CRAY T3E and SGI Origin2000: Merging Architectures from the User's Point of View

Stephan Seidl and Wolfgang E. Nagel

Center for High Performance Computing (ZHR)
Dresden University of Technology
D-01062 Dresden, Germany
E-mail: {seidl | nagel}@zhr.tu-dresden.de

Abstract

While the T3E is very well established as a highly parallel machine in many compute intensive environments, large Origin2000 sites still have to optimize their usage profile to get effective cycles for parallel codes even for moderate numbers of processors. The paper compares T3E and Origin2000 systems, highlighting some details with respect to parallel programming and runtime behavior of appropriate applications. The goal is not to favor one system over the other, but to give recommendations how to design applications which are able to run efficiently on both architectures.

Users are mainly faced with two differences between both systems. First, on a T3E a parallel application is statically parallel from the beginning. In case of the Origin2000, an application gets parallel during execution time when the user has control. Second, once started on a T3E, a parallel application is always running as fast as possible. On an Origin2000, this does only happen under certain circumstances. The paper will give some background about these facts and will demonstrate the strong dependence of the runtime behavior of parallel programs on different runtime situations. Comparative performance illustrations of both machines will color the overall picture of the merging worlds.

1 Introduction

A few months ago, Dresden University of Technology (TU Dresden) has ordered a SGI/CRAY SN-1 machine. The system will be shipped in the second half of 1999. Part of the contract is to deliver a T3E beforehand. Since this will be a 300 MHz model, a lot of measurements have not only been done for the T3E-900. On the other hand, the TU Dresden is the largest German Origin2000 site with, over all, 56 MIPS R10k processors, 18 GBytes memory, about 300 GBytes disks, and, of course, a 21-monthed exciting story of run.

With release 6.5, the IRIX operating system comes now with a version of Miser which can also handle MPI-parallelized jobs. This means that now the task scheduler holds certain information which allow to recognize the tasks of a MPI-parallelized application as a group which is to be served as an entirety. In fact, this is very encouraging because all the prerequisites are finally fulfilled to run MPI-based parallel applications efficiently under normal batch conditions, too. The merged machine of the future will have essential characteristics of the Origin2000 concept. Nevertheless, the IRIX operating system is to be gained upon the level a user sees on a T3E, but, of course, without loosing the flexibility of an Origin2000.

On a T3E, there is no fight for resources while a job is running on the application nodes, except for pushing a packet through the connecting wires. On the Origin2000 and its successors, respectively, this should also be true in case of Miser-controlled jobs. All the other load coming from interactive requests should further fight for processors etc. Clearly, things like Miser seem to be the right way here to dynamically 'partition' the S2MP architecture into a critical batch job part with high priorities under strict resource control, and a remaining part with an UNIX-like resource management.

What remains is a question which should be answered by the vendor. What does Miser do in case of an application which performs ordinary *fork()* system calls, or, what is done in case of PVM? Early investigations have shown that so far only the parent process gets the 'batch critical' attribute.

2 Architecture Overview

A T3E can have up to 2048 processors over a 3D torus interconnect. The 3D torus links have a raw bandwidth of 600 MBytes/s in each direction. One node of the system consists of an Alpha microprocessor, a system control chip, local memory, and a network router. The system logic runs at 75 MHz, and the processor runs with a multiple of this, i.e. at 300 MHz for the T3E which is called T3E-600 in this paper, at 450 MHz for the T3E-900, or at 600 MHz for the T3E-1200 (delivered at a few sites in Germany over the last few weeks). The latter has not been taken into account here. The Alpha processor is capable to do one floating-point add and one floating-point multiply at the same time. Each processor contains an 8 KBytes direct-mapped primary cache, an 8 KBytes instruction cache, and a 96 KBytes unified three-way associative second-level cache. Instead of a large, board-level cache, there is a small set of stream buffers to improve the access to stride-1 or small-stride vectors by prefetching. The remote communication and synchronization is done between a large set of so-called E-registers and the memory.

An Origin2000 can have up to 128 processors where up to 4 eight-vertex hypercubes are connected with each other. One node board of the system consists of two R10000 microprocessors with 4 MBytes external second-level cache each, one HUB ASIC, and local memory. Two processors share the same memory portion through 780 MBytes/s peak. Two node boards are connected by a six-port router unit. The processors run at 195 MHz. They are able to execute two floating-point operations per cycle. Each processor contains a 32 KBytes two-way set associative primary cache, and a 32 KBytes two-way set associative instruction cache. The main memory is located in a single shared address space, hence the Origin2000 is capable to run large multi-threaded applications too. For cache coherency a directory-based protocol is applied, using extra memory hardware which is not accessible by the user.

Picking up real-time information completely differs on the T3E and the Origin2000. On T3E, there is a recommended intrinsic function *_rtc()* which returns a 64 bit integer. To get seconds, this integer is to be divided by the return value of *sysconf(_SC_CLK_TCK)*. On the Origin2000 the recommended code is described in *syssgi(2)*. Its basic idea is to map a 64 bit counter into the user's address space via *mmap()*. A step unit of 800 ns is occasionally not sufficient for performance analyzing. One gets the following results.

Table 1: Realtime timers

	Stepunit	Stepunit^{-1}	Measured overhead to access
T3E	$13.\overline{3}$ ns	75 MHz	$<$ 230ns
Origin2000	800 ns	1.25 MHz	$<$ 320 ns

3 Performance Comparisons

3.1 PE Performance

To draw an exact picture of the per-PE performance of at least one well known program kernel on each of the three machines (T3E-600, T3E-900, and Origin2000), 24 variants performing matrix multiplication have been studied. These variants come from two languages, C, and Fortran, from either operating over the output matrix itself or over its transposed one, and from six possible permutations to order the loops.

