OpenMP tutorial

by Brent Swartz

February 18, 2014
Room 575 Walter
1-4 P.M.
Outline

• OpenMP Definition
• MSI hardware Overview
• Hyper-threading
• Programming Models
• OpenMP tutorial
• Affinity
• OpenMP 4.0 Support / Features
• OpenMP Optimization Tips
OpenMP Definition

The OpenMP Application Program Interface (API) is a multi-platform shared-memory parallel programming model for the C, C++ and Fortran programming languages. Further information can be found at http://www.openmp.org/
OpenMP Definition

Jointly defined by a group of major computer hardware and software vendors and the user community, OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from multicore systems and SMPs, to embedded systems.
MSI Hardware

- MSI hardware is described here:

https://www.msi.umn.edu/hpc
MSI Hardware

The number of cores per node varies with the type of Xeon on that node. We define: \( \text{MAXCORES} = \text{number of cores/node} \), e.g. Nehalem=8, Westmere=12, SandyBridge=16, IvyBridge=20.
MSI Hardware

Since MAXCORES will not vary within the PBS job (the itasca queues are split by Xeon type), you can determine this at the start of the job (in bash):

```
MAXCORES=`grep "core id" /proc/cpuinfo | wc -l`
```
MSI Hardware

Most Itasca nodes (all batch queue nodes) are Nehalem nodes. The 51 Sandy Bridge nodes are described in:

https://www.msi.umn.edu/content/itascasb

Itasca nodes are not shared between users.
Hyper-Threading

Hyper-threading is Intel's proprietary simultaneous multithreading (SMT) implementation used to improve parallelization of computations (doing multiple tasks at once). It is described here:

http://en.wikipedia.org/wiki/Hyper-threading
Hyper-Threading

For each processor core that is physically present, the operating system addresses two virtual or logical cores, and shares the workload between them when possible.
Hyper-Threading

The main function of hyper-threading is to decrease the number of dependent instructions on the pipeline. It takes advantage of superscalar architecture (multiple instructions operating on separate data in parallel).
Hyper-Threading

They appear to the OS as two processors, thus the OS can schedule two processes at once. In addition two or more processes can use the same resources. If resources for one process are not available, then another process can continue if its resources are available.
Hyper-Threading

Hyper-threading is disabled on all itasca nodes, except for the Ivy Bridge (gagliardi_ivy queue) nodes. When hyper-threading is enabled, programs can use double the number of threads, since 2 threads are mapped per core. This can give performance changes of from -5% to +30%, depending upon the application.
Programming Models

There are several ways to implement parallel programs. For best efficiency in all cases, you should use SIMD (vectors) on the innermost loops. Three of the more common models are:
Programming Models

- MPI only (Distributed memory / InterNode Communications). Usually mapped as one process per core. Able to run across multiple nodes, and thus use much more memory.
Programming Models

- MPI only: Message-passing has become accepted as a portable style of parallel programming, but has several significant weaknesses that limit its effectiveness and scalability. Message-passing in general is difficult to program and doesn't support incremental parallelization of an existing sequential program.
Programming Models

- MPI only: Message-passing was initially defined for client/server applications running across a network, and so includes costly semantics (including message queuing and selection and the assumption of wholly separate memories) that are often not required by tightly-coded scientific applications running on modern scalable systems with globally addressable and cache coherent distributed memories.
Programming Models

- OpenMP only (Shared memory / IntraNode Communications). Usually mapped one thread per core, unless hyper-threading is enabled, in which case 2 threads are mapped per core. Generally easier to program, but you are limited to your shared memory space (one node on Itasca).
Programming Models

Hybrid MPI/OpenMP program organization:

- MPI (Distributed memory / InterNode Communications) mapped one MPI process per node (or one process per socket).
- OpenMP (Shared memory / IntraNode Communications) mapped one thread/core.
- SIMD (vector computation on a single core/thread on the innermost loop(s))
OpenMP tutorial link

• The OpenMP tutorial we will cover is described here:

https://computing.llnl.gov/tutorials/openMP/
Affinity

The Intel® runtime library has the ability to bind OpenMP threads to physical processing units. The interface is controlled using the KMP_AFFINITY environment variable. Depending on the system (machine) topology, application, and operating system, thread affinity can have a dramatic effect on the application speed.
Affinity

Most MSI systems have hyper-threading turned off, so the physical processing units are the cores (processors) on a node. A full description of KMP_AFFINITY is found here: http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/2011Update/cpp/lin/optaps/common/optaps_openmp_thread_affinity.htm
Affinity

A full description of KMP_AFFINITY is found here:
http://software.intel.com/en-us/node/463446
**Affinity**

MSI staff found KMP_AFFINITY=compact to be optimal when hyper-threading is not enabled (at least for the code we were using on SandyBridge Xeons). When hyper-threading is enabled, KMP_AFFINITY=scatter was optimal. Every application has different performance characteristics, so we recommend that you try running your application with these different settings (compact, scatter) to see which is optimal for it.
Affinity

The Intel compiler options to use to get an executable that can efficiently use both Nehalem and SandyBridge (and IvyBridge) nodes are:

-xSSE4.2 -axAVX
The GNU compilers have an environment variable called GOMP_CPU_AFFINITY similar to KMP_AFFINITY, although you must specify the CPU numbers explicitly. See:

http://gcc.gnu.org/onlinedocs/gcc-4.3.0/libgomp/GOMP_CPU_AFFINITY.html
MSI OpenMP 4.0 Support

