Predicting the Memory Bandwidth and Optimal Core Allocations for Multi-threaded Applications on Large-scale NUMA Machines

Wei Wang, Jack W. Davidson, and Mary Lou Soffa
Department of Computer Science
University of Virginia
{wwang, jwd, soffa}@virginia.edu

ABSTRACT

Modern NUMA platforms offer large numbers of cores to boost performance through parallelism and multi-threading. However, because performance scalability is limited by available memory bandwidth, the strategy of allocating all cores can result in degraded performance. Consequently, accurately predicting optimal (best performing) core allocations, and executing applications with these allocations are crucial for achieving the best performance.

Previous research focused on the prediction of optimal numbers of cores. However, in this paper, we show that, because of the asymmetric NUMA memory configuration and the asymmetric application memory behavior, optimal core allocations are not merely optimal numbers of cores. Additionally, previous studies do not adequately consider NUMA memory resources, which further limits their ability to accurately predict optimal core allocations.

In this paper, we present a model, NuCore, which predicts both memory bandwidth usage and optimal core allocations. NuCore considers various memory resources and NUMA asymmetry, and employs Integer Programming to achieve high accuracy and low overhead. Experimental results from real NUMA machines show that the core allocations predicted by NuCore provide 1.27x average speedup over using all cores with only 75.6% cores allocated. NuCore also provides 1.18x and 1.21x average speedups over two state-of-the-art techniques. Our results also show that NuCore faithfully models NUMA memory systems and predicts memory bandwidth usages with only 10% average error.

1. INTRODUCTION

Non-uniform memory access (NUMA) platforms are widely used in data centers and high performance computing centers, because they offer large numbers of cores and high memory bandwidth, allowing the simultaneous execution of massive numbers of threads to achieve high performance. However, the performances of multi-threaded applications do not always scale linearly with the increasing numbers of cores on NUMA machines [1, 2]. In fact, for certain multi-threaded applications, the scalability is so limited that allocating all available cores of a NUMA machine hurts performance. For example, we observed that on a 48-core NUMA machine, the PARSEC benchmark *streamcluster* achieves the best performance using only six cores – the execution time when using six cores is 3.34 times faster than using all 48 cores [3]. The observation that allocating all cores may degrade performance is also reported on many other NUMA platforms [4, 5, 6, 7, 8, 9].

Because the potential performance penalty of allocating too many cores on a NUMA machine is significant, it is important that we execute multi-threaded applications with the minimum core allocation that provides the best performance. We call such a core allocation the *optimal core allocation*.[1]

Although previous studies investigated this core allocation problem, these studies did not accurately determine the optimal core allocations for two reasons:

1) Previous work only focused on determining the optimal numbers of cores [4, 5, 6, 7, 8, 9]. However, because of the asymmetric memory configuration and asymmetric application memory behavior, optimal core allocations are not simply optimal numbers of cores. A NUMA machine consists of several memory nodes, where each node has its own DRAM connection. The nodes are connected using inter-node links to allow inter-node communications. However, the inter-node links within a NUMA machine can have different bit-widths and maximum bandwidths. The connection topology of these links may also be non-uniform. Moreover, a multi-threaded application's memory behavior is also asymmetric in that it typically does not use every memory link similarly. This hardware and software asymmetry demands each node to be treated differently when determining optimal core allocations.

2) Furthermore, prior work did not adequately consider NUMA memory resources and factors. We observed that on large-scale NUMA platforms, memory bandwidth is the primary scalability limitation that impacts optimal core alloca-

[1]Note that, in this paper, we assume one thread per core. Therefore, we use core and thread inter-changeably.

tions. The maximum memory bandwidth usage of an application is affected by many memory system factors, such as inter-node connections and memory contention. Currently, many of these factors cannot be accurately and quantitatively evaluated, which further limits the ability to accurately determine the optimal core allocations.

In this paper, we present a novel model, NuCore, which predicts both optimal core allocations and memory bandwidth usages with high accuracy and low overhead for memory-intensive multi-threaded applications on large-scale NUMA machines. NuCore predicts optimal core allocations by modeling various NUMA memory factors and considering the asymmetry. We first analyzed optimal core allocations of different multi-threaded applications on real NUMA machines to determine the memory system factors that impact optimal core allocation. These factors include local DRAM accesses, the maximum bandwidth of inter-node links, and the contention among local and inter-node memory accesses. We then developed methods to quantitatively evaluate the impacts of these factors with constraint functions.

To handle the asymmetry and achieve fast prediction, we employed Integer Programming (IP). We express the memory system factors for each link and node with its own constraint functions so that they can be considered differently. We also developed a technique to express non-linear memory contention factors with linear functions. Additionally, we express the prediction of optimal core allocations with a linear objective function. Our model, NuCore, combines the linear constraints and objective function, and predicts optimal core allocations using an IP solver. Additionally, because NuCore extensively models memory resources, it can also predict the memory bandwidth usage of a multi-threaded application when it is executing with an arbitrary core allocation.

We evaluated our model on two NUMA platforms. Experimental results show that NuCore can correctly predict the optimal core allocations of 22 out of 25 benchmarks from various benchmark suites. For the other three benchmarks, NuCore's predictions differed by at most one core per node from the experimentally determined optimal core allocations. The predicted optimal core allocations perform within 1.0% of real optimal allocations on average. The average speedup of predicted core allocations is 1.27x, with only 75.6% of all cores allocated on average. Moreover, the core allocations predicted by NuCore perform 1.18x and 1.21x faster than two state-of-the-art techniques on average. The results also show that NuCore can be solved with a IP solver in 0.02 seconds, indicating that NuCore can be applied at run-time. Our results also show that NuCore can predict bandwidth usages with only 10% average error for memory-intensive benchmarks.

The contributions of this paper include:

1. A detailed analysis of the optimal core allocations on real large-scale NUMA machines. The analysis reveals the memory factors that impact optimal core allocations and illustrates the necessity of considering the asymmetry of NUMA systems.

2. An IP-based model that predicts both memory bandwidth usage and optimal core allocation with high accuracy and low overhead.

3. Comprehensive experimental evaluations using two large real NUMA machines, 25 benchmarks, and Extreme Value Theory to validate NuCore's accuracy and benefit.

This paper is organized as follows: Section 2 provides the motivation for this research; section 3 discusses the asymmetric property of NUMA platforms; section 4 analyzes various optimal core allocations; section 5 provides an overview of NuCore model; section 6 describes NuCore in detail; section 7 evaluates NuCore experimentally; section 8 discusses limitations and future work; section 9 discusses related work; and section 10 summarizes the work.

2. MOTIVATION

We first provide the motivation for this work with an example.

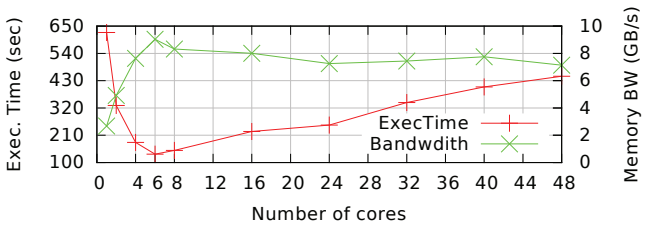


Figure 1: Performance and memory bandwidth usage of *streamcluster* on an AMD NUMA machine.

Figure 1 shows the performance of the PARSEC benchmark *streamcluster* using different core allocations on a NUMA machine [3]. This machine has four AMD 12-core processors, and each processor has two memory nodes. As Figure 1 shows, *streamcluster* achieves its best performance using only six cores.

Using performance counters, we discovered that the increased CPU time is primarily from increased memory latency. Because the cache miss rate (misses per instruction) of *streamcluster* remains constant, the increased memory latency is caused by saturated memory bandwidth. Figure 1 also gives the bandwidth usages of each core allocation, which also peaks using six cores.

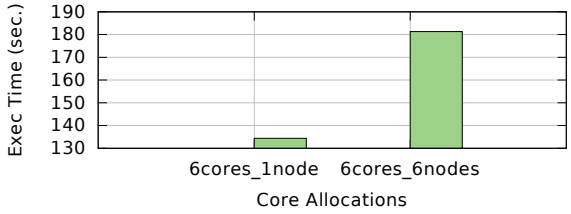


Figure 2: Performance of *streamcluster* with two 6-core allocations. "6c_1n": all six cores allocated from on node; "6c_6n": six cores each from a node.

However, not every six-core allocation is the optimal. Figure 2 gives the performance of *streamcluster* using two different six-core allocations. The first core allocation allocates

all six cores from one memory node, whereas the second core allocation allocates a core from six memory nodes. As Figure 2 shows, only the first allocation is the optimal. The first allocation is 1.34 times faster than the second one. This result indicates that the optimal core allocation is not only about the number of cores, but also about the core allocation of each individual memory node. In fact, without knowing per-node core allocation, it is not possible to determine the optimal number of cores currently.

