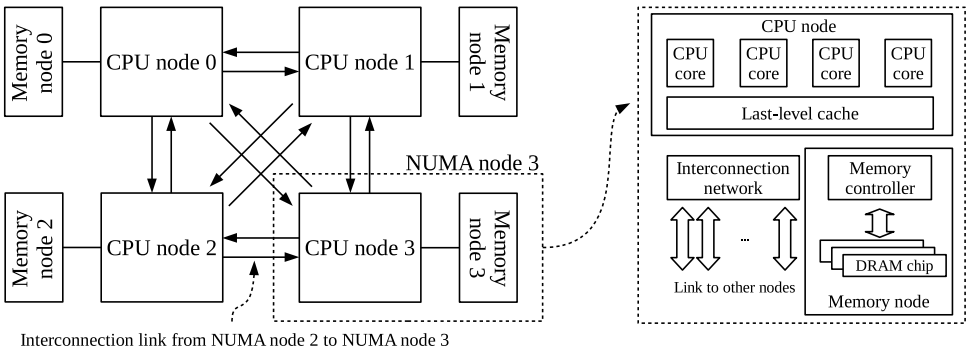

Fig. 1. Generalized NUMA architecture considered in this paper.

- A memory read operation issued by a CPU core completes after being served by both an interconnection link and a memory controller.
- Multiple memory operations by several CPU cores located behind the same CPU node to the same memory node are serialized in FIFO order.
- Multiple memory operations by several CPU cores from a single CPU node to different memory nodes do not cause contention because they are served by different interconnection links.
- Different CPU nodes can issue memory operations to the same memory node. This causes serialization of the requests at the memory controller in a FIFO queue. Since each CPU node has a dedicated interconnection link to every memory node, there is no contention for the interconnection link.

4. Overview of stochastic reward nets

Petri nets (PNs) are mathematical and graphical tools that provide a formal description of systems whose dynamics are characterized by concurrency, parallelism, synchronization, nondeterminism, mutual exclusion, and conflict; all typical features of distributed systems [29,35]. A Petri net can be defined as a 5-tuple, $PN = (P, T, F, W, M_0)$, where

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*,
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions*,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed *arcs*,
- $W : F \to \mathbb{N}$ assigns a *weight* to each arc, and
- $M_0 : P \to \mathbb{Z}^+$ defines the *initial marking*.

The initial marking M_0 assigns a number of *tokens* to the places P. The transitions and the arcs define how tokens move from place to place. Arcs connect places with transitions and vice-versa. The weight function W defines the multiplicity of an arc. A transition is enabled and can fire if all input places hold at least as many tokens as its input arcs. Upon firing, one token for each input arc is removed from its input place, and one token per output arc is deposited in the corresponding output place.

PNs are represented graphically using four elements [7]. Figs. 2 to 5 are examples of such a graphical representation.

- *Places* are represented by **circles**.
- *Transitions* are represented by **bars**.
- *Arcs* are shown as **directed edges**. The multiplicity of an arc is represented with a slash and the weight on the edge. Regular arcs are drawn with an arrow at the head, while an edge with a circle head represents an inhibitor arc.
- *Tokens*, if shown, are printed as little **dots.**

In classical PNs, there is no difference between transitions and each transition can fire as soon as it is enabled, i.e., there is no priority among enabled transitions. Although this definition of PNs is adequate for verifying certain properties of a system, such as liveness, boundedness, or invariance, it is not suitable for a quantitative evaluation of system behavior. In order to allow for a quantitative evaluation, PNs are equipped with a notion of *time* [18]. Time-augmented PNs can be further classified into Timed Petri Nets (TPNs; time is deterministic) or Stochastic Petri Nets (SPNs; time is stochastic). In SPNs, an exponentially distributed time delay is associated to each transition [5]. To model immediate actions, Generalized SPNs (GSPNs) [30] support *immediate* and *timed* transitions. Immediate transitions fire immediately after being enabled, while timed transitions fire after a random, exponentially distributed delay. In the graphical representation of GSPNs, immediate transitions are shown as filled rectangles, while timed transitions are represented by rectangular boxes.

Stochastic extensions were added to GSPNs to permit the indication of reward rates at the net level, resulting in Stochastic Reward Nets (SRNs). An SRN is obtained by associating reward rates with markings of a GSPN [12,32]. The graphical representation of an SRN is very similar to a GSPN; however, their application is different. SRNs allow the automated generation of Markov Reward Models (MRMs), facilitating the combined evaluation of performance and dependability of degradable fault-tolerant systems. Since the modeling power of SRNs is the same as that of MRMs, the wide range of measures available for MRMs can be easily obtained for SRNs.

The Stochastic Petri Net Package (SPNP) [13] is a well-known tool for numerically solving SRN models. Models are expressed in CSPL (C-based SPN Language), an extension of the C programming language. Advanced features of SPN models are available in SPNP, such as marking-dependent arc multiplicities, enabling functions, arrays of places or transitions, and subnets. Sophisticated steady-state and transient solvers are available including cumulative and up-to-absorption measures. The predefined measures can be easily extended by user-specified expressions. An SRN can be numerically analyzed with SPNP by first assigning reward rates to each marking of the SRN and then by computing the expected rewards. For example, in the steady-state analysis, if r_i represents the reward rate assigned to a marking i of an SRN model and π_i denotes the probability of the SRN for being in marking i in the steady-state, then the expected steady-state reward can be computed as $\sum_i r_i \pi_i$.

5. Proposed models

This section describes the SRNs proposed to model and evaluate the Mean Response Time (MRT) of memory accesses in NUMA architectures. In Section 5.1, we first present a basic single-core, single-controller SRN that models a single CPU node with one memory node. This model is then extended to capture the entire

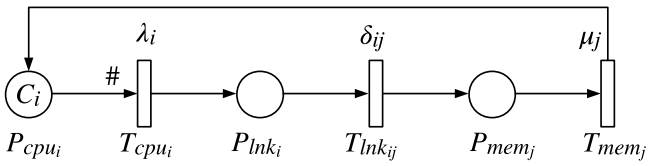

Fig. 2. SRN model for CPU node i, memory node j, and the interconnection link.

system architecture in a monolithic way in Section 5.2. To cope with the state space explosion problem of the monolithic model, two approximate models based on the folding method and the fixed-point iteration technique are proposed in Sections 5.3 and 5.4, respectively.

5.1. Single-core, single-controller SRN model

Fig. 2 shows the structure of the basic SRN to model CPU node i, memory node j, and the interconnection link connecting i to j. C_i denotes the number of active cores in CPU node i. Timed transitions T_{cpu_i}, $T_{lnk_{ij}}$, and T_{mem_j} model the LLC misses triggered by the C_i cores, the data transfer between CPU node i and memory node j, and the service process of memory node j, respectively. The memory access pattern of memory-intensive parallel workloads has been shown to follow an exponential distribution [11]; hence, we consider exponential distributions for the firing functions of transitions T_{cpu_i}, $T_{lnk_{ij}}$, and T_{mem_j}.

Initially, there are C_i tokens in place P_{cpu_i} to represent the active cores in CPU node i that can trigger an LLC miss. The firing rate of transition T_{cpu_i}, modeling the LLC misses triggered by the cores, is λ_i and can be considered as the inter-request rate of a core to an interconnection link. The pound sign (#) on the arc connecting place P_{cpu_i} to transition T_{cpu_i} denotes that the firing rate is marking dependent, that is, the rate of the transition depends on the number of tokens in the input place and varies over time. Thus, the firing rate of T_{cpu_i} is $[\#P_{cpu_i}] \cdot \lambda_i$, where $[\#P_{cpu_i}]$ denotes the number of tokens in place P_{cpu_i}. When transition T_{cpu_i} fires, one token is removed from place P_{cpu_i} and one token is deposited into place P_{lnk_i}. The existence of a token in place P_{lnk_i} enables timed transition $T_{lnk_{ij}}$ that models the data transfer through the link connecting CPU node i to memory node j. The time associated with transition $T_{lnk_{ij}}$ is exponentially distributed with rate δ_{ij}, where $1/\delta_{ij}$ stands for the mean data transfer time between CPU node i and memory node j.

Upon firing transition $T_{lnk_{ij}}$, a token moves from place P_{lnk_i} to place P_{mem_j} enabling timed transition T_{mem_j}. The time associated with this transition follows an exponential distribution with rate μ_j, where $1/\mu_j$ is the mean time taken by memory node j to serve a memory read/write request. After firing transition T_{mem_j}, a token is removed from place P_{mem_j} and put into place P_{cpu_i} to indicate that a core has been served by a memory node and can continue processing and generating more memory access requests. This basic SRN model is a simplified case that does not consider contention in a link or a memory node. In this model, a dedicated interconnection link connects CPU node i to memory node j, and memory node j serves the requests issued by the cores of CPU node i with FIFO discipline. The basic SRN lays the groundwork for the full system model presented in the next section.

5.2. Monolithic model

To model the NUMA architecture described in Section 3, the single-core, single-controller SRN from Fig. 2 is extended into a monolithic SRN that represents the complete system as shown in Fig. 3. In this SRN, there are M memory nodes and N CPU nodes, and each CPU node i contains C_i active cores. In the following, the bounds for variables i, j, and k are $1 \le i \le N$, $1 \le j \le M$, and $1 \le k \le C_i$, respectively.

Table 1
Guard functions of the SRN model represented in Fig. 3.

Guard function	Value
$[g_i]$	1 ; if $([\#P_{cpu_i}] + [\#P_{lnk_i}] < C_i)$
$1 \le i \le N$	0 ; otherwise

Table 2
Probability functions of immediate transitions of the SRN model represented in Fig. 3.

Immediate transition	Probability function
t_{cpu_i} $1 \le i \le N$	Return $\left(\frac{C_i}{\sum_{k=1}^{N} C_k}\right)$

Similar to the SRN presented in Fig. 2, place P_{cpu_i} models CPU node i. Each place P_{cpu_i} contains C_i tokens representing the active cores in CPU node i. The time associated with transition T_{cpu_i} represents the inter-request time of the cores in CPU node i and is considered to be exponentially distributed with rate λ_i. The firing time of transition T_{cpu_i} is marking dependent, i.e., with k tokens in place P_{cpu_i}, so the firing rate of T_{cpu_i} is $k \cdot \lambda_i$. If there is a token in place P_{lnk_i}, all transitions $T_{lnk_{ij}}$ are enabled. Similar to related work [10], we assume that all cores are executing the same application and that the data is evenly distributed among the memory nodes. Consequently, the SRN models presented here assume that memory requests are forwarded to the individual memory nodes with equal probability. This is a realistic assumption for NUMA multi-core systems executing applications optimized to utilize all available bandwidth of all memory nodes with parallel programming models such as OpenMP [15]. Since the number of memories is M, the token in place P_{lnk_i} can move to each place P_{mem_j} with probability $\frac{1}{M}$, i.e., the firing probability of each transition $T_{lnk_{ij}}$ is $\frac{1}{M}$. Upon firing one of the transitions $T_{lnk_{ij}}$, place P_{mem_j} receives a token. Inhibitor arcs are used to model the fact that the data is equally distributed among the memories and all cores triggering LLC misses experience an equal memory service time. Inhibitor arcs connecting places P_{mem_j} to transitions $T_{lnk_{ij}}$ prevent these transitions from firing when the number of tokens in places P_{mem_j} reaches the arc multiplicity m. Since there are M memory nodes and N CPU nodes in the system, and each CPU node i contains C_i active cores, we assign $m = \left\lceil \frac{\sum_{k=1}^{N} C_k}{M} \right\rceil$ to each inhibitor arc to balance the load among the memory nodes.

The existence of a token in place P_{mem_j} enables timed transition T_{mem_j}, modeling the service time of memory node j. After firing T_{mem_j} with rate μ_j, a token is moved from place P_{mem_j} to place P_{dis}. Place P_{dis} is an auxiliary place helping us to return the tokens exhausted from transitions T_{mem_j} to the right places. Without this mechanism, we would not know anymore from which CPU node this request has originated after depositing the token into place P_{mem_j}. The N immediate transitions t_{cpu_i} are used to dispatch the tokens of place P_{dis} to places P_{cpu_i}. A guard function, $[g_i]$ as shown in Table 1, is associated with each transition t_{cpu_i} to ensure that the number of cores in each CPU node does not exceed the predefined value C_i. Therefore, each transition t_{cpu_i} can fire only when the number of tokens in places P_{cpu_i} and P_{lnk_i} is less than C_i. In addition to the guard functions $[g_i]$, probability functions p_i are associated with transitions t_{cpu_i} as shown in Table 2. These functions ensure that a CPU node u with more active cores than CPU node v has a higher chance to receive tokens from place P_{dis} because it sends tokens to places P_{mem_j} more frequently.