Here, the results from C are of interest since they reflect some hardware properties. Typically, C compilers just generate straight code, even with '`-O3`'. On the other hand, current Fortran compilers try to recognize patterns to replace them by highly-optimized code sequences to get, at least, one floating-point instruction per cycle[1]. The MFLOP rates are depicted here are based on $2n^3$ operations, where n denotes the dimension of the $n \times n-$matrices. What we should see is that the MIPS R10k processor gets high profit from its 4 MBytes second-level cache. The curves for the transposed case look similarly, except that prefetching does not longer help in some cases.

The Fortran90 examples give an impression of the excellent T3E compilers, even so the observed compile-times are often many times longer than typical others. Bad loop orders are clearly recognized and replaced by the optimum one. One more detail is of interest here. Once started on the T3E application nodes, repeated executions exactly show the same behavior with respect to their time consumption. This is different from the situation on an Origin2000 where the reproducibility is on a lower level.

Summarizing here, the per-PE performance of a real application is not predictable. It strongly depends on the problem and on the code design. On the Origin2000 one has to specify, at least, '`-O3`' to get results which are comparable with T3E performance values. In future, program developers still should think about using the Fortran dialect Fortran90 as the implementation language to get the best performance on such HPC systems.

[1]The results represented by figures 1 ... 12 are based on UNICOS/mk 2.0.3.23, with C 6.0.2.1, and F90 3.0.2.3, and IRIX64 6.5, with C 7.2.1, and F90 7.2.1. All the compilers have been invoked with '`-O3`' only, whereat the SGI compilers were under control of '`abi=n32:isa=mips4:proc=r10k`'. IEEE-754 Double Precision has been used as the floating-point number format.

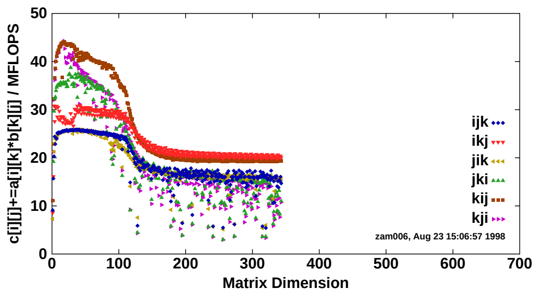

Figure 1: C-coded matrix multiplication on T3E-600 with streams off

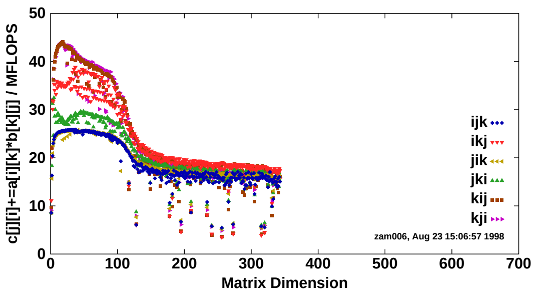

Figure 2: C-coded matrix multiplication on T3E-600 with streams off, transposed case

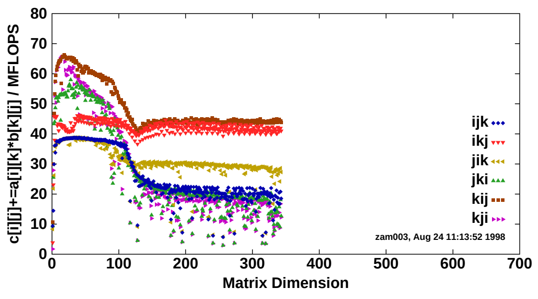

Figure 3: C-coded matrix multiplication on T3E-900 with streams on

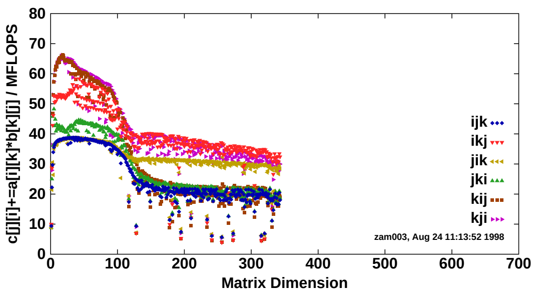

Figure 4: C-coded matrix multiplication on T3E-900 with streams on, transposed case

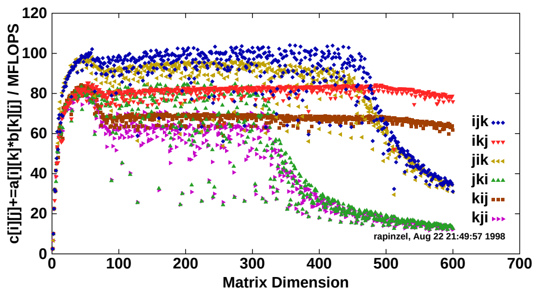

Figure 5: C-coded matrix multiplication on Origin2000

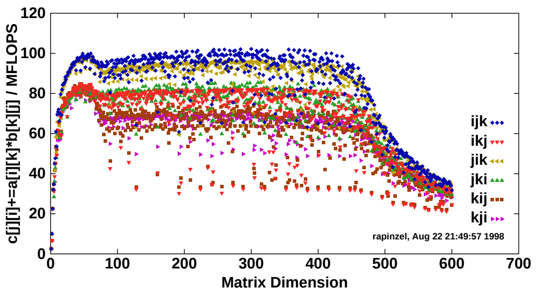

Figure 6: C-coded matrix multiplication on Origin2000, transposed case

3.2 MPI Performance

The measurements here are based on a kernel application which interchanges messages between three processes, using the *MPI_Send()-MPI_Recv()* pair, and the *MPI_Ssend()-MPI_Recv()* pair, respectively. 'Normalized' denotes that the measured times have been divided by three.

