Published in the proceedings of the Fifth European Workshop on OpenMP, EWOMP '03, Aachen, Germany, Sept. 22-26, 2003, www.compunity.org

Hybrid Parallel Programming on HPC Platforms

Rolf Rabenseifner*

Summary

Most HPC systems are clusters of shared memory nodes. Parallel programming must combine the distributed memory parallelization on the node inter-connect with the shared memory parallelization inside of each node. Various hybrid MPI+OpenMP programming models are compared with pure MPI. Benchmark results of several platforms are presented. This paper analyzes the strength and weakness of several parallel programming models on clusters of SMP nodes. Benchmark results on a Myrinet cluster and on recent Cray, NEC, IBM, Hitachi, SUN and SGI platforms show, that the hybrid-masteronly programming model can be used more efficiently on some vector-type systems, but also on clusters of dual-CPUs. On other systems, one CPU is not able to saturate the inter-node network and the commonly used masteronly programming model suffers from insufficient inter-node bandwidth. This paper analyses strategies to overcome typical drawbacks of this easily usable programming scheme on systems with weaker interconnects. Best performance can be achieved with overlapping communication and computation, but this scheme is lacking in ease of use.

Keywords. OpenMP, MPI, Hybrid Parallel Programming, Threads and MPI, HPC, Performance.

1 Introduction

Most systems in High Performance Computing are clusters of shared memory nodes. Such hybrid systems range from small clusters of dual-CPU PCs up to largest systems like the Earth Simulator consisting of 640 SMP nodes connected by a single-stage cross-bar and with SMP nodes combining 8 vector CPUs on a shared memory [3, 5]. Optimal parallel programming schemes enable the application programmer to use the hybrid hardware in a most efficient way, i.e., without any overhead induced by the programming scheme.

On distributed memory systems, message passing, especially with MPI [4, 12, 13], has shown to be the mainly used programming paradigm. One reason of the success of MPI was the clear separation of the optimization: communication could be improved by the MPI library, while the numerics had to be optimized by the compiler. On shared memory systems, directive-based parallelization was standardized with OpenMP [15], but there is also a long history of proprietary compiler-directives for parallelization. The directives handle mainly the work sharing; there is no data distribution.

On hybrid systems, i.e., on clusters of SMP nodes, parallel programming can be done in several ways: one can use pure MPI, or some schemes combining MPI and OpenMP, e.g., calling MPI routines only outside of parallel regions (which is herein named the *masteronly* style), or using OpenMP on top of a (virtual) distributed shared memory (DSM) system. A classification on MPI and OpenMP based parallel programming schemes on hybrid architectures is given in Sect. 2. Unfortunately, there are several mismatch problems between the (hybrid) programming schemes and the hybrid hardware architectures. Often, one can see in publications, that applications may or may not benefit from hybrid programming depending on some application parameters, e.g., in [7, 10, 22].

Sect. 3 gives a list of major problems often causing a degradation of the speed-up, i.e., causing that the parallel hardware is utilized only partially. Sect. 4 shows, that there isn't a silver bullet to achieve an optimal speed-up. Benchmark results show that different hardware platforms are more or less prepared for the hybrid programming models. Sect. 5 discusses optimization strategies to overcome typical drawbacks of the hybrid masteronly style. With these optimizations, efficiency can be achieved together with the ease of parallel programming on clusters of SMPs. The conclusions are provided in Sect. 6.

*High-Performance Computing-Center (HLRS), University of Stuttgart, Allmandring 30, D-70550 Stuttgart, Germany `rabenseifner@hlrs.de`, `www.hlrs.de/people/rabenseifner/`

2 Parallel programming on hybrid systems, a classification

Often, *hybrid MPI+OpenMP programming* denotes a programming style with OpenMP shared memory parallelization inside the MPI processes (i.e., each MPI process itself has several OpenMP threads) and communicating with MPI between the MPI processes, but *only outside of parallel regions*. For example, the MPI parallelization is based on a domain decomposition, the MPI communication mainly exchanges the halo information after each iteration of the outer numerical loop, and these numerical iterations itself are parallelized with OpenMP, i.e., (inner) loops inside of the MPI processes are parallelized with OpenMP work-sharing directives. However, this scheme is only one style in a set of different hybrid programming methods. This hybrid programming scheme will be named *masteronly* in the following classification, which is based on the question, when and by which thread(s) the messages are sent between the MPI processes:

1. **Pure MPI**: each CPU of the cluster of SMP nodes is used for one MPI process. The hybrid system is treated as a flat massively parallel processing (MPP) system. The MPI library has to optimize the communication by using shared memory based methods between MPI processes on the same SMP node, and the cluster interconnect for MPI processes on different nodes.

2. Hybrid MPI+OpenMP without overlapping calls to MPI routines with other numerical application code in other threads:

2a. **Hybrid masteronly**: MPI is called only outside parallel regions, i.e., by the master thread.

2b. **Hybrid multiple/masteronly**: MPI is called outside the parallel regions of the application code, but the MPI communication is done itself by several CPUs: The thread parallelization of the MPI communication can be done
- automatically by the MPI library routines, or
- explicitly by the application, using a full thread-safe MPI library.

In this category, the non-communicating threads are sleeping (or executing some other applications, if non-dedicated nodes are used). This problem of idling CPUs is solved in the next category:

3. Overlapping communication and computation: While the communication is done by the master thread (or a few threads), all other non-communicating threads are executing application code. This category requires, that the application code is separated into two parts: the code that can be overlapped with the communication of the halo data, and that code that must be deferred until the halo data is received. Inside of this category, we can distinguish two types of sub-categories:

• How many threads communicate:

(A) Hybrid funneled: Only the master thread calls MPI routines, i.e., all communication is funneled to the master thread.

(B) Hybrid multiple: Each thread handles its own communication needs (B1), or the communication is funneled to more than one thread (B2).

• Except in case B1, the communication load of the threads is inherently unbalanced. To balance the load between threads that communicate and threads that do not communicate, the following load balancing strategies can be used:

(I) Fixed reservation: reserving a fixed amount of threads for communication and using a fixed load balance for the application between the communicating and non-communicating threads.

(II) Adaptive.

4. **Pure OpenMP**: based on virtual distributed shared memory systems (DSM), the total application is parallelized only with shared memory directives.

Each of these categories of hybrid programming has different reasons, why it is not appropriate for some classes of applications or classes of hybrid hardware architectures. The paper focuses on the first two methods. Overlapping communication and computation is studied in more detail in [17, 16]. Regarding pure OpenMP approaches, the reader is referred to [1, 6, 8, 11, 18, 19, 20]. Different SMP parallelization strategies in the hybrid model are studied in [21] and in [2] for the NAS parallel benchmarks. The following section shows major problems of mismatches between programming and hardware architecture.

3 Mismatch problems

All these programming styles on clusters of SMP nodes have advantages, but also serious disadvantages based on mismatch problems between the (hybrid) programming scheme and the hybrid architecture:

- With pure MPI, minimizing of the inter-node communication requires that the application-domain's neighborhood-topology matches with the hardware topology.
- Pure MPI also introduces intra-node communication on the SMP nodes that can be omitted with hybrid programming.
- On the other hand, such MPI+OpenMP programming is not able to achieve full inter-node bandwidth on all platforms for any subset of inter-communicating threads.
- With masteronly style, all non-communicating threads are idling.
- CPU time is also wasted, if all CPUs of an SMP node communicate, although a few CPUs are already able to saturate the inter-node bandwidth.
- With hybrid masteronly programming, additional overhead is induced by all OpenMP synchronization, but also by additional cache flushing between the generation of data in parallel regions and the consumption in subsequent message passing routines and calculations in subsequent parallel sections.

Overlapping of communication and computation is a chance for an optimal usage of the hardware, but

- causes serious programming effort in the application itself to separate numerical code that neds halo data and that cannot be overlapped with the communication therefore,
- causes overhead due to the additional parallelization level (OpenMP), and
- communicating and non-communicating threads must be load balanced.

A few of these problems will be discussed im more detail and based on benchmark results in the following sections.

3.1 The inter-node bandwidth problem

With hybrid masteronly or funneled style, all communication must be done by the master thread. The benchmark measurements in Fig. 3 and the inter-node results in Tab. 1 show, that on several platforms, the available aggregated inter-node bandwidth can be achieved only,

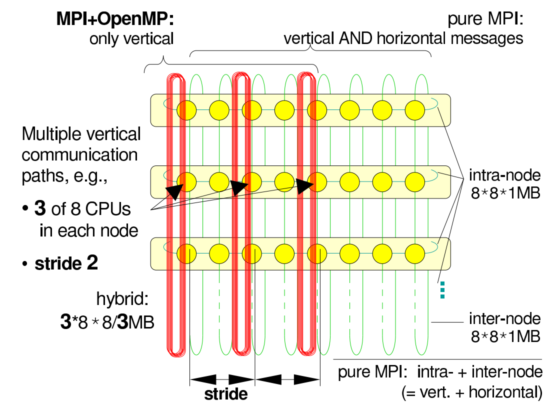

Figure 1: Communication pattern with *hybrid MPI+OpenMP* style and with *pure MPI* style.

if more than one thread is used for the communication with other nodes.

In this benchmark, all SMP nodes are located in a logical ring. Each CPU sends messages to the corresponding CPU in the next node and receives from the previous node in the ring. The benchmark is done with pure MPI, i.e., one MPI process on each CPU, except for Cray X1, where we used as smallest entity an MSP (which itself has 4 SSPs [=CPUs]). Fig. 1 shows the communication patterns. The aggregated bandwidth per node is defined as the number of all bytes of all messages on the inter-node network divided by the time needed for the communication and divided by the number of nodes. Note that in this definition, each message is counted only once, and not twice.[1] Fig. 2 shows the absolute bandwidth over the number of CPUs (or MSPs at Cray X1), Fig. 3 shows relative values, i.e., the percentage of the achieved peak bandwidth in each system over the percentage of CPUs of a node.

One can see, that only on the NEC SX-6, Cray X1 systems, and on the Myrinet based cluster of dual-CPU PCs, one can achieve more than 75 % of the peak bandwidth already with **one** CPU (or MSP on Cray X1) per node (see highlighted values in Tab. 1, Col. 3).

On the other systems, the hybrid masteronly or funneled programming scheme can achieve only a small percentage of the peak inter-node bandwidth. [17] has

[1]The hardware specification typically presents the duplex inter-node bandwidth by counting each message twice, i.e. as incoming and outgoing message at a node, e.g., on a Cray X1, 25.6 GB/s = 2*12.8 GB/s; the measured 12 GB/s (shmen_put) must be compared with the 12.8 GB/s of the hardware specification.

	Master-only, inter-node bandw. [GB/s]	pure MPI, inter-node bandw. [GB/s]	Master-only bw / max. inter-node bw [%]	pure MPI, intra-node bandw. [GB/s]	memory band-width bandw. [GB/s]	Peak and *Linpack* perfor-mance [GFLOP/s]	max. inter-node bw / peak *or* *Linpack* perf. [B/FLOP]	#nodes ∗ #CPUs per SMP node
Cray X1, shmem_put preliminary results	9.27	**12.34**	**75 %**	33.0	136	**51.20** *45.03*	**0.241** *0.274*	8 * 4 MSPs
Cray X1, MPI preliminary results	4.52	**5.52**	**82 %**	19.5	136	**51.20** *45.03*	**0.108** *0.123*	8 * 4 MSPs
NEC SX-6, MPI with global memory	**7.56**	4.98	**100 %**	78.7 93.7+)	256	**64** *61.83*	**0.118** *0.122*	4 * 8 CPUs
NEC SX-5Be local memory	2.27	**2.50** a)	**91 %**	35.1	512	**64** *60.50*	**0.039** *0.041*	2 *16 CPUs a) only 8 CPUs
Hitachi SR8000	0.45	**0.91**	49 %	5.0	32+32	**8** *6.82*	**0.114** *0.133*	8 * 8 CPUs
IBM SP Power3+	0.16	**0.57+)**	28 %	2.0	16	**24** *14.27*	**0.023** *0.040*	8 *16 CPUs
SGI O3800 600MHz (2 MB messages)	0.427+)	**1.74+)**	25 %	1.73+)	3.2	**4.80** *3.64*	**0.363** *0.478*	16 *4 CPUs
SGI O3800 600MHz (16 MB messages)	0.156	**0.400**	39 %	0.580	3.2	**4.80** *3.64*	**0.083** *0.110*	16 *4 CPUs
SGI O3000 400MHz (preliminary results)	0.10	**0.30+)**	33 %	0.39+)	3.2	**3.20** *2.46*	**0.094** *0.122*	16 *4 CPUs
SUN Fire 6800[a] (preliminary results)	0.15	**0.85**	18 %	1.68		**43.1** *23.3*	**0.019** *0.036*	4 *24 CPUs
HELICS Dual-PC cluster with Myrinet	0.127+)	**0.129+)**	**98 %**	0.186+)		**2.80** *1.61*	**0.046** *0.080*	32 * 2 CPUs
HELICS Dual-PC cluster with Myrinet	**0.105**	0.091	**100 %**	0.192		**2.80** *1.61*	**0.038** *0.065*	32 * 2 CPUs
HELICS Dual-PC cluster with Myrinet	0.118+)	**0.119+)**	**99 %**	0.104+)		**2.80** *1.61*	**0.043** *0.074*	128 * 2 CPUs
HELICS Dual-PC cluster with Myrinet	**0.093**	0.082	**100 %**	0.101		**2.80** *1.61*	**0.033** *0.058*	128 * 2 CPUs
HELICS Dual-PC cluster with Myrinet	**0.087**	0.077	**100 %**	0.047		**2.80** *1.61*	**0.031** *0.054*	239 * 2 CPUs
Column[b]	1	2	3	4	5	6	7	8

Table 1: Inter- and Intra-node bandwidth for large messages compared with memory bandwidth and peak performance. All values are aggregated over one SMP node. Each message counts only once for the bandwidth calculation. Message size is 16 MB, except +) with 2 MB.

[a] A degradation may be caused by system processes because the benchmark used all processors of the SMP nodes.

[b] Columns 1, 2, 4 are benchmark results, Col. 3 is calculated from Col. 1 & 2, Col. 5 & 6 "peak" are theoretical values, Col. 6 "Linpack" is based on the TOP500 values for the total system [14], and Col. 7 is calculated from Col. 1, 2 & 6.

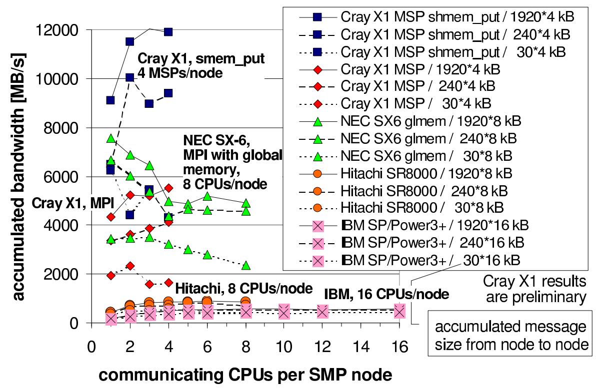

Figure 2: Aggregated bandwidth per SMP node.

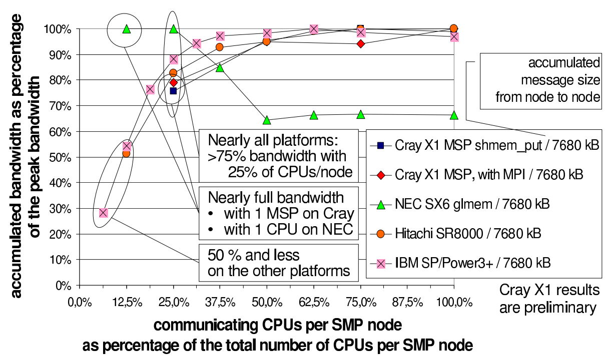

Figure 3: Aggregated bandwidth per SMP node.

compared the pure MPI with the masteronly scheme. For this comparison, each MPI process in the pure MPI scheme has also to exchange messages between the processes in the same node. These intra-node messages have the same size as the inter-node messages, (c.f. Fig. 1). Fig. 4 shows the ratio of the inter-node communication time of hybrid MPI+OpenMP masteronly style divided by the time needed for the inter- and intra-node communication with pure MPI. In case of hybrid masteronly style, the messages must transfer the accumulated data of the parallel inter-node messages in pure MPI style, i.e., the message size is multiplied with the number of CPUs of an SMP node. In Fig. 4, one can see a significantly better communication time with pure MPI, on those platforms, on which the inter-node network cannot be saturated by the master thread. In this benchmark, the master thread in the masteronly scheme was emulated by an MPI process (and the other threads by MPI process waiting

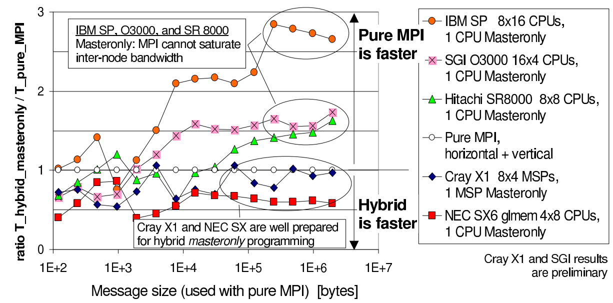

Figure 4: Ratio of hybrid communication time to pure MPI communication time.

in a barrier). On most platforms, the measurements were verified with a benchmark using hybrid compilation and hybrid application start-up. The diagram compares experiments with the same aggregated message-size in the inter-node communication; on the x-axis, the corresponding number of bytes in the pure-MPI experiment is shown. This means, e.g., that the message size in the hybrid-masteronly experiment on a 16-CPU-per-node system is 16 times larger than in the experiments with pure MPI.

Benchmark platforms were: a Cray X1 with 16 nodes at Cray; the NEC SX-6 with 24 nodes and IXS interconnect at the DKRZ, Hamburg, Germany; the Hitachi SR8000 with 16 nodes at HLRS, Stuttgart, Germany; the IBM SP-Power3 at NERSC, USA; the SGI Origin 3000 (400 MHz) *Lomax* with 512 CPUs at NASA/Ames Research Center, NAS, USA; an SGI Origin 3800 (600 MHz) at SGI; the SUN Fire 6800 cluster with Sun Fire Link at the RWTH Aachen, Germany; and HELICS, a Myrinet 2 GBit/s full bisection network based cluster of 256 dual AMD Athlon 1.4 GHz PCs at IWR, University of Heidelberg.

3.2 The sleeping-threads problem and the saturation problem

The two most simple programming models on hybrid systems have both the same problem although they look quite different: With hybrid masteronly style the non-master threads are sleeping while the master communicates, and with pure MPI, all threads try to communicate while only a few (or one) threads already can saturate the inter-node network bandwidth (expecting that the application is organized in communicating and computing epochs). If one thread is able to achieve the full inter-node bandwidth (e.g., NEC SX-6, see Fig. 2), then both problems are equivalent. If one thread can only achieve a small percentage (e.g., 28 % on IBM SP), then the problem with masteronly style is significantly higher.

As example on the IBM system, if an application communicates 1 sec in the pure MPI style (i.e. 1×16 = 16 CPUsec), then this program would need about 16/0.28 = 57 CPUsec in masteronly style, and if one would use 4 CPUs for the inter-node communication (4 CPUs achieve 88.3 %) and the other 12 threads for overlapping computation, then only 4/0.883 = 4.5 CPUsec would be necessary.

If the inter-node bandwidth cannot be achieved by one thread, then it may be a better choice to split each SMP node into several MPI processes that are itself multithreaded. Then, the inter-node bandwidth in the pure MPI and hybrid masteronly model are similar and mainly the topology, intra-node communication, and OpenMP-overhead problems determine which of both programming styles are more effective. When overlapping communication and computation, this splitting can also solve the inter-node bandwidth problem described in the previous section.

4 Bite the bullet

Each parallel programming scheme on hybrid architectures has one or more significant drawbacks. Depending on the needed resources of an applications, the drawbacks may be major or only minor.

Programming without overlap of communication and computation

One of the two problems, *sleeping-threads* and *saturation problem* is indispensable. The major design criterion may be the topology problem:

- If it cannot be solved, pure MPI may cause too much inter-node traffic, but the masteronly scheme implies on some platforms a slow inter-node communication due to the inter-node bandwidth problem described above.
- If the topology problem can be solved, then we can compare hybrid masteronly with pure MPI: On some platforms, wasting inter-node bandwidth with masteronly style is the major problem; it causes more CPUs longer idling than with pure MPI. For example on an IBM SP system with 16 Power3+ CPUs on each SMP node, Fig. 5 shows the aggregated bandwidth per node with the experiment described in Sect. 3.1. The *pure MPI horizontal+vertical* bandwidth is defined in this diagram by dividing the amount of inter-node message bytes (without counting the intra-node messages[2]) by the time needed for inter- and intra-node communication, i.e., the intra-node communication is treated as overhead. One can see, that more than 4 CPUs per node must communicate in parallel to achieve full inter-node bandwidth. At least 3 CPU per node must communicate in the hybrid model to beat the pure MPI model. Fig. 6 shows the ratio of the execution time in the hybrid models to the pure MPI model. A ratio greater than 1 shows that the hybrid model is slower than the pure MPI model.

On systems with 8 CPUs per node, the problem may be reduced, e.g., as one can see on a Hitachi SR 8000 in Fig. 7. On some vector type systems, one CPU may already be able to saturate the inter-node network, as shown in Fig. 8–10. Note: the aggregated inter-node bandwidth on the SX-6 is reduced, if more than one CPU per node tries to communicate at the same time over the IXS. Fig. 9 and 10 show preliminary results on a Cray X1 system with 16 nodes. Each SMP node consists of 4 MSPs (multi streaming processors). Each MSP itself consists of 4 SSPs (single streaming processors). With MSP-based programming, each MSP is treated as a CPU, i.e., each SMP node has 4 CPUs (=MSPs) that internally use an (automatic) thread-based parallelization (= *streaming*). With SSP-based programming, each SMP node has 16 CPUs (=SSPs). Preliminary results with the SSP-mode have shown, that the inter-node bandwidth is partially bound to the CPUs, i.e., that the behavior is similar to the 16-way IBM system.

Similar to the multi-threaded implementation of MPI on the Cray MSPs, it would be also possible on all other platforms to use multiple threads inside of the MPI communication routines if the application uses the hybrid masteronly scheme. The MPI library can easily detect whether the application is inside or outside of a parallel region. With this potential optimization (described in more detail in Sect. 5), the communication time of the hybrid masteronly model should always be shorter than the communication time in the pure MPI scheme.

[2]Because the intra-node messages must be treated as overhead if we compare pure MPI with hybrid communication strategies.

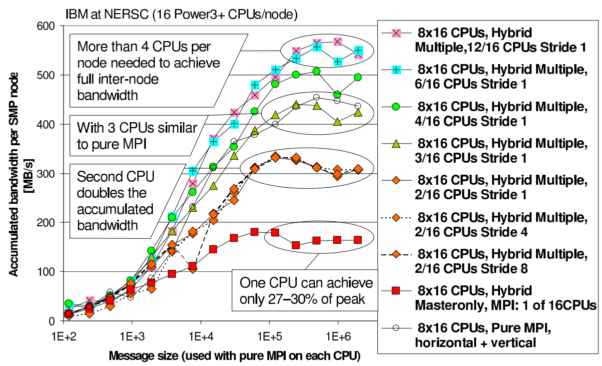

Figure 5: Aggregated bandwidth per SMP node on IBM SP with 16 Power3+ CPUs per node.

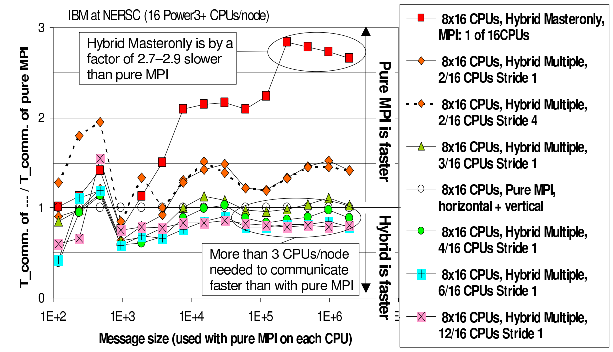

Figure 6: Ratio of communication time in hybrid models to pure MPI programming on IBM SP.

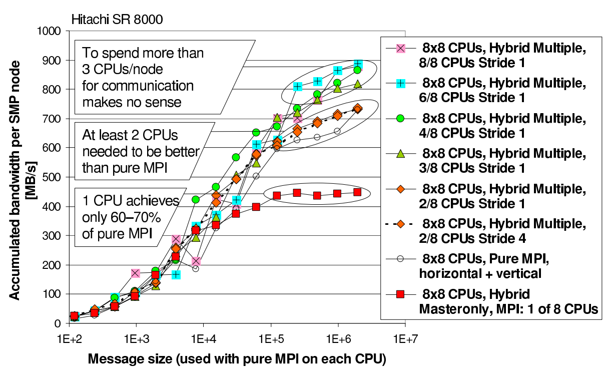

Figure 7: Aggregated bandwidth per SMP node on Hitachi SR 8000.

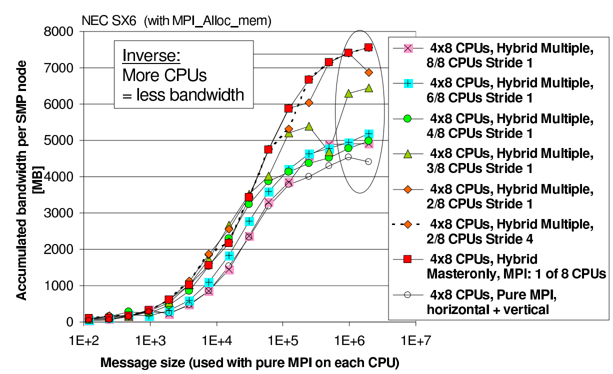

Figure 8: Aggregated bandwidth per SMP node on NEC SX-6.

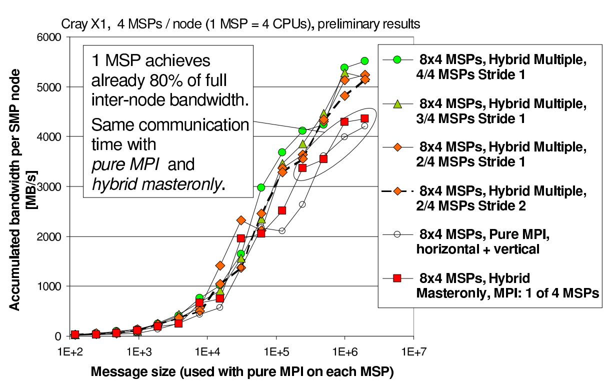

Figure 9: Aggregated bandwidth per SMP node on Cray X1, MSP-based MPI-parallelization.

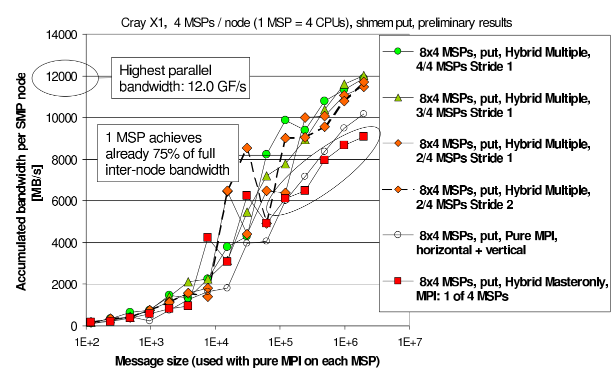

Figure 10: Aggregated bandwidth per SMP node on Cray X1. MSP-based and MPI_Sendrecv is substituted by shmem_put.

On the other hand, looking on the Myrinet cluster with only 2 CPUs per SMP node, the hybrid communication model hasn't any drawback on such clusters because one CPU is already able to saturate the inter-node network (see lowermost rows in Tab. 1).

Programming with overlap of communication and computation

Although overlapping communication with computation is the chance to achieve fastest execution, this parallel programming style isn't widely used due to the lack of ease of use. It requires a coarse-grained and thread-rank-based OpenMP parallelization, the separation of halo-based computation from the computation that can be overlapped with communication, and the threads with different tasks must be load balanced.

Advantages of the overlapping scheme are: (a) the problem that one CPU may not achieve the inter-node bandwidth is no longer relevant as long as there is enough computational work that can be overlapped with the communication; (b) the saturation problem is solved as long as not more CPUs communicate in parallel than necessary to achieve the inter-node bandwidth; (c) the sleeping threads problem is solved as long as all computation and communication is load balanced among the threads.

A detailed analysis of the performance benefits of overlapping communication and computation can be found in [17].

5 Optimization Chance

On Cray X1 with MSP-based programming and on NEC SX-6, the *hybrid masteronly* communication pattern is faster than the *pure MPI*. Although both systems have vector-type CPUs, the reasons for these performance results are quite different: On the NEC SX-6, the hardware of one CPU is really able to saturate the inter-node network if the user data resides in global memory. On the Cray X1, each MSP consist of 4 SSPs (=CPUs). MPI communication issued by one MSP seems internally to be multi-streamed by all 4 SSPs. With this multi-threaded implementation of the communication, Cray can achieve 75–80 % of the full inter-node bandwidth, i.e., of the bandwidth that can be achieved if all MSPs (or all SSPs) communicate in parallel.

This approach can be generalized for the masteronly style. Depending on whether the application itself is translated for pure MPI approach, hybrid MPI + automatic SMP-parallelization, or hybrid MPI+OpenMP, the linked MPI library itself can also be parallelized with OpenMP directives or vendor-specific directives.

Often, the major basic capabilities of an MPI library are to put data into a shared memory region of the destination process (RDMA put), or to get data from the source process (RDMA get), or to locally calculate reduction operations on a vector, or to handle derived datatypes and data. All these operations (and not the envelop handling of the message passing interface) can be implemented multi-threaded, e.g., inside of a parallel region. In the case, that the application calls the MPI routines outside of parallel *application* regions, the parallel region inside of the MPI routines will allow a thread-parallel handling of these basic capabilities. In the case, the application overlaps communication and computation, the parallel region inside of the MPI library is a nested region and will get only the (one) thread on which it is already running. Of course, the parallel region inside of MPI should only be launched, if the amount of data that must be transferred (or reduced) exceeds a given threshold.

This method optimizes the bandwidth without a significant penalty to the latency. On the Cray X1, currently only 4 SSPs are used to stream the communication in MSP mode achieving only 75–80 % of peak. It may be possible to achieve full inter-node bandwidth, if the SSPs of an additional MSP would also be applied. With such a multi-threaded implementation of the MPI communication for masteronly-style applications, there is no further need (with respect to the communication time) to split large SMP nodes into several MPI processes each with a reduced number of threads (as proposed in Sect. 3.2).

6 Conclusions

Different programming schemes on clusters of SMPs show different performance benefits or penalties on the hardware platforms benchmarked in this paper. Table 1 summarizes the results. Cray X1 with MSP-based programming and NEC SX-6 are well designed for the hybrid MPI+OpenMP masteronly scheme. On the other platforms, as well as on the Cray X1 with SSP-based programming, the master thread cannot saturate the inter-node network which is a significant performance bottleneck for the masteronly style.

To overcome this disadvantage, a multi-threaded implementation of the basic device capabilities in the MPI libraries is proposed in Sect. 5. Partially, this method is already implemented in the Cray X1 MSP-based MPI-library. Such MPI optimization would allow the saturation of the network bandwidth in the masteronly style. The implementation of this feature is important especially on platforms with more than 8 CPUs per SMP node.

This enhancement of current MPI implementations implies that the hybrid masteronly communication should be always faster than pure MPI communication. Both methods still include the sleeping threads or saturated network problem, i.e., that more CPUs are used for communicating than really needed to saturate the network. This drawback can be solved with overlapping of communication and computation, but this programming style needs extreme programming effort.

To achieve an optimal usage of the hardware, one can also try to use the idling CPUs for other applications, especially low-priority single-threaded or multi-threaded non-MPI applications if the parallel high-priority hybrid application does not use the total memory of the SMP nodes.

Acknowledgments

The author would like to acknowledge his colleagues and all the people that supported this project with suggestions and helpful discussions. He would especially like to thank Gerhard Wellein at RRZE, Dieter an Mey at RWTH Aachen, Thomas Ludwig, Stefan Friedel, Ana Kovatcheva, and Andreas Bogacki at IWR, Monika Wierse, Wilfried Oed, and Tom Goozen at CRAY, Holger Berger at NEC, Reiner Vogelsang at SGI, Gabriele Jost at NASA, and Horst Simon at NERSC for their assistance in executing the benchmark on their platforms. This research used resources of the HLRS Stuttgart, LRZ Munich, RWTH Aachen, University of Heidelberg, Cray Inc., NEC, SGI, NASA/AMES, and resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

References

[1] Rudulf Berrendorf, Michael Gerndt, Wolfgang E. Nagel and Joachim Prumerr, *SVM Fortran*, Technical Report IB-9322, KFA Jlich, Germany, 1993. `www.fz-juelich.de/zam/docs/printable/ib/ib-93/ib-9322.ps`

[2] Frank Cappello and Daniel Etiemble, *MPI versus MPI+OpenMP on the IBM SP for the NAS benchmarks*, in Proc. Supercomputing'00, Dallas, TX, 2000. `http://citeseer.nj.nec.com/cappello00mpi.html` `www.sc2000.org/techpapr/papers/pap.pap214.pdf`

[3] The Earth Simulator. `www.es.jamstec.go.jp`

[4] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard*, in Parallel Computing 22–6, Sep. 1996, pp 789–828. `http://citeseer.nj.nec.com/gropp96highperformance.html`

[5] Shinichi Habataa, Mitsuo Yokokawa, and Shigemune Kitawaki, *The Earth Simulator System*, in NEC Research & Development, Vol. 44, No. 1, Jan. 2003, Special Issue on High Performance Computing.

[6] Jonathan Harris, *Extending OpenMP for NUMA Architectures*, in proceedings of the Second European Workshop on OpenMP, EWOMP 2000. `www.epcc.ed.ac.uk/ewomp2000/proceedings.html`

[7] D. S. Henty, *Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling*, in Proc. Supercomputing'00, Dallas, TX, 2000. `http://citeseer.nj.nec.com/henty00performance.html` `www.sc2000.org/techpapr/papers/pap.pap154.pdf`

[8] Matthias Hess, Gabriele Jost, Matthias Müller, and Roland Rühle, *Experiences using OpenMP based on Compiler Directed Software DSM on a PC Cluster*, in WOMPAT2002: Workshop on OpenMP Applications and Tools, Arctic Region Supercomputing Center, University of Alaska, Fairbanks, Aug. 5–7, 2002. `http://www.hlrs.de/people/mueller/papers/wompat2002/wompat2002.pdf`

[9] Georg Karypis and Vipin Kumar. *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, Journal of Parallel and Distributed Computing, 48(1): 71–95, 1998. `http://citeseer.nj.nec.com/karypis98parallel.html` `http://www-users.cs.umn.edu/ karypis/metis/`

[10] Richard D. Loft, Stephen J. Thomas, and John M. Dennis, *Terascale spectral element dynamical core for atmospheric general circulation models*, in proceedings, SC 2001, Nov. 2001, Denver, USA. www.sc2001.org/papers/pap.pap189.pdf

[11] John Merlin, *Distributed OpenMP: Extensions to OpenMP for SMP Clusters*, in proceedings of the Second European Workshop on OpenMP, EWOMP 2000. www.epcc.ed.ac.uk/ewomp2000/proceedings.html

[12] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Rel. 1.1, June 1995, www.mpi-forum.org.

[13] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, July 1997, www.mpi-forum.org.

[14] Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst D. Simon, Universities of Mannheim and Tennessee, *TOP500 Supercomputer Sites*, www.top500.org.

[15] OpenMP Group, www.openmp.org.

[16] Rolf Rabenseifner, *Hybrid Parallel Programming: Performance Problems and Chances*, in proceedings of the 45th CUG Conference 2003, Columbus, Ohio, USA, May 12–16, 2003, www.cug.org.

[17] Rolf Rabenseifner and Gerhard Wellein, *Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures*, International Journal of High Performance Computing Applications, Sage Science Press, Vol. 17, No. 1, 2003, pp 49–62.

[18] Mitsuhisa Sato, Shigehisa Satoh, Kazuhiro Kusano, and Yoshio Tanaka, *Design of OpenMP Compiler for an SMP Cluster*, in proceedings of the 1st European Workshop on OpenMP (EWOMP'99), Lund, Sweden, Sep. 1999, pp 32–39. http://citeseer.nj.nec.com/sato99design.html

[19] Alex Scherer, Honghui Lu, Thomas Gross, and Willy Zwaenepoel, *Transparent Adaptive Parallelism on NOWs using OpenMP*, in proceedings of the Seventh Conference on Principles and Practice of Parallel Programming (PPoPP '99), May 1999, pp 96–106.

[20] Weisong Shi, Weiwu Hu, and Zhimin Tang, *Shared Virtual Memory: A Survey*, Technical report No. 980005, Center for High Performance Computing, Institute of Computing Technology, Chinese Academy of Sciences, 1998, www.ict.ac.cn/chpc/dsm/tr980005.ps.

[21] Lorna Smith and Mark Bull, *Development of Mixed Mode MPI / OpenMP Applications*, in proceedings of Workshop on OpenMP Applications and Tools (WOMPAT 2000), San Diego, July 2000. www.cs.uh.edu/wompat2000/

[22] Gerhard Wellein, Georg Hager, Achim Basermann, and Holger Fehske, *Fast sparse matrix-vector multiplication for TeraFlop/s computers*, in proceedings of VECPAR'2002, 5th Int'l Conference on High Performance Computing and Computational Science, Porto, Portugal, June 26–28, 2002, part I, pp 57–70. http://vecpar.fe.up.pt/