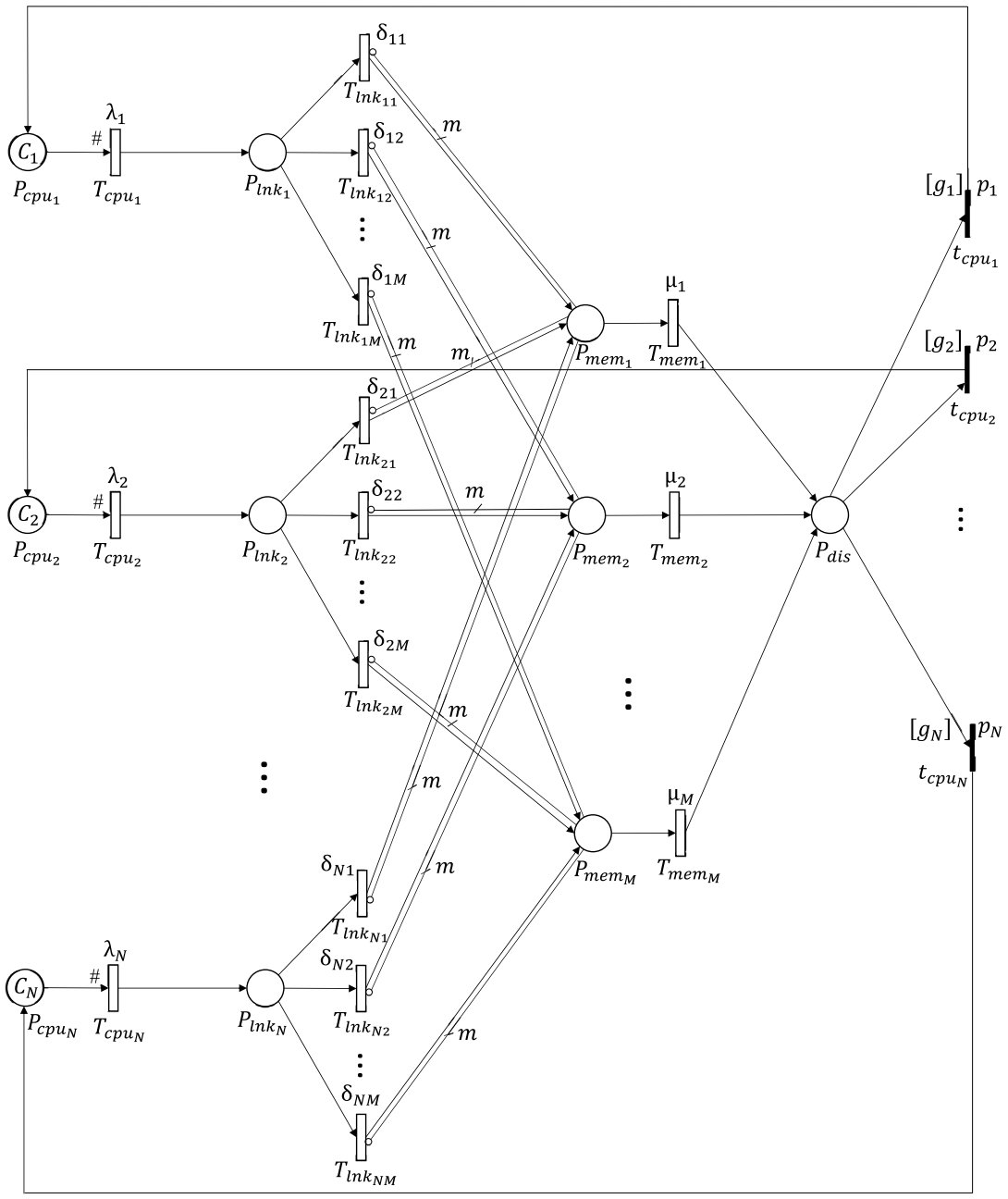

Fig. 3. Monolithic SRN model for the NUMA architectures under study.

The SRN shown in Fig. 3 can theoretically be used to model any number of CPU and memory nodes and any number of active cores in each CPU node. In practice, however, the model suffers from an exponential state space explosion in the underlying Markov chain of the SRN to the extent that existing tools cannot solve the model anymore. For example, for a system with nine active cores in eight CPU nodes and eight memory nodes, i.e., $N = 8$, $C_1 = 2$, $C_2 = C_3 = \cdots = C_8 = 1$, and $M = 8$, the underlying Markov chain of the monolithic model contains about seven hundred thousand states. By adding only one more active core to the system such that $C_1 = C_2 = 2$, $C_3 = C_4 = \cdots = C_8 = 1$, the number of states in the underlying Markov chain grows exponentially to over 10 million states. In other words, while the monolithic SRN models the entire system architecture, it is impractical to employ this model to large-scale NUMA systems. In addition, the structure of the net needs to be modified whenever the number of CPU or memory nodes in the system changes. In the following sections, two approximate models are presented that cope with these difficulties.

5.3. Folded model

In the monolithic SRN model, the structures of the individual CPU and memory nodes are identical, and LLC misses are redirected with equal probability to each memory node. This symmetry allows us to model the architecture with a folded SRN [20]. In a folded model, only one CPU and one memory node are modeled explicitly (the so-called *tagged* nodes) while the remaining CPU/memory nodes are folded into a combined sub-model.

Fig. 4 depicts the structure of the folded model. CPU node t is considered to be the *tagged CPU node* and all remaining $N - 1$ CPU nodes are folded into a submodel labeled *folded CPU nodes*. In a similar way, memory node t is treated as the *tagged memory*

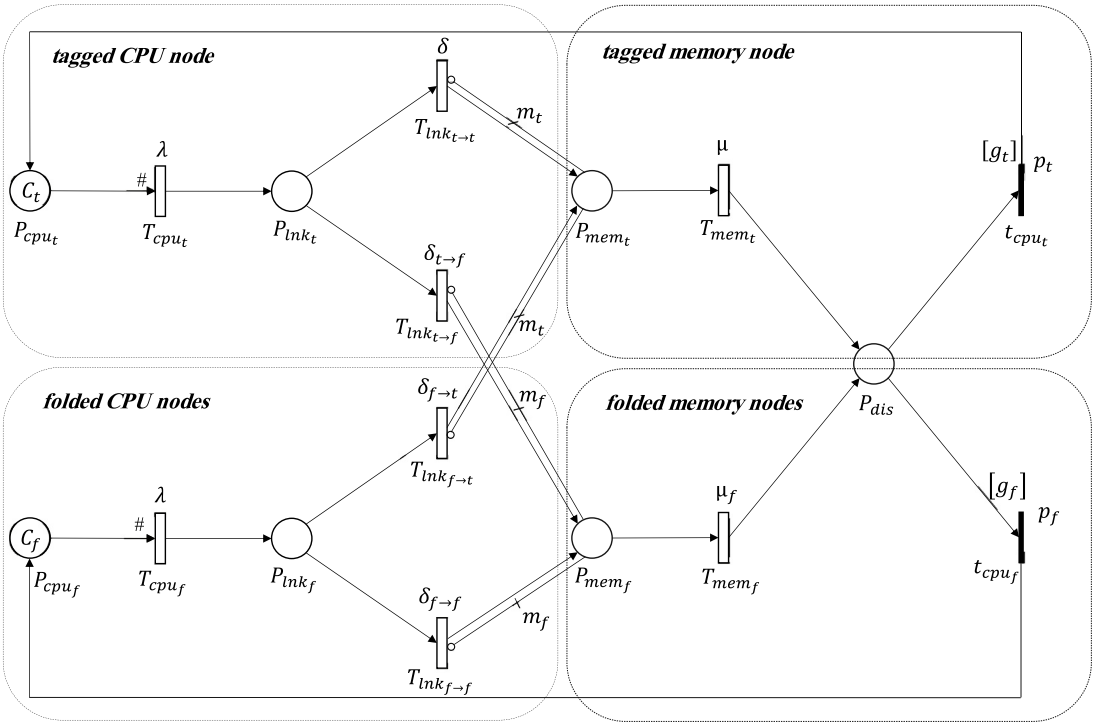

Fig. 4. Folded approximate SRN model for the NUMA architectures under study.

node and all remaining $M-1$ memory nodes are folded into submodel *folded memory nodes*. The *tagged CPU node* together with the *tagged memory node* can be used to compute the MRT of a memory access in the system, and the *folded CPU nodes* and the *folded memory nodes* are folded sub-models that influence the tagged sub-models.

To fold the monolithic SRN into the folded model, we assign the average of the transition rates to transitions in the folded model. That is, the mean inter-request time of cores, $1/\lambda$, is set to the average of $1/\lambda_i$ from Fig. 3. Similarly, the mean service time of memory nodes is set to $1/\mu = \frac{1}{M}\sum_{j=1}^{M}\frac{1}{\mu_j}$. The mean link transition time $1/\delta$ is set to the average of all link transition times in the monolithic SRN, i.e., $1/\delta = \frac{1}{N\cdot M}\sum_{i=1}^{N}\sum_{j=1}^{M}\frac{1}{\delta_{ij}}$. Since we focus on computing the steady-state MRT of memory accesses in the average case, the impact of these simplifications on accuracy is negligible.

Initially, there are C_t and $C_f = \sum_{k=1,k\neq t}^{N} C_k$ tokens in places P_{cpu_t} and P_{cpu_f}, respectively. Each token in place P_{cpu_t} represents an active core in the *tagged CPU node*. When an LLC miss occurs, a token is moved from place P_{cpu_t} to place P_{lnk_t}. The existence of a token in place P_{lnk_t} enables timed transitions $T_{lnk_{t\to t}}$ and $T_{lnk_{t\to f}}$, representing the interconnection links from the *tagged CPU node* to the *tagged memory node* and the *folded memory nodes*, respectively. Since the interconnection link between *tagged CPU node* and *tagged memory node* is a simple link, the firing rate of transition $T_{lnk_{t\to t}}$ is δ. Timed transition $T_{lnk_{t\to f}}$, however, is associated with a rate function $\delta_{t\to f}$ that takes the number of folded memories in *folded memory nodes* into account. Comparing the folded with the monolithic SRN, transition $T_{lnk_{t\to f}}$ in Fig. 4 corresponds to transitions $T_{lnk_{ij}}$, $i = t$ and $1 \le j \le M$, $j \neq t$, in Fig. 3. Since there are N CPU and M memory nodes in the system, each memory can serve at most $\left\lceil \frac{\sum_{k=1}^{N} C_k}{M} \right\rceil = \left\lceil \frac{C_t+C_f}{M} \right\rceil$ requests. This limitation of serving memory read/write requests by the memory nodes is modeled by using the multiplicity of inhibitor arcs connecting places P_{mem_t} and P_{mem_f} to the four timed transitions $T_{lnk_{t/f\to t/f}}$ modeling data transmissions. The maximum number of requests that can be served by *tagged memory node* and *folded memory node*, labeled m_t and m_f in Fig. 4, is defined by Eqs. (1) and (2).

$$m_t = \left\lceil \frac{C_t + C_f}{M} \right\rceil \tag{1}$$

$$m_f = (M-1)\cdot \left\lceil \frac{C_t + C_f}{M} \right\rceil \tag{2}$$

For transition $T_{lnk_{t\to f}}$, in addition to the inhibitor arc multiplicity, the number of folded memory nodes and the number of tokens in place P_{mem_f} need to be taken into account when transferring data from the *tagged CPU node* to the *folded memory nodes*. This is accomplished by the rate function $\delta_{t\to f}$ associated with $T_{lnk_{t\to f}}$ as given in Table 3. The $M-1$ memory nodes are represented by *folded memory nodes*. Each memory node has a capacity of $\left\lceil \frac{C_t+C_f}{M} \right\rceil$ requests, hence the number of memories in *folded memory nodes* with sufficient capacity to host a request from cores of *tagged CPU node* is given by Eq. (3).

$$(M-1) - \left\lfloor \frac{[\#P_{mem_f}]}{\left\lceil \frac{C_t+C_f}{M} \right\rceil} \right\rfloor \tag{3}$$

where $[\#P_{mem_f}]$ stands for the number of tokens in place P_{mem_f}. As the data transfer rate between a CPU node and a memory node is δ, we multiply Eq. (3) by δ to capture the value of $\delta_{t\to f}$, the firing rate of transition $T_{lnk_{t\to f}}$.