This example illustrates the significant benefit of optimal core allocations. It also shows that memory bandwidth is a major scalability limitation, and that optimal core allocation is not simply the optimal number of cores. In the next two sections, we further explore NUMA memory system and analyze optimal core allocations to study how to accurately predict them while considering the core allocation of each node differently.

3. THE ASYMMETRIC NUMA ARCHITECTURE

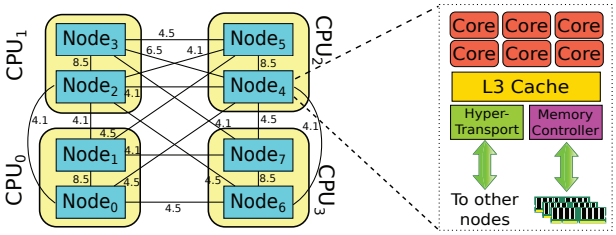


Figure 3: A NUMA machine with four 12-core AMD Opteron 6174 processors. The numbers on the inter-node links represents their maximum bandwidth (GB/s).

Contemporary large-scale NUMA platforms consist of several multi-core processors. A multi-core processor is usually composed of one or more groups of cores, which are called nodes. Each node is connected to its own set of DRAMs. The nodes are also connected using inter-node connections, allowing inter-node communication [10, 11]. Figure 3 gives the sketch of a NUMA machine with four AMD Opteron 6174 processors (a total of 48 cores) [12]. Each processor has two six-core nodes.

The connections between nodes in Figure 3 are the inter-node connections (HyperTransport links [11]). As Figure 3 shows, the inter-node connections on this NUMA platform is not symmetric. More specifically, the asymmetry exists in two aspects:

1) the inter-node connection topology is asymmetric in that not all nodes are directly connected. Some nodes are directly connected (e.g., node 1 and 2), whereas some are connected through two physical links (e.g., node 0 and 7);

2), the maximum bandwidth of inter-node connections is asymmetric in that different connections have different maximum bandwidth. Figure 3 also gives the maximum bandwidth (GB/s) of each inter-node connection (measured using the bandwidth measurement tool *bw_mem* from *lmBench* [13]), which ranges from 8.5GB/s to 4.1GB/s.

The fundamental cause of this asymmetry is that each node has a limited number of physical inter-node links. On this AMD platform, each node has only four physical links [11]. Consequently, there are not enough physical links to ensure fully connected topology. Additionally, to accommodate more direct connections, a physical link may be partitioned into two inter-node connections. Depending on the partition, the bit-widths of inter-node connections vary from 32 bits to 16 bits, which causes the different bandwidths.

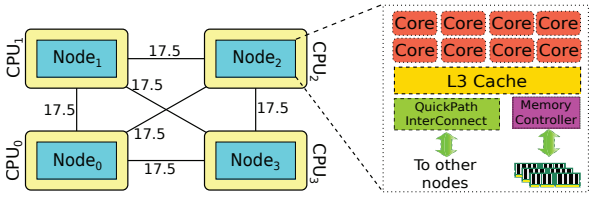


Figure 4: A NUMA machine with four 8-core Intel Xeon X7550 processors. The numbers on the inter-node links represents their maximum bandwidth (GB/s).

Figure 4 gives another example of a NUMA machine with four Intel Xeon X7550 processors, where each processor has one eight-core node (32 cores in total). Figure 4 shows the connection topology and maximum bandwidth of the inter-node connections. On this Intel platform, the nodes are connected with QuickPath Interconnect (QPI) [10]. Because this Intel NUMA machine has only four nodes, there are enough physical QPI links so that all nodes are directly connected, and all links have the same maximum bandwidth. Note that, because the asymmetry also exists in application memory behavior, even NUMA machines with symmetric connections have to treat each node differently as shown later in Section 4.3.1.

4. ANALYSIS OF OPTIMAL CORE ALLOCATIONS

We first analyzed the optimal core allocations to reveal the NUMA memory resources and factors that impacts optimal core allocations. Our experiments show that there are three limitations in NUMA memory systems that impact scalability and core allocations.

A core allocation is described using a vector $\{a_0, a_1, a_2, \cdots, a_i, \cdots, a_n\}$, where a_i represents the number of cores allocated on node i.

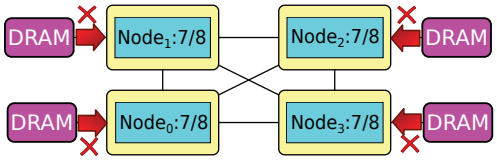


Figure 5: Benchmark *md.D-resid* is limited by local memory bandwidth. Values "X/Y" give the optimal allocation (X) and maximum core count (Y) for each node. A "cross" indicates a saturated memory link.

4.1 Limitation 1: Local Memory BW

Local memory bandwidth usage refers to the bandwidth consumed by processor cores (and their executing threads)

when they access their directly connected DRAM modules. If the local memory bandwidth demand is high, the local DRAM bandwidth can impact optimal core allocations.

For example, the core allocation for the NPB benchmark *mg.D* on the 32-core Intel NUMA machine is limited by local memory bandwidth when executing the function *resid* [14]. The optimal core allocation for *mg.D-resid* and the saturated memory connections are illustrated in Figure 5. As the figure shows, the optimal core allocation for *md.D-resid* is $\{7,7,7,7\}$, i.e., 7 cores for each node. This allocation is 10% faster than using all 32 cores. *Mg.D* has only local memory accesses when executing the function *resid*. However, the DRAMs cannot provide enough bandwidth. That is, the DRAM bandwidth is only enough to support seven cores on each node.

4.2 Limitation 2: Inter-node Memory BW

Inter-node memory bandwidth usage refers to the data usage for inter-node communications. Inter-node connections have limited maximum memory bandwidth. Consequently, if the inter-node communication has high bandwidth demand, than the inter-node connections may limit scalability.

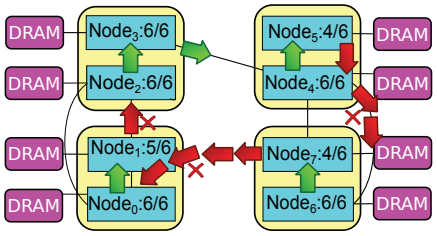


Figure 6: NPB benchmark *mg.D-rprj3* is limited by inter-node memory bandwidth. Values "X/Y" give the optimal allocation (X) and maximum core count (Y) for each node. A "cross" indicates a saturated memory connection.

For example, the NPB benchmark *mg.D* has intensive inter-node communications when it is in the phase of executing the function *rprj3* [14]. During this phase, *mg.D*'s threads communicate in a ring fashion as illustrated in Figure 6, i.e., the threads on $node_0$ send data to $node_1$, the threads on $node_1$ send data to $node_2$, and so on.

The optimal core allocation of *mg.D-rprj3* is significantly impacted by inter-node connections and their asymmetry (differences in maximum bandwidth as shown in Figure 3). The cores (and their threads) on $node_0$, $node_2$, $node_3$, $node_4$, and $node_6$ send data through high-bandwidth inter-node links. Therefore, all cores on these nodes can be allocated. However, the connection between $node_1$ and $node_2$ has lower bitwidth and smaller bandwidth. Therefore, only five cores can be allocated on $node_1$. $Node_5$ and $node_7$ have to send data using two-hop connections, which have even smaller maximum bandwidth. Consequently, only four cores on each of $node_5$ and $node_7$ can be allocated. In summary, the optimal core allocation for this case is $\{6,5,6,6,6,4,6,4\}$ on the AMD platform, which is 9% faster than using all 48 cores.

4.3 Limitation 3: Interference of Local and Inter-node Memory Accesses

Local memory accesses and inter-node memory accesses usually exist in a NUMA machine at the same time and interfere with each other. This interference can also impact optimal core allocations. Because this interference varies with data locations, there are two cases to consider.

4.3.1 Case 1: Fully Shared Data

For some multi-threaded applications, all of their data are shared by their threads. The data are usually held in the DRAMs of a few nodes. Without loss of generality, consider the case where all shared-data are in $node_i$. Because all data are in $node_i$, $node_i$'s DRAM has two tasks: 1) sending data to $node_i$'s cores; 2) sending data to other nodes. However, if the data demand is high, the maximum output bandwidth of $node_i$'s DRAM may not be large enough to satisfy the needs of all cores, i.e., not all cores can be allocated. Because all data are located on $node_i$, it is best to satisfy the data demand of $node_i$'s cores first. After $node_i$'s cores' demand is met, we can use the remaining bandwidth to satisfy the needs of some cores on the other nodes.