The protocol for sending/receiving the messages is switched, depending on the message length. In some cases, this is pretty much optimized, sometimes not. Hence, there is still enough room for some improvement. In case of the T3Es, the important differences between the results based on *MPI_Send()* and *MPI_Ssend()*, respectively, cannot be explained in detail here. Evidently, users should use the synchronous calls wherever possible. A little bit surprising are the results for short messages, see figures 19 through 24. While the Origin2000 yields

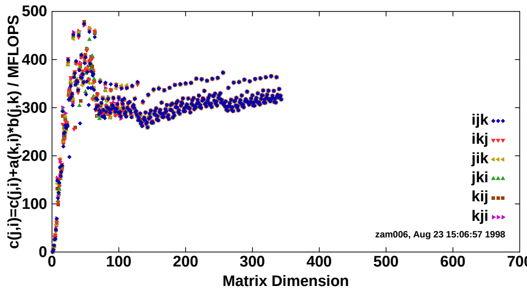

Figure 7: Fortran-coded matrix multiplication on T3E-600 with streams off

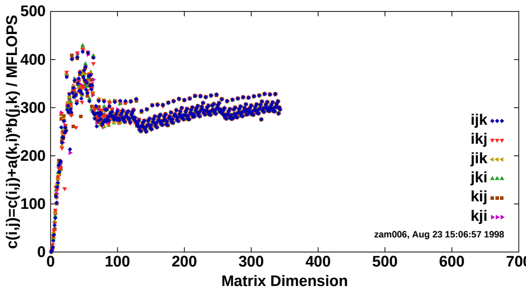

Figure 8: Fortran-coded matrix multiplication on T3E-600 with streams off, transposed case

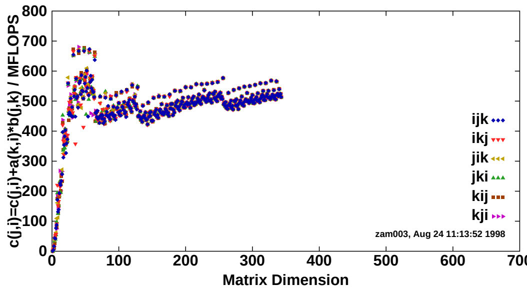

Figure 9: Fortran-coded matrix multiplication on T3E-900 with streams on

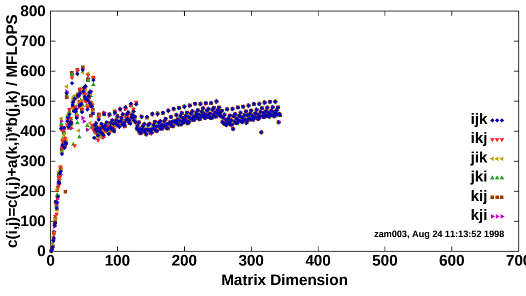

Figure 10: Fortran-coded matrix multiplication on T3E-900 with streams on, transposed case

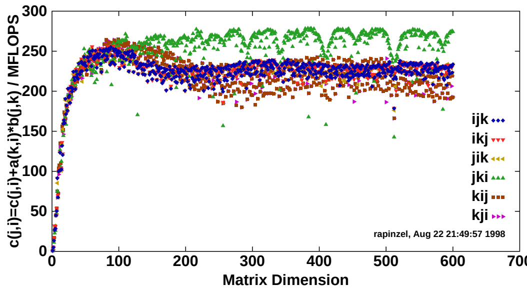

Figure 11: Fortran-coded matrix multiplication on Origin2000

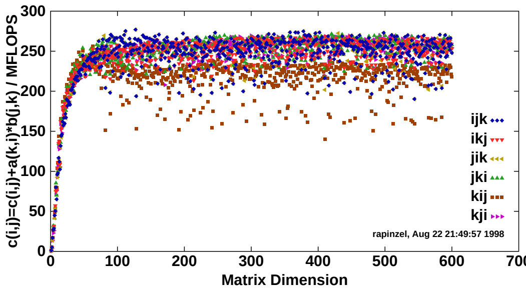

Figure 12: Fortran-coded matrix multiplication on Origin2000, transposed case

reproducible values with small standard deviations, the round-trip times strongly varies on the T3Es. Nevertheless, the slope of the average curves of the T3Es is significantly smaller than that of the Origin2000. Using the *MPI_Send()*-*MPI_Recv()* pair, the MPI point-to-point performance of the Origin2000 is of the same order as the one of the T3E-600, except for very large message lengths. It is less than the performance of the T3E-900 at all. With respect to this pair, the values agree with [4].

Summarizing the measurement values, the great question is why the overall performance of the Origin2000 is still pretty low even if a synchronous *MPI_Ssend()* is used while the latter boosts the performance of a T3E by a factor of 2. On the other hand, on T3Es, the MPI implementation should silently change to synchronous mode in case of messages which are large enough, even if the user has invoked *MPI_Send()*.

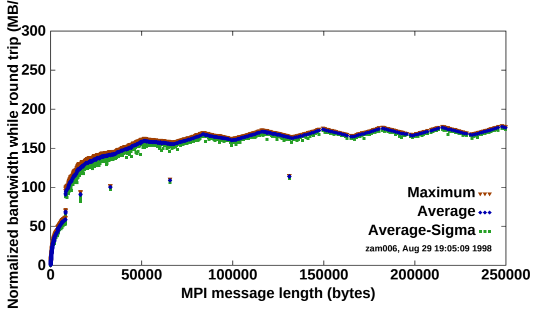

Figure 13: MPI point-to-point communication on T3E-600 with streams off, long messages with *MPI_Send()*