As shown in Fig. 4, there is contention between timed transitions $T_{lnk_{t\to t}}$ and $T_{lnk_{t\to f}}$ to grab a token from place P_{lnk_t}. If transition $T_{lnk_{t\to t}}$ wins, the token is moved to place P_{mem_t}; otherwise, $T_{lnk_{t\to f}}$ wins and the token is moved to place P_{mem_f}. If there is a token in place P_{mem_t}, timed transition T_{mem_t} is enabled and can fire with rate μ to represent the serving of a memory request by the *tagged memory node*. Upon firing T_{mem_t}, a token is removed from place P_{mem_t} and another token is deposited into place P_{dis}. The functionality of place P_{dis} and its output transitions, t_{cpu_t} and t_{cpu_f}, are the same as the related components

Table 3
Rate functions of the folded SRN model represented in Fig. 4.

Rate function	Value
$\delta_{t\to f}$	$\left((M-1)-\left\lfloor \frac{[\#P_{mem_f}]}{\left\lceil \frac{C_t+C_f}{M} \right\rceil} \right\rfloor\right)\cdot\delta$
$\delta_{f\to t}$	$[\#P_{lnk_f}]\cdot\delta$; if $[\#P_{lnk_f}]\leq(N-1)$ $(N-1)\cdot\delta$; otherwise
$\delta_{f\to f}$	$[\#P_{lnk_f}]\cdot(N-1)\cdot\delta$; if $[\#P_{lnk_f}]\leq\left((M-1)-\left\lfloor \frac{[\#P_{mem_f}]}{\left\lceil \frac{C_t+C_f}{M} \right\rceil} \right\rfloor\right)$ $\left((M-1)-\left\lfloor \frac{[\#P_{mem_f}]}{\left\lceil \frac{C_t+C_f}{M} \right\rceil} \right\rfloor\right)\cdot(N-1)\cdot\delta$; otherwise
μ_f	$[\#P_{mem_f}]\cdot\mu$; if $[\#P_{mem_f}]\leq(M-1)$ $(M-1)\cdot\mu$; otherwise

in the monolithic SRN model. Guard functions $[g_t]$ and $[g_f]$ and probability functions p_t and p_f, associated with transitions t_{cpu_t} and t_{cpu_f}, respectively, can be inferred from Tables 1 and 2 by replacing index i with t and f.

In the folded sub-model comprising *folded CPU nodes* and *folded memory nodes*, the existence of at least one token of C_f tokens in place P_{cpu_f} enables timed transition T_{cpu_f}. Since the firing rate of transition T_{cpu_f} is marking dependent and the number of tokens in folded place P_{cpu_f} is equal to the number of active cores in $N-1$ CPU nodes, we do not change the rate λ. Hence, the rate of T_{cpu_f} is $[\#P_{cpu_f}]\cdot\lambda$, where $[\#P_{cpu_f}]$ is the mark of place P_{cpu_f}. Upon firing T_{cpu_f}, place P_{lnk_f} is fed with a token and both timed transitions $T_{lnk_{f\to t}}$ and $T_{lnk_{f\to f}}$ are enabled. Transition $T_{lnk_{f\to t}}$ redirects memory accesses to *tagged memory node* and transition $T_{lnk_{f\to f}}$ redirects them to *folded memory nodes*. The two rate functions $\delta_{f\to t}$ and $\delta_{f\to f}$, associated with transitions $T_{lnk_{f\to t}}$ and $T_{lnk_{f\to f}}$, respectively, are defined in Table 3.

Timed transition $T_{lnk_{f\to t}}$ in Fig. 4 corresponds to transitions $T_{lnk_{ij}}$, $1\leq i\leq N, i\neq t, j=t$, in Fig. 3. The firing rate of $T_{lnk_{f\to t}}$ considers the number of folded CPU nodes to find the number of links connecting those CPU nodes to the *tagged memory node*. The rate function $\delta_{f\to t}$ first checks the number of tokens in place P_{lnk_f}, denoted by $[\#P_{lnk_f}]$. If that number is less than or equal to $N-1$, the rate function fires with rate $[\#P_{lnk_f}]\cdot\delta$, otherwise with $\delta_{f\to t}=(N-1)\cdot\delta$. While transition $T_{lnk_{f\to t}}$ folds only $N-1$ transitions, transition $T_{lnk_{f\to f}}$ contains a wide spectrum of folded transitions, corresponding to transitions $T_{lnk_{ij}}$, $1\leq i\leq N, i\neq t$, and $1\leq j\leq M, j\neq t$, of Fig. 3, i.e., $(N-1)\cdot(M-1)$ separate transitions. The firing rate $\delta_{f\to f}$ of $T_{lnk_{f\to f}}$ must not only take the number of folded CPU nodes into account, but it also has to consider the number of folded memory nodes and the number of tokens in place P_{mem_f}. Therefore, the rate function $\delta_{f\to f}$, shown in Table 3, compares the number of tokens in place P_{mem_f} with the total capacity of *folded memory nodes*. If there is enough capacity to host a new request in *folded memory nodes*, it checks the number of tokens in place P_{lnk_f} to find the number of interconnection links between *folded CPU nodes* and memory nodes with sufficient capacity. After computing the number of links and memory nodes with sufficient capacity, rate function $\delta_{f\to f}$ multiplies the total number of existing paths by δ to obtain the final value of the rate associated with transition $T_{lnk_{f\to f}}$.

Upon firing transitions $T_{lnk_{f\to f}}$ and $T_{lnk_{f\to t}}$, a token is moved from place P_{lnk_f} to places P_{mem_f} and P_{mem_t}, respectively. Place P_{mem_f}, which accepts tokens from both transitions $T_{lnk_{t\to f}}$ and $T_{lnk_{f\to f}}$, models the folded memory nodes. The rate function μ_f is associated with this transition to compute its firing rate. As shown in Table 3, function μ_f checks the number of tokens in place P_{mem_f}. If this number is less than or equal to $M-1$, i.e., the number of folded memory nodes, it assigns $[\#P_{mem_f}]\cdot\mu$ to μ_f, where $[\#P_{mem_f}]$ is the mark of place P_{mem_f}; otherwise, $\mu_f=(M-1)\cdot\mu$. Therefore, the maximum speed of the folded memory node in serving requests is $(M-1)\cdot\mu$. After firing transition T_{mem_f}, a token is put in place P_{dis} to be returned to P_{cpu_t} or P_{cpu_f}.

The state space of the folded model grows much slower than that of the monolithic model. The folded model is able to analyze all configurations of the NUMA architecture considered in this paper. Nevertheless, the growth rate of the number of states in the underlying Markov chain of this model is still a concern. As an example, consider a NUMA architecture with eight memory nodes and six CPU nodes with 48 active cores ($M=8$, $N=6$, $C_t=6$, $C_f=42$). The Markov chain of the this folded SRN model contains around 170 thousand states. Adding eight, then 16 active cores to the system by increasing the number of CPU nodes to 7 and 8, respectively, the number of states reaches 340 and 620 thousand states. To analyze larger NUMA architectures, the following section presents a more scalable model.

5.4. Fixed-point model

To analyze future NUMA architectures with tens of CPU and memory nodes, a scalable approximate model based on the fixed-point iteration technique is presented. The fixed-point iteration technique is a viable approach to analyze systems with interrelated sub-systems [3,8,17,40]. Each sub-system is analyzed separately while the remainder of the system is represented in a simplified manner. The technique iteratively solves the simplified complement of the sub-system and modifies the input parameters of other sub-systems in accordance to the output of the current sub-system. This procedure is repeated until a fixed-point is reached, i.e., the difference between the values of a measure from two successive iterations falls below a given threshold.

From Fig. 4, we observe that the folded model is divided into four parts labeled *tagged CPU node*, *tagged memory node*, *folded CPU nodes*, and *folded memory nodes*. In the folded model, the *tagged memory node* and the *folded memory nodes* act as delays with respect to the other sub-models of the system. In the proposed fixed-point approximate model shown in Fig. 5, the two parts are replaced with two timed transitions labeled T_{mem_t} and T_{mem_f}. The rates assigned to these transitions are determined by considering the folded model. The two remaining parts are used to derive sub-models of the fixed-point SRN model. In Fig. 5, *sub-model 1* and *sub-model 2*, respectively corresponding to the tagged CPU/memory nodes and the folded CPU/memory nodes, have almost the same structure, but their dynamic behaviors and functions are different. To maintain consistency with the folded model, the same names are used for components in the fixed-point model. For the sake of brevity, the components of Fig. 5 that are similar with those introduced in Fig. 4, mostly subscribed by t and f, are not described in this section.

We first describe *sub-model 2* which corresponds to *folded CPU nodes* and *folded memory nodes* in the folded model. A token in place P_{lnk_f} enables timed transitions $T_{lnk_{f\to f}}$ and $T'_{lnk_{sub2\to sub1}}$. Transition $T_{lnk_{f\to f}}$ assigns requests issued from the cores of *folded CPU nodes* to *folded memory nodes*, and transition $T'_{lnk_{sub2\to sub1}}$ sends the requests to the *tagged memory node*. Transition $T_{lnk_{f\to f}}$ has the same functionality as in the folded model and its rate is $\delta_{f\to f}$ as shown in Table 3. However, the firing rate of $T'_{lnk_{sub2\to sub1}}$, denoted by $\delta'_{sub2\to sub1}$, is affected by the number of tokens in place P_{mem_t} of *sub-model 1* and computed by Eq. (4).

$$\delta'_{sub2\to sub1}=p_{ava\_mem_t}\cdot\delta \tag{4}$$

where $p_{ava\_mem_t}$ stands for the probability of *tagged memory node* being available for the requests sent from *folded CPU nodes*. This

Table 4
Rate functions defined for the fixed-point SRN model shown in Fig. 5.

Rate function	Value
$\delta'_{sub2\to sub1}$	$Pr\left(\#P_{mem_t} < \left\lceil \frac{C_t+C_f}{M} \right\rceil\right) \cdot \delta$
$\delta'_{sub1\to sub2}$	$\left((M-1) - \left\lfloor \frac{[\#P_{mem_f}]}{\left\lceil \frac{C_t+C_f}{M} \right\rceil} \right\rfloor\right) \cdot Pr(\#P_{lnk_t} > 0) \cdot \delta$
μ^{eff}	$Pr(\#P_{mem_t} > 0) \cdot \mu$
$\delta_{sub1\to sub2}$	$Pr\left((M-1) - \left\lfloor \frac{[\#P_{mem_f}]}{\left\lceil \frac{C_t+C_f}{M} \right\rceil} \right\rfloor = j\right) \cdot j \cdot \delta$
$\delta_{sub2\to sub1}$	$Pr\left(\#P_{mem_t} < \left\lceil \frac{C_t+C_f}{M} \right\rceil\right) \cdot Pr(\#P_{lnk_f} > 0) \cdot \delta'_{sub2\to sub1}$
μ_f^{eff}	$\sum_{j=1}^{M-1} Pr(\#P_{mem_f} = j) \cdot j \cdot \mu + \left(1 - \sum_{j=1}^{M-1} Pr(\#P_{mem_f} = j)\right) \cdot (M-1) \cdot \mu$

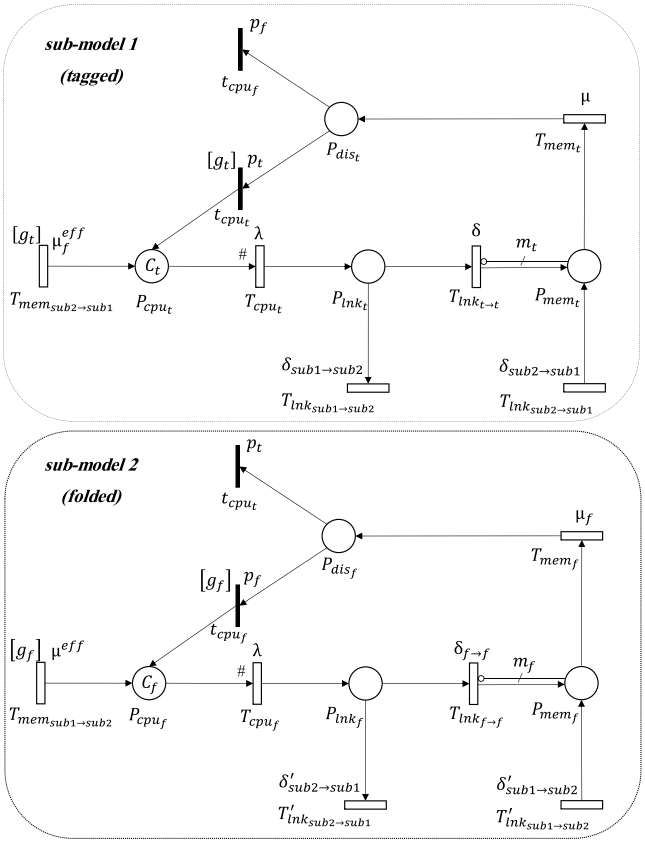

Fig. 5. Fixed-point approximate SRN model for the NUMA architectures under study.

parameter, which is passed from *sub-model 1* to *sub-model 2*, is defined by Eq. (5).

$$p_{ava\_mem_t} = Pr\left(\#P_{mem_t} < \left\lceil \frac{C_t + C_f}{M} \right\rceil\right) \tag{5}$$

where $Pr(\#P_{mem_t} < i)$ is the probability that there are less than i tokens in place P_{mem_t}.