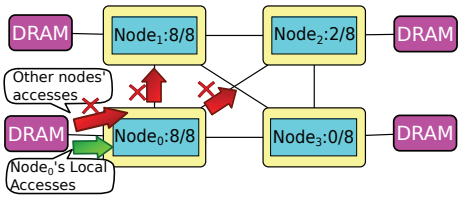


Figure 7: Benchmark *streamcluster* is limited by local and inter-node access interference. Values "X/Y" give the optimal allocation (X) and maximum core count (Y) for each node. A "cross" indicates a saturated memory connection.

For example, the PARSEC benchmark *streamcluster* has this all-data-shared behavior [3]. Figure 7 gives the optimal core allocation for *streamcluster* and the saturated memory connections when it executes on the 32-core Intel NUMA machine. As Figure 7 shows, the maximum output memory bandwidth of $node_0$, which contains shared data, cannot satisfy the needs of all cores. After meeting the demand of all eight cores on $node_0$, the remaining bandwidth can only support the execution of 10 cores on the other nodes. Therefore, the optimal core allocation for *streamcluster* is $\{8,8,2,0\}$, which is 79% faster than the use-all-cores allocation. This example also shows that even for NUMA machines with symmetric connections, each node still has to be treated differently due to software asymmetry.

4.3.2 Case 2: Partially Shared Data

For some applications, the majority of their data are private to their threads, while only a small portion of the data is shared. That is, most data are distributed across the nodes, whereas the shared data are usually located on a few nodes. Without loss of generality, let the node with shared-data be $node_i$. Similar to case 1, $node_i$ has to send data to $node_i$'s cores, as well as the cores on the other nodes. If the data demand is high, the maximum output bandwidth of $node_i$'s DRAM may not be large enough to satisfy the needs of all cores. Unlike case 1, in this case, as most data are located outside $node_i$, it is best to satisfy the needs of the cores on the other nodes first. After the needs of the other nodes are

met, then the remaining bandwidth can be used to satisfy the needs of some cores on $node_i$.

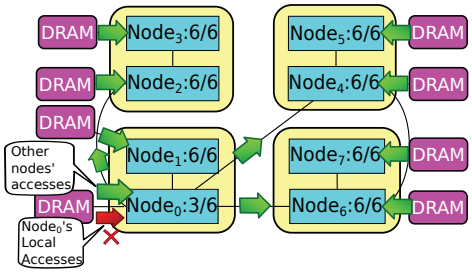


Figure 8: BLAS routine *dgemm* is limited by local and inter-node access interference. Values "X/Y" give the optimal allocation (X) and maximum core count (Y) for each node. A "cross" indicates a saturated memory connection.

For example, the BLAS matrix multiplication routine *dgemm* implemented by AMD Core Math Library (ACML) has this partial data-sharing behavior [15]. Figure 8 gives the optimal core allocation for *dgemm* and the saturated memory connections on the AMD NUMA machine. As Figure 8 shows, most of *dgemm*'s data are distributed, while shared data are located on $node_0$. Because of the high data demand, $node_0$'s memory cannot provide enough bandwidth for both local and inter-node accesses. Because the majority of the data are located on $node_1$ to $node_7$, it is best to ensure the data needs of the cores on $node_1$ to $node_7$ are met. Therefore, the optimal core allocation for *dgemm* only allocates three cores on $node_0$, leaving enough of $node_0$'s memory bandwidth to satisfy the needs of other nodes. That is, for *dgemm*, the optimal core allocation is $\{3,6,6,6,6,6,6,6\}$, which is 12% faster than using all 48 cores.

4.4 Summary of Insights

The above examples illustrate that there are four factors from NUMA memory system that may impact optimal core allocations for memory-intensive applications. These factors are: [2]

1. Hardware configuration, application memory behavior, and their asymmetry.
2. Local memory bandwidth and DRAM contention;
3. Maximum inter-node memory bandwidth;
4. Interference of local and inter-node accesses.

The first factor (asymmetry) dictates that each node to be treated differently when predicting optimal core allocations. It is also the fundamental reason why optimal core allocations are not optimal numbers of cores. The last three factors are memory system factors that directly limit maximum BW usage.

[2]We believe this list of factors is exhaustive as it covers all memory resources below shared-cache. They also include all factors identified by previous research on NUMA memory system [16, 17, 18].

5. OVERVIEW OF THE NuCore MODEL

Because the four memory factors in Section 4.4 determine memory bandwidth usages and optimal core allocations, we predict the bandwidth usage and optimal allocation by modeling these factors. This section gives an overview for our prediction model, NuCore.

5.1 Handling the Four Memory Factors

5.1.1 *Handling the Asymmetry*

The asymmetry requires treating each node differently, which greatly increases the solution space of optimal core allocations. For instance, the 48-core/8-node AMD NUMA machine has about $7^8 = 5,764,801$ possible core allocations. This huge solution space makes it very challenging to generate predictions within reasonable amount of time.

In this paper, we argue that Integer Programming (IP) can be used to address this challenge. First, IP naturally considers each node and memory link differently as long as they are represented with individual variables. Second, with recent advancement, IP solvers can determine the optimal solutions for most problems in polynomial time [19, 20]. Because of these properties, we built the NuCore model based on IP.

5.1.2 *Handling Other Factors*

IP requires that all factors and limitations be expressed with linear constraint functions. To satisfy these requirements, we developed techniques to quantitatively describe the local and inter-node memory factors with linear functions. For the non-linear local DRAM contention, we employed a technique to express this contention with linear functions. Section 6 gives the details of these techniques.

5.2 Applying NuCore

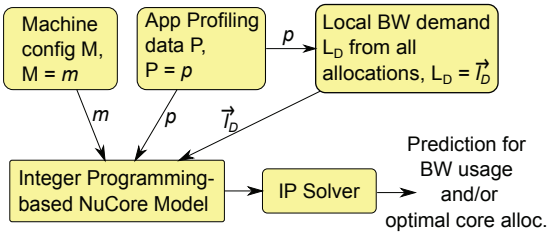


Figure 9: Overview of the NuCore model.

Figure 9 provides an overview of NuCore. The input parameters to NuCore are:

1. The configuration of the NUMA machine, denoted by M, such as the node connection topology, and the maximum bandwidth of the inter-node links.
2. Profiling data, denoted by P, describes the memory behavior of the application. To collect profiling data, at the beginning of each program phase, we execute the application briefly with a core allocation of one core per node, while hardware performance monitoring units (PMU) are used to collect its local and inter-node bandwidth usages. This profiling can be performed online during application execution. For this work,

we used a phase detection technique similar to prior work [4].

3. The predictions of per-node local DRAM bandwidth demands of the application when it runs with different number of cores on each node, denoted by L_D. A local bandwidth model, DraMon, is used to obtain this prediction [21].

The actual values of M, P, and L_D (i.e., m, p, and $\vec{l_d}$) are passed to NuCore to predict the optimal core allocation with an IP solver for a specific application and machine. Note that, in this paper, we only focus on the prediction of the optimal core allocations. The run-time adaptation to the optimal core allocations has been addressed by previous work, and is beyond the scope of this paper [7, 9, 22, 23].

6. NuCore DETAILS

This section discusses the NuCore model in detail. We explain how NuCore expresses each memory system factor as linear constraints, and how it expresses the prediction of optimal core allocations as a linear objective function.

6.1 $Constraint_1$: HW Config. – Max. Core Count per Node

The number of cores allocated on a node cannot exceed that node's maximum core count. Let a_i be the number of cores allocated on $node_i$. Let n_i be the maximum core count of $node_i$. The core allocation a_i must be smaller than n_i, i.e.,

$$0 \le a_i \le n_i. \tag{1}$$

6.2 $Constraint_2$: App. Mem. Behavior – Max. Data Rate

The bandwidth usage of an application is limited by the maximum rates that it can issue memory requests. The inter-node bandwidth usage from $node_j$ to $node_i$, I_{ji}, includes the usages of the responses to the read requests from $node_i$, and the write requests from $node_j$. Let $I_{r,ji,solo}$ be the bandwidth usage of an application running on $node_i$ and read-accessing $node_j$ when it uses one (solo) core on $node_i$. The $I_{r,ji,solo}$ is acquired from profiling. Intuitively, if an application with one core is using $I_{r,ji,solo}$ bandwidth, its bandwidth usage ($I_{r,ji}$) with a_i cores is no larger than a_i times $I_{r,ji,solo}$:

$$I_{r,ji} \le a_i \times I_{r,ji,solo}. \tag{2}$$

Similarly, if the threads on $node_j$ are also writing to $node_i$, then the write-bandwidth usage $I_{w,ji}$ is limited by

$$I_{w,ji} \le a_j \times I_{w,ji,solo}. \tag{3}$$

The total inter-node bandwidth usage from $node_j$ to $node_i$ is the sum of the read and write bandwidth:

$$I_{ji} = I_{r,ji} + I_{w,ji}. \tag{4}$$

Additionally, let $L_{D,i}$ be the local bandwidth demand of a_i cores on $node_i$. $L_{D,i}$ is acquired using the DRAM model DraMon [21]. The actual local bandwidth usage of $node_i$, denoted by L_i, is limited by its local bandwidth demand:

$$L_i \le L_{D,i}, \tag{5}$$

6.3 $Constraint_3$: Local DRAM BW and Contention

The local bandwidth demand $L_{D,i}$ of $node_i$ is a function of the numbers of cores allocated on $node_i$ (i.e., a_i). Unfortunately, this function is not linear due to DRAM row buffer contention [21]. Consequently, we describe the local bandwidth demand $L_{D,i}$ with a discrete function given in Equation (6), where $b_{c,i}$ represents the value of the local bandwidth demand when c cores are used on $node_i$ (c is an auxiliary variable).

