Comparison of Parallel Programming Models on Clusters of SMP Nodes

R. Rabenseifner[1], and G. Wellein[2]

Abstract: In High Performance Computing, most systems are clusters and each node of a cluster is a parallel shared memory system (SMP node). The SMP nodes vary in size and quality of CPUs in a wide range. There are dual-CPU PC systems with BUS based memory access, nodes with 8, 16 or 24 CPUs with higher memory bandwidth or independent memory paths in the best case, and memory and CPU may implement a cache based access, or a pseudo-vector or full vector access. All these hybrid clusters of SMP nodes can be programmed with pure MPI, i.e., one MPI process is running on each CPU. This programming model implies an additional message transfer overhead between the CPUs within an SMP node. To prohibit this overhead, hybrid programming models are proposed, combining MPI with OpenMP.

MPI is used only for the inter-node communication and OpenMP (or automatic parallelization) is used for the parallelization inside of an SMP node. In the most commonly used hybrid model, the MPI routines are called only outside of OpenMP parallel regions, i.e., they are called only from the master threads and only while all other threads in the calling node are sleeping. Although this model avoids massaging overheads inside of the SMP nodes, there are even so serious drawbacks: First, on most platforms, the master thread is not able to use the full bandwidth of the inter-node network. This is shown with different benchmarks on several platforms. Second, if all threads except the master thread are sleeping and dedicated CPUs are used, then all MPI communication must be weighted by the number of threads used in each MPI process.

To overcome these drawbacks, several strategies are discussed. On large SMP nodes, the node can be virtually separated into several MPI processes with a smaller amount of threads to guarantee that the inter-node bandwidth can be used, and that therefore the total cost of communication is minimized. To minimize idle time of sleeping threads, the communication and computation can be overlapped. The implications on the application itself and on the usage of OpenMP and MPI and on load balancing issues are discussed. A benchmark with a Jacobi-Davidson solver shows a significant performance win (50 %) with a strategy that reduces the complexity of the load balancing by reserving a fixed amount of resources for communicating.

[1] High-Performance Computing-Center (HLRS), University of Stuttgart
Allmandring 30, D-70550 Stuttgart, Germany, *rabenseifner@hlrs.de*

[2] Regionales Rechenzentrum Erlangen
Martensstraße 1, D-91058 Erlangen, Germany, *gerhard.wellein@rrze.uni-erlangen.de*