Scaling OpenMP Programs to Thousand Cores on the Numascale Architecture

Dirk Schmidl
IT Center, RWTH Aachen University
schmidl@itc.rwth-aachen.de

Atle Vesterkjær
Numascale AS
av@numascale.com

I. Introduction

The downside of shared memory programming compared to message passing is the limitation to run on a single system, whereas message passing allows to run applications on a cluster of shared memory nodes. Numascale's interconnect couples several machines in a cache coherent way to form a single system on multiple boards which allows shared memory programming on the complete machine. However, this does not necessarily mean that shared memory programs deliver satisfying performance on such a system. In this work we investigated a Numascale machine with 1728 cores hosted at the University of Oslo's Center for Information Technology. The system consists of 72 IBM x3755 M3 nodes coupled in a 3D torus network topology with Numascale's interconnect. We investigate the memory bandwidth with kernel benchmarks and furthermore look at an application developed at the Institute of Combustion Technology at RWTH Aachen University, namely TrajSearch. We describe all tuning steps done so far to optimize the application for large SMP machines like the Numascale machine and present good performance results for OpenMP runs with 1024 threads on the Oslo system. The structure of this work is as follows: first, we describe the Numascale technology and the test system in Sec. II. Second, we present performance results on the system and describe our tuning steps for the TrajSearch code in Sec. III before we conclude and discuss future steps needed in Sec. IV.

II. The Numascale System

The University of Oslos Center for Information Technology, USIT operates the Numascale installation used for this study. The systems is a PRACE prototype that it is financially supported by the PRACE-1IP project.

Configuration:

- 72 IBM x3755 M3 nodes
- 144 AMD 6174 CPUs
- 1728 Cores
- 4.6 TB Memory
- 3D Torus Interconnect (3x6x4)

The Numascale technology enables the realization of large scale shared memory systems from commodity servers enabling large shared memory systems to be built at the price of clusters. This is achieved through the use of Numascales adapter card connected to the HyperTransport processor bus of standard AMD based servers. The technology uses a low latency torus fabric based on the SCI standard. The resulting system is a cache coherent Non-Uniform Memory Access (ccNUMA) machine.

Shared memory is attractive to developers, as any processor can access any data in the system through direct load and store operations, thus making the programming of the system easier and the code less error-prone. Performance scaling on large scale systems is generally not straightforward. The allocation of shared data can have a significant impact on the performance. On the other hand, tuning of an application may be very beneficial for performance and much less demanding than rewriting an application for a message passing model.

The Numascale system presents itself as a single system with many cores and large memory for the programmer. It runs a single image standard Linux operating system and requires no virtualization software layers. Running a single OS image simplifies the operation of the system. The Numascale adapter translates the standard server snooping cache coherency protocol to its own directory based protocol and handles all remote memory accesses system wide. All IO for the system is also controlled and run under the single OS.

III. Performance Results

Many scientific applications are limited by the memory bandwidth of modern computers. NUMA systems theoretically increase the available memory bandwidth with every socket in the system, since a memory controller and memory DIMMs are added with every processor. In practice the bandwidth does not increase linearly with the number of sockets, since the cache coherence traffic introduces a bit of overhead and the overhead may rise with the number of sockets as well. We therefore investigated the reached bandwidth with the STREAM [1] benchmark on the Numascale system for an increasing number of boards used. The code was compiled with the Oracle Solaris Studio 12.3 Compiler.

Figure 1 shows the reached bandwidth for the triad kernel of the STREAM benchmark for an increasing number of boards used and for a memory footprint of 2 GB per board. As expected the bandwidth does not increase perfectly linear, but an increase of about 7 can be observed going from 8 to 72 boards. Overall a bandwidth of more than 2 TB/s is reached on the complete machine.

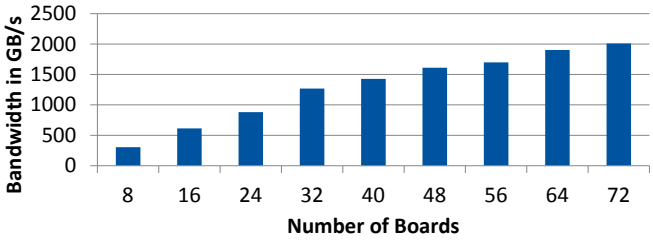

Fig. 1. Scalability of the memory bandwidth reached on the system with an increasing number of boards used.

