OpenMP vs. MPI on a Shared Memory Multiprocessor

J. Behrens (GWDG), O. Haan (GWDG), L. Kornblueh (MPI f. Meteorology)

Abstract

The change in parallel computer architecture from distributed memory systems, as e.g. Cray T3E, to (clusters of) shared memory multiprocessors, as e.g. IBM SP or pSeries 690, was accompanied by a widening gap between processor and communication speed, e.g. the product of peak performance and latency for MPI point-to-point communication is 6 for T3E1200 and 50 for pSeries 690 in units Gflop/s times microseconds. The speed up of a parallel application will be reduced accordingly on the new platform

Two routes can be taken to improve the situation: 1. Optimising the communication scheme for the application with regard to the communication granularity. Reducing the granularity diminishes the impact of latency. 2. Using the shared memory programming model on the shared memory system. This will reduce the software overhead of the MPI implementation, which is largely responsible for the latency of message passing in shared memory.

We report on results for this two step procedure for a particular example, the climate simulation program ECHAM5. With MPI parallelisation ECHAM5 for a 128x64x19 grid achieved a speed up of 30 on a 32 processor Cray T3E. With minor changes the same program on a 32 processor pSeries690, using the IBM's shared memory MPI implementation sped up to only 14.

The granularity of communication could be reduced for the transport phase of ECHAM5, which rearranges the global data structure between different phases of local computations. For this part of the code the parallel efficiency improved from 11 to 15. This improvement is the net effect from several modifications of the original communication scheme: 1. combining the rearrangements for all layers into a single communication phase, 2. changing the distribution of layers to tasks, 3. increasing the message size by using derived MPI data types.

The shared memory parallelisation of the transport part of ECHAM5 was straightforward, starting from the serial version. Inserting parallelisation directives around the loops stepping through the horizontal layers and declaring common blocks with intermediary data as thread-private were the only changes needed. The speed up was poor, because cache misses increased with the number of threads. This unexpected behaviour disappeared after reducing the data size during the local computations, which interleave the data rearrangements. After this reduction the increase of cache misses disappeared, the OpenMP parallelisation showed a speed up of 20 on 32 processors.

There have been comparisons of shared memory and message passing parallelisation for various applications in the literature. Here we add an example with clear advantage to the use of shared memory. The direct use of shared memory communication obviously reduces the communication overhead in comparison to the case when the additional layer of MPI calls is involved. A further advantage within the OpenMP framework is the availability of dynamic load scheduling which can be used to overcome speedup limitation due to load imbalance. Unfortunately with increasing numbers of processors and small iteration space this procedure is not efficient. We implemented a different scheduler that extrapolates collected timings and considers distribution overhead that leads to a much higher speedup with OpenMP. The MPI version does not benefit from this scheme due to additional overhead, e.g., initialisation of domain decompositions, and the importance of communication limitations.