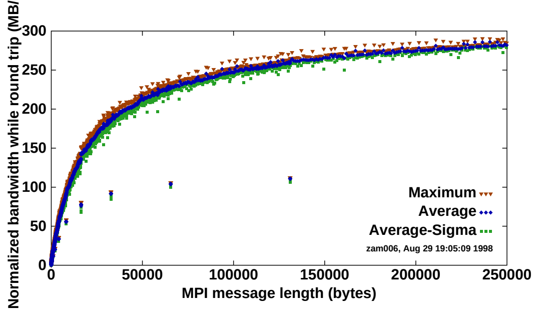

Figure 14: MPI point-to-point communication on T3E-600 with streams off, long messages with *MPI_Ssend()*

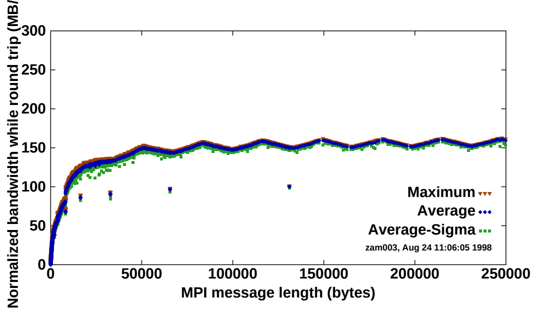

Figure 15: MPI point-to-point communication on T3E-900 with streams on, long messages with *MPI_Send()*

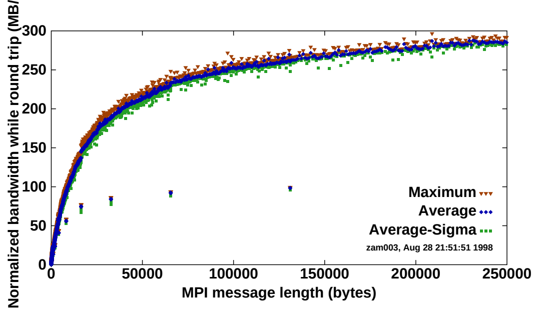

Figure 16: MPI point-to-point communication on T3E-900 with streams on, long messages with *MPI_Ssend()*

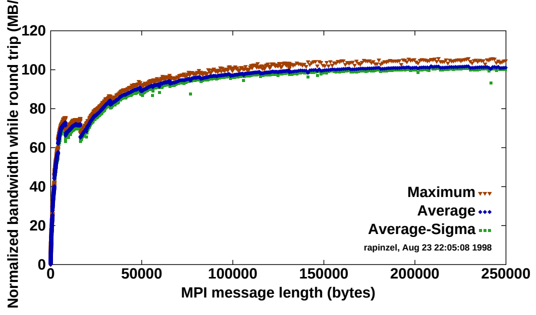

Figure 17: MPI point-to-point communication on Origin2000, long messages with *MPI_Send()*

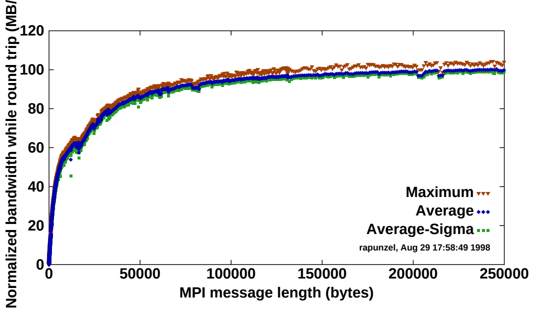

Figure 18: MPI point-to-point communication on Origin2000, long messages with *MPI_Ssend()*

3.3 PVM Performance

The measurements with respect to PVM are based on a kernel application which is similar to that for MPI. It uses the *pvm_psend()*-*pvm_precv()* pair.

SGI has replaced the whole PVM in July 1998 with the result that the communication rates of the current release 3.1.1.0 with MPT 1.2.1.0 (3.3.10) are the same now as these of the open release 3.4beta6. PVM seems to be still not working together with Miser.

As shown in figure 27, one has very short startup times on the T3E-900. This is surprising because they are shorter than the MPI startup times.

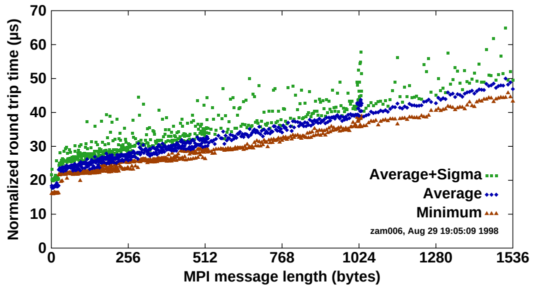

Figure 19: MPI point-to-point communication on T3E-600 with streams off, short messages with *MPI_Send()*

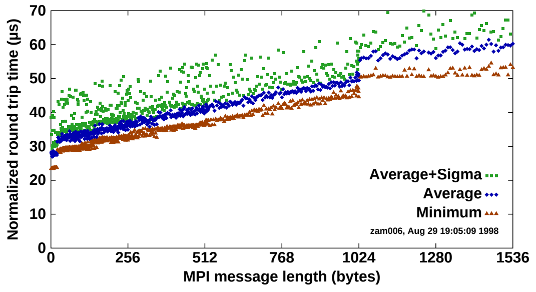

Figure 20: MPI point-to-point communication on T3E-600 with streams off, short messages with *MPI_Ssend()*

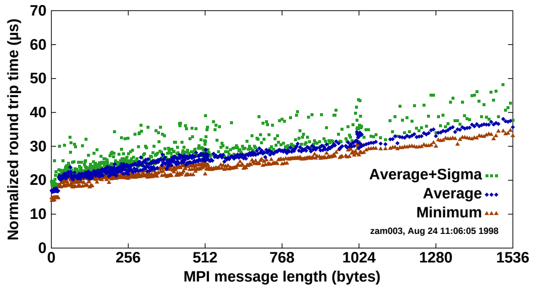

Figure 21: MPI point-to-point communication on T3E-900 with streams on, short messages with *MPI_Send()*