$$L_{D,i} = f(a_i) = \begin{cases} b_{0,i}, & \text{if } a_i = 0 \\ b_{1,i}, & \text{if } a_i = 1 \\ \cdots & \\ b_{c,i}, & \text{if } a_i = c \\ \cdots & \\ b_{n_i,i}, & \text{if } a_i = n_i \end{cases} \tag{6}$$

Discrete functions cannot be used as IP constraints. We employed a technique which converts discrete functions to linear functions. First, we convert Equation (6) into a linear function by introducing auxiliary variables $y_{c,i}$.

$$\begin{aligned} &L_{D,i} = y_{0,i} \cdot b_{0,i} + y_{1,i} \cdot (b_{1,i} - b_{0,i}) + \cdots + y_{n_i,i} \cdot (b_{n_i,i} - b_{n_i-1,i}), \\ &y_{c,i} \text{ is an } \textbf{integer}, 0 \le y_{c,i} \le 1; 0 \le c \le n_i \\ &y_{c,i} = 1 \text{ if } a_i \ge c; \text{ else } y_{c,i} = 0 \end{aligned} \tag{7}$$

When $a_i = c$, Equation (7) is $L_{D,i} = b_{0,i} + b_{1,i} - b_{0,i} + \cdots + b_{c,i} - b_{c-1,i} = b_{c,i}$, equivalent to Equation (6). Next, we use a common method to convert the $if-else$ condition in Equation (7) to linear constraints with Equation (8), where B and ε are arbitrary constants, $0 < \varepsilon < 1$, $B \gg n_i$ [24].

$$\begin{aligned} &B \cdot y_{c,i} \ge a_i - c + \varepsilon, \\ &c \cdot y_{c,i} \le a_i. \end{aligned} \tag{8}$$

In short, with Equations (7) and (8), we can convert the discrete local bandwidth demands of Equation (6) into linear constraints.

6.4 $Constraint_4$: Max. Inter-node Connection BW

Unidirectional Max The bandwidth usage from $node_j$ to $node_i$, I_{ji}, is also limited by the frequency and the number of bits of the inter-node link that connects $node_j$ to $node_i$. Let $UniMax_{ji}$ denote this physical limit. This constraint is

$$I_{ji} \le UniMax_{ji}, \tag{9}$$

where $UniMax_{ji}$ is an input parameter as machine configuration. Although, its value can be determined theoretically based on the frequency and bit-width of the link, the real maximum bandwidth is usually smaller than the theoretical value for two reasons. First, there are overheads from the packet header and CRC code for each data packet [10, 11]. Second, the link is also used to send snooping messages. Therefore, we used the bandwidth measurement tool *bw_mem* from *lmBench* benchmark suite to determine the value of $UniMax_{ji}$ experimentally [13, 25].

Bidirectional Max In a NUMA machine, $node_i$ and $node_j$ can access each other simultaneously over the same inter-node link (to send requests and responses). These accesses contend for the shared link. That is, the sum of the bandwidth usages of I_{ji} and I_{ij} is limited by this contention. Let

$BiMax_{ij}$ denote the maximum bi-directional bandwidth of the physical link between $node_i$ and $node_j$. This constraint is then

$$I_{ji} + I_{ij} \leq BiMax_{ij}. \tag{10}$$

$BiMax_{ij}$ is is also determined experimentally using the *lmBench*.

6.5 $Constraint_5$: Local/Inter-node Access Contention

For memory-intensive applications, because both the remote requests and local requests to a node access that node's local DRAM, their bandwidth is limited by the maximum available bandwidth of that node's DRAM. Our experimental results reveal that the maximum of the sum of the outgoing inter-node bandwidth of a node has a linear relationship with its local bandwidth demand. Figure 10 depicts this linear relationship for a node on the 32-core Intel NUMA machine. The y-intercept and the slope of the linear equation, α_i and β_i, are 20.26 and 0.24 for this machine (determined with *bw_mem*). The correlation coefficient is 0.96 indicating a strong linear relationship.

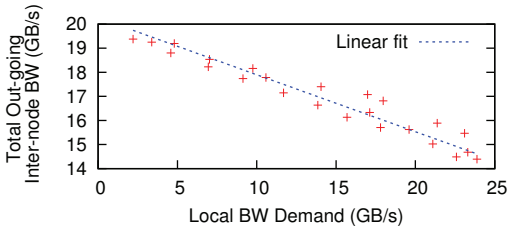


Figure 10: Linear relationship of the total outgoing bandwidth and the local bandwidth demand of the 32-core Intel NUMA platform.

Let N denote the number of nodes. Recall that $L_{D,i}$ denote the local bandwidth demand on $node_i$ predicted with the DraMon model [21]. Based on this linear relationship, the constraint of the local and remote access contention can be expressed as

$$\sum_{j=1}^{N} I_{ij} + \beta_i \times L_{D,i} \leq \alpha_i, \tag{11}$$

where the values of α_i and β_i are determined using linear regression with *bw_mem*.

We deduce that this linear relationship is a reflection of the processor's attempt to ensure every requesting source, including the inter-node links and the local cores, gets a fair share of its memory bandwidth. Intuitively, α_i represents the maximum bandwidth of $node_i$'s memory, and β_i represents the lowest share of $node_i$'s bandwidth reserved for local cores. This insight leads to another constraint that binds the local bandwidth usage on any $node_i$, shown in Equation (12). Intuitively, the sum of the total out-going bandwidth and the local bandwidth of $node_i$, cannot exceed its maximum bandwidth α_i.

$$\sum_{j=1}^{N} I_{ij} + L_i \leq \alpha_i. \tag{12}$$

6.6 Handling Multi-hop Links

As shown in Section 3, two nodes may be connected using multiple physical inter-node links. Without loss of generality, assume two nodes $node_l$ and $node_k$ are connected using a virtual link through d nodes $l, l+1, ..., l+d, k$. The bandwidth usage I_{lk} is also subject to the physical limit and bi-directional contention as described in Equation (9) and (10). $UniMax_{lk}$ and $BiMax_{lk}$ are also determined experimentally using *bw_mem*. Additionally, because the virtual link sends data through its physical links, its bandwidth usage I_{lk} should also be added to its physical links:

$$I_{ji} = I_{ji,r} + I_{ji,w} + \sum_{l,k} I_{lk}, \forall\ node_l \text{ and } node_k \text{ connected through link } j \rightarrow i. \tag{13}$$

6.7 Optimization Objective Function

By definition, the optimal core allocations for memory-intensive applications are the minimum core allocations with the best performance and highest bandwidth usage. Therefore, predicting the optimal core allocation requires satisfying two goals: maximizing the total bandwidth usage and minimizing core allocation. The objective function for maximizing the sum of local and inter-node bandwidth is

$$\text{maximize: } \sum_i L_i + \sum_{i,j} I_{ij}. \tag{14}$$

The second goal of minimizing core allocation is

$$\text{minimize: } \sum_i a_i. \tag{15}$$

Because minimizing a function is similar to maximizing the negative of it, we combine Equation (14) and (15) into one objective function. To emphasize that our priority is maximizing bandwidth, we multiply the bandwidth goal by a constant C.

$$\text{maximize: } C \cdot (\sum_i L_i + \sum_{i,j} I_{ij}) - \sum_i a_i. \tag{16}$$

6.8 NuCore Summary

Equation (17) summarizes the linear constraints and objective function of NuCore.

