Blocking vs. Non-blocking Communication under MPI on a Master-Worker Problem

André Fachat, Karl Heinz Hoffmann
Institut für Physik
TU Chemnitz
D-09107 Chemnitz
Germany
e-mail: fachat@physik.tu-chemnitz.de
Phone: ++49-371-531-3550
Fax: ++49-371-531-3233

June 5, 1998

Abstract

In this report we describe the conversion of a simple Master-Worker parallel program from global blocking communications to non-blocking communications. The program is MPI-based and has been run on different computer architectures. By moving the communication to the background the processors can use the former waiting time for computation. However we find that the computing time increases by the time the communication time decreases in the used MPICH implementation on a cluster of workstations. Also using non-global communication instead of the global communication slows the algorithm down on computers with optimized global communication routines like the Cray T3D.

Keywords: MPI, MPICH, blocking communication, non-blocking communication

1 Introduction

The available computing power has vastly increased since the invention of the microprocessor. Moore's law even says that the increase in computing power per processor goes exponentially in time. However, the price for the custom processors with the best performance often is prohibitivly larger than for off-the-shelf processors. And the need for computing power is increasing faster than the processor development can keep up with.

This led to parallel computers, where several, sometimes up to thousands of cheap, off-the-shelf processors are coupled to one large multprocessing machine. In this machine all processors run in parallel, thus increasing the power by the factor N, the number of processors. However, most programs are written to be run on *one* processor, not many. Therefore the programs have to be rewritten [1].

There are two major parallel programming doctrines. The first is to let the compiler automatically parallelize a given serial program. But the compiler cannot know all parts of the program that can be parallelized. More effective is the second way, to explicitly code parallelism into the algorithm. This is normally done with message-passing libraries (There are shared-memory machines but we do not handle them here). These libraries are subroutines that provide means of communication between processors to the program. In this paper we use the *Message Passing Interface MPI* [2, 3]. It is a successor to PVM and is supported by all major vendors of parallel computers. There are implementations for systems from Linux workstation clusters up the Cray T3E for example.

If, in a parallel program, one processor wants to tell another processor something the processor has to send a message. For this the processor sets up a message buffer, stores the message in the buffer and calls the `send` subroutines. Then the buffer is given to the operating system's communication routines, and sent to the other processor(s). The receiving processor then delivers the message to a `receive` call.

One important point is *when* the sending process returns from the `send` call. *Blocking* communication lets the sender return from the call only when the receiving process has taken over the buffer. *Non-blocking* communication immediately returns from the `send` call, but the communication buffer is still in use because the message has not been sent already. Therefore the sending process has to check if it can release the send buffer with an extra library call. The time between those calls can be used for other computations for example.

Another important point is if the communication involves more than two communicating processes. MPI allows, with *one* library call, to send data from one to all processors, or gather data from all processes or even more complex operations. Operations where all processors are involved are called *global* communication operations. Two-process communication is called *non-global* or also *point-to-point* communication.

In this paper we will investigate the timing behaviour of a specific MPI master-worker application with global blocking communication compared to non-blocking non-global communication.

2 Our test problem

The test problem we used is our simulated annealing [4, 5] production code. The code is described in more detail in [6]. The main goal is to find the global minimum of an objective function with many local minima in a high-dimensional parameter space.

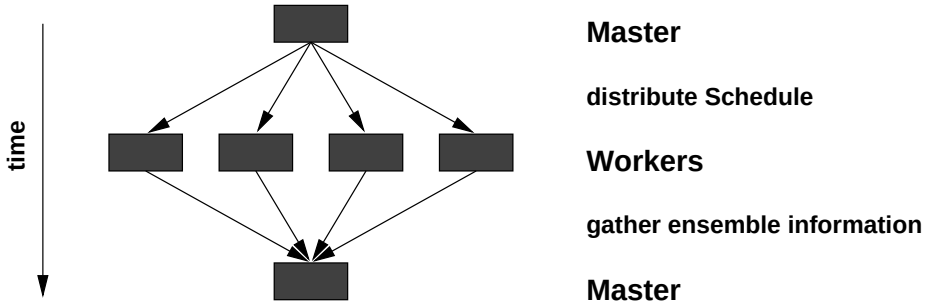

Figure 1: The original implementation with global, blocking communication.