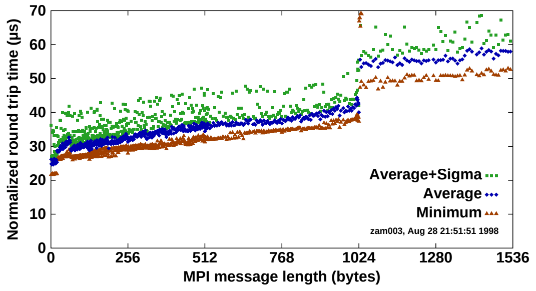

Figure 22: MPI point-to-point communication on T3E-900 with streams on, short messages with *MPI_Ssend()*

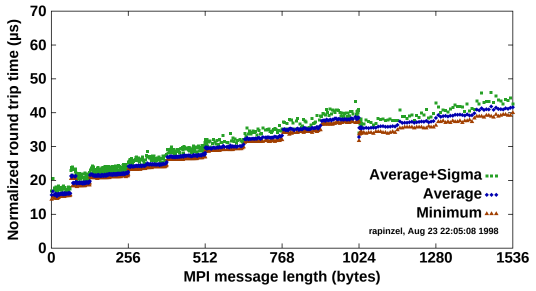

Figure 23: MPI point-to-point communication on Origin2000, short messages with *MPI_Send()*

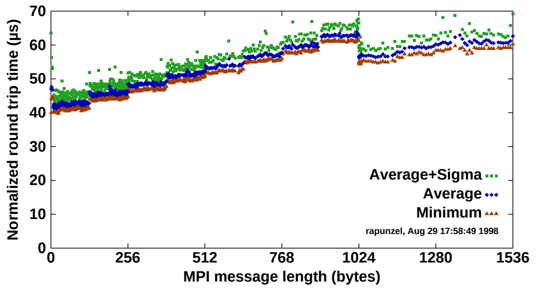

Figure 24: MPI point-to-point communication on Origin2000, short messages with *MPI_Ssend()*

Remains to give some coding hints. Interchangeable PVM codes should not contain *pvm_halt()* and *pvm_start_pvmd()* calls. On T3Es, these routines make no sense, except the application should run on the command nodes which is a bad intention. On the Origin2000, the *pvm_halt()* call does never return currently, and the group server still core-dumps, when remotely accessed, fortunately, after the application has finished. On a T3E one should branch around *pvm_spawn()* which has no meaning there. There is an appropriate stub in the library. At least on SGI hosts, MPI - as the emerging standard for message-passing applications - should be used instead of PVM wherever possible.

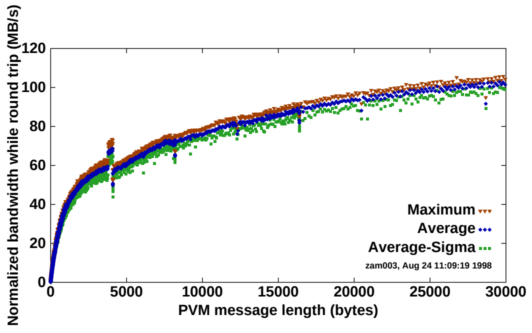

Figure 25: PVM point-to-point communication on T3E-900 with streams on, medium size messages

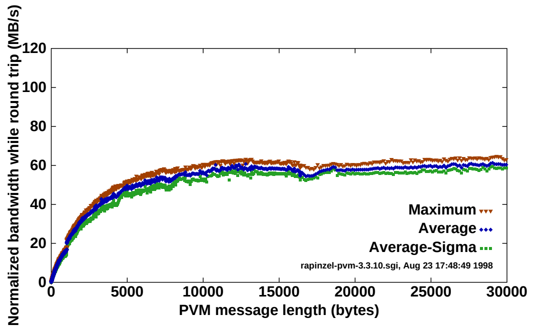

Figure 26: PVM point-to-point communication on Origin2000, medium size messages

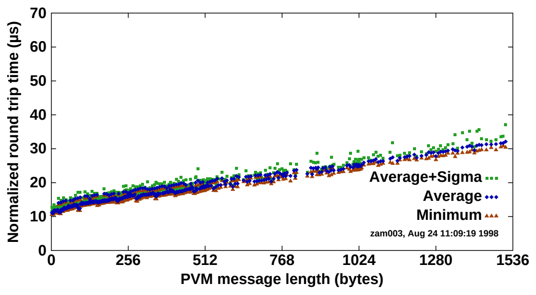

Figure 27: PVM point-to-point communication on T3E-900 with streams on, short messages

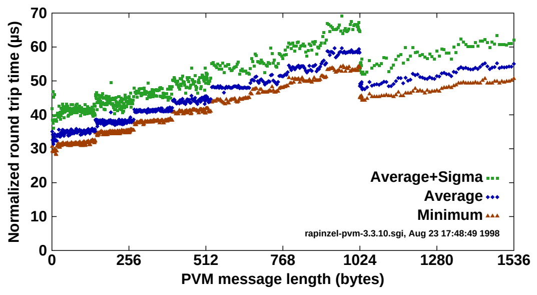

Figure 28: PVM point-to-point communication on Origin2000, short messages

3.4 Communication-to-computation performance ratio

To guess the communication-to-computation performance ratio of the T3E-900 and the Origin2000, the runtime behavior of an adapted version of the well known 2D-decomposed Jacobi iteration MPI example code [3] has been investigated which comes with the VAMPIR performance analysis tool [5].