$$\begin{aligned}
&\text{maximize: } C \cdot (\sum_i L_i + \sum_{i,j} I_{ij}) - \sum_i a_i \\
&\text{subject to: } \forall i, j: \\
&\quad \text{constraint}_1\text{: } 0 \leq a_i \leq n_i, \\
&\quad \text{constraint}_2\text{: } I_{r,ji} \leq a_i \times I_{r,ji,solo},\ I_{w,ji} \leq n_j \times I_{w,ji,single}, \\
&\qquad\qquad I_{ji} = I_{ij,r} + I_{ij,w} + \sum_{l,k} I_{lk}, \forall\ node_l \text{ and } node_k \\
&\qquad\qquad \text{connected through link } j \rightarrow i., \\
&\quad \text{constraint}_3\text{: } L_{D,i} = \sum_{c=0}^{n_i} y_{c,i} \cdot (b_{c,i} - b_{c-1,i}), \\
&\qquad\qquad \forall c, 0 \leq c \leq n_i\text{:} \\
&\qquad\qquad\quad B \cdot y_{c,i} \geq a_i - c + \varepsilon, \\
&\qquad\qquad\quad c \cdot y_{c,i} \leq a_i,\ y_{c,i} \text{ is integer}, 0 \leq y_{c,i} \leq 1 \\
&\quad \text{constraint}_4\text{: } I_{ji} \leq UniMax_{ji},\ I_{ji} + I_{ij} \leq BiMax_{ij}, \\
&\quad \text{constraint}_5\text{: } \sum_{j=1}^{M} I_{ij} + \beta_i \times L_{D,i} \leq \alpha_i, \sum_{j=1}^{M} I_{ij} + L_i \leq \alpha_i,
\end{aligned} \tag{17}$$

6.9 Predicting Bandwidth Usage

The solution provided by an IP solver includes predictions for memory bandwidth usages as well. Therefore, if we specify a core allocation in NuCore (i.e., specify the value of a_i), NuCore can also predict the memory bandwidth usage for a multi-threaded application under that specific core allocation.

7. EXPERIMENTAL EVALUATION

This section gives the experimental evaluation of the accuracy and performance benefits of our model.

7.1 Experimental Setup

Platforms We evaluated our model on the two NUMA platforms discussed in Section 3. The 48-core AMD platform has four Opteron 6174 processors with Linux 2.6.23 (Figure 3). Each processor has two six-core nodes sharing 6MB L3 cache and 12GB memory. Each core has a 128KB L1 Cache and 256KB L2 cache. The 32-core Intel platform has four Xeon X7550 processors with Linux 3.8.0. (Figure 4). Each processor has one eight-core node with 18MB L3 cache and 64GB memory. Each core also has 64KB L1 cache and 256KB L2 cache.

Benchmarks We used PARSEC 2.1 benchmarks and NPB-OMP 3.3.1 benchmarks in our evaluation [3, 14]. We also used the BLAS matrix multiplication routine *dgemm* from Intel Math Kernel Library (MKL) 11.1 and AMD Core Math Library (ACML) 5.3.1, because its wide application and high bandwidth usage [15, 26, 27]. The evaluation included 25 benchmarks, among which 7 are memory-intensive (listed in Table 1a). Three PARSEC *bodytrack*, *dedup*, and *x264* are I/O bound. The remaining 15 benchmarks are CPU-intensive. For PARSEC, the "native" input sets are used, and we predicted optimal core allocations for its parallel regions (ROI). For NPB, the largest executable input sets, "C" or "D", are used. For *dgemm*, two $1.6K \times 1.6K$ matrices with random values are multiplied to fully exercise the memory system [28].

Core Allocation Performance Metric We compared the performance of NuCore predicted core allocation with the use-all-cores allocation and report the speedup of NuCore's predictions. Additionally, we compare NuCore's predictions with two state-of-the-art predictive techniques [4, 9]. The speedup is defined as

$$SpeedUp = \frac{ExecTime_{use-all-cores}}{ExecTime_{predicted}}. \quad (18)$$

Bandwidth Prediction Accuracy Metric For each benchmark, we predicted its bandwidth usages using NuCore for ten randomly selected core allocations (we will show later that this sample selection is statistically sound). We compare the predicted values and the actual values obtained from PMUs, and report the mean absolute percentage error (MAPE) for each benchmark, which is defined as [29],

$$MAPE = \frac{1}{10}\sum_{alloc=1}^{10}\left|\frac{BW_{alloc,actual} - BW_{alloc,predicted}}{BW_{alloc,actual}}\right|. \quad (19)$$

7.2 Determining the Real Optimal Allocation

To determine the accuracy of NuCore, we compare NuCore's core allocations predictions with real optimal core allocations. We use two methods to determine the read optimal.

Experimental Approach We determined the real optimal core allocation experimentally by evaluating the performance of adjacent core allocations of NuCore's predicted optimal allocation. Two core allocations A and A' are considered adjacent if they differ by only one core:

$$A = \{a_0, a_1, \ldots, a_i, \ldots, a_N\} \text{ and } A' = \{a'_0, a'_1, \ldots, a'_i, \ldots, a'_N\}$$
$$\text{are adjacent} \iff \{\exists! i, |a_i - a'_i| = 1\} \bigwedge \{\forall j, j \neq i, a_j = a'_j\} \quad (20)$$

A core allocation is considered optimal if it performs better than all of its adjacent core allocations. We started with a "seed" allocation (NuCore predicted core allocation), and compared it with its adjacent allocations. If the "seed" allocation performs better than its adjacent allocations, then the "seed" is optimal. Otherwise, we selected the best performing adjacent core allocation as the new "seed," and repeat this procedure until we find the optimal one. Note that the optimal allocation determined by this method may be only local optimal. We call the optimal allocation determined by this approach as *experimental optimal* or *exp optimal*.

Extreme Value Theory (EVT) Finding the global optimal from millions of core allocations experimentally is impractical. Instead, we determine the speedup upper bound of all core allocations. If this upper bound has a tight confidence interval with high confidence, it can be viewed as the real optimal performance. We determined this upper bound with EVT [30, 31, 32]. We sampled the performance of more than 2000 randomly selected core allocations to ensure a 3% confident interval [30]. Based on the distribution of the samples, the maximum value of the sample space, i.e., the speedup upper bound, can be estimated with EVT. For the rest of the paper, we call this upper bound the *real optimal*. Note that EVT only gives the speedup upper-bound, it does not give which core allocation has the speedup of this upper-bound.

7.3 Optimal Core Allocation Prediction

Accuracy of NuCore Table 1 gives the optimal core allocations determined by NuCore and the experimental approach for memory-intensive benchmarks. NPB benchmarks have several phases and the predictions are made for each phase. The function name of each phase is supplied for NPB benchmarks in Table 1. Table 1 also gives the major scalability limitation for each benchmark. These constraints correspond to the three limitations in Section 4, which are local BW limited, inter-node BW limited, and local/inter-node interference limited due to all or partial data sharing.

Table 1 shows that NuCore's predictions are very close to the experimental optimal. Only one PARSEC benchmark and four phases of two NPB benchmarks, *streamcluster*, *mg.D-psinv*, *mg.D-rprj3*, *mg.D-interp* and *sp.C-rhs* are mispredicted. However, these mispredicted core allocations differ with the experimental optimal by only one core per node.

For the rest **CPU- or IO-bound** (not bandwidth limited) benchmarks, they all perform best using all cores, which are also accurately predicted by NuCore. These results indicate that NuCore can predict the optimal core allocations for various types of applications with high accuracy.

Benchmark	NuCore (%)	Exp. Optimal (%)	Varuna (%)	CRUST (%)	Scale Limit
streamcluster	{6,0,0,0,0,0,0,0} (12.5%)	{6,0,0,0,0,0,0,0} (12.5%)	{2,2,1,1,1,1,1,1} (20.8%)	{1,1,1,1,1,1,1,1} (16.7%)	All-shared
canneal	{6,6,6,3,0,0,0,0} (43.8%)	{6,6,6,3,0,0,0,0} (43.8%)	{6,6,6,6,6,6,6,6} (100%)	{5,5,5,5,5,5,5,5} (83.3%)	All-shared
facesim	{6,6,4,0,0,0,0,0} (33.3%)	{6,6,4,0,0,0,0,0} (33.3%)	{2,2,2,2,2,2,2,2} (33.3%)	{4,4,4,4,4,4,4,4} (66.7%)	All-shared
mg.D-resid	{6,6,6,6,6,6,6,6} (100%)	{6,6,6,6,6,6,6,6} (100%)	{3,3,3,3,3,3,3,2} (47.9%)	{6,6,6,6,6,6,6,6} (100%)	None
mg.D-psinv	{5,5,5,5,5,5,5,5} (83.3%)	{6,6,6,6,6,6,6,6} (100%)	{6,6,6,6,6,6,6,6} (100%)	{6,6,6,6,6,6,6,6} (100%)	None
mg.D-rprj3	{6,5,6,6,6,4,6,4} (89.6%)	{6,5,6,6,6,4,6,5} (91.7%)	{6,6,6,6,6,6,6,6} (100%)	{6,6,6,6,6,6,6,6} (100%)	Inter BW
mg.D-interp	{6,5,6,6,6,4,6,4} (89.6%)	{6,5,6,6,6,4,6,5} (91.7%)	{6,6,6,6,6,6,6,6} (100%)	{6,6,6,6,6,6,6,6} (100%)	Inter BW
sp.C-x/y/zsovle	{1,1,1,1,1,1,1,1} (16.7%)	{1,1,1,1,1,1,1,1} (16.7%)	{2,2,2,1,1,1,1,1} (22.9%)	{1,1,1,1,1,1,1,1} (16.7%)	Partial Shared
sp.C-rhs	{6,5,6,4,5,4,5,4} (81.3%)	{6,5,6,5,5,4,4,4} (81.3%)	{4,4,4,3,3,3,3,3} (56.3%)	{2,2,2,2,2,2,2,2} (33.3%)	Local BW
dgemm (MKL)	{2,6,6,6,6,6,6,6} (91.7%)	{2,6,6,6,6,6,6,6} (91.7%)	{6,6,6,6,6,6,6,6} (100%)	{2,2,2,2,2,2,2,2} (33.3%)	Partial-shared
dgemm (ACML)	{3,6,6,6,6,6,6,6} (93.8%)	{3,6,6,6,6,6,6,6} (93.8%)	{6,6,6,6,6,6,6,6} (100%)	{6,6,6,6,6,6,6,6} (100%)	Partial-shared

(a) Optimal Core Allocations on the AMD Platform

Benchmark	NuCore (%)	Exp. Search. (%)	Varuna (%)	CRUST (%)	Constraint
streamcluster	{8,8,3,0} (59.4%)	{8,8,2,0} (56.3%)	{8,8,8,8} (100%)	{7,7,7,7} (87.5%)	All-shared
canneal	{8,8,8,4} (87.5%)	{8,8,8,4} (87.5%)	{8,8,8,8} (100%)	{7,7,7,7} (87.5%)	All-shared
facesim	{8,8,0,0} (50.0%)	{8,8,0,0} (50.0%)	{3,3,3,3} (37.5%)	{4,4,4,4} (50.0%)	All-shared
mg.D-resid	{7,7,7,7} (87.5%)	{7,7,7,7} (87.5%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	Local BW
mg.D-psinv	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	None
mg.D-rprj3	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	None
mg.D-interp	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	None
sp.C-x/y/zsovle	{3,3,3,3} (37.5%)	{3,3,3,3} (37.5%)	{5,5,4,4} (56.3%)	{2,2,2,2} (25.0%)	Partial-shared
sp.C-rhs	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{4,4,4,4} (50.0%)	None
dgemm (MKL)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	None
dgemm (ACML)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	{8,8,8,8} (100%)	None

(b) Optimal Core Allocations on the Intel Platform

Table 1: Optimal core allocations determined by NuCore, experimental search, and two state-of-the-art predictive methods on the AMD and Intel platforms.

Performance of NuCore Predictions Figure 11 gives the performance of the optimal core allocations determined by NuCore. Figure 11 shows that NuCore's predictions can significantly improve performance over using-all-cores. NuCore's predictions have a maximum speedup of 3.34x when executing *streamcluster*. This speedup is achieved with only 12.5% cores allocated (Table 1). The average speedup for memory-intensive benchmarks on the two platforms is 1.27x.

Figure 11 also gives the performance of the optimal core allocations determined experimentally, and the real optimal performance (speedup upper bound) determined by EVT. The average performance difference between NuCore predictions and experimental optimal allocations is only 0.5%. The average performance difference between real optimal and NuCore predictions is only 1.0%. These small differences further demonstrate that NuCore can predict the optimal core allocation with high accuracy.

Comparing to State-of-the-Art Techniques Table 1 also gives the optimal core allocations determined by two state-of-the-art techniques, Varuna and CRUST [4, 9]. As Table 1 shows, the accuracy of both Varuna and CRUST is lower than NuCore. The performance of these core allocations are given in Figure 11. Figure 11 shows that NuCore's predicted allocations perform 1.18x times better than those predicted by Varuna, and 1.21x times better than those predicted by CRUST on average.

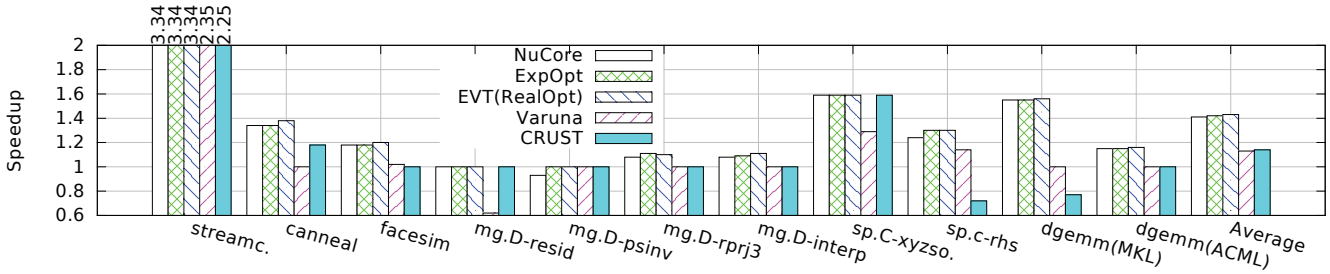
We observe that there are two factors that impact the accuracy of Varuna and CRUST. First, both models do not consider NUMA asymmetry. Therefore, they cannot accurately predict the optimal core allocations when an optimal allocation allocates different number of cores on different nodes. Second, both models do not consider memory resources adequately. CRUST only considers shared cache and cache miss rates. Varuna is resource-agnostic. Because neither of the models consider DRAM links and inter-node memory links, they cannot accurately determine when (i.e., the core allocation) a memory link is saturated.

7.4 Bandwidth Usage Prediction

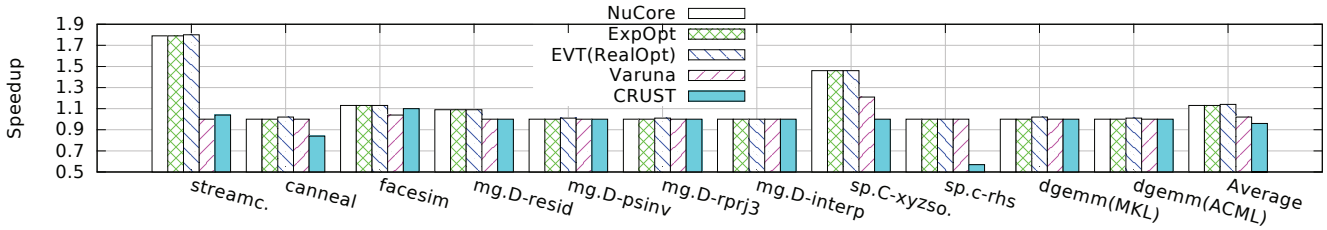
Benchmark	MAPE					
	Intel Platform			AMD Platform		
	Local	Inter	Total	Local	Inter	Total
streamcluster	8.8%	5.8%	4.0%	16.1%	6.4%	8.0%
canneal	10.2%	2.4%	3.3%	12.9%	6.9%	6.4%
facesim	8.3%	6.8%	9.7%	3.3%	4.8%	2.1%
mg.D-resid	8.2%	0.0%	8.2%	7.8%	0.0%	7.8%
mg.D-psinv	6.9%	0.0%	6.9%	7.6%	0.0%	7.6%
mg.D-rprj3	1.9%	4.3%	2.0%	10.9%	4.9%	8.2%
mg.D-interp	10.5%	6.7%	7.5%	9.8%	6.9%	8.4%
sp.C-x/y/zsovle	10.1%	8.3%	5.1%	10.4%	5.4%	8.7%
sp.C-rhs	7.1%	4.8%	6.7%	6.5%	3.9%	5.3%
dgemm (MKL)	9.1%	2.1%	3.2%	5.2%	2.8%	4.3%
dgemm (ACML)	6.0%	6.5%	4.8%	5.7%	8.7%	5.6%
Average	7.9%	4.3%	5.6%	8.7%	4.7%	6.7%

Table 2: Average MAPE of bandwidth usage prediction for memory-intensive benchmarks.

Table 2 gives the average bandwidth prediction errors for the seven memory-intensive benchmarks (and their phases). Note that there are several cases where the benchmarks have



(a) Speedups on AMD Platform



(b) Speedups on Intel Platform

Figure 11: Performance of the optimal core allocations determined by NuCore, experimental approach, and two state-of-the-art predictive methods, over use all cores allocation on the AMD and Intel platforms.

no inter-node bandwidth usage, where NuCore has 0% inter-node bandwidth prediction error. Table 2 shows that NuCore is highly accurate for memory bandwidth prediction for memory-intensive benchmarks.