• At MSI, the only compiler which currently has OpenMP 4.0 support (combined directives are not supported yet) is the latest Intel/cluster module, loaded using:

% module load intel/cluster
OpenMP 4.0 was released in July 2013. The new features included in OpenMP 4.0 are:

- Support for accelerators. The OpenMP 4.0 specification effort included significant participation by all the major vendors in order to support a wide variety of compute devices.
MSI OpenMP 4.0 Support

OpenMP provides mechanisms to describe regions of code where data and/or computation should be moved to another computing device. Several prototypes for the accelerator proposal have already been implemented.
MSI OpenMP 4.0 Support

- SIMD constructs to vectorize both serial as well as parallelized loops. With the advent of SIMD units in all major processor chips, portable support for accessing them is essential. OpenMP 4.0 provides mechanisms to describe when multiple iterations of the loop can be executed concurrently using SIMD instructions and to describe how to create versions of functions that can be invoked across SIMD lanes.
MSI OpenMP 4.0 Support

• Thread affinity. OpenMP 4.0 provides mechanisms to define where to execute OpenMP threads. Platform-specific data and algorithm-specific properties are separated, offering a deterministic behavior and simplicity in use. The advantages for the user are better locality, less false sharing and more memory bandwidth.
MSI OpenMP 4.0 Support

- Error handling. OpenMP 4.0 defines error handling capabilities to improve the resiliency and stability of OpenMP applications in the presence of system-level, runtime-level, and user-defined errors. Features to abort parallel OpenMP execution cleanly have been defined, based on conditional cancellation and user-defined cancellation points.
MSI OpenMP 4.0 Support

• Tasking extensions. OpenMP 4.0 provides several extensions to its task-based parallelism support. Tasks can be grouped to support deep task synchronization and task groups can be aborted to reflect completion of cooperative tasking activities such as search. Task-to-task synchronization is now supported through the specification of task dependency.
MSI OpenMP 4.0 Support

• Support for Fortran 2003. The Fortran 2003 standard adds many modern computer language features. Having these features in the specification allows users to parallelize Fortran 2003 compliant programs. This includes interoperability of Fortran and C, which is one of the most popular features in Fortran 2003.
MSI OpenMP 4.0 Support

- User-defined reductions. Previously, OpenMP only supported reductions with base language operators and intrinsic procedures. With OpenMP 4.0, user-defined reductions are now also supported.
MSI OpenMP 4.0 Support

- Sequentially consistent atomics. A clause has been added to allow a programmer to enforce sequential consistency when a specific storage location is accessed atomically.
MSI OpenMP 4.0

• Intel OpenMP 4.0 features:

MSI OpenMP 4.0

- Intel OpenMP 4.0 features (includes a team examples):

MSI OpenMP 4.0

- Intel OpenMP 4.0 features (includes a first touch example):

http://openmp.org/sc13/OpenMP4.0_Intro_YonghyongYan_SC13.pdf
MSI OpenMP 4.0

• SIMD video from Intel:

MSI OpenMP 4.0

- OpenMP 4.0 examples:

http://openmp.org/wp/2014/02/updated-openmp-40-examples/
MSI OpenMP 4.0

OpenMP and OpenACC are actively merging their specification while continuing to evolve. A first step at merging has been made with the release of OpenMP 4.0. OpenACC implementations can be considered to be a beta test of the OpenMP accelerator specification.

OpenACC has been created and implemented by NVIDIA, PGI, Cray, and CAPS in order to address their immediate customer needs.
OpenACC has been created and implemented by NVIDIA, PGI, Cray, and CAPS in order to address their immediate customer needs.

An OpenACC tutorial will be given at MSI on March 25. OpenACC supports nVidia GPGPUs via the PGI compiler at MSI.
OpenMP Optimization Tips

- In general it should be fastest to parallelize the outer loop only, provided there is sufficient parallelism to keep the threads busy and load balanced. Parallelizing the inner loop only adds an overhead for every parallel region encountered which (although dependent on the implementation and the number of threads) is typically of the order of tens of microseconds. Parallelizing both loops is unlikely to be the most efficient solution, except maybe in some corner cases.
OpenMP Optimization Tips

- Sometimes, if the outer loop doesn't have much parallelism but the inner loop does, there is the option to put the `#pragma omp parallel` outside the outer loop, and the `#pragma omp for` before the inner loop. This amortizes the overhead of creating the parallelism outside the outer loop, runs the outer loop 'redundantly' on all threads, and work-shares the iterations of the inner loop, which can be better than parallelizing either the outer or the inner loop alone.

University of Minnesota
OpenMP Optimization Tips

- Avoid thread migration for better data locality. All scripts on MSI systems should set env var OMP_PROC_BIND=true although the current system gcc (4.4.6) does not use it, gcc versions 4.7 and above do use it (And Intel compiled apps, too).
OpenMP Optimization Tips

Setting OMP_PROC_BIND=true will eliminate cache invalidations due to processes switching to different cores. It will not necessarily force OPTIMAL thread to core binding (to do this you need to set KMP_AFFINITY for Intel compiled apps, GOMP_CPU_AFFINITY for gcc compiled apps). OMP_PROC_BIND was added in OpenMP 3.1.
OpenMP Optimization Tips

False sharing can cause significant performance degradation. See:

User Support Info

User Support:
E-mail: help@msi.umn.edu
Phone: (612) 626-0806
Webpage:
http://www.msi.umn.edu