Figure 29 shows the excellent communication patterns produced on a T3E-900, running Jacobi iterations. The time line goes from left to right. Time sections where one of the processors stays in the application code are colored with blue here, yellow is for different *MPI-Sendrecv()* calls, and red for *MPI-Allreduce()*. *MPI-Allreduce()* sends and receives 8 bytes, i.e., very short messages, and, only *MPI-Allreduce()* is discussed from now. Since we need average values below a mean per-processor communication rate, B_1 is introduced is determined by

$$B_1 \quad = \quad \frac{2l(p-1)}{\overline{t_1}}$$

l is the message length in bytes, with $l = 8\,$Bytes here, p is the number of processors, with $p = 50$, and $\overline{t_1}$ denotes the average time one of the processors spends in the *MPI-Allreduce()* routine. With $\overline{t_1} = 686\,\mu$s from VAMPIR one gets $B_1 = 1100\,$KBytes/s.

In case of a dedicated Origin2000, the application code runs about two times faster, and *MPI-Allreduce()* this needs about twice the time of a T3E-900. Although the program runs

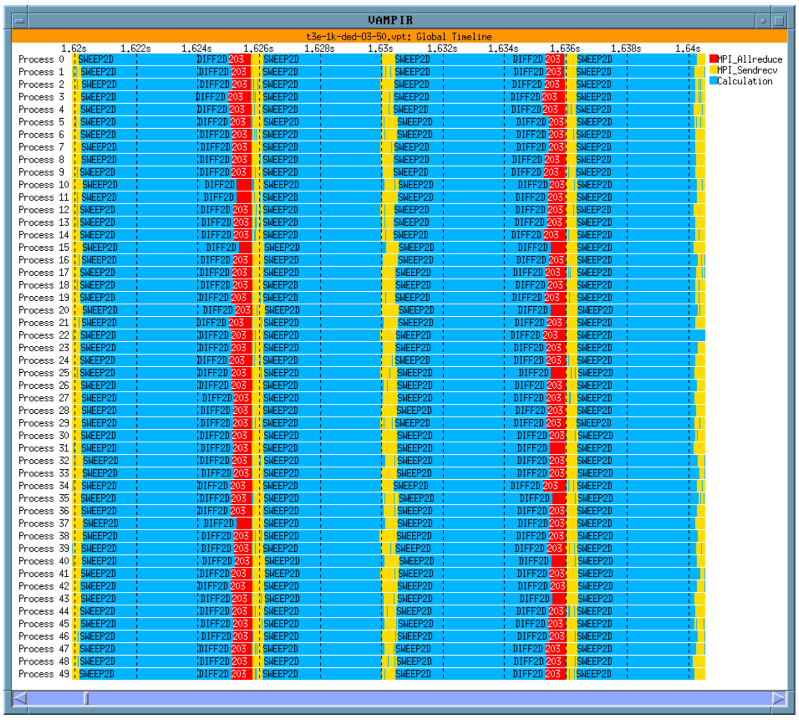

Figure 29: One of the Jacobi iterations (NX=1000), running on 50 processors of a T3E-900

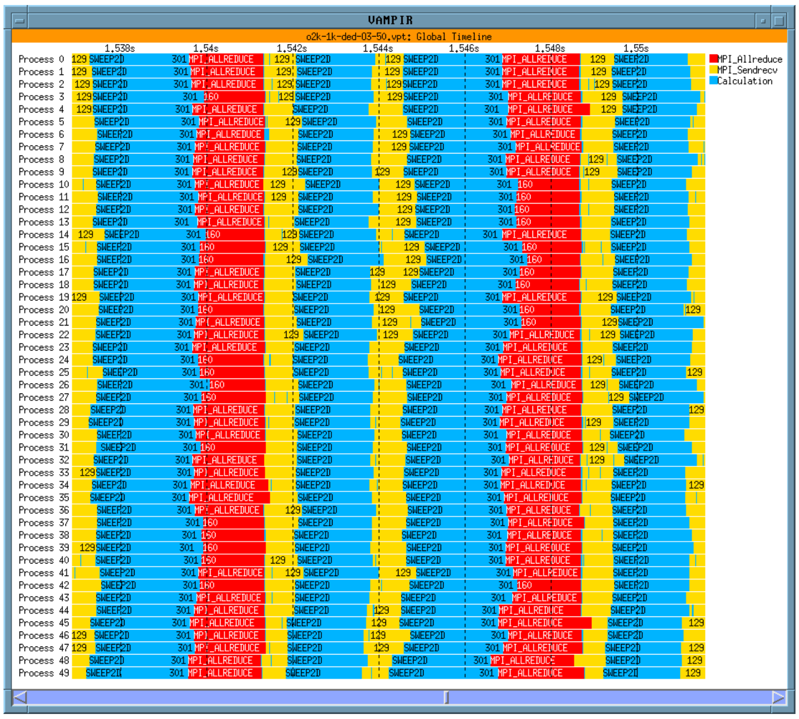

Figure 30: One of the Jacobi iterations (NX=1000), running on 50 processors of a dedicated Origin2000

faster at all, roughly half of the whole time is spent in the communication routines. With $\overline{t_1} = 1590\,\mu$s from VAMPIR, B_1 equals 480 KBytes/s.

Considering only *MPI-Allreduce()* in case of the Jacobi iteration MPI example code, the communication-to-computation performance ratio of the T3E-900 is about four times the appropriate value of a dedicated Origin2000. This performance ratio is suitable to guess the parallel efficiency of a given application, using a certain number of processors.

4 Scheduling Aspects

With respect to scheduling, we should distinguish between job scheduling and task scheduling. In principle, both of them have to solve discrete optimization problems. The job scheduler has to reorder jobs with well defined resource requirements to start them preventing any resource conflict.

On T3Es, the optimization problem of the job scheduler has two variables, the number of PEs and the time. On shared memory machines, there is one more variable, which is the memory needed to execute the job. Assuming that the job scheduler has done its work, i.e. there is a plan of action, the question is how to enforce that this plan is executed in time. A common way is to remove all the interactive load from the machine. Interactive load cannot be taken into account since its resource requirements are unknown and strongly varying. Another way is to qualify the task scheduler to distinguish between task objects which are part of the plan of action or not. In figure 31, mission critical jobs are blue, green, and yellow, the red part denotes interactive load which is running under time-sharing conditions, using only resources which are not needed by the others. One should ask the question here whether it is possible to make all the pages sticky which are associated with the batch jobs.

Clearly, writing a task scheduler which is able to distribute thousands of task objects over hundreds of processors, distinguishing between mission critical tasks and uncritical ones, is one of the pretty big problems of our days. Being successful here, large application servers will win over clusters of SMP nodes, and, scientific users will win too, because they can have lots of PEs and large memory portions at the same time. Miser is a first step into this direction; nevertheless, Miser still has too many restrictions today. It will be the task of the next months for all application groups world-wide to give input about important requirements for effective scheduling issues back to the vendor to get the next generation of system software on SGI hardware 'just right'.

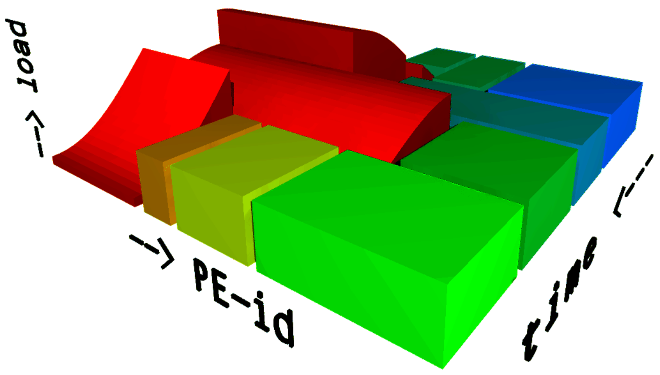

Figure 31: One possible scheduling concept for large shared memory application servers

5 Sharing Cache-Lines

For the memory access conditions which a single Origin2000 processor does have far from saturation effects, see the discussion in [6]. Like the Origin, the SGI/CRAY SN-1 machine is a shared memory system, which will allow to run large multi-threaded applications too. Explicit multi-threading is not very common in the field of scientific applications. Moreover, multi-threaded applications matter little to T3Es. Even so, a kernel has been written to study performance degradations caused by concurrent write access to data which are located in the same cache line. Figure 32 shows the result of the 40 processor multi-threaded kernel in case of non-shared cache lines. It has run under Miser to enforce its execution on top of more than 120 per cent user load base.

The time runs horizontally for each thread from left to right. One color change from blue to yellow and back denotes the same portion of work. The work itself is to increment an unsigned short integer ignoring wraparounds. Unsigned short integers have been taken to ensure that one cache line is capable to contain all of them. After 25 portions of work, there is a barrier which synchronizes all the threads. Despite of some disturbances, the execution pattern is very regular. Figure 33 shows the result using the same kernel, expect that all the 40 unsigned short integers are localized in the same cache line. It seems that one can see the hardware of the Origin2000, two threads are much more faster than the others, perhaps, they are running on the processors of that node board which contains the memory where the origin of the cache line is localized. After these two processors have finished their part, all the others get able to accelerate their work. The more threads finish their work, the more the remaining threads increase in speed.

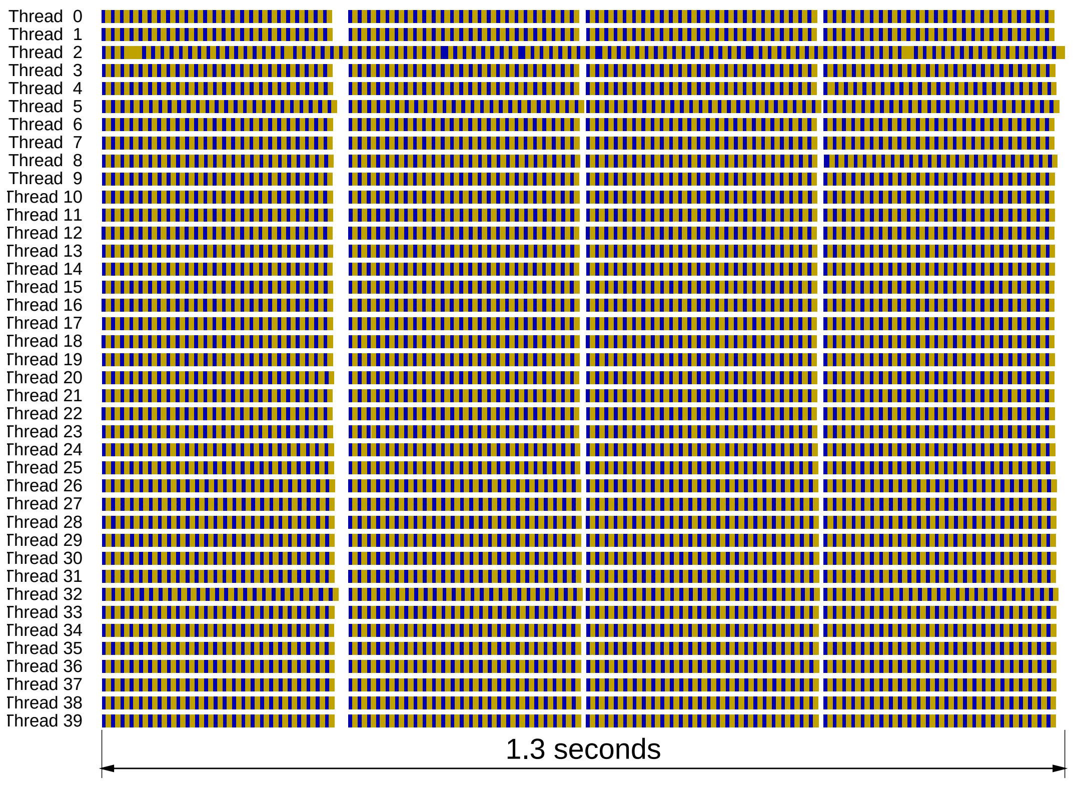

Figure 32: Runtime behavior of a multi-threaded kernel without data access collision