A. *TrajSearch*

The STREAM benchmark has shown a good scalability on the Numascale system, so we next tried to run the application TrajSearch. TrajSearch is used to investigate turbulences which occur during combustion. It is a post-processing code for dissipation element analysis developed at the Institute of Combustion Technology of RWTH Aachen University by Peters and Wang [2]. A dissipation element is defined as a volume from which all trajectories reach the same minimum and maximum point. The dissipation element analysis provides a deeper understanding of turbulence and can be employed to reconstruct important statistical properties as has been shown by Gampert and Göbbert [3]. The code has been parallelized with OpenMP and optimized for large SMP machines as described in [4].

The optimization steps done before include:

- Some data structures were replicated for every thread which reduced the cache traffic between nodes.
- The synchronization overhead through OpenMP critical sections has been reduced by introducing local buffers and writing back larger chunks in the shared data structures.
- A thread local random number generator (RNG) has been implemented to replace the standard Fortran RNG which requires synchronization between threads. The numbers generated are no longer statistically independent between different threads, but they are sufficient for the purpose of the code.
- The memory placement was optimized to increase the number of local memory accesses in the shared data structures.
- The OpenMP loop scheduling, which was dynamic to act against the load imbalance in the computation loop, was replaced by a self implemented NUMA aware loop scheduler to further maximize the number of local memory accesses.

A more detailed description of these steps can be found in [4]. We have shown good scalability to over a hundred cores for TrajSearch in the previous work. On the Numascale machine we optimized the parameters of the scheduler to use larger chunks and we increased the size of the local buffers. Since the machine offers 4.5 TB of memory the larger buffers fit into memory and they helped to further reduce the synchronization overhead. Again we used the Oracle Solaris Studio 12.3 Compiler and we used a medium dataset with a memory footprint of about 45 GB.

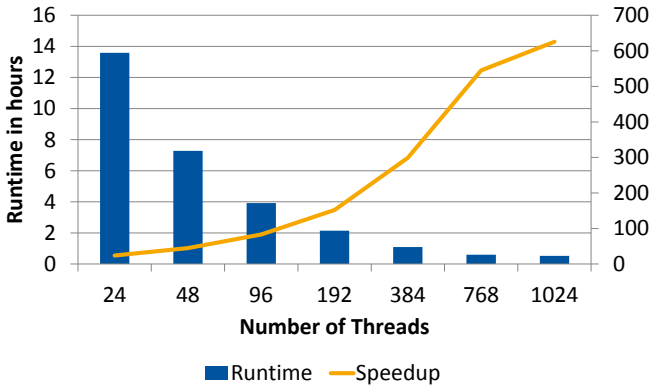

Fig. 2. Runtime and speedup of the TrajSearch code for an increasing number of threads.

Figure 2 shows the runtime and speedup on the Numascale machine. Because a serial run would have taken about 2 weeks, we took the 24 threads run as basis for all other speedup calculations. We assume a speedup of 24 for 24 threads. For an increasing number of threads the runtime declines up to 1024 threads, where a speedup of about 625 can be observed. We could not use more than 1024 threads so far, because the Oracle compiler only supports thread placement for up to 1024 threads.

IV. Conclusion and Future Work

In this work we investigate a 1728 core Numascale system hosted at the University of Oslo. We have shown that the available bandwidth increases with the number of boards in the system and that a total bandwidth of over 2 TB/s is available for users. Furthermore, we presented tuning steps done for the TrajSearch code to optimize for large NUMA systems. With these optimizations we achieved a speedup of about 625 using 1024 threads on the system. Future work is to implement thread binding in the code itself, to circumvent the limitations of the Oracle compiler for thread placement. This should allow test runs on all 1728 cores of the system. Although the scalability is quite good with 625 on 1024 threads, we want to figure out the root cause for the limits in scalabilty and if possible further optimize the code to achieve even better performance results.

References

[1] J. D. McCalpin, "STREAM: Sustainable Memory Bandwidth in High Performance Computers," 1995. [Online]. Available: http://www.cs.virginia.edu/stream/

[2] N. Peters and L. Wang, "Dissipation element analysis of scalar fields in turbulence," *C. R. Mechanique*, vol. 334, pp. 493–506, 2006.

[3] M. Gampert, J. H. Göbbert, M. Gauding, P. Schfer, and N. Peters, "Extensive strain along gradient trajectories in the turbulent kinetic energy field," *New Journal of Physics*, 2011.

[4] N. Berr, D. Schmidl, J. H. Göbbert, S. Lankes, D. an Mey, T. Bemmerl, and C. Bischof, "Trajectory-Search on ScaleMP's vSMP Architecture," in *Applications, Tools and Techniques on the Road to Exascale Computing : proceedings of the 14th biennial ParCo conference ; ParCo2011 ; held in Ghent, Belgium*, ser. Advances in Parallel Computing ; 22. New York, NY: IOS Press, 2012.