Profiling for Asymmetric NUMA Systems

David Daharewa Gureya: dgureya@gsd.inesc-id.pt
Advisor: João Barreto: joao.barreto@tecnico.ulisboa.pt
INESC-ID Lisboa/ Instituto Superior Técnico, Universidade de Lisboa, Portugal

Resumo—Modern NUMA multicore machines present complex memory access latency and bandwidth characteristics, making it hard to allocate memory optimally for a given program's access patterns. However, sub-optimal allocation can significantly impact programs execution performance. Therefore, it is important for a programmer or system designer to understand the architectural layout of a given system in which a program executes so as to utilize the underlying systems' resources efficiently. However, NUMA systems present the challenge to the programmer on how to allocate data in such a way that memory access latency is minimized and bandwidth is maximized. Additionally, NUMA architectures form a challenge for the programming models and runtime systems that need to effectively distribute execution load on available resources. Therefore, the main goal of this PhD thesis is to design solutions that will benefit programmers and runtime systems for asymmetrically connected NUMA systems.

Index Terms—Performance, Asymmetric NUMA Systems

1 Introduction

Non-uniform memory access (NUMA) architectures present the challenge to the multithreading software developer on how to allocate data in such a way that memory access latency is minimized and bandwidth is maximized. Additionally, NUMA architectures form a challenge for the programming models and runtime systems that need to effectively distribute execution load on available resources. For applications and benchmarks that deal with high amount of data and tasks, the need for an effective load distribution becomes of great importance. Executing such applications on NUMA systems without considering hardware architecture characteristics and application's memory behavior, has major impact on performance and results in significant slowdowns. More precisely, other than remote memory accesses, shared resource contention can lead to performance degradation. This is because the performance of parallel application depends on; memory allocation, thread placement, and data structures used (distributed/replicated). The problems that may arise on a NUMA system include: Congestion on the interconnect, congestion on the memory controllers, load imbalance of the memory controllers, and suboptimal allocation can significantly impact the performance of parallel programs.

Therefore, applications sensitive to performance can require complex logic to handle memory with divergent memory characteristics. NUMA support has been around for awhile in various operating systems. In most scenarios, an operating system can simply be run on a NUMA system, providing decent performance for typical applications. In these cases, kernel NUMA support frequently optimizes process execution without the need for user intervention. However, when the heuristics provided by the operating system do not provide satisfactory application performance to the end user, special NUMA configurations through tools and kernel configurations are needed. For instance, in high-performance computing, high-frequency trading, and for realtime applications, this is typically the case. For regular enterprise-class applications, these issues have also recently become more significant [16].

The main goal of this PhD thesis[1] is to design solutions that will benefit programmers and runtime systems for asymmetrically connected NUMA systems. This is challenging for a number of reasons: Efficient online measurement of communication patterns is challenging. For instance, it is important for a programmer or system designer to understand the architectural layout of a given system in which the program executes so as to utilize the underlying systems' resources efficiently e.g. to avoid remote data accesses as much as possible or to reduce pressure on congested resources. However, existing tools for examining NUMA topology, such as *numactl*, do not provide enough information about the asymmetry and heterogeneity presented by modern NUMA architectures. In addition, vendor specifications are often incomplete or vague. Worse hardware diversity is increasing as much as complexity. Secondly, changing the placement of threads and memory may incur high overhead. For example, migrating large amounts of memory can be extremely costly, hence thread migration must be done in way that minimizes memory migration. Running multiple applications simultaneously is also another challenge. Applications may have conflicting preferences and different communication patterns and are thus differently impacted by the connectivity between the nodes they run on. Lastly, finding the optimal placement is combinatorially difficult. The number of possible application placements can be very large and a brute-force approach to the problem is not possible as the search space grows rapidly [9].

1. This work was supported by national funds through Fundação para aCiência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

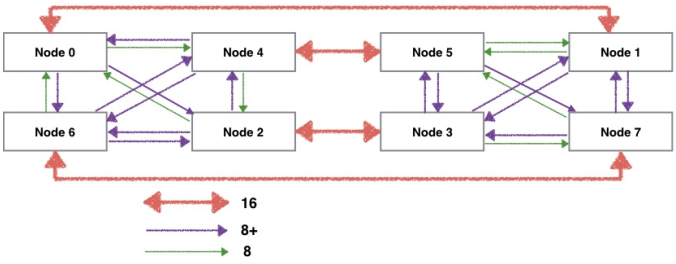

Figura 1. Modern NUMA systems, with eight nodes

2 Background and Related Work

Figure 1 depicts an AMD Opteron NUMA machine with eight nodes (each hosting six cores). Interconnect links exhibit many disparities. For instance, links have different bandwidths: some are 16-bit wide, some are 8-bit wide and others are 8plus(+)-bit wide. Furthermore, the asymmetry of interconnect links has dramatic and at times surprising effects on performance.

