MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks

Franck Cappello and Daniel Etiemble
LRI, Université Paris-Sud, 91405, Orsay, France
Email: fci@lri.fr

Abstract

The hybrid memory model of clusters of multiprocessors raises two issues: programming model and performance. Many parallel programs have been written by using the MPI standard. To evaluate the pertinence of hybrid models for existing MPI codes, we compare a unified model (MPI) and a hybrid one (OpenMP fine grain parallelization after profiling) for the NAS 2.3 benchmarks on two IBM SP systems. The superiority of one model depends on 1) the level of shared memory model parallelization, 2) the communication patterns and 3) the memory access patterns. The relative speeds of the main architecture components (CPU, memory, and network) are of tremendous importance for selecting one model. With the used hybrid model, our results show that a unified MPI approach is better for most of the benchmarks. The hybrid approach becomes better only when fast processors make the communication performance significant and the level of parallelization is sufficient.

1 Introduction

Some primary supercomputer manufacturers like IBM and Compaq use CLUsters of MUltiProcessors (CLUMP) to provide scalable high performance computers. The future ASCI White machine corresponds to this architecture with 512 16-way SMP nodes. CLUMPs may use different memory models for the interconnection of the SMP nodes: NUMA or Message-Passing. The SGI Origin computers use the NUMA approach as well as the Convex Exemplar ones. The IBM SPs and the Compaq SC Clusters use message-passing. In this paper, we focus on CLUMPs with a hybrid memory model (shared memory inside nodes and message passing between nodes) at the hardware level. Two main issues should be addressed for selecting a programming model: the ease of use and performance.

First, we must choose between a unified programming model or a hybrid one. In the unified programming models, the programmer uses a single API to describe the communications inside the multiprocessor nodes and between them. All the "message passing" or DSM systems belong to this category. The hybrid memory models mix shared memory inside the multiprocessor and message passing between the nodes. MPI+OpenMP, MPI+threads are two hybrid models. Because the hybrid programming models require managing two different memory models, they are much more complex for the programmer. Also, many existing applications use a MPI implementation. For these applications, the relevance of the hybrid programming models must be carefully examined because, as we will demonstrate, even optimized hybrid codes may provide insignificant performance improvement compared to the original MPI version.

The second issue is performance. It depends on at least three parameters: a) the sharing of communication support (memory system and network interface) between the processors for the unified model i.e. how the per processor latency and bandwidth evolve when several processors use a same network interface (and protocol), b) the degree of shared memory parallelism for the hybrid model and c) the speed-up on the parallel section for both models that can be different because loop nests may be executed in a different way.

Section 2 presents the programming models and the methodology. Section 3 compares the performance of MPI unified model and the MPI+ OpenMP hybrid model for the NAS 2.3 Benchmarks. We present detailed results for different cluster sizes and data set sizes (CLASS A and CLASS B) in section 4. Section 5 presents related works. Finally, section 6 concludes.

2 Programming models and methodology

In this paper, we focus on existing MPI programs that have been developed for traditional parallel machines. Because most of the manufacturers provide extended versions of their communication library for clusters of multiprocessors, existing MPI codes can be directly used with a unified MPI model. The alternative is mixing MPI with a shared

0-7803-9802-5/2000/$10.00 (c) 2000 IEEE.

memory model such as OpenMP. In that case, different possibilities exist, which must be compared according to the performance and programming effort trade-off.

2.1 Programming efforts for mixing MPI + OpenMP

2.1.1 Fine-grain parallelization

From an existing MPI code, the simplest approach is the incremental one: it consists in OpenMP parallelization of the loop nests in the computation part of the MPI code. This approach is also called OpenMP fine-grain or loop level parallelization. Several options can be used according to 1) the programming effort and 2) the choice of the loop nests to parallelize.