The simulated annealing algorithm is a stochastic algortihm. A random state is generated as a starting point. This state is then randomly modified in a certain way to get a new state. If the objective function of the modified state is lower than the objective function of the original state, it is used as a new starting point. If the objective function is higher then the modified state is only used with a certain probability. This simulation is often called a 'random walker'. An important variable is the so-called *Temperature* T. It gives a measure for how often a new state with higher energy is accepted and is reduced during the run to freeze the walker in a low-lying minimum. The way to lower this temperature, i.e. the series T_i for $i = 0, 1, ..$ is called the *Schedule*. Here a fixed number of walker steps are done at the same temperature, before T is changed again.

To increase the sampled area in the state space an ensemble approach is used. This means that a large number of random walkers are simulated in parallel [7]. All walkers use the same temperature. Using an ensemble also means that certain statistic information is now available. This information is used to compute a new temperature for the next simulation step.

Now the algorithm has to be parallelized. An overview on parallelization techniques has been written by Greening [8]. The parallelization of the used algorithm is simple, the ensemble members are distributed among the different processors. One processor is designated as 'master'. It computes the schedule. This schedule is then sent to all 'worker' processors with a broadcast operation. All workers then do a fixed number of simulation steps on their part of the ensemble. After that the statistic information is sent back from the workers to the masters with a gather operation. Then the new schedule is computed and sent back to the workers.

The schedule sent to the workers consists of one `double` and two `int` values. This is the (integer) number of steps to do at the (double) temperature that is also sent. Also an integer flag is sent to control load balancing for example. The statistical information sent back consists of 4 `double` and one `int`. The workers sent the sum of all energies of their part of the ensemble, the sum of the energies squared, the time needed for the simulation and the best energy

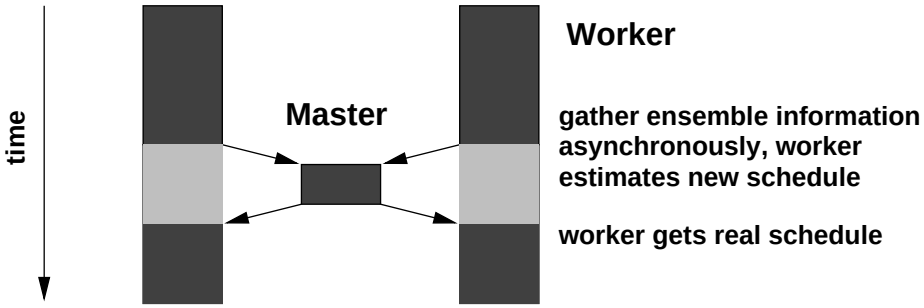

Figure 2: The algorithm using non-blocking communication and estimated schedule. The workers do a fixed number of steps between sending the statistic to the master. After sending they estimate their schedule and use this until the master sends the real schedule.

reached so far. Also the total number of accepted moves accross the local part of the ensemble is sent. The message size is independent of the ensemble size.

The approach with a master and several workers is also called *master-worker* approach. Our implementation, however, allows the master to be a worker between the schedule computations as well.

3 Non-blocking communication

During the development of our stochastic optimization code we found that in our workstation cluster the program needs about 5-20% of the time for the communication subroutines. In order to improve performance we wanted to know whether putting the communication into the background with the non-blocking communication would improve the performance.

This has several implications. First and most important is that the workers need something to do during the time that is now saved. They need the information from the master (the schedule) to do the simulation. This information is derived from an ensemble statistics. But, if a worker has several ensemble members, it can derive such an ensemble statistics for the local ensemble members. This can be used as an estimate until the real schedule arrives from the master.

This of course changes the character of the whole algorithm, as the communication time directly influences the schedule: It is not 'exact' in the sense that the schedule used is not the globally valid schedule for the whole ensemble. Instead a certain fraction of the simulation time the locally estimated schedule is used. Furthermore it is not 'reproducable', as factors outside program control – network latency and load, each CPU load – influence the schedule by changing the time where the estimated schedule is used.

