Parallel Computing 43 (2015) 43–63

Contents lists available at ScienceDirect

Parallel Computing

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

Communication-aware process and thread mapping using online communication detection

Matthias Diener [a,b,*], Eduardo H.M. Cruz [a], Philippe O.A. Navaux [a], Anselm Busse [b], Hans-Ulrich Heiß [b]

[a] Informatics Institute, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
[b] Communication and Operating Systems Group, Technische Universität Berlin, Germany

ARTICLE INFO

Article history:
Received 5 April 2014
Received in revised form 7 November 2014
Accepted 22 January 2015
Available online 2 February 2015

Keywords:
Shared memory
Parallel applications
Communication optimization
Mapping

ABSTRACT

The rising complexity of memory hierarchies and interconnections in parallel shared memory architectures leads to differences in the communication performance. These differences can be exploited to perform a communication-aware mapping of parallel applications to the hardware topology, improving their performance and energy efficiency. To perform the mapping, it is necessary to determine the communication behavior of the processes and threads of the application. Previous methods rely on static communication traces to detect communication, require hardware changes or support only a subset of parallelization models.

We propose CDSM, Communication Detection in Shared Memory, a mechanism that detects communication in from page faults and uses this information to perform the mapping. CDSM works on the operating system level during the execution of the parallel application and supports all parallelization models that use shared memory for communication. It does not require modifications to the applications, previous knowledge about their behavior, or changes to the hardware and runtime libraries. Experiments with the MPI, MPI+OpenMP and OpenMP implementations of the NAS parallel benchmarks, the HPCC benchmark and the PARSEC benchmark suite on a shared memory machine show that CDSM has a high detection accuracy with a negligible overhead. Execution time and processor energy consumption were reduced by up to 35.9% and 18.9%, respectively (10.2% and 7.3%, on average). Experiments on a cluster system, where CDSM optimizes the communication within each node, showed an average execution time reduction of 10.4%.

1. Introduction

In parallel shared memory architectures, the number of cores per processor and number of processors per system are increasing [1]. This leads to new challenges for the memory and interconnection subsystems, increasing their complexity and introducing a hierarchy for the memory requests from cores to the memory. With this hierarchy, memory accesses can be divided into different groups according to their performance. Accesses to local processor caches or local non-uniform memory access (NUMA) memories have a higher performance than accesses to remote caches or memories. For this reason,

* Corresponding author at: Informatics Institute, Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil.
E-mail addresses: mdiener@inf.ufrgs.br (M. Diener), ehmcruz@inf.ufrgs.br (E.H.M. Cruz), navaux@inf.ufrgs.br (P.O.A. Navaux), anselm.busse@tu-berlin.de (A. Busse), hans-ulrich.heiss@tu-berlin.de (H.-U. Heiß).

http://dx.doi.org/10.1016/j.parco.2015.01.005
0167-8191/

improving the *locality* of memory accesses is an important design goal for operating systems and applications, as it can increase the performance and energy efficiency of computer systems [2,3].

Parallel applications need to communicate in order to perform their tasks. This communication can be *explicit*, by using dedicated functions to send and receive data, or *implicit*, through memory accesses to shared memory segments. Programming models for explicit communication include the Message Passing Interface (MPI), while OpenMP and Pthreads use implicit communication. As communication via shared memory has a lower overhead, many common implementations of MPI provide extensions that enable faster intra-node communication through the use of shared memory, such as Nemesis [4] for MPICH2 [5] and KNEM [6] for Open MPI [7]. To exploit several levels of interconnections more efficiently, hybrid applications combine several paradigms, such as MPI and OpenMP [8,9]. As the communication performance varies greatly on modern system architectures, using the communication pattern to map processes and threads to cores is important to increase performance and save energy. We call this type of mapping *communication-aware mapping*.

To illustrate how the memory hierarchy can affect the communication performance, consider the shared memory architecture shown in Fig. 1, which consists of two NUMA nodes. Each node contains a multicore processor that also supports simultaneous multithreading (SMT). Two processes can communicate in three ways in this architecture. They are labeled a, b and c in the figure. (a) Processes that are executing on the same core on the SMT siblings can communicate via the fast L1 and L2 caches; (b) Processes that are executing on different cores of the same processor have to communicate via the slower L3 cache, but can still benefit from the fast intra-chip interconnection; (c) Processes that are executing on different processors need to communicate via the slow off-chip interconnection and have the lowest communication performance.

Communication-aware mapping in shared memory architectures has three benefits. First, it improves the efficiency of the interconnections, reducing inter-chip traffic that has a higher latency and lower bandwidth than intra-chip interconnections. Second, it reduces the number of cache misses of parallel applications. In a read–read situation, where two processes read the same data, the mapping reduces the replication of data in different caches, thereby increasing the cache space available to the application [10]. In the read–write or write–write case, an optimized mapping additionally reduces the number of cache lines that need to be invalidated in case of a write operation. Third, communication-aware mapping improves the memory locality on NUMA machines by keeping communicating processes on the same NUMA node.

In this paper, we propose *Communication Detection in Shared Memory (CDSM)*, a mechanism to perform communication-aware mapping in shared memory architectures. Related work in this area uses communication traces from previous executions to perform the mapping [11–13]. These mechanisms can not be used if the application changes its communication behavior with different inputs or a different number of processes. Most mechanisms for MPI-based applications focus on optimizing inter-node traffic in cluster environments, but do not improve the mapping inside each node [14–16]. Many proposals for mapping in shared memory systems use hardware counters that are highly hardware dependent [17], provide only indirect information about the communication behavior [18], or require changes to the hardware [19]. Finally, some related mechanisms only work for applications with specific parallelization models, such as ForestGOMP [20] for applications based on OpenMP. Our proposal overcomes these limitations. Although CDSM performs mapping in shared memory architectures, it also supports programming models for distributed memory, such as MPI, as long as they use shared memory to communicate.

Since communication is usually performed through memory accesses in shared memory platforms, these accesses can be used to detect the communication between processes and threads. CDSM gathers information about the memory access behavior of parallel applications and uses this information to map processes and threads to processing units according to their communication behavior. Communication is detected by analyzing the page faults caused by the parallel applications on the operating system level. Page faults in the same memory block caused by different processes or threads are considered as communication. The detected communication pattern is used together with information about the hardware topology to calculate the mapping with the objective of optimizing the communication of the application. We use an efficient mapping

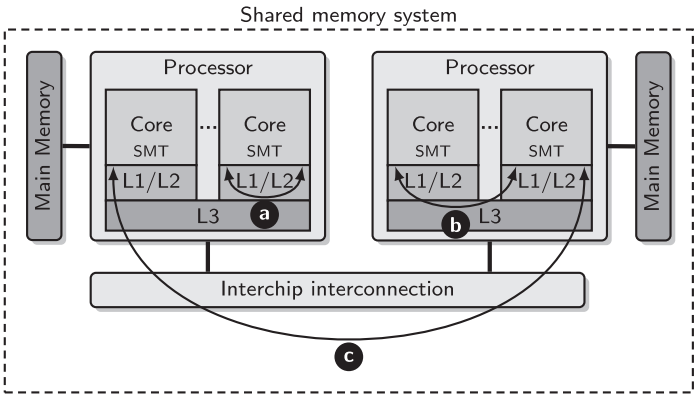

Fig. 1. Architecture of a shared memory system with three communication possibilities between two processes *a*, *b* and *c*.