Tokens in place P_{mem_f} model the serving of requests in *folded memory nodes*. The firing rate μ_f of timed transition T_{mem_f} is the same as in the folded model and given in Table 3. Similarly, the multiplicity of the inhibitor arc connecting P_{mem_f} to $T_{lnk_{f\to f}}$, defining the maximum number of requests that can be served by the *folded memory nodes*, is given by Eq. (2). While timed transition $T_{lnk_{f\to f}}$ models accesses to *folded memory nodes* by the cores of *folded CPU nodes*, transition $T'_{lnk_{sub1\to sub2}}$ models accesses from the cores of the *tagged CPU node*. As in the folded model, the rate of redirecting requests from the *tagged CPU node* to *folded memory nodes* depends on the number of available memory nodes in the folded $M-1$ memory nodes. In *sub-model 2* of the fixed-point model, the rate of $T'_{lnk_{sub1\to sub2}}$ depends not only on the number of tokens in place P_{mem_f} but also on the probability of at least one token being in place P_{lnk_t}, which is the enabling condition of transition $T_{lnk_{sub1\to sub2}}$ in *sub-model 1*. Therefore, the firing rate of $T'_{lnk_{sub1\to sub2}}$ is computed by Eq. (6).

$$\delta'_{sub1\to sub2} = \left((M-1) - \left\lfloor \frac{[\#P_{mem_f}]}{\left\lceil \frac{C_t+C_f}{M} \right\rceil} \right\rfloor\right) \cdot \quad Pr(\#P_{lnk_t} > 0) \cdot \delta \tag{6}$$

where $Pr(\#P_{lnk_t} > 0)$ denotes the probability that at least one token is in place P_{lnk_t}. This probability is an output of *sub-model 1* and used here as an input for *sub-model 2*.

When T_{mem_f} fires, a token is moved from place P_{mem_f} to place P_{dis_f}. Place P_{dis_f} is necessary to properly dispatch the requests served by *folded memory nodes* to *folded CPU nodes*. The two outgoing transitions t_{cpu_t} and t_{cpu_f} from place P_{dis_f} are assigned the probability functions p_t and p_f that are identical to those in the folded model. The guard function $[g_f]$, assigned to transition t_{cpu_f}, compares the sum of tokens in places P_{cpu_f} and P_{lnk_f} with C_f. If the sum is less than C_f, then the guard evaluates to true and both transitions t_{cpu_t} and t_{cpu_f} can fire with probabilities p_{cpu_t} and p_{cpu_f}, respectively. Otherwise, only t_{cpu_t} fires and empties place P_{dis_f}. Unlike in the folded model, no guard function is assigned to transition t_{cpu_t} since it is only used to balance the fraction of tokens redirected to *sub-model 2* and empty place P_{dis_f} when t_{cpu_f} is disabled due to an unsatisfied guard.

Place P_{cpu_f} accepts tokens from both transitions t_{cpu_f} and $T_{mem_{sub1\to sub2}}$. Immediate transition t_{cpu_f} models the output stream of *folded memory nodes* to *folded CPU nodes*, whereas timed transition $T_{mem_{sub1\to sub2}}$ models the output stream of the *tagged memory node* to *folded CPU nodes*. The guard function $[g_f]$ is also associated with $T_{mem_{sub1\to sub2}}$ to ensure that the number of tokens in *folded CPU nodes* does not exceed C_f. The firing rate of $T_{mem_{sub1\to sub2}}$ depends on the effective firing rate of transition T_{mem_t} in *sub-model 1* and is computed by Eq. (7).

$$\mu^{eff} = Pr(\#P_{mem_t} > 0) \cdot \mu \tag{7}$$

where $Pr(\#P_{mem_t} > 0)$ is the probability of there being at least one token in place P_{mem_t}. The rate functions $\delta'_{sub2\to sub1}$, $\delta'_{sub1\to sub2}$, and μ^{eff}, finally, are given in Table 4. The functions $\delta_{f\to f}$ and μ_f are identical to those in the folded model (see Table 3).

In *sub-model 1*, timed transition $T_{lnk_{sub1\to sub2}}$ models the memory requests from cores in *tagged CPU node* to *folded memory nodes*. The firing rate of this transition is affected by the number of tokens in place P_{mem_f} of *sub-model 2* and given in Eq. (8).

$$\delta_{sub1\to sub2} = p_{ava\_mem_f}(j) \cdot j \cdot \delta \tag{8}$$

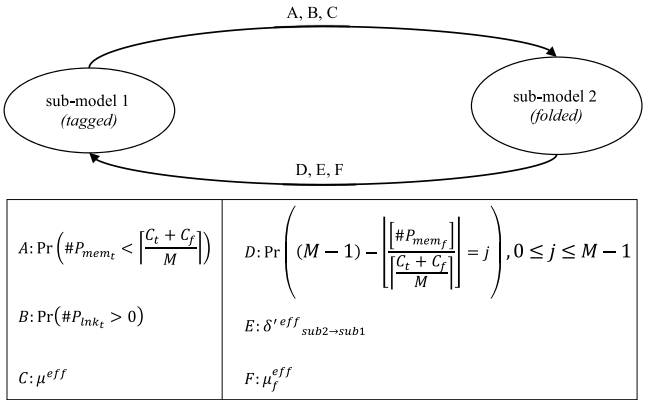

Fig. 6. Import graph of the sub-models of the fixed-point model.

where $p_{ava\_mem_f}(j)$, $0 \le j \le M-1$, an output of *sub-model 2* and input to *sub-model 1*, denotes the probability of there being j available memory nodes in *folded memory nodes*. This probability is computed by Eq. (9).

$$p_{ava\_mem_f}(j) = Pr\left((M-1) - \left\lfloor \frac{[\#P_{mem_f}]}{\lceil \frac{C_t + C_f}{M} \rceil} \right\rfloor = j\right) \tag{9}$$

Timed transition $T_{lnk_{t \to t}}$ models the accesses to *tagged memory node* by the cores of *tagged CPU node*, and transition $T_{lnk_{sub2 \to sub1}}$ models these accesses by the cores of the *folded CPU nodes*. The rate of $T_{lnk_{sub2 \to sub1}}$ is computed by Eq. (10).

$$\delta_{sub2 \to sub1} = p_{ava\_mem_t} \cdot \delta'^{eff}{}_{sub2 \to sub1} \tag{10}$$

where $p_{ava\_mem_t}$ is the probability of *tagged memory node* not being full, obtained by Eq. (5), and $\delta'^{eff}{}_{sub2 \to sub1}$, passed from *sub-model 2* to *sub-model 1*, represents the effective firing rate of transition $T'_{lnk_{sub2 \to sub1}}$ computed by Eq. (11).

$$\delta'^{eff}{}_{sub2 \to sub1} = Pr(\#P_{lnk_f} > 0) \cdot \delta'_{sub2 \to sub1} \tag{11}$$

where $Pr(\#P_{lnk_f} > 0)$ is the probability that there is at least one token in place P_{lnk_f}.

Timed transition T_{mem_t} shows the serving process of requests by the *tagged memory node* and its rate is μ. The functionality of the components P_{dis_t}, t_{cpu_t}, t_{cpu_f}, and their related functions are the same as the corresponding components and functions introduced for *sub-model 2*. Place P_{cpu_t} has two lines of token arrivals: immediate transition t_{cpu_t} representing the tokens arriving from *tagged memory node*, and timed transition $T_{mem_{sub2 \to sub1}}$ modeling the tokens coming from the *folded memory nodes*. The guard function $[g_t]$, which is associated with both transitions t_{cpu_t} and $T_{mem_{sub2 \to sub1}}$, ensures that the number of tokens in places P_{cpu_t} and P_{lnk_t} does not exceed the predefined threshold C_t. The firing rate of transition $T_{mem_{sub2 \to sub1}}$ depends on the effective firing rate of transition T_{mem_f} in *sub-model 2*. According to the firing rate of T_{mem_f}, denoted by μ_f, the effective rate μ_f^{eff} is computed by Eq. (12).

$$\begin{aligned} \mu_f^{eff} &= \sum_{j=1}^{M-1} Pr(\#P_{mem_f} = j) \cdot j \cdot \mu \\ &+ \left(1 - \sum_{j=1}^{M-1} Pr(\#P_{mem_F} = j)\right) \cdot (M-1) \cdot \mu \end{aligned} \tag{12}$$

where $Pr(\#P_{mem_f} = j)$ is the probability of there being exactly j tokens in place P_{mem_f}. The rate functions $\delta_{sub1 \to sub2}$, $\delta_{sub2 \to sub1}$, and μ_f^{eff} are represented in Table 4.

The iterative solving process starts with arbitrary values for parameters passed between the sub-models. From the second iteration on, input parameters are set according to the output values obtained in the previous iteration. This process continues until the difference between the MRT of a memory access for two successive iterations becomes smaller than a tolerance bound. The MRT obtained in the last iteration is reported as the final result. The import graph, representing the parameters passed between the two sub-models of the fixed-point model, is shown in Fig. 6.

6. Memory performance evaluation of a real NUMA architecture

This section introduces the NUMA machine under test, the target applications, and the methodology used to obtain the input parameters for the proposed SRN models.

6.1. Target NUMA architecture

We validate the proposed models on a 64-core AMD NUMA platform with four AMD Opteron 6380 processors [1]. A block diagram of the architecture is given in Fig. 7. The machine has eight CPU nodes and eight memory nodes. Each CPU node contains eight CPU cores and one shared LLC, and each memory node has one memory controller. All CPU and memory nodes are connected

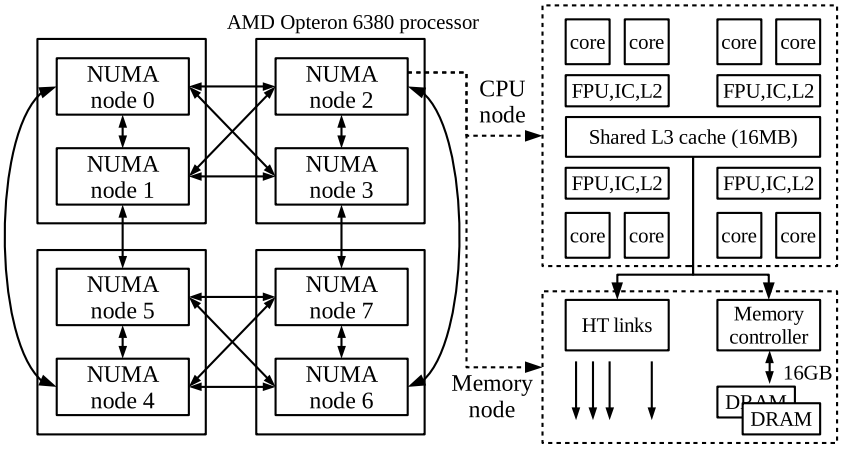

Fig. 7. The 64-core AMD multi-socket system.

by AMD's HyperTransport (HT) [37]. The AMD HT interconnection topology is asymmetric and not all nodes are directly connected. As a consequence, the interconnection links between a CPU node and a memory controller observe varying latencies in dependence of the number of hops. The AMD server architecture was chosen over the Intel architecture with its regular QuckPath Interconnect [33] to demonstrate that the presented models can cope with irregular and indirect interconnects that exhibit varying latencies to the different memory nodes.

The layout of the CPU, the memory nodes, and the HT interconnect fit into the generalized NUMA system from Section 3. AMD Opteron processors contain the following two additional shared resources that are not modeled by the generalized NUMA system:

1. Pairs of neighboring cores, called Compute Units (CU), share the Floating Point Unit (FPU), the first-level Instruction Cache (IC), and the second-level (L2) cache. This can cause contention in these resources between the two cores of a CU.
2. The shared L3 cache has a limited size. Multiple cores contending for LLC resources can cause a super-linear LLC miss rate.

Section 6.3 addresses these contention points when profiling the parameters of the models.

6.2. Target applications

The proposed SRNs are validated with eight memory-intensive high-performance computing applications. The evaluation uses *FT*, *SP*, *CG*, *BT*, *MG* and *LU* from the NAS Parallel Benchmarks (NPB) version 1 [4]. We use a C implementation of the applications [39]. *EP* and *IS* were excluded because these benchmarks generate almost no memory accesses. *EP* is compute-bound and the working set of *IS* fits into the on-chip caches. The workloads *SpMV* (Sparse-Matrix and Vector multiplication) using the Compressed Sparse Row (CSR) format and *Pagerank* from the GraphBIG benchmark [34] represent two workloads from the machine-learning and big data application domain. To obtain the hardware-dependent parameters, the *Stream Write* microbenchmark from *Stream benchmark* [31] is used. *Stream Write* generates memory read and write operations with a configurable stride pattern and is a completely memory-bound with no computation or opportunities for caching.

6.3. Extracting model parameters

To validate the proposed SRNs in terms of the mean memory response time *MRT* for a given number of cores, one workload is executed with a varying number of active cores. Each core is assigned a constant amount of work. An increasing number of active cores causes more contention in the interconnection network and the memory nodes, which in turn leads to a longer *MRT* and a higher workload turnaround time. We defined the turnaround time (*total_time*) as shown in Eq. (13).

$$total\_time = CPU\_time + LLC\_miss \cdot MRT \tag{13}$$

CPU_time represents the workload's execution time for the assigned work excluding the time spent for memory accesses. The time consumed by memory accesses is computed by multiplying the number of LLC misses (*LLC_miss*) by the mean memory response time (*MRT*). The value of *MRT* is obtained by dividing the total memory response time by the number of LLC misses as shown in Eq. (14).

$$MRT = \frac{total\_time - CPU\_time}{LLC\_miss} \tag{14}$$

The *MRT* obtained from experiments on the AMD machine is compared with the value computed by the SRN models.

To compute the workload-specific model parameter λ, each workload is executed in isolation on one CPU core and its local memory node to measure the turnaround time of the workload, $total\_time_1$, and the number of LLC misses, LLC_miss_1, obtained from the hardware's performance counters [2]. The two parameters are used to compute the value of λ. If the target platform has additional sources of shared-resource interference, more information about the workload may be required. In the following, we show how to compute the *MRT* for the AMD architecture with the aforementioned shared resources (Section 6.1).

A workload's inter-request rate λ represents the number of LLC misses per time unit from a core in the steady-state and is defined by Eq. (15).

$$\lambda = \frac{LLC\_miss}{CPU\_time} \tag{15}$$

LLC_miss represents the average of per-core LLC misses for all allocated cores. Let $LLC\_miss_{C_i}$ denote the average number of LLC misses of the C_i CPU cores in CPU node i. Then, *LLC_miss* is computed by Eq. (16) as follows.

$$LLC\_miss = \frac{\sum_{i=1}^{N} (LLC\_miss_{C_i} \cdot C_i)}{\sum_{i=1}^{N} C_i} \tag{16}$$

On our target architecture, each CPU node contains at most eight cores, and $LLC\_miss_{C_i}$ changes for larger numbers of active cores due to LLC interference. To consider this interference, we execute the workload using all $P = 8$ cores of a CPU node and measure the number of LLC misses. LLC_miss_P represents the average number of LLC misses observed by one core under full load and is computed by dividing the LLC misses by P. The average number of LLC misses for a given C_i is interpolated from the difference between the LLC misses for one and eight cores, $diff = LLC\_miss_P - LLC\_miss_1$ as shown in Eq. (17).

$$LLC\_miss_{C_i} = LLC\_miss_1 + diff \cdot \frac{C_i - 1}{P - 1} \tag{17}$$

Interference in the CUs and the LLC can affect the *CPU_time*. *CPU_time* is computed using Eq. (18)

$$CPU\_time = CPU\_time_1 + if_{CU} + if_{LLC} \tag{18}$$

where CPU_time_1 represents execution time on one CPU core without any interference, and if_{CU} and if_{LLC} represent the overhead of CU and LLC interference, respectively.

To compute CPU_time_1, we use the values of $total\_time_1$ and LLC_miss_1 obtained from the initial profiling run using one core. According to Eq. (13), CPU_time_1 is computed by Eq. (19) as follows.

$$CPU\_time_1 = total\_time_1 - (LLC\_miss_1 \cdot MRT_1) \tag{19}$$

where MRT_1 is obtained by executing *Stream Write*. Since *Stream Write* has no computations or cache hits, we can simply measure its mean memory response time.

To determine if_{CU}, the turnaround times of two runs, one using two cores in the same CU, the other one using two cores from separate CUs, are compared. The difference, denoted by β, captures the time caused by interference in a CU. The average overhead of interference caused by resource sharing in a CU is then given by Eq. (20).

$$if_{CU} = \begin{cases} \beta \cdot \frac{(\sum_{i=1}^{N} C_i) - 1}{\sum_{i=1}^{N} C_i}, & \text{if } C_N \% 2 = 1 \\ \beta, & \text{otherwise} \end{cases} \tag{20}$$

To account for interference in the LLC, if_{LLC} computes the time increase by considering the difference of the number of LLC misses as shown in Eq. (21).

$$if_{LLC} = \alpha \cdot (LLC\_miss - LLC\_miss_1) \tag{21}$$

Workloads may experience a longer CPU time with an increasing number of LLC misses caused by interference (capacity misses) in the LLC. Based on measured results with 1 and the maximum number of available cores in a CPU node, P, we set α to the empirically determined value $\alpha = 0.025$ if $LLC\_miss_1/Total\_time_1 > LLC\_miss_P/Total\_time_P$. Otherwise, $\alpha = 0$. A sophisticated scheme may provide better accuracy, however, the effect of if_{LLC} on accuracy has been found to be marginal.

The machine-dependent parameters $\delta_{i,j}$, representing the data transfer rate between CPU node i and memory node j, and μ_j, the service rate of memory node j, are computed by considering the different link transfer times in the NUMA interconnection network. The *Stream Write* benchmark is employed to discover the different interconnection link transfer times by executing *Stream Write* on one CPU core and accessing all possible memory controllers while measuring the transfer/service times of the different links.

7. Numerical evaluation

The presented models are evaluated using the Stochastic Petri Net Package (SPNP) [13] in the steady-state. The measure of interest is the MRT of a memory access. For the folded and the fixed-point model, this measure is computed for the cores in the *tagged CPU node* and is obtained by applying Little's law [41] to both the interconnection link and the *tagged memory node*. The monolithic model does not have an explicitly tagged CPU node, hence any of the CPU nodes, its interconnection link and memory node can be chosen.

The accuracy of the models is given as the Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{1}{n} \cdot \sum_{k=1}^{n} \left| \frac{a_k - m_k}{a_k} \right| \tag{22}$$

where a_k represents the actual value obtained from the measurement on a real system, and m_k is the value computed by the analytic model. MAPE is a popular statistical measure of a forecast systems' accuracy. As a measure of error, lower values represent higher prediction accuracy; the mean absolute percent accuracy is obtained by subtracting the MAPE from 1. MAPE has been chosen due to its advantages of scale-independence and interpretability. Another popular metric that quantifies the similarity of two sets of values is R^2, computed as the square of the correlation between the predicted and the actual values. The R^2 metric may show unintuitive behavior in situations where the trend of actual data values is close to linear. This can lead to seemingly bad R^2 values despite small absolute errors and a strong visual correlation between the predicted and the actual values. The presented SRN models are compared a multi-level queueing system recently proposed by Cho [10] that models the interconnection link and the memory controller using a M/M/1/N/N queueing system.

Tables 5 and 6 list the input parameters of the models along with the observed values in our experiments. Table 5 gives the data transfer rate for each memory interconnection link and the service rate of a memory controller; the values were measured using the *Stream Write* microbenchmark. The link transfer rate δ used in the folded and fixed-point SRN models is set to the average of the individual link transfer rates. Table 6 lists the LLC miss rate λ of one CPU core for all target applications. Note that the value of λ changes in dependence of the number of active cores. All hardware-dependent parameters are computed as described in Section 6.3. The SRN models are evaluated using three different scenarios with a varying number of CPU cores and memory nodes.

Table 5
Architecture-specific parameters of the models.

Parameter	Distance to memory node	Rate (1/µs)
$\delta_{i,j}$, $1 \le i,j \le 8$	0 hops: local access	285.7
	1 hop: remote access within same processor	142.9
	1 hop: remote access to different processor	90.9
	2 hops: remote access to different processor	49.3
μ_j, $1 \le j \le 8$	–	87.0

Table 6
Application-specific parameters of the models.

Rate (req./us)	Stream write	FT	SP	CG	BT	MG	LU	SpMV	Pagerank
λ	1235	12	32	57	7	40	45	49	27

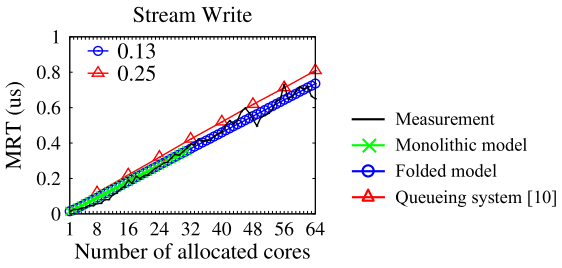

Fig. 8. MRT of Stream Write for one memory node and a varying number of cores (Section 7.1). The values show the MAPE of the models against actual measurements. The MAPE of the monolithic model is not reported because it cannot be solved by the SPNP tool for more than 32 cores.

7.1. One memory node, varying number of cores

In a first scenario, performance is evaluated for one memory node with an increasing number of active cores. The memory-intensive application *Stream Write* is used to validate the SRN models for this heavily congested scenario. Fig. 8 shows the computed and measured MRT from 1 to 64 cores obtained by solving the presented monolithic and folded models together with the data gathered from the experiments on the 64-core AMD system. The memory controller and the interconnection link are saturated at a small number of active cores, consequently, the MRT increases almost linearly with the number of cores. The presented folded model achieves a MAPE of 0.13 and is significantly more accurate than the queueing-based model presented in [10] with a MAPE of 0.25. As visible in Fig. 8, the SPNP tool cannot solve the monolithic model for more than 32 cores because of a state space explosion in the underlying Markov chain. The fixed-point model cannot be applied to this scenario because it requires the number of CPU and memory nodes to be bigger than one.

7.2. One CPU node, varying number of memory nodes

In this second scenario, one CPU node with eight active cores is paired with one to eight memory nodes. Fig. 9 presents the results of this scenario for the eight target applications. Similar to the first scenario, the fixed-point model cannot be evaluated for only a single CPU node. We observe that by increasing the number of memory nodes, the mean response time of memory accesses decreases. The results shown in Fig. 9 demonstrate the prominence of the presented models in estimating the MRT of memory accesses against the previously presented approach [10].

