Towards Autotuning by Alternating Communication Methods

Adrian Tineo
Swiss National Supercomputing Centre
Via Cantonale - Galleria 2
6928 Manno (TI), Switzerland
atineo@cscs.ch

Sadaf R Alam
Swiss National Supercomputing Centre
Via Cantonale - Galleria 2
6928 Manno (TI), Switzerland
alam@cscs.ch

Thomas C Schulthess
Swiss National Supercomputing Centre
Via Cantonale - Galleria 2
6928 Manno (TI), Switzerland
thomas.schulthess@cscs.ch

ABSTRACT

Interconnects in emerging high performance computing systems feature hardware support for one-sided, asynchronous communication and global address space programming models in order to improve parallel efficiency and productivity by allowing communication and computation overlap and out-of-order delivery. In practice though, complex interactions between the software stack and the communication hardware make it challenging to obtain optimum performance for a full application expressed with a one-sided programming paradigm. Here, we present a proof-of-concept study for an autotuning framework that instantiates hybrid kernels based on refactored codes using available communication libraries or languages on a Cray XE6 and a SGI Altix UV 1000. We validate our approach by improving performance for bandwidth- and latency-bound kernels of interest in quantum physics and astrophysics by up to 35% and 80% respectively.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: [Modeling techniques, Performance attributes]

General Terms

Performance

1. INTRODUCTION

A major challenge for scalable scientific applications is the widening gap in performance between processor and memory, a phenomena aggravated in the context of large distributed memory machines. With multi-chip multi-core (MCMC) nodes or accelerated nodes with GPU devices increasing the FLOP capacity per node, the interconnection problem is only bound to become ever more demanding. Memory locality is key, as the rate of access to data is determined by its distance to the processor, ranging from local memory, to a nearby socket or a distant network node.

Massively Parallel Processing (MPP) systems consisting of thousands of processors offer integrated solutions of hardware and software to meet the requirements of demanding scientific applications. Traditionally, the interconnection network in MPP systems has been optimized for distributed-memory, message-passing communication paradigms. However, the message-passing model may impose unnecessary communication overheads, so native remote memory access support is becoming increasingly attractive for highly scalable platforms. Such support is present for the Gemini interconnect in the Cray XE6 and the NUMAlink5 architecture in the SGI Altix UV 1000.

A variety of communication solutions for parallel programming are offered on these high-end MPP systems, including the message-passing MPI library as well as models that explicitly rely on one-sided and asynchronous communication primitives such as *put* or *get*. In these systems, the interaction between the compiler and runtime with the hardware layer becomes a complex one, especially if we consider multiple communication methods. The maturity of the software layer and its tuning with the networking architecture are key factors. In our experience, choosing the optimal communication solution for a scientific kernel on MCMC MPPs is highly challenging.

Although there is relevant work aimed to improve communication efficiency, by using non-blocking collectives [3], compiler-generated non-blocking communication [2], or joint compiler/runtime support [1], the combined use of available communication paradigms is largely unexplored.

We contribute to these efforts by combining, even within a single kernel instantiation, several communication methods available on the system, into *hybrid* solutions. We organize this process within an autotuning framework, which is able to generate optimum communication-intensive kernels for several target platforms, and could serve as the basis for the development of autotuning tools by system vendors. We validate our approach with two scientific kernels that belong to the areas of quantum physics [4] and astrophysics [5], which represent bandwidth- and latency-bound workloads respectively, achieving a relative improvement over a base version of up to 35% and 80%.

2. AUTOTUNING FRAMEWORK

Our autotuning framework is composed of three basic stages as shown in Fig. 1. First is *code refactoring*, which exposes the communication pattern so that it can be expressed with one-sided communication primitives. When possible, out-of-order message delivery is tolerated too. This transformation allows for maximum flexibility for the hardware and runtime to schedule the communication in the most efficient manner. Second, a thorough *platform profiling* phase

PMBS'11, November 13, 2011, Seattle, Washington, USA.
ACM 978-1-4503-1102-1/11/11.