The highest error is 30.0% when predicting the local bandwidth usage of *streamcluster* with core allocation $\{2,2,2,2,2,2,1,1\}$. This high error is partially caused by the fluctuation of the PMU readings, which in turn is caused by *streamcluster*'s short memory bursts. These bursts are too short to be stably caught by the PMUs [33].

Because it is impossible to evaluate NuCore for every core allocation, we predicted ten randomly-picked allocations for each benchmark. With Student's test, we find that this experiment design is statistically sound: these results show that, for memory-intensive applications, NuCore's average error is lower than 10% for bandwidth predictions with 99% confidence.

7.5 Prediction Time of NuCore

We used a state-of-the-art IP solver, SCIP, to solve NuCore instances [34]. The maximum time to solve a NuCore instance on both platforms is 0.02 seconds. The profiling phase requires running a program with one thread/core per node for 0.05 seconds to sample PMU readings. Therefore, the total prediction time is 0.07 seconds. This low overhead makes NuCore suitable for run-time optimization.

8. DISCUSSION

Impact of Program Parallelism, Cache Contention and Synchronization on Core Allocation Prediction We observer that the major scalability limitation on our systems is memory bandwidth. Because we used large input sets designed for evaluating large systems, our benchmarks have abundant parallelism. We did observe that synchronization and cache contention affect scalability. However, because of fast inter-node links and large input sets (i.e., working sets do not fit in cache), synchronization and cache contention has less impact on core allocation decisions than memory bandwidth on our systems. For systems with smaller caches, slower inter-node links or smaller workloads, the impact of limited-parallelism, synchronization and cache contention may be significant. As these factors have been studied before, we plan to combine existing models for these factors with NuCore in the future [4, 5, 6, 7, 8, 35].

Cache Impact on Bandwidth Prediction Most memory-intensive benchmarks already have high cache miss rates, and their memory behaviors are not significantly affected by cache contention [36, 37]. Therefore, NuCore can predict their bandwidth usage with high accuracy without a cache model.

Most of the CPU-intensive (not bandwidth limited) benchmarks have very low cache miss rates, and are also not significantly affected by the cache contention [36, 37]. Consequently, NuCore can also predict the bandwidth usages for most CPU-intensive benchmarks with high accuracy. However, some applications, such as the benchmarks of *freqmine* and *ep.D*, are more sensitive to the contention and data sharing in the cache than other benchmarks. Including a cache model can improve the accuracy of the bandwidth usage prediction for them [4, 38, 39, 40].

Prefetcher Impact The prefetchers of the AMD platform were enabled in our experiments. AMD prefetchers are less-aggressive and can reduce prefetching requests in case of contention [12]. Therefore, these prefetchers do not affect our model's accuracy. The prefetchers on the Intel platform were disabled in our experiments. However, preliminary results with synthetic benchmarks show that Intel prefetchers do not significantly reduce NuCore's accuracy. Nonetheless, it must be noted that an aggressive prefetcher may change application memory behavior and may require its own model.

9. RELATED WORK

There are several studies investigated the core allocation problem. Li et al. proposed a dynamic core allocation algorithm to reduce power consumption [41]. Suleman et al. addressed the scalability limitations of memory bandwidth and critical sections with core allocation [6]. Lee et al. proposed a run-time system to dynamically change thread count [7]. Chadha et al. proposed a run-time system to determine the optimal thread count, processor voltage and frequency [8]. Kayiran et al. investigated the thread management on GPGPUS [5]. CRUST is a novel system designed to improve core allocations on many-core chips [4]. CRUST focused on the data sharing in cache, as well as cache-bandwidth interaction. Pusukuri et al. studied the optimal thread count problem using OS observations [35]. Sirdharan et al. proposed a model called Varuna based on Amdahl's law to predict optimal thread count [9]. Varuna is designed to be resource-agnostic and generic so that it can be applied to architectures with unknown properties. Similarly, Sasaki et al. investigated node allocations on NUMA machines using Amdahl's law [42]. These studies focused on the prediction of optimal core/thread count, not the optimal core allocation. Moreover, they did not consider DRAM contention, inter-node connections, as well as hardware and software asymmetry. However, as we showed in this paper, without considering these factors, it is impossible to accurately predict optimal core allocations.

Kim et al. and Wang et al. modeled local memory bandwidth for multi-core processors [21, 43]. Eklov et al. characterized the performance impact of memory contention [44]. Wang et al. modeled the performance impact of bandwidth partitioning [45]. These techniques do not consider remote memory accesses, and thus they cannot be directly used to predict the memory bandwidth usages on NUMA machines.

Blagodurov et al. and Majo et al. analyzed the bandwidth constraints on NUMA machines [17, 46]. These studies provided valuable insights on NUMA bandwidth constraints that helped us develop our models.

Integer Programming has long been used to solve scheduling problems on computer system[47, 48]. Nowatzki et al. proposed a generic IP framework for scheduling programs on spatial architectures [49]. However, they pointed out that existing IP-based solutions are limited to problems with "directly expressible" and linear-only constraints. However, the core of the memory bandwidth problem, i.e., the memory contention, is neither linear nor directly expressible. Our insights regarding NUMA characteristics in Section 6 and our technique of converting non-linear constraints to linear make it possible to apply IP to memory-related problems.

Some studies investigated memory page migration to colocate the data and computation on the same memory node [50, 51]. In particular, Lepers et al. improved memory page migration algorithms to consider memory asymmetry [18]. There is also research that investigated data placement on NUMA machines [28, 52, 53, 54]. These techniques aim to reduce memory latency rather than reducing bandwidth usage. In fact, the reduced memory latency improves performance and potentially increases bandwidth usage. Therefore, NuCore is complementary to these techniques. Novel hardware is also proposed to improve memory system [55, 56, 57, 58, 59, 60]. Because these techniques do not eliminate the bandwidth over-saturation problem, NuCore can be used with them to address bandwidth issues comprehensively.

10. SUMMARY

NUMA machines offer large numbers of cores to boost performance. However, because of memory bandwidth limitation, allocating all cores to a multi-threaded program may significantly degrade performance. This paper presented a detailed analysis of optimal (best performing) core allocations on real NUMA machines. This analysis revealed that accurately predicting optimal core allocations must consider NUMA memory resources, memory contention and HW/SW asymmetry. This analysis also showed that optimal core allocations are not simply optimal numbers of cores. Based on this analysis, we designed a novel model, NuCore, to predict optimal core allocations for multi-threaded applications. NuCore achieves high accuracy by extensively modeling various NUMA memory resources and factors. NuCore also employed novel techniques to convert the prediction problem into an Integer Programming problem, allowing low overhead predictions. When evaluated on a real NUMA machine, the optimal core allocations predicted by NuCore provides 1.27x speedup over use-all-cores allocations on average, and 1.18x and 1.21x speedup over two state-of-the-art prediction techniques. Additionally, NuCore can also predict memory bandwidth usages of memory-intensive applications with only 10% errors on average.

Acknowledgments This work was supported by the National Science Foundation under grants CCF-0811689 and CNS-0964627, and by the Air Force Research Laboratories (AFRL) Contract No. FA8750-15-C-0118. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of NSF, AFRL, the DoD, or the U.S. Government. We appreciate the insightful comments and constructive suggestions from the anonymous reviewers. We would also like to thank Tanima Dey for her valuable input.

References

[1] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich, "An Analysis of Linux Scalability to Many Cores," in *USENIX Conference on Operating Systems Design and Implementation*, 2010.

[2] X. Liu and J. Mellor-Crummey, "A Tool to Analyze the Performance of Multithreaded Programs on NUMA Architectures," in *Int'l Symp. on Principles and Practice of Parallel Programming*, 2014.

[3] C. Bienia, *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, 2011.

[4] W. Heirman, T. Carlson, K. Van Craeynest, I. Hur, A. Jaleel, and L. Eeckhout, "Undersubscribed Threading on Clustered Cache Architectures," in *Int'l Symp. on High Performance Computer Architecture*, 2014.

[5] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither More nor Less: Optimizing Thread-level Parallelism for GPGPUs," in *Proc. of Int'l Conf. on Parallel Architectures and Compilation Techniques*, 2013.

[6] M. A. Suleman, M. K. Qureshi, and Y. N. Patt, "Feedback-driven Threading: Power-efficient and High-performance Execution of Multi-threaded Workloads on CMPs," in *Proc. of Architectural Support for Programming Languages and Operating Sys.*, 2008.

[7] J. Lee, H. Wu, M. Ravichandran, and N. Clark, "Thread Tailor: Dynamically Weaving Threads Together for Efficient, Adaptive Parallel Applications," in *Int'l Symp. on Computer Architecture*, 2010.

[8] G. Chadha, S. Mahlke, and S. Narayanasamy, "When Less Is MOre (LIMO): Controlled Parallelism for Improved Efficiency," in *Int'l Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, 2012.