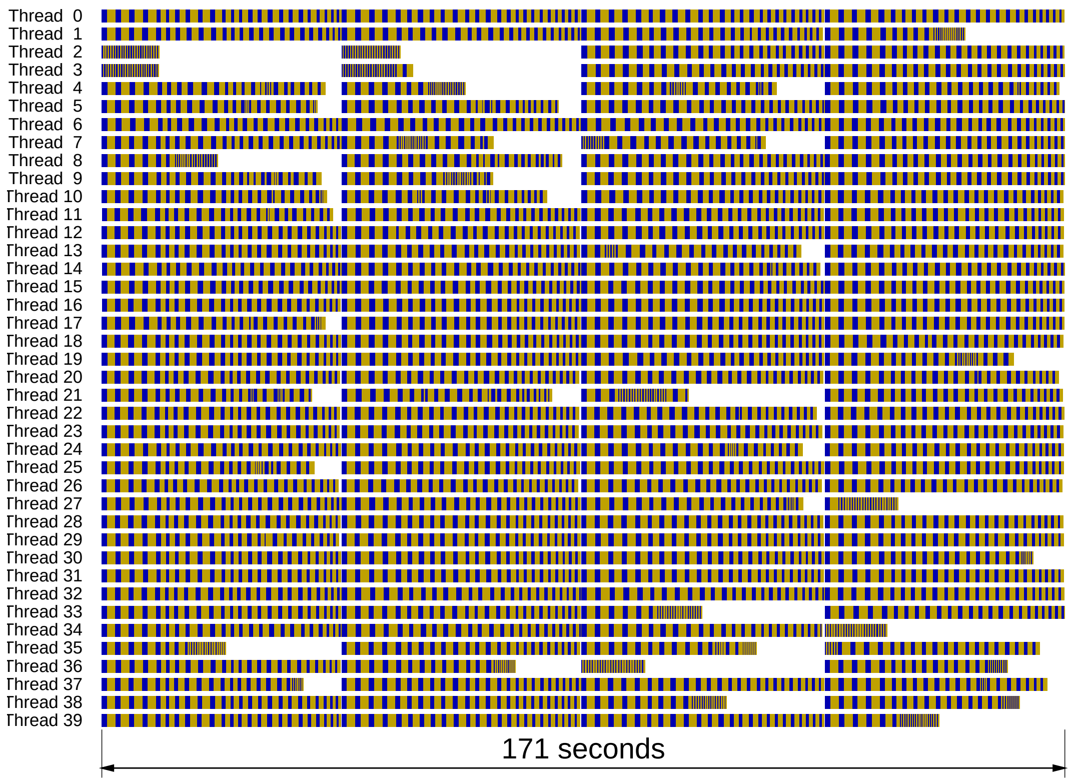

Figure 33: Runtime behavior of a multi-threaded kernel with maximum data access collision

Summarizing here, shared memory machines do not only allow to write slow code which is based on the message-passing parallelism, one has also the freedom to write slowly running multi-threaded applications. On the other hand, in normal situations the overhead introduced by this situation of 'wrong-sharing' data is - because of the still working principle of 'locality' in most cases - not a major problem. Nevertheless, if applications are developed, e.g. based on OpenMP, the user should carefully look to the time-consuming program parts.

6 Conclusion

We have compared the CRAY T3E, i. e. *the* massive parallel machine of these days, and the SGI Origin2000, a new-style shared memory system. Both systems are based on excellent hardware concepts. The merged SGI/CRAY machine of the future will be closer to an Origin2000 than to a T3E. Nevertheless, all excellent parameter values of the T3E, the transfer bandwidth, for example, should be carried on - and somewhat improved if possible - in the new architecture. IRIX, as the operating system, has done important steps to control the complicated situation of large shared memory application servers. Nevertheless, it still has to be more and more improved to ensure that, once, the job scheduling system will have the power to enforce its plan of action via task scheduling mechanisms in the presence of any interactive load.

References

[1] Anderson E., Brooks J., Grassl C., and Scott S.: *Performance of the CRAY T3E Multiprocessor*. Proc. Supercomputing'97, San Jose, USA

[2] Dowd K.: *High Performance Computing*. O'Reilly & Associates, Inc., Sebastopol, 1993.

[3] Gropp W., Lusk E., and Skjellum A.: *Using MPI*. MIT Press, ISBN 0-262-57104-8, 1996.

[4] Luecke R.G. and Coyle J.J.: *Comparing the Performance of MPI on the CRAY T3E-900, the CRAY Origin 2000 and the IBM P2SC*. Iowa State University, April 1998.

[5] Nagel W.E., Arnold A., Weber M., Hoppe H-C., and Solchenbach, K.: *VAMPIR: Visualization and Analysis of MPI Resources.* Supercomputer 63, Vol. 12, No. 1, pp. 69-80, 1996.

[6] Seidl S.: *Access Delays Related to the Main Memory Hierarchy on SGI Origin-2000.* http://armoise.saclay.cea.fr/~workshop/Documents/Final_Papers/Stephan_Seidl_-11_Perf_opt_1.ps.

[7] Silicon Graphics Inc.: *Origin Servers Technical Report.* April 1997.

Originally published in Proceedings of the Fourth European SGI/Cray MPP Workshop (Sep 10-11, 1998, IPP, Garching, Germany), H. Lederer and F. Hertweck, eds., IPP R/46, Oct 1998, pp. 6-19.
Also available as internal report ZHR-IR-9803.