Optimizing thread and memory placement on NUMA systems has been extensively studied [11], [12], [1], [3], [2], [15]. The most common strategy for thread and data placement in NUMA is locating data as close as possible to cores. However, dynamic conditions of the architecture resources often leads to another allocation decisions. For instance, exploiting 2-hop distances remote memory rather than 1-hop distance, can reduce the memory access latency depending on the status of interconnect [1]. More precisely, when the nodes are connected by links of different bandwidth, we must consider not only whether the threads and data are placed on the same or different nodes, but how these nodes are connected. As another example, it turns out that interleaving dramatically reduces memory controller and interconnect congestion by alleviating the load imbalance and mitigating traffic hotspots, thereby improving the memory latency [2]. Additionally, when an IMC linked to local memory is congested, placing data in remote memory instead of local memory could yield better performance [3].

As a general comparison, we observed that most related work either perform thread or data mapping, but not both of them together. Thread mapping mechanisms such as [17], are not able to reduce the amount of remote memory accesses on NUMA architectures. On the other hand, data mapping mechanisms such as [2], are not able to reduce cache misses or correctly handle the mapping of shared pages. Existing mechanisms that perform both mappings together have several disadvantages. For instance, [1] uses a simpler/best-effort approach to find the best thread placements which might not be optimal and also only considers interconnect asymmetry as the only NUMA memory resource. On the other hand, [18]'s solution requires hardware support. Several other proposals require specific architectures, APIs or programming languages to work, limiting their applicability.

Recent research has shown that the performance effects of NUMA are significant and the problem is nontrivial. The relevance of this emerging problem is evident by the recent attention that it has received from the research community. However, this open problem is still at its infancy.

Therefore, our main idea for this thesis is to provide a thread and data mapping model that may lead to more accurate/optimal solutions. More precisely, we will extend the [1] solution with the [17] model. We will model the NUMA memory system factors that may impact optimal core allocations for memory intensive applications using Integer Programming techniques. Such factors include: hardware configuration, application memory behavior, and their asymmetry; local memory bandwidth and DRAM contention; and maximum inter-node memory bandwidths.

Referências

[1] Baptiste Lepers, Vivien Quéma, and Alexandra Fedorova. Thread and memory placement on NUMA systems: asymmetry matters. In Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '15). USENIX Association, Berkeley, CA, USA, 277-289.

[2] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quéma and Mark Roth. Traffic management: a holistic approach to memory placement on numa systems. In ASPLOS,2013.

[3] Z. Majo and T.R. Gross. Memory System Performance in a NUMA Multicore Multiprocessor. In SYSTOR, 2011.

[4] Bandwidth: a memory bandwidth benchmark for x86/x86_64 based Linux/Windows/MacOSX. http://zsmith.co/bandwidth.html

[5] Kim, Haecheon and Lim, Seungmin and Yoon, Junkee and Baek, Seungjae and Choi, Jongmoo and Cho, Seong-je. Analysis of Microarchitecture Resources Interference on Multicore NUMA Systems. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16).

[6] J.Rao, K. Wang, X. Zhou and C. Xu. Optimizing Virtual Machine Scheduling in NUMA Multicore Systems. In HPCA, 2013.

[7] PARSEC Benchmark Suite, http://parsec.cs.princeton.edu/

[8] Andreas Kleen. A numa api for linux. Technical report, SUSE Labs, April 2005.

[9] Kaestle, Stefan and Achermann, Reto and Haecki, Roni and Hoffmann, Moritz and Ramos, Sabela and Roscoe, Timothy. Machine-aware Atomic Broadcast Trees for Multicores. In OSDI, 2016.

[10] numactl, https://linux.die.net/man/8/numactl

[11] Renaud Lachaize, Baptiste Lepers, and Vivien Quéma. MemProf: A memory Profiler for NUMA Multicore Systems. In USENIX ATC, 2012

[12] Collins, Alexander and Harris, Tim and Cole, Murray and Fensch, Christian. LIRA: Adaptive Contention-Aware Thread Placement for Parallel Runtime Systems. In ROSS, 2015

[13] AutoNUMA: the other approach to NUMA scheduling. LWN.net, March 2012, https://lwn.net/Articles/488709/

[14] Toward better NUMA scheduling. Linux Weekly News, March 2012. https://lwn.net/Articles/486858/

[15] Majo, Zoltan and Gross, Thomas R. Matching Memory Access Patterns and Data Placement for NUMA Systems. In CGO, 2012

[16] Lameter, Christoph. NUMA (Non-Uniform Memory Access): An Overview. ACM Queue, July 2013.

[17] W. Wang, J. W. Davidson and M. L. Soffa, "Predicting the memory bandwidth and optimal core allocations for multi-threaded applications on large-scale NUMA machines,"2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), Barcelona, 2016, pp. 419-431.

[18] E. H. M. Cruz, M. Diener, M. A. Z. Alves, L. L. Pilla and P. O. A. Navaux, "Optimizing Memory Locality Using a Locality-Aware Page Table,"2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing, Jussieu, 2014, pp. 198-205.