Performance Evaluation 88–89 (2015) 18–36

Contents lists available at ScienceDirect

Performance Evaluation

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

Characterizing communication and page usage of parallel applications for thread and data mapping

Matthias Diener [a,*], Eduardo H.M. Cruz [a], Laércio L. Pilla [c], Fabrice Dupros [b], Philippe O.A. Navaux [a]

[a] Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
[b] BRGM, Orléans, France
[c] Department of Informatics and Statistics, Federal University of Santa Catarina, Florianópolis, Brazil

ARTICLE INFO

Article history:
Received 23 June 2014
Received in revised form 5 February 2015
Accepted 12 March 2015
Available online 20 March 2015

Keywords:
Shared memory
Thread mapping
Data mapping
Multicore
NUMA

ABSTRACT

The parallelism in shared-memory systems has increased significantly with the advent and evolution of multicore processors. Current systems include several multicore and multithreaded processors with Non-Uniform Memory Access (NUMA) characteristics. These architectures require the adoption of two strategies for the efficient execution of parallel applications: (i) threads sharing data should be placed in such a way in the memory hierarchy that they execute on shared caches; and (ii) a thread should have the data that it accesses placed on the NUMA node where it is executing. We refer to these techniques as thread and data mapping, respectively. Both strategies require knowledge of the application's memory access behavior to identify the communication between threads and processes as well as their usage of memory pages.

In this paper, we introduce a profiling method to establish the suitability of parallel applications for improved mappings that take the memory hierarchy into account, based on a mathematical description of their memory access behaviors. Experiments with a large set of parallel workloads that are based on a variety of parallel APIs (MPI, OpenMP, Pthreads, and MPI+OpenMP) show that most applications can benefit from improved mappings. We provide a mechanism to compute optimized thread and data mappings. Experimental results with this mechanism showed performance improvements of up to 54% (20% on average), as well as reductions of the energy consumption of up to 37% (11% on average), compared to the default mapping by the operating system. Furthermore, our results show that thread and data mapping have to be performed jointly in order to achieve optimal improvements.

1. Introduction

Since reaching the limits of Instruction Level Parallelism (ILP), Thread Level Parallelism (TLP) has become important to continue increasing the performance of shared-memory computer systems. Increases in the TLP are accompanied by more complex memory hierarchies, consisting of several private and shared cache levels, as well as multiple memory controllers that introduce Non-Uniform Memory Access (NUMA) characteristics. As a result, the performance of memory accesses depends on the location of the data [1,2]. Accesses to data that is located on local caches and NUMA nodes have a higher

* Corresponding author.
E-mail address: mdiener@inf.ufrgs.br (M. Diener).

http://dx.doi.org/10.1016/j.peva.2015.03.001
0166-5316/

bandwidth and lower latency than accesses to remote caches or nodes [3]. Improving the *locality* of memory accesses is therefore an important way to achieve optimal performance in modern architectures [4].

For parallel applications, locality can be improved in two ways. First, by executing threads that access shared data close to each other in the memory hierarchy, they can benefit from shared caches and faster intra-chip interconnections [5,6]. We refer to accesses to shared data as *communication* in this paper, and call an optimized mapping of threads to processing units that takes communication into account a communication-aware *thread mapping*. Most parallel programming APIs for shared memory, such as OpenMP and Pthreads, directly use memory accesses to communicate. Even implementations of the Message Passing Interface (MPI), which uses explicit functions to communicate, contain optimizations to communicate via shared memory, such as Nemesis [7] for MPICH2 [8] and KNEM [9] for Open MPI [10]. Second, the memory pages that a thread accesses should be placed on NUMA nodes close to where it is executing to reduce the inter-node traffic, as well as to increase the performance of accesses to the main memory [11]. We call this technique *data mapping*.

The goal of this paper is to characterize the memory access behavior of parallel applications to determine their suitability for mapping and evaluate their performance improvements using mappings that optimize locality. To characterize the communication, we introduce metrics that describe the spatial, temporal and volume properties of memory accesses to shared memory areas. In contrast to previous work that uses a logical definition of communication [12,13], we use a broader definition that focuses on the architectural impact of these accesses. We characterize the memory page usage of the applications by analyzing the distribution of accesses from different NUMA nodes during the execution. The characterizations are then used to perform an optimized thread and data mapping. Related work in this area mostly treats thread and data mapping as separate problems and only handles one of them [14,15]. We make the case that mapping has to be performed in an integrated way to achieve maximum benefits.

The main contributions of this paper are:

- We introduce metrics and a methodology to evaluate the communication and page usage of parallel applications running on shared memory architectures and use it to analyze their potential for thread and data mapping.
- We present a mechanism to employ this information and calculate thread and data mappings that optimize memory access locality.
- We characterize a large set of parallel applications and evaluate their performance and energy consumption improvements using the optimized mappings.

The rest of this paper is organized as follows. The next section briefly discusses the benefits of improved mappings and introduces the workloads that we use in this paper. Section 3 presents related work about characterization of memory access behavior and mapping. Our communication and page usage characterization methodologies are presented in Sections 4 and 5, respectively, together with evaluations of the workloads. Section 6 introduces our optimized mapping mechanism that uses the memory access behavior to perform thread and data mapping that maximizes locality. The experimental results of this mapping mechanism are presented in Section 7. Finally, Section 8 summarizes our conclusions and presents ideas for future work.

2. Background

2.1. Benefits of improved mappings

Thread and data mapping aim to improve the memory accesses to shared and private data in parallel applications. Thread mapping improves the usage of the interconnections by reducing inter-chip traffic that has a higher latency and lower bandwidth than intra-chip interconnections. It also reduces the number of cache misses of parallel applications. In situations where threads read the same data, executing the threads on the same shared cache reduces data replication, thereby increasing the cache space available for the application [16]. It can also reduce cache-to-cache transfers. In situations where threads write to the same memory addresses, an optimized thread mapping also reduces cache line invalidations. Data mapping improves the memory locality on NUMA machines by reducing the number of accesses to remote memory banks. As thread mapping, it improves the usage of the interconnections. This increases the memory bandwidth available in the system and reduces the average memory access latency.

It is important to note that thread mapping is a prerequisite for data mapping, for the two reasons depicted in Fig. 1. In the figure, an architecture consisting of two NUMA nodes with two cores per node is shown. Consider that two threads T_1 and T_2 are accessing a page P. The first benefit of thread mapping for data mapping is that it prevents unnecessary thread migrations between NUMA nodes, such that threads can benefit from the local data accesses. In Fig. 1(a), if page P was placed on the NUMA node where T_1 is executing and T_1 is migrated to the other node, the data mapping is ineffective.

The second reason is that data mapping alone is not able to improve locality when more than one thread accesses the same page, since the threads may be executing on cores of different NUMA nodes. In this situation, only the threads that are executing on the same node where the data is located can benefit from the increased locality. As shown in Fig. 1(b), when mapping T_1 and T_2 to the same NUMA node, accesses to page P by both threads will be considered as accesses to the local node. In contrast, if T_1 was executing on the other NUMA node, only accesses by T_2 would be considered local. By performing the thread mapping, threads that share a lot of data are executed on the same node, improving the effectiveness of the data mapping.