The second implication is that it is not possible to use global communication routines anymore, as in MPI they are available as blocking only. The global

Setup	computation time [s]	communication time [s]	worker comp. time [s]	worker comm. time [s]
original	307.6	60.3	312.6	56.7
estimated	326.3	41.0	320.9	49.2
decoupled	300.4	0.0	296.2	0.0

Table 1: Typical result, with times for a master and a worker. When using non-blocking communication the time saved during the communication routines is added to the computation time. Sometimes the communication time is even reduced to 1% of the original time, but the computing time is increased by that amount again.

routines have therefore been broken up into the available non-blocking point-to-point operations. This does not matter much on a workstation cluster. But, as it is shown later, it does matter on the Cray T3D with its highly optimized global communication routines that are now not used anymore.

The resulting workflow can be seen in figure 2.

4 Results

The results are in general presented for three different scenarios. The first is the original implementation with global schedule. The second is the described non-global communication algorithm with partly estimated schedule. As a comparision to that there is a third scenario where the communication is completely disabled. Each processor computes its local schedule from the statistics of its local part of the ensemble, so it is a parallel run of several simulations with smaller ensemble size each. The three scenarios are called 'original', 'estimated' and 'decoupled' resp.

Simulations have been carried out on a four processor workstation cluster (Intel Pentium Pro 200 with Linux 2.0, connected with a 10 MBit Ethernet) as well as the Cray T3D at the Edinburgh Parallel Computing Center EPCC [9]. The Cray had a larger ensemble size than the Cluster, though. The MPI implementation used on the cluster is MPICH [3], that uses standard Unix networking and is available for many platforms that can interoperate. The Cray MPI version is a proprietary, optimized MPI implementation, that cannot interoperate with processors outside the T3D.

Several simulations have been run. A typical result for a run on our workstation cluster is shown in figure 3 and table 1. In other configuration or test (simulated annealing) problems the numbers vary, but they have one thing in common. It can be seen that using non-blocking communication does not give an improved performance. Instead the time that is saved with not waiting in the blocking communication routines seems to be added to the computation time. By comparing the decoupled results with the original and estimated results one can see the amount of time used for communication.

The situation on the Cray T3D for example is different. Here we have special

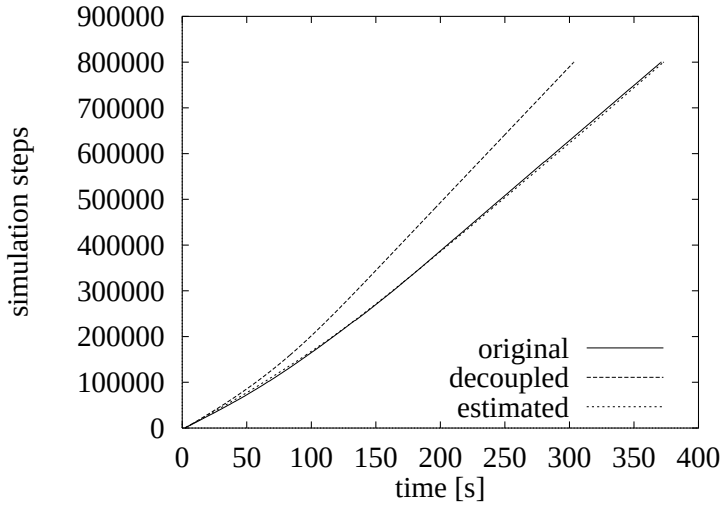

Figure 3: Timing results for a cluster of four workstations. The plots show the number of simulation steps over time.