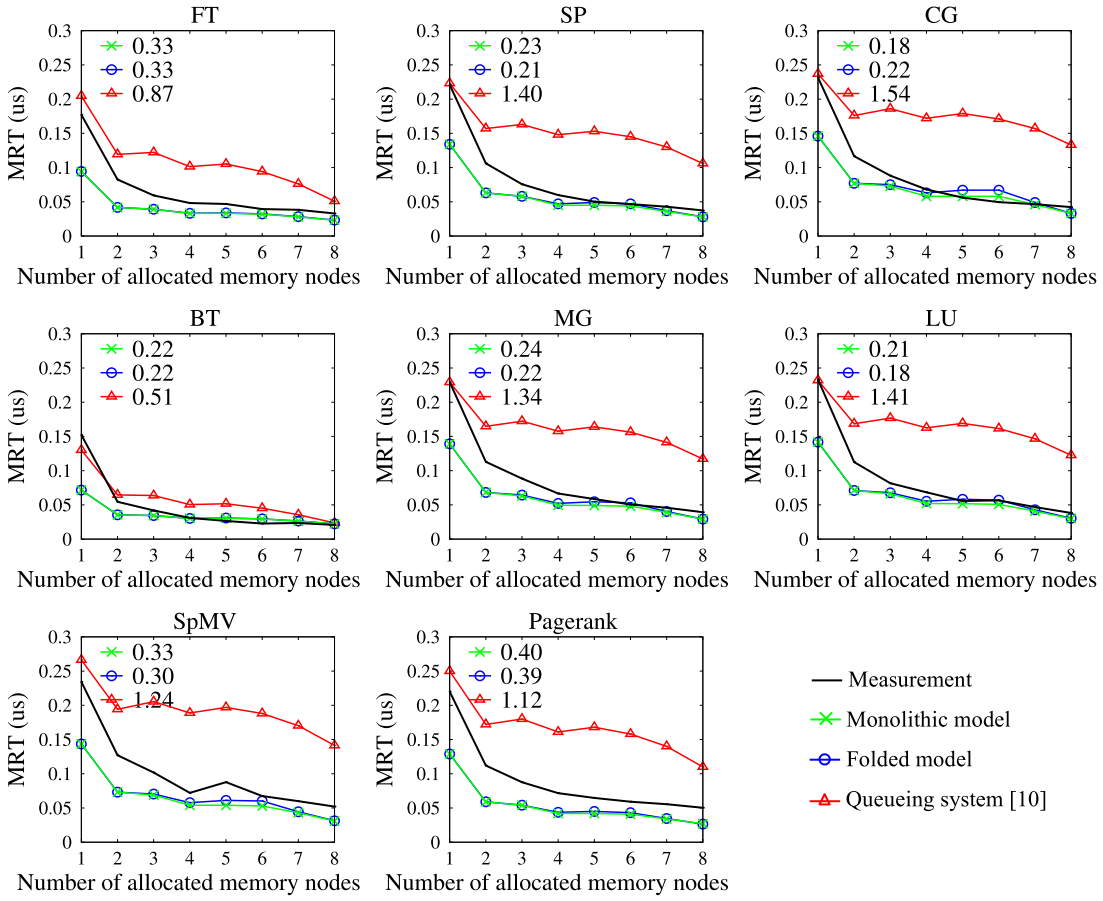

Fig. 9. MRT of memory-intensive applications for one CPU node and a varying number of memory nodes (Section 7.2). The values in the plot show the MAPE of the models against the measurements.

The queueing system-based approach fails to capture the trend of the MRT for the majority of all applications, while both the monolithic and the folded model consistently capture the trend of the MRT and outperform the queueing-based model by a significant margin. The simple architecture model of the queueing system matches the setup with one CPU node and one memory node well, while the SRN models, built for multiple CPU and memory nodes, underestimate the MRT in this case. With an increasing number of memory nodes the SRN models clearly outperform the queueing system-based approach.

7.3. Eight CPU and memory nodes, varying number of cores

The third scenario analyzes the performance of the memory system for configurations using all eight CPU and memory nodes of the 64-core AMD architecture. The application's memory data is distributed equally among the eight memory nodes, and the number of allocated CPU cores is varied from 1 to 64. Active cores are distributed to CPU nodes in a round-robin fashion, i.e., core i is allocated to CPU node ($i \bmod |\mathit{CPU\ nodes}|$); on the AMD system $|\mathit{CPU\ nodes}| = 8$. With this allocation, interference in the LLCs starts with core 9 and in the CUs with core 33. The allocation also matches the symmetry of core allocation assumed by the folded and fixed-point model.

Fig. 10 shows the results of this scenario for the eight target applications. The monolithic model can only be evaluated for up to 8 due to the state space explosion. Both approximate models are able to evaluate all configurations of the 64-core AMD system. We observe that the presented SRN models are able to model the MRT of the system with good accuracy with moderate MAPE values from 0.06 to 0.34. Small measured MRT values leading to large relative errors even for small prediction errors of around 0.01 μs are the cause for the highest MAPE of 0.34 observed for *BT* with the folded model; nevertheless, the SRN models capture the trend of the MRT for *BT* well as can be observed in Fig. 10. The SRN models consistently outperform the queueing system-based approach [10] and succeed at predicting the trend of the memory response time well. We observe that the models show a slightly larger error with a higher number of active cores; the most prominent example is the *LU* application with 48 to 64 cores. This divergence is caused by the simple interference modeling (Section 6.3) rather than a systematic limitation of the SRN models. The *LU* and *SpMV* applications generate a high volume of memory reads that lead to contention in the LLCs. Although the presented models show a higher error for *LU* and *SpMV* compared to other workloads, such as *FT*, that have less intra-node inference, the prediction accuracy is acceptable. Our approach for modeling intra-CPU node interference is based on a direct measurement and analysis methodology similar to [28]. The large shared caches of modern processors render achieving high accuracy from direct measurements a challenge. Interference modeling can potentially produce more accurate results with shared cache partitioning, however, cache partitioning requires modifications to the hardware [38] or suffers from a runtime overhead.

The generality of the presented approach is demonstrated by applying it to a 72-core Intel Xeon NUMA system comprising four Intel Xeon E7-8870 v3 processors [21] each with 18 cores and 45MB of LLC. Fig. 11 plots the results of the models for the target

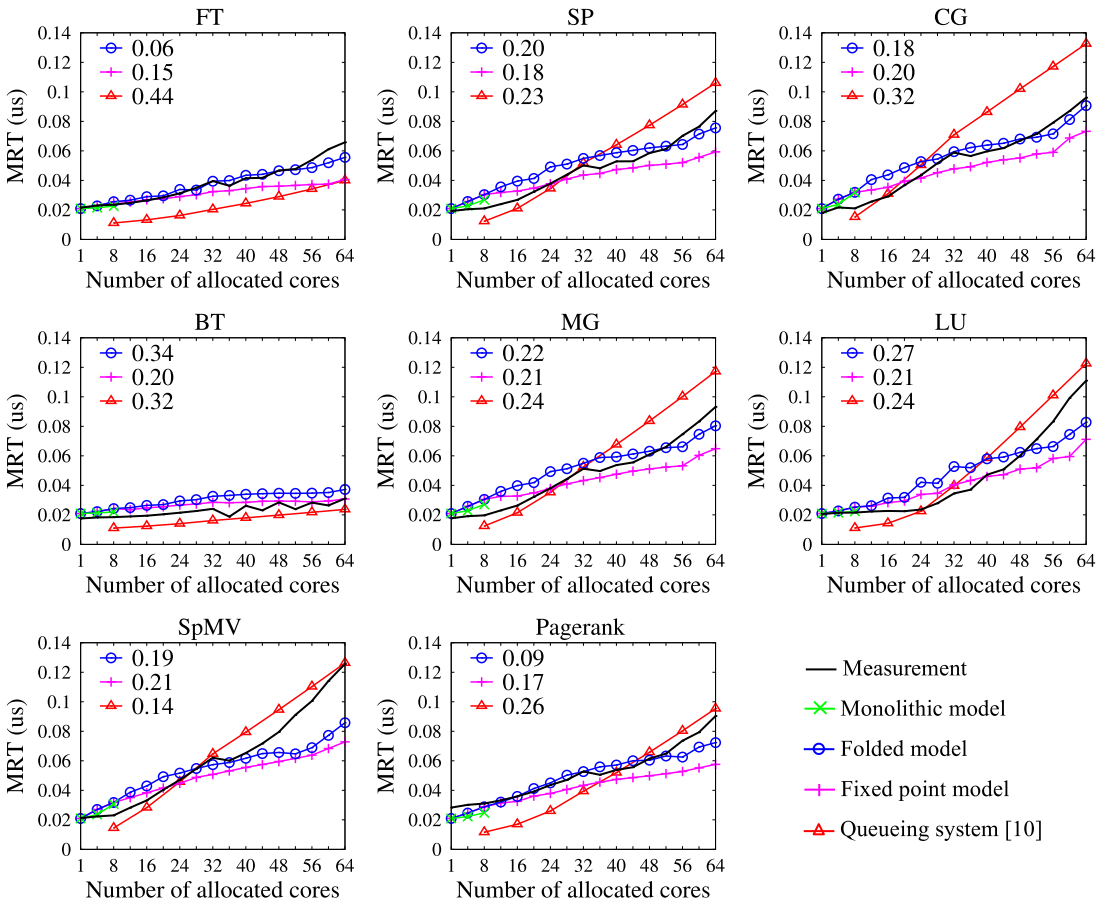

Fig. 10. MRT of memory-intensive applications for a varying number of cores with interference modeling on the 64-core AMD system. The values in the plots represent the MAPE. No MAPE is given for the monolithic model as the SPNP tool cannot solve the model for more than 8 cores.

NPB applications, SpMV (CSR format), and Pagerank (Brin and Page's algorithm [6]). Since the Intel system has more cores and fewer memory nodes compared to the AMD system (72 vs. 64 cores; 4 vs. 8 memory nodes), the peak MRT is significantly higher on the Intel platform. For most scenarios, the presented models outperform the queueing-based model with modest errors on both the AMD and Intel system, showing the strength of the presented modeling approach. Comparing the results of the two systems, we observe a larger error on the Intel platform for a number of applications such as *CG* and *SpMV*. The reason for this behavior is differences in the interconnection network and memory-level optimizations that are abstracted away by the SRN models.

7.4. Complexity analysis and speed of convergence

An important concern of SRN models is the number of states and non-zero entries of the generated Markov chains. This computational complexity directly affects the solvability of the models in terms of space and time requirements. Except for the simplest scenario with only one CPU node (Section 7.1), the monolithic model is unable to solve the other scenarios for a larger number of active cores (Figs. 8 and 10). The number of states and non-zero entries of the generated Markov chain matrices are given in Fig. 12 for the scenario from Fig. 10. The results show that the folded model reduces the complexity of the underlying Markov model and can thus handle all situations considered in the 64-core machine. Nevertheless, for high-end or future architectures with more CPU and memory nodes, the state space explosion also affects the folded model. This limitation is mitigated by the scalable fixed-point model, rendering it adequate to model large-scale systems. Fig. 13 shows the speed of convergence of the fixed-point model to the final result for 64 active cores and 8 memory nodes. In this figure, the initial values of the input parameters of the model are set to different values to show that all settings converge to the same result after only few iterations. As described in Section 5.4, we solve the interactive sub-models until the difference between the MRTs of two consecutive iterations is less than a given tolerance bound. While this bound has an influence on the number of iterations before the model stops, less than 10 iterations are sufficient to obtain the final result.

8. Modeling larger systems and future directions

This section first applies the fixed-point SRN model to large-scale NUMA systems to demonstrate the applicability of the model to predict the performance of future systems, before discussing extensions of the presented models and directions for future work.

8.1. Modeling larger systems

The scalability of the fixed-point model is a helpful attribute for users to evaluate memory performance for larger NUMA systems. Solving the fixed-point model allows us to predict the MRT with an increasing number of cores, CPU nodes, and memory nodes beyond what are physically available, providing helpful guidelines for hardware developers of future NUMA systems. Fig. 14 plots the results of the fixed-point SRN model for four different configurations. Each configuration employs a different

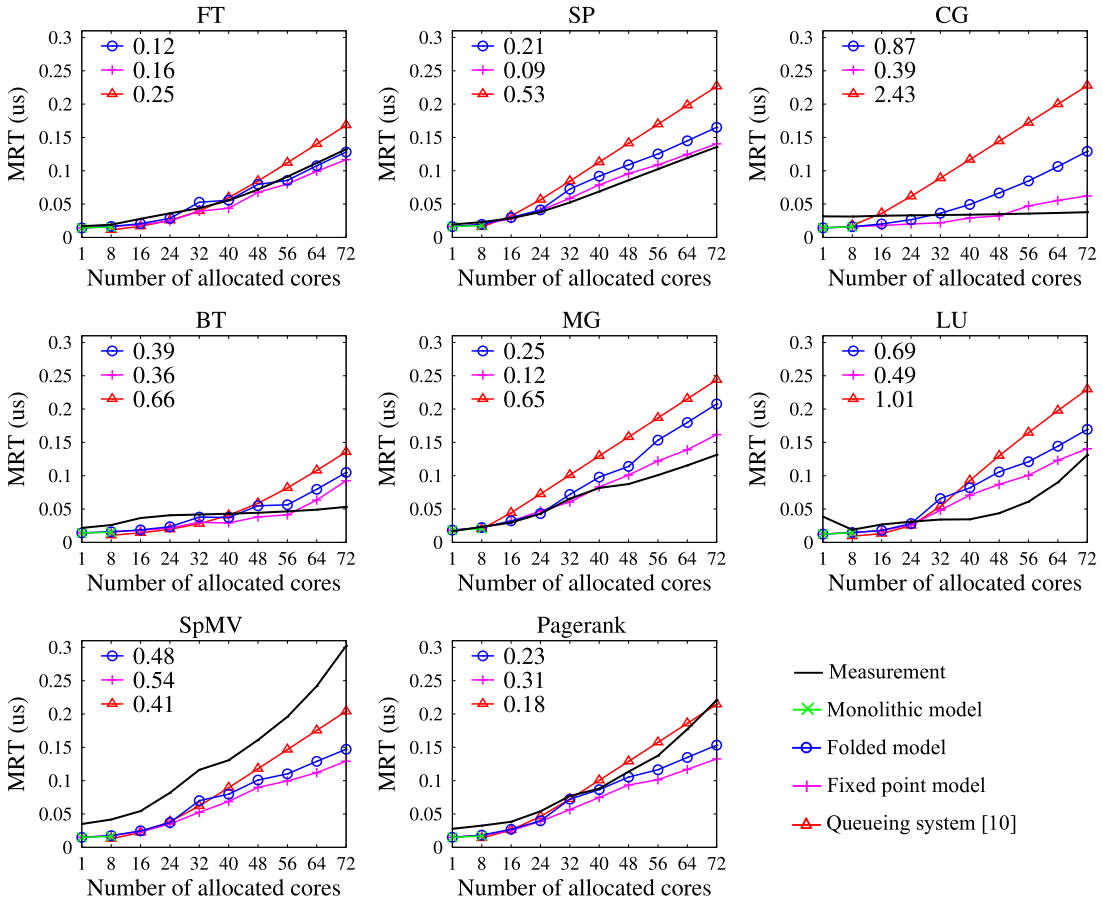

Fig. 11. MRT of memory-intensive applications for a varying number of cores with interference modeling on the 72-core Intel system. The monolithic model cannot be solved for more than 8 cores.