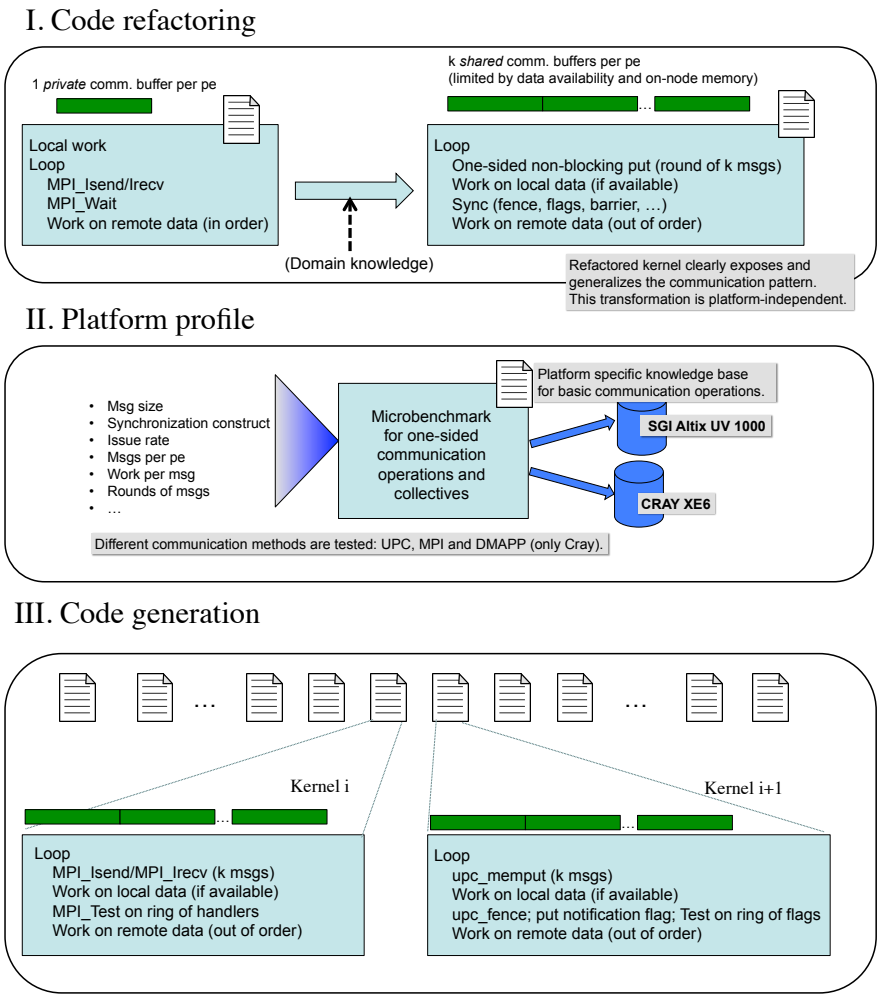

Figure 1: Autotuning framework, organized in 3 stages for (I) code refactoring, (II) platform profiling, and (III) code generation.

is performed to steer the code generation process. Vendors or users can create a knowledge database for the different communication methods available in the system, for a broad number of processors, messages sizes, etc. Third is *code generation*. By using the refactored algorithm template with flexible communication and the pruning provided by the platform profiling, different kernels can be generated, combining communication methods appropriately. The resulting implementations can then be explored by executing and evaluating them to locate the most efficient version for each combination of running conditions, such as problem size or number of processors.

3. CONCLUSIONS

We present a proof-of-concept for an autotuning framework targeting different communication methods on two different HPC architectures with specific support for remote memory access, a Cray XE6 and a SGI Altix UV 1000. While several communication strategies can be deployed in such systems, it is very challenging to find the optimal implementation for a specific kernel in all running conditions of interest. Our framework defines the strategy to perform the automatic selection of such optimal kernel version. The full paper can be found in [6].

4. REFERENCES

[1] Chen, W.-Y., Bonachea, D., Iancu, C., and Yelick, K. Automatic Nonblocking Communication for Partitioned Global Address Space Programs. In *Proceedings of the 21st annual international conference on Supercomputing* (New York, NY, USA, 2007), ICS '07, ACM, pp. 158–167.

[2] Danalis, A., Pollock, L., Swany, M., and Cavazos, J. MPI-aware Compiler Optimizations for Improving Communication-Computation Overlap. In *Proceedings of the 23rd international conference on Supercomputing* (New York, NY, USA, 2009), ICS '09, ACM, pp. 316–325.

[3] Hoefler, T., Gottschling, P., Lumsdaine, A., and Rehm, W. Optimizing a Conjugate Gradient Solver with Non-blocking Collective Operations. *Parallel Computing 33*, 9 (2007), 624 – 633.

[4] Moessner, R., and Sondhi, S. L. Ising Models of Quantum Frustration. *Phys. Rev. B 63* (May 2001), 224401.

[5] Speck, R., Gibbon, P., and Hoffmann, M. *Advances in Parallel Computing*. 2010, ch. Efficiency and Scalability of the Parallel Barnes-Hut Tree Code PEPC, pp. 35 – 42.

[6] Tineo, A., Alam, S.R., and Schulthess, T.C. Towards Autotuning by Alternating Communication Methods *SIGMETRICS Performance Evaluation Review 40(2)* (2012)