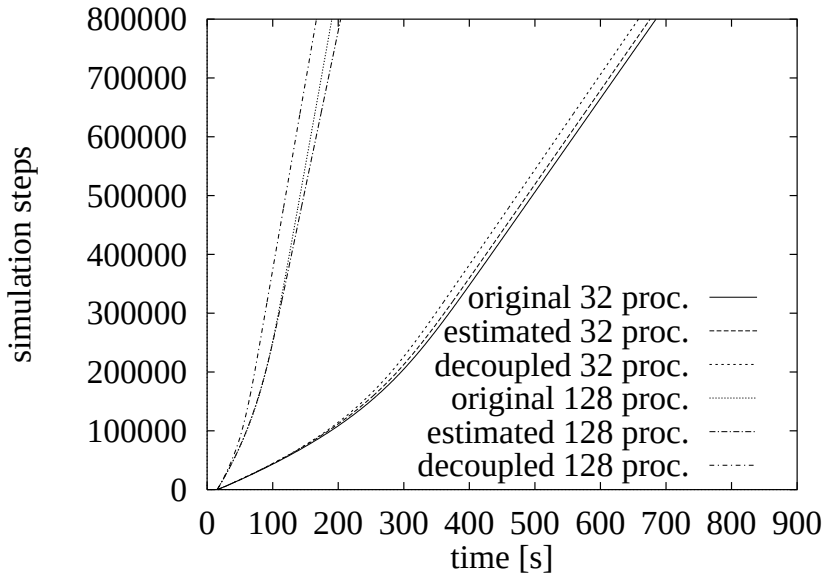

Figure 4: Timing results for a run on the Cray T3D. The plots show the number of simulation steps over time.

hardware dedicated to the fast communication between the processors. Figure 4 shows the timing for the Cray T3D. It can be seen that for a small number of processors (32) the hardware can transfer the message in parallel to the computation, the non-blocking communication gives a (small) performance increase. However, with a large number of processors (128) another effect takes over. The global (blocking) MPI routines in the Cray MPI implementation are highly optimized and use the geometry of the machine's communication topography. The non-blocking communication routines are point-to-point communication routines and are not optimized for the Cray. Here the global comunication routines are faster than the non-blocking routines.

A drawback is that the estimated schedule used in the non-blocking communication slightly reduces the quality of the optimization result.

5 Conclusions

In this paper we discussed the transition of a production code from global blocking communication to non-blocking point-to-point communication. Also we compared the performance of the two algorithms. We found that in a commonly used MPI implementation (MPICH) the communication has a high CPU-usage. Using non-blocking communication does not reduce this CPU-usage but only moves it to the background. This increases the real time measured for the computation, as the CPU is time-shared with the computation. On other architectures this can be different, as has been seen with the Cray T3D. A low number of processors can indeed profit from non-blocking communication, because the hardware can send the message with less CPU usage. However, with a large number of processors the master-worker model is better equipped with the highly optimized global communication routines.

6 Acknowledgements

We acknowledge the support of the german national science foundation (DFG) for the Sonderforschungsbereich 393, "Numerische Simulation auf massiv parallelen Rechnern". Acknowledgements also go to the European Union, for providing access to the Cray T3D at the Edinburgh Parallel Computing Center by the means of the TRACS programme (Training and Research on Advanced Computer Systems). Further we thank the TRACS staff from EPCC, and esp. Stephen S. Booth from the EPCC support for dicussions about random number generators and for optimized random number generation code for the Cray T3D.

References

[1] B. Monien, R. Diekmann, R. Feldmann, R. Klasing, R. Lüling, K. Menzel, T. Römke, and U.-P. Schroeder. *Efficient Use of Parallel & Distributed Systems: From Theory to Practice*. Springer-Verlag, 1995.

[2] Message passing interface (mpi) standard. `http://www.mpi-forum.org`.

[3] Mpich homepage. `http://www.mcs.anl.gov/mpi/mpich/`.

[4] V. Černy. Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization, Theory and Application*, 45:41–51, 1985.

[5] S. Kirkpatrick, C.D. Gelatti, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[6] A. Fachat and K.H. Hoffmann. Implementation of ensemble based simulated annealing with dynamic load balancing under mpi. *Comp. Phys. Comm*, 107:49–53, 1997.

[7] K.H. Hoffmann, P. Sibani, J.M. Pedersen, and P. Salamon. Optimal ensemble size for parallel implementations of simulated annealing. *Appl. Math. Lett.*, 3(3):53–56, 1990.

[8] D.R. Greening. Parallel simulated annealing techniques. *Physica D*, 42:293–306, 1990.

[9] Edinburgh parallel computing center (epcc) homepage. `http://www.epcc.ed.ac.uk`.