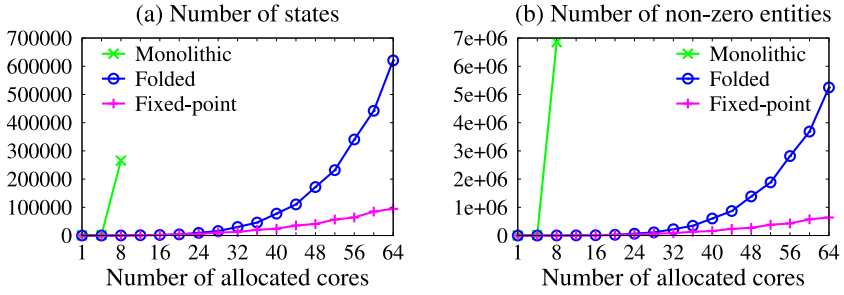

Fig. 12. The size of the Markov chains generated by the proposed SRN models.

number of cores per CPU node and/or memory nodes. We use the data transfer rate obtained on the 64-core AMD machine and the per-core memory access rates measured for memory-intensive applications in Table 6 to perform the modeling. This study provides some insights for analysts. First, comparing configurations with 8 memory nodes (configuration 1–2) to those with 16 memory nodes (configuration 3–4), we obtain the expected result that the MRT increases more slowly when more memory nodes are available. Independent of the number of memory nodes, once the memory system is saturated the growth of the MRT becomes almost linear. The presented models can thus be used to estimate when the memory system is saturated and to select an adequate number of cores for an application that does not overcommit CPU resources. Moreover, the models also make it possible to select the adequate number of memory nodes for a given application. For example, for *FT* on 64 cores, the model reveals that 8 memory nodes are sufficient because the MRT up to 64 cores is almost identical regardless of the number of memory nodes. Such an analysis assists users in selecting the adequate hardware configuration when renting computational resources in a data center. Interestingly, comparing configuration 3 (8 cores per node) and 4 (16 cores per node), we observe that for the same number of total cores the MRT increases more slowly with the simpler CPU node configuration (8 cores per node). This implies that it is better to scale-out the CPU node architecture if we can optimize an application to equally utilize all memory nodes. The same observation is not prominent between configuration 1 and 2 because the memory systems of both configurations are saturated at a relatively small number of cores.

8.2. Directions for future work

The monolithic, folded and fixed-point SRNs presented in this paper model homogeneous computing cores with NUMA memory

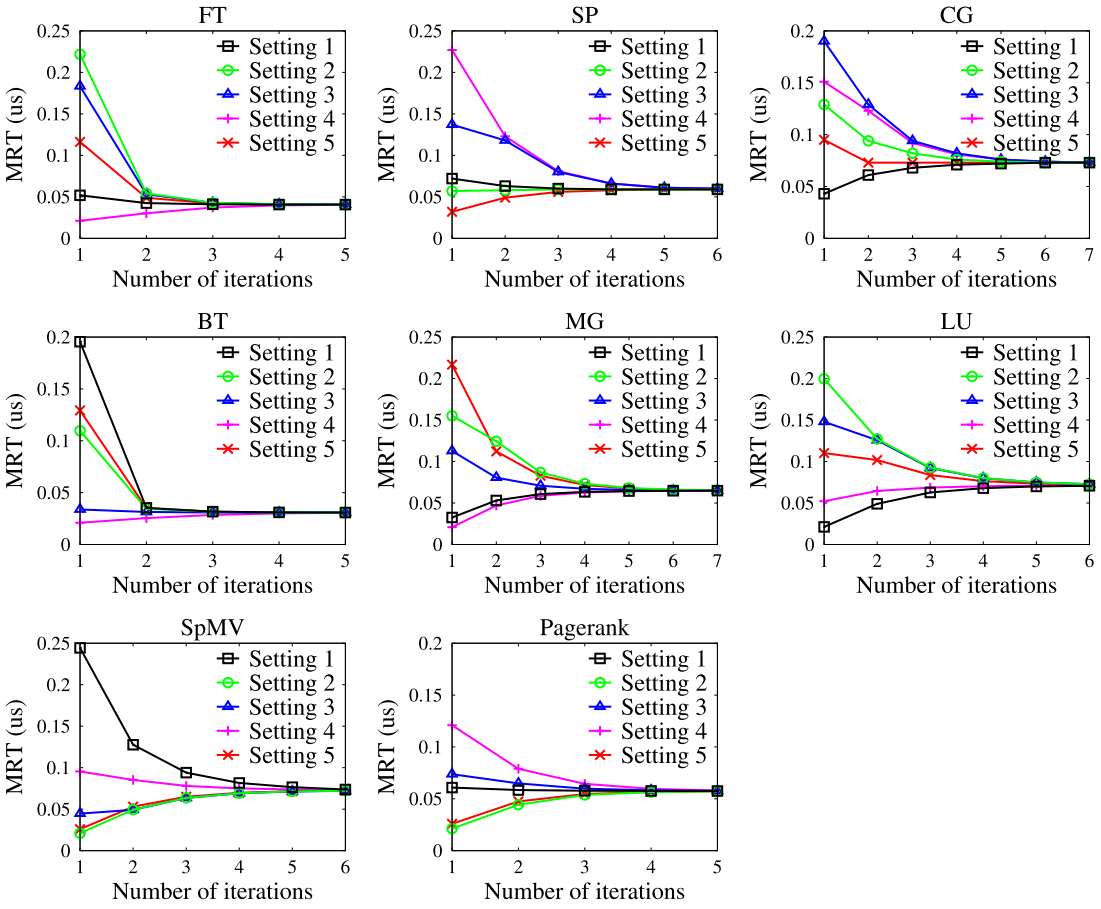

Fig. 13. The convergence speed of the proposed fixed-point model over iterations.

nodes. Although the models can appropriately model NUMA systems and evaluate the MRT of existing systems, they have some limitations. The SRN models employ an analytical approach to predict speedup, which is a simplification of a complex real-world problem that only considers the core characteristics of the system. This causes the models to show a relatively low prediction accuracy for some applications, such as SpMV and LU. Considering the behavior of these applications, it should be noted that the proposed models do not properly model the complex interactions of all parts of the system such as, for example, interference in the CUs or LLCs (see Eq. (18)). Furthermore, since the connections between the components of the SRN models are wired-down and do not change, the models can only be applied to systems with similar memory architectures. Extending the models to more accurately model interference in the CUs and LLC is one possible direction for future work.

A second open research problem is to extend this approach to heterogeneous computing systems and distributed systems. Referring to the monolithic model in Fig. 3, we observe that there is no assumption about the physical location of the memory nodes; the timed transitions T_{mem_i} model the time required to access a memory node. To model distributed memory systems, these timed transitions need to be replaced by a series of places, arcs, and transitions that model physically remote memory accessible through a network. The processing speed of a CPU core is modeled using the timed transitions T_{cpu_i}. To accommodate heterogeneous processing cores, the rate of these transitions can be adjusted to represent the different processing speed of the cores.

Another future direction of research applies the presented models to maximize the performance of parallel applications. The models can help to determine the optimal number of cores that maximizes the overall performance of an application. Another possibility is to employ the proposed SRN models in a job scheduler. The scheduler can base its scheduling decisions on the results of the models predicting the performance and/or utilization of the memory system.

9. Conclusions

Understanding memory performance in multi-core platforms is a prerequisite to perform optimizations. This paper has presented an analytical approach based on SRNs to model and evaluate the performance of memory accesses in NUMA architectures. A monolithic model considering the entire system provides the intuition to understand memory accesses in the system. For practical applications of the proposed approach, two approximation models are proposed for larger numbers of CPU cores and memory nodes. The validation of the models on a 64-core AMD Opteron server and a 72-core Intel platform and a comparison to a previously published NUMA performance model shows that the proposed SRN models are able to accurately evaluate memory performance of existing NUMA machines.

The input parameters of the models are architecture- and application-specific. The architecture-specific parameters, including data transfer rates of interconnection links and the service rate of the memory controllers, can be measured by running a synthetic workload. The application-specific parameters, such as the LLC miss rate per core, is obtained by profiling an application. Based on these values, the proposed SRN models can evaluate the performance of memory accesses on various NUMA system configurations.

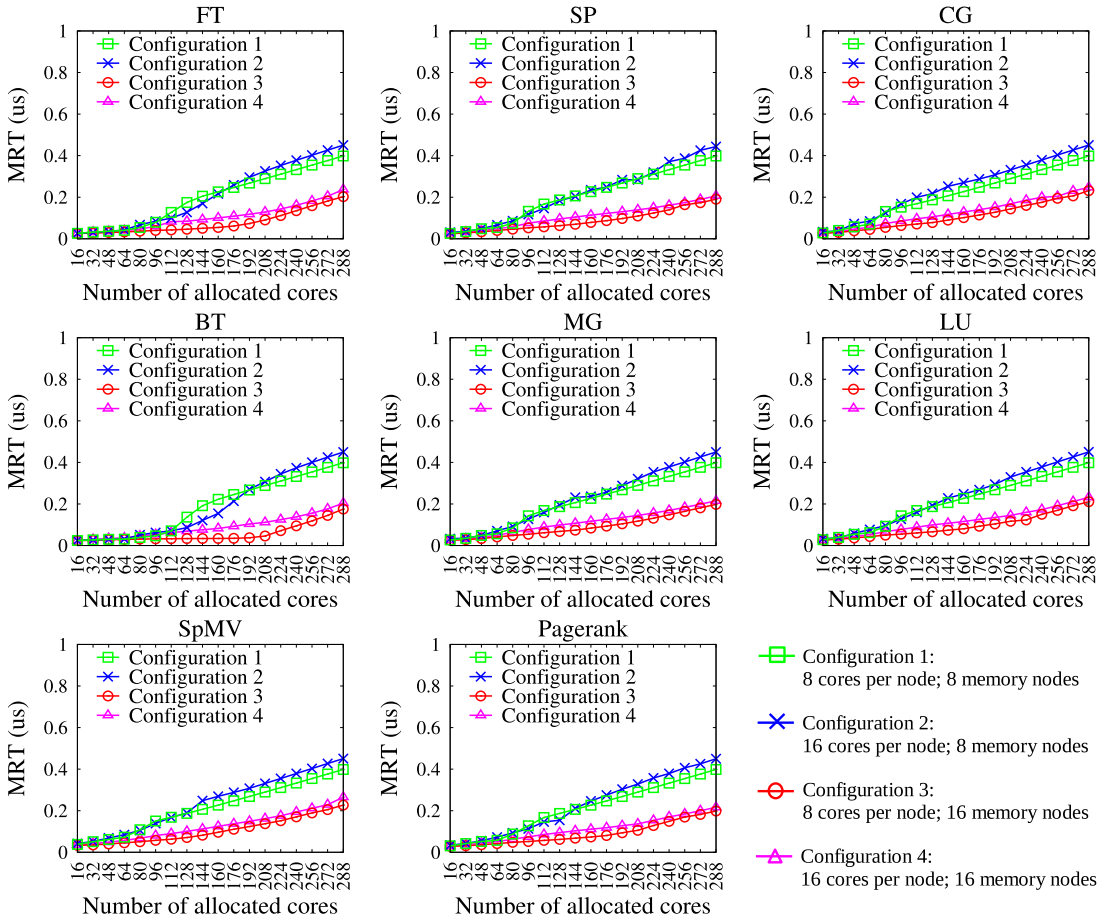

Fig. 14. MRT of memory-intensive applications for larger systems and different NUMA system configurations.