[9] S. Sridharan, G. Gupta, and G. S. Sohi, "Adaptive, Efficient, Parallel Execution of Parallel Programs," in *Int'l Conf. on Programming Language Design and Implementation*, 2014.

[10] Intel, "An Introduction to the Intel QuickPath Interconnect," 2009.

[11] T. H. Consortium, "HyperTransport I/O Technology Overview," 2004.

[12] AMD, "BIOS and Kernel Developer's Guide (BKDG) For AMD Family 10h Processors," 2013.

[13] L. W. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis," in *USENIX Annual Technical Conference*, 1996.

[14] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and its Performance," tech. rep., NASA Ames Research Center, 1999.

[15] AMD, "AMD Core Math Library (ACML)," 2013.

[16] Z. Majo, *Modeling Memory System Performance of NUMA Multicore-Multiprocessors*. PhD thesis, ETH Zurich, 2014.

[17] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A Case for NUMA-aware Contention Management on Multicore Systems," in *USENIX Annual Technical Conference*, 2011.

[18] B. Lepers, V. Quema, and A. Fedorova, "Thread and Memory Placement on NUMA System: Asymmetry Matters," in *Proceedings of USENIX Annual Technical Conference*, 2015.

[19] H. W. Lenstra Jr., "Integer Programming with a Fixed Number of Variables," *Mathematics of Operations Research*, vol. 8, no. 4, 1983.

[20] G. Pataki, M. Tural, and E. B. Wong, "Basis Reduction and the Complexity of Branch-and-Bound," in *ACM-SIAM Symp. on Discrete Algorithms*, 2010.

[21] W. Wang, T. Dey, J. W. Davidson, and M. L. Soffa, "DraMon: Predicting Memory Bandwidth Usage of Multi-threaded Programs with High Accuracy and Low Overhead," in *Int'l Symp. on High Performance Computer Architecture*, 2014.

[22] R. W. Moore and B. R. Childers, "Inflation and Deflation of Self-adaptive Applications," in *Int'l Symp. on Software Engineering for Adaptive and Self-Managing Systems*, 2011.

[23] ""GNU GOMP libgomp Documentation"," 2014.

[24] T. Koch, *Rapid Mathematical Prototyping*. PhD thesis, Technische Universität Berlin, 2004.

[25] L. Peng, J.-K. Peir, T. K. Prakash, C. Staelin, Y.-K. Chen, and D. Koppelman, "Memory Hierarchy Performance Measurement of Commercial Dual-core Desktop Processors," *Journal of Systems Architecture*, vol. 54, no. 8, 2008.

[26] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An Updated Set of Basic Linear Algebra Subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, 2002.

[27] Intel, "Reference Manual for Intel Math Kernel Library 11.1 Update 3," 2013.

[28] W. Y. Alkowaileet, "*NUMA-aware multicore Matrix Multiplication*," Master's thesis, University of California, Irvine, 2013.

[29] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, 2006.

[30] P. Radojković, V. Čakarević, M. Moretó, J. Verdú, A. Pajuelo, F. J. Cazorla, M. Nemirovsky, and M. Valero, "Optimal Task Assignment in Multithreaded Processors: A Statistical Approach," in *Proc. of Architectural Support for Programming Languages and Operating Systems*, 2012.

[31] J. Beirlant, Y. Goegebeur, J. Segers, and J. Teugels, *Statistics of Extremes: Theory and Applications*. John Wiley & Sons, 2006.

[32] E. Castillo, *Extreme Value Theory in Engineering*. Elsevier, 1988.

[33] V. Weaver, D. Terpstra, and S. Moore, "Non-determinism and Overcount on Modern Hardware Performance Counter Implementations," in *Int'l Symp. on Performance Analysis of Systems and Software*, 2013.

[34] T. Achterberg, "SCIP: solving sonstraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, 2009.

[35] K. K. Pusukuri, R. Gupta, and L. N. Bhuyan, "Thread Reinforcer: Dynamically Determining Number of Threads via OS Level Monitoring," in *Int'l Symp. on Workload Characterization*, 2011.

[36] Y. Xie and G. H. Loh, "Dynamic Classification of Program Memory Behaviors in CMPs," in *Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2008.

[37] E. Z. Zhang, Y. Jiang, and X. Shen, "Does Cache Sharing on Modern CMP Matter to the Performance of Contemporary Multithreaded Programs?," in *ACM Symp. on Principles and Practice of Parallel Programming*, 2010.

[38] X. Xiang, B. Bao, C. Ding, and K. Shen, "Cache Conscious Task Regrouping on Multicore Processors," in *Int'l Symp. on Cluster, Cloud and Grid Computing*, 2012.

[39] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm, "RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations," in *Proc. of Architectural Support for Programming Languages and Operating Sys.*, 2009.

[40] D. Eklov, D. Black-Schaffer, and E. Hagersten, "Fast Modeling of Shared Caches in Multicore Systems," in *Proc. of Int'l Conf. on High Performance and Embedded Architectures and Compilers*, 2011.

[41] J. Li and J. F. Martinez, "Dynamic Power-performance Adaptation of Parallel Computation on Chip Multiprocessors," in *Int'l Symp. on High-Performance Computer Architecture*, 2006.

[42] H. Sasaki, T. Tanimoto, K. Inoue, and H. Nakamura, "Scalability-Based Manycore Partitioning," in *Int'l Conf. on Parallel Architectures and Compilation Techniques*, 2012.

[43] M. Kim, P. Kumar, H. Kim, and B. Brett, "Predicting Potential Speedup of Serial Code via Lightweight Profiling and Emulations with Memory Performance Model," in *Int'l Symp. on Parallel and Distributed Processing Symposium*, 2012.

[44] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten, "Bandwidth Bandit: Quantitative Characterization of Memory Contention," in *Int'l Symp. on Code Generation and Optimization*, 2013.

[45] R. Wang, L. Chen, and T. M. Pinkston, "An Analytical Performance Model for Partitioning Off-Chip Memory Bandwidth," in *Int'l Symp. on Parallel and Distributed Processing*, 2013.

[46] Z. Majo and T. R. Gross, "Memory System Performance in a NUMA Multicore Multiprocessor," in *Int'l Conf. on Systems and Storage*, 2011.

[47] M. Kudlur and S. Mahlke, "Orchestrating the Execution of Stream Programs on Multicore Platforms," in *Proc. of Programming Language Design and Implementation*, 2008.

[48] H. M. Wagner, "An integer linear-programming model for machine scheduling," *Naval Research Logistics Quarterly*, vol. 6, no. 2, 1959.

[49] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, "A General Constraint-centric Scheduling Framework for Spatial Architectures," in *Proc. of Programming Language Design and Implementation*, 2013.

[50] M. Awasthi, D. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, "Handling the Problems and Opportunities Posed by Multiple On-chip Memory Controllers," in *Int'l Conf. on Parallel Architectures and Compilation Techniques*, 2010.

[51] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, "Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems," in *Int'l Conf. on Architectural Support for Programming Languages and Oper. Sys.*, 2013.

[52] J. Bircsak, P. Craig, R. Crowell, Z. Cvetanovic, J. Harris, C. A. Nelson, and C. D. Offner, "Extending OpenMP for NUMA Machines," in *Int'l Conf. on Supercomputing*, 2000.

[53] Z. Majo and T. R. Gross, "(Mis)Understanding the NUMA Memory System Performance of Multithreaded Workloads," in *IEEE Int'l Symp. on Workload Characterization*, 2013.

[54] L. Nai, Y. Xia, C.-Y. Lin, B. Hong, and H.-H. S. Lee, "Cache-conscious Graph Collaborative Filtering on Multi-socket Multicore Systems," in *Proc. of Computing Frontiers*, 2014.

[55] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *Int'l Symp. on Computer Architecture*, 2008.

[56] W. Jia, K. A. Shaw, and M. Martonosi, "MRPB: Memory request prioritization for massively parallel processors," in *Int'l Conf. on High Performance Computer Architecture*, 2014.

[57] Z. Fang, L. Zhang, J. B. Carter, S. A. Mckee, A. Ibrahim, M. A. Parker, and X. Jiang, "Active Memory Controller," *The Journal of Supercomputing*, vol. 62, no. 1, 2012.

[58] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: Near-optimal Block Placement and Replication in Distributed Caches," in *Int'l Symp. on Computer Architecture*, 2009.

[59] S. Volos, J. Picorel, B. Falsafi, and B. Grot, "BuMP: Bulk Memory Access Prediction and Streaming," in *Proc. of Int'l Symp. on Microarchitecture*, 2014.

[60] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A Fully Associative, Tagless DRAM Cache," in *Proc. of Int'l Symp. on Computer Architecture*, 2015.