Several levels of programming effort can be used. First possibility consists in parallelizing loop nests in the computation part of the MPI code without any manual optimization. Only the correctness of the parallel version versus the sequential version semantic is checked. But the incremental approach can be significantly improved by applying several manual optimizations (loop permutation, loop exchange, use of temporary variables). These optimizations are required 1) to transform non parallel loop nests into parallel ones and 2) to improve the parallel efficiency by avoiding false sharing or reducing the number of synchronization points (critical sections, barriers).

Another issue is the choice of the loop nests to parallelize. One option is to parallelize all loop nests. This option has two drawbacks: it increases the programming effort and the parallelization of loop nests that doesn't contribute significantly to the global execution time can be counterproductive. The alternative consists in selecting by profiling the loop nests that contribute significantly to the global execution time. In this work, we have selected the loops to parallelize according to the framework shown in figure 1. The complete description of intra-node parallelization process for the NAS parallel benchmarks is described in [1].

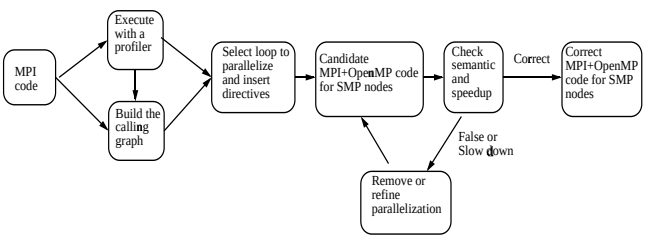

Figure 1. The parallelization framework

2.1.2 Coarse-grain parallelization

Instead of applying a two level parallelization (process level and loop level), another currently investigated approach is the coarse-grain OpenMP SPMD parallelization. In this approach, OpenMP is still used to take advantage of the shared memory inside the SMP nodes but a SPMD programming style is used instead of the traditional shared memory multithread approach. In this mode, OpenMP is used to spawn N threads in the main program, having each thread act similarly to a MPI process. The OpenMP `Parallel` directive is used at the outermost level of the program. The principle is to spawn the threads just after the spawn of the MPI processes (some initializations may separate the two spawns). As for the message passing SPMD approach, the programmer must take care of several issues: array distribution among threads, work distribution among threads and coordination between threads. Since the array distribution is done assuming a shared memory, the distribution of the arrays only concerns the attribution of different array regions to the different running threads. For maximum performance, these regions should not overlap for write references. The work distribution is made according to the array distribution. Typically, the OpenMP `DO` directive is not used for distributing the loop iterations among threads. Instead, the programmer inserts some calculations of the loop boundaries that depends on the thread number. Coordinating the threads involves managing critical sections (I/O, MPI calls) using either OpenMP directives like `Master` or thread library calls like `omp_get_thread_num()` to guard conditional statements. Few results have been published on the coarse-grain mixed OpenMP+MPI parallelization.

In this paper, we only consider the fine grain incremental approach including manual optimizations and profiling to choice the loop nests to parallelize. The comparison results between MPI and MPI+OpenMP performance presented in this paper are only valid for this approach.

2.2 MPI versus MPI+OpenMP

MPI The MPI processes within nodes communicates through the memory (MP-SHARED-MEMORY option on the IBM SP) without using the network interface. The user space mode has been used for highest possible performance. With this model, existing MPI codes run without modification (except for time measurements).

MPI+OpenMP Parallelizing an application for the SPMD MPI paradigm often produces a program with the typical layout presented in the left part of figure 2. It corresponds to the parallelization for a cluster of uniprocessor nodes, where a MPI process is allocated on each node. The right part shows how the computation part of the MPI process is split into threads by using OpenMP directives. The number of threads corresponds to the number of CPUs in each node. In Figure 2, P is the part of MPI code that can-

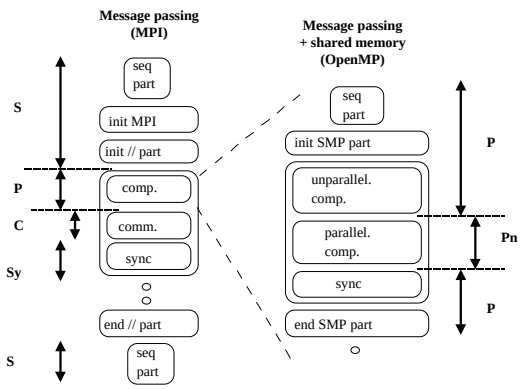

Figure 2. Parallelizing a MPI code with OpenMP by using the fine grain approach: the computation part of the MPI code on a node is parallelized with OpenMP directives.

not be parallelized with OpenMP and Pn is the OpenMP parallel part.

2.3 IBM SP systems

Two IBM SP systems were used for the experiments because they exhibit two different balances between the performance of the main components (CPU, Memory, Network) performance. The first system gathers 32 WinterHawk II (WH2) multiprocessors with 4 Power3+ CPUs per node running at 375 MHz. Memory system for these nodes is based on a bus with a 1.6 GB/s maximum bandwidth. The second system connects 8 NightHawk I (NH1) multiprocessors based on Power3 CPUs running at 222 MHz. Each NH1 multiprocessor provides a theoretical 14.2 GB/s peak memory bandwidth under specific assumptions (memory bank population). The Power3 CPU has a 64 KB data cache. The unified L2 cache sizes are respectively 8 MB (WH2 nodes) and 4 MB (NH1 nodes) per CPU.

The multi-stage interconnection network is the same for both SP systems. Each multiprocessor node has one network interface. The maximum per node network bandwidth is 150-MB/sec unidirectional and 300-MB/sec bidirectional.

The SP software environment includes the AIX 4.3 operating system, the XLF 6.1 Fortran compiler that includes OpenMP directives and the PPE 2.4 Parallel Environment (intra-node MPI communications use the shared memory). The following compiler options have been used: -O3 -qarch=pwr3 -qtune=pw3 -qcache=auto for MPI. The Power3 microprocessor options allow the use of prefetch instructions. The -qcache=auto option lets the compiler decides whether optimizing or not according to the features of memory hierarchy. For OpenMP, the following options are needed: -qsmp=noauto:schedule=static. It means that the compiler must not parallelize automatically, but obey the directives (OpenMP or other ones). The scheduling option indicates how the loop iterations must be distributed among the different threads. The static option without argument means a block distribution with a block size of I/n, where I is the number of loop iterations and n is the number of parallel threads that will run concurrently.

Table 1 details the bidirectional communication performance (asynchronous echo test) with WH2 nodes. Note that no compiler optimizations were used for these measures.

	Uniprocessor	4-way node External comm.	4-way node Internal comm.
Bandwidth	193 MB/s	49 MB/s (per CPU)	360 MB/s (1 comm.) 174 Mo/s (2 comm.)
Latency	17 μs	37 μs	7.8 μs (1 comm.) 8.4 μs (2 comm.)

Table 1. Communication performance for bidirectional point to point tests (1 comm. means that only two processors are communicating. 2 comm. means that two couples of processors are communicating simultaneously).

Table 1 shows that each CPU can use 1/4 of the maximum bandwidth delivered by the network board. The measured standard deviation is very low. It is lower than 5% for the per processor communication bandwidth when 4 CPUs from one node communicate with 4 CPUs of another node (extended echo test for clusters of SMP nodes). For this communication scheme, each CPU experiences a latency of 37 μs. For a given number of small messages to send from a node, the total communication time is thus two times smaller with 4 CPUs per node than with only one. The reason is the overlap between the computation parts of the communications. The internal communication performance is far greater than the external one. Each node has a bus between the CPUs and the memory. This is why the bandwidth decreases by a half for two simultaneous communications within the SMP node.