CRediT authorship contribution statement

Reza Entezari-Maleki: Methodology, Software, Validation, Resources, Data curation, Visualization, Writing - original draft, Writing - review & editing. **Younghyun Cho:** Software, Validation, Resources, Data curation, Visualization, Writing - original draft, Writing - review & editing. **Bernhard Egger:** Conceptualization, Methodology, Resources, Writing - original draft, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) funded by the Korean government, in part, by grants NRF-2015K1A3A1A14021288, 2016R1A2B4009193, by the BK21 Plus for Pioneers in Innovative Computing (Dept. of Computer Science and Engineering, SNU, grant 21A20151113068), and by the Promising-Pioneering Researcher Program of Seoul National University in 2015. The visit of Dr. Reza Entezari-Maleki was funded by the Foreign Scholar Invitation Program of Seoul National University in 2019. ICT at Seoul National University provided research facilities for this study.

References

[1] AMD, AMD Opteron 6300 Series Processors, 2020, https://www.amd.com/en-us/products/server/opteron/6000/6300, [online; accessed 2020].

[2] AMD, BIOS and kernel developer's guide (BKDG) for AMD family 15h models 00h-0fh processors, 2020, https://www.amd.com/system/files/TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf, [online; accessed 2020].

[3] E. Ataie, R. Entezari-Maleki, L. Rashidi, K.S. Trivedi, D. Ardagna, A. Movaghar, Hierarchical stochastic models for performance, availability, and power consumption analysis of IaaS clouds, IEEE Trans. Cloud Comput. 7 (4) (2019) 1039–1056.

[4] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, S.K. Weeratunga, The NAS parallel benchmarks, Int. J. Supercomput. Appl. 5 (3) (1991) 63–73.

[5] F. Bause, P.S. Kritzinger, Stochastic Petri Nets: An Introduction to the Theory, second ed., Vieweg+Teubner Verlag, 2002.

[6] S. Brin, L. Page, Reprint of: The anatomy of a large-scale hypertextual web search engine, Comput. Netw. 56 (18) (2012) 3825–3833.

[7] J. Celko, Chapter 13 - petri nets, in: Joe Celko's Trees and Hierarchies in SQL for Smarties, second ed., in: The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, Boston, 2012, pp. 239–244.

[8] X. Chang, R. Xia, J.K. Muppala, K.S. Trivedi, J. Liu, Effective modeling approach for IaaS data center performance analysis under heterogeneous workload, IEEE Trans. Cloud Comput. 6 (4) (2018) 991–1003.

[9] Y. Cho, C.A.C. Guzman, B. Egger, Maximizing system utilization via parallelism management for co-located parallel applications, in: Proceedings of the 2018 International Conference on Parallel Architectures and Compilation (PACT), ACM, Limassol, Cyprus, 2018.

[10] Y. Cho, S. Oh, B. Egger, Online scalability characterization of data-parallel programs on many cores, in: Proceedings of The 2016 International Conference on Parallel Architectures and Compilation (PACT), Haifa, Israel, 2016, pp. 191–205.

[11] Y. Cho, S. Oh, B. Egger, Performance modeling of parallel loops on multi-socket platforms using queueing systems, IEEE Trans. Parallel Distrib. Syst. 31 (2) (2020) 318–331.

[12] G. Ciardo, A. Blakemore, P.F. Chimento, J.K. Muppala, K.S. Trivedi, Automated generation and analysis of Markov reward models using stochastic reward nets, in: C.D. Meyer, R.J. Plemmons (Eds.), Linear Algebra, Markov Chains, and Queueing Models, Springer, 1993, pp. 145–191.
[13] G. Ciardo, J.K. Muppala, K.S. Trivedi, SPNP: stochastic Petri net package, in: The 3rd International Workshop on Petri Nets and Performance Models (PNPM), Kyoto, Japan, 1989, pp. 142–151.
[14] M. Curtis-Maury, F. Blagojevic, C.D. Antonopoulos, D.S. Nikolopoulos, Prediction-based power-performance adaptation of multithreaded scientific codes, IEEE Trans. Parallel Distrib. Syst. 19 (10) (2008) 1396–1410.
[15] L. Dagum, R. Enon, OpenMP: an industry standard API for shared-memory programming, IEEE Comput. Sci. Eng. 5 (1) (1998) 46–55.
[16] T.T. Dao, J. Kim, S. Seo, B. Egger, J. Lee, A performance model for GPUs with caches, IEEE Trans. Parallel Distrib. Syst. 26 (7) (2015) 1800–1813.
[17] R. Entezari-Maleki, K.S. Trivedi, A. Movaghar, Performability evaluation of grid environments using stochastic reward nets, IEEE Trans. Dependable Secure Comput. 12 (2) (2015) 204–216.
[18] A. Furfaro, L. Nigro, F. Pupo, Distributed simulation of timed colored Petri nets, in: The 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications, TX, USA, 2002, pp. 159–166.
[19] S. Hong, H. Kim, An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness, in: The 36th Annual International Symposium on Computer Architecture (ISCA), Austin, TX, 2009, pp. 152–163.
[20] O.C. Ibe, H. Choi, K.S. Trivedi, Performance evaluation of client–server systems, IEEE Trans. Parallel Distrib. Syst. 4 (11) (1993) 1217–1229.
[21] Intel, Intel Xeon Processor E7-8870 V3, 2020, http://ark.intel.com/products/84682/Intel-Xeon-Processor-E7-8870-v3-45M-Cache-2_10-GHz, [online; accessed 2020].
[22] H. Jonkers, Queueing models of shared-memory parallel applications, in: Proceedings of UK Performance Engineering Workshop for Computer and Telecommunication Systems, Loughborough, UK, 1993.
[23] H. Jonkers, Queueing models of parallel applications: The Glamis methodology, in: G. Haring, G. Kotsis (Eds.), International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, in: Lecture Notes in Computer Science, vol. 794, Springer, 1994, pp. 123–138.
[24] M. Kim, P. Kumar, H. Kim, B. Brett, Predicting potential speedup of serial code via lightweight profiling and emulations with memory performance model, in: The IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS), Shanghai, China, 2012, pp. 1318–1329.
[25] D. Kroft, Lockup-free instruction fetch/prefetch cache organization, in: Proceedings of The 8th Annual Symposium on Computer Architecture (ISCA), Minneapolis, Minnesota, USA, 1981, pp. 81–87.
[26] A. Lastovetsky, R.R. Manumachu, New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2017) 1119–1133.
[27] F. Liu, X. Jiang, Y. Solihin, Understanding how off-chip memory bandwidth partitioning in chip multiprocessors affects system performance, in: The IEEE 16th International Symposium on High-Performance Computer Architecture (HPCA), Bangalore, India, 2010, pp. 1–12.
[28] J. Mars, L. Tang, M.L. Soffa, Directly characterizing cross core interference through contention synthesis, in: Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers, in: HiPEAC '11, ACM, New York, NY, USA, 2011, pp. 167–176.
[29] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modeling with Generalized Stochastic Petri Nets, first ed., John Wiley & Sons, 1995.
[30] M.A. Marsan, G. Conte, G. Balbo, A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems, ACM Trans. Comput. Syst. 2 (2) (1984) 93–122.
[31] J.D. McCalpin, STREAM: Sustainable Memory BandWidth in High Performance Computers, Tech. rep., University of Virginia, Charlottesville, Virginia, a continually updated technical report (1991-2007). URL http://www.cs.virginia.edu/stream/.
[32] J.K. Muppala, K.S. Trivedi, Composite performance and availability analysis using a hierarchy of stochastic reward nets, in: G. Balbo, G. Serazzi (Eds.), Computer Performance Evaluation, Modelling Techniques and Tools, Elsevier Science Publishers B.V. (North-Holland), 1992, pp. 335–349.
[33] B. Mutnury, F. Paglia, J. Mobley, G.K. Singh, R. Bellomio, QuickPath Interconnect (QPI) design and analysis in high speed servers, in: The IEEE 19th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), Austin, TX, 2010, pp. 265–268.
[34] L. Nai, Y. Xia, I.G. Tanase, H. Kim, C.-Y. Lin, Graphbig: understanding graph computing in the context of industrial solutions, in: SC15: International Conference for High Performance Computing, Networking, Storage and Analysis, 2015, pp. 1–12, http://dx.doi.org/10.1145/2807591.2807626.
[35] J.L. Peterson, Petri Net Theory and the Modeling of Systems, first ed., Prentice Hall, 1981.
[36] S. Rixner, W.J. Dally, U.J. Kapasi, P. Mattson, J.D. Owens, Memory access scheduling, in: Proceedings of the 27th Annual International Symposium on Computer Architecture, in: ISCA '00, ACM, New York, NY, USA, 2000, pp. 128–138.
[37] G. Sartori, Hypertransport Technology, Platform Conference.
[38] V. Selfa, J. Sahuquillo, S. Petit, M.E. Gómez, A hardware approach to fairly balance the inter-thread interference in shared caches, IEEE Trans. Parallel Distrib. Syst. 28 (11) (2017) 3021–3032.
[39] S. Seo, G. Jo, J. Lee, Performance characterization of the NAS parallel benchmarks in OpenCL, in: The IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, 2011, pp. 137–148.
[40] L.A. Tomek, K.S. Trivedi, Fixed point iteration in availability modeling, in: M.D. Cin, W. Hohl (Eds.), Fault-Tolerant Computing Systems, in: Informatik-Fachberichte, vol. 283, Springer, 1991, pp. 229–240.
[41] K.S. Trivedi, Probability & Statistics with Reliability, Queuing and Computer Science Applications, second ed., John Wiley & Sons, 2008.
[42] T.-F. Tsuei, W. Yamamoto, Queuing simulation model for multiprocessor systems, Computer 36 (2) (2003) 58–64.
[43] B.M. Tudor, Y.M. Teo, S. See, Understanding off-chip memory contention of parallel programs in multicore systems, in: The International Conference on Parallel Processing (ICPP), Taipei, Taiwan, 2011, pp. 602–611.
[44] W. Wang, J.W. Davidson, M.L. Soffa, Predicting the memory bandwidth and optimal core allocations for multi-threaded applications on large-scale NUMA machines, in: The IEEE 22nd International Symposium on High Performance Computer Architecture (HPCA), Barcelona, Spain, 2016, pp. 419–431.
[45] W. Wang, T. Dey, J.W. Davidson, M.L. Soffa, DraMon: Predicting memory bandwidth usage of multi-threaded programs with high accuracy and low overhead, in: The IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), Orlando, FL, 2014, pp. 380–391.

Reza Entezari-Maleki received the B.S. and M.S. degrees from the Iran University of Science and Technology, Tehran, Iran, in 2007 and 2009, respectively, and the Ph.D. degree from the Sharif University of Technology, Tehran, in 2014, all in computer engineering (software). He worked as a post-doctoral researcher at the School of Computer Science, Institute for Research in Fundamental Sciences (IPM) from 2015 to 2018. He is currently an assistant professor in Iran University of Science and Technology and an associate researcher at INESC-ID, Instituto Superior Tecnico (IST), Universidade de Lisboa. His main research interests are performance/dependability modeling and evaluation, distributed computing systems, cloud computing and task scheduling algorithms.

Younghyun Cho received the B.S. degree in computer science from the University of Seoul in 2013. He is currently working toward his Ph.D. degree at the Computer Systems and Platforms Laboratory at Seoul National University. His research interests include runtime systems for manycore systems and performance modeling of parallel applications.

Bernhard Egger received the diploma in computer science from the Swiss Federal Institute of Technology, Zürich (ETHZ) in 2001 and the PhD degree in computer science and engineering from Seoul National University in 2008. He worked as a senior research engineer at SAIT, Samsung Electronic's research institute, from 2008 to 2011. He joined Seoul National University as a faculty member in 2011 where he currently is a professor in the Department of Computer Science and Engineering. His research interests include programming language design, compilers, and operating systems for heterogeneous manycore systems. He is a member of the IEEE and ACM. More information is available at https://csap.snu.ac.kr/.