2.4 Using SMP nodes

2.4.1 Scalability of the NAS benchmarks with 1-way nodes

Before examinig the scalability of the NAS benchmarks with the two different memory models on clusters of multiprocessors, we present some background results about their scalability on a cluster of uniprocessors. The behaviour of

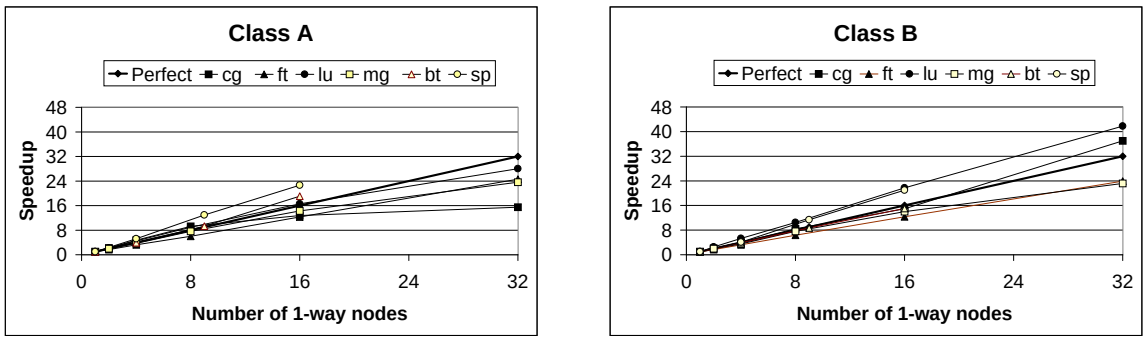

Figure 3. Speedup for the NAS benchmarks with WH2 nodes. The left part corresponds to Class A and the right part to Class B.

the NAS benchmark highly depends on the balance between the main components of the parallel architectures. In [2] most of the Class A benchmarks except IS and FT have a linear speedup up to 32 processors on recent architectures. The SGI Origin even provides superlinear speedup for some benchmarks. The figure 3 presents the speedup of the Class A and class B benchmarks for the SP3 with WH2 nodes. The observed speedups, which depend on the communication/computation ratio, are sensitive to the dataset sizes, as shown by CG and LU (sub-linear for Class A and superlinear for Class B for 32 nodes). Previous results available on the NAS Benchmark Web page generally show sub-linear speedups for most of the computers. [2] presents a deep analysis of the reasons behind the different behaviours of the NAS benchmarks on different machines. The main parameters of the scalability are the sequential node architecture and the performance of the communication system. The authors demonstrate that although the SGI Origine has a relatively low communication efficiency on the NAS communication patterns, some superlinear speedups are achievable because of the node architecture (cache, memory system).

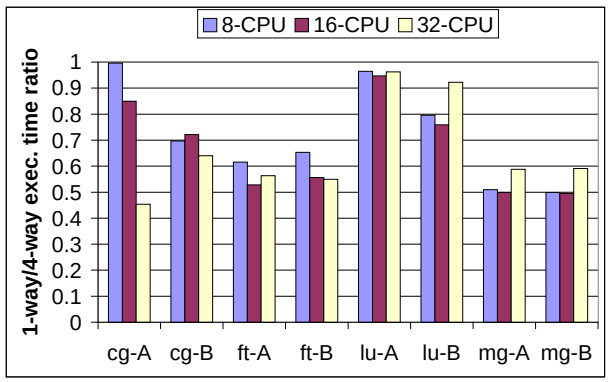

Figure 4. Ratio of the MPI execution time with 1-way WH2 nodes over the MPI execution time with 4-way nodes according to the number of CPUs for the class A and class B benchmarks. The values are less than 1 when 1-way nodes are better.

2.4.2 1-way nodes versus 4-way nodes

Before going into details of the performance comparison, we want to confirm the intuition that sharing the memory system and network interface makes a cluster of SMP nodes less efficient than a cluster of uniprocessor nodes using the same number of CPUs. On the SP system with 32 WH2 nodes, we have compared the execution times of class A and class B MPI benchmarks when using the same number of CPUs, with either 1-way nodes or 4-way nodes. The comparison have thus been done with 8, 16 and 32 CPUs. Results are shown in figure 4. The figure shows that the ratio between clusters of 1-way nodes and clusters of 4-way nodes is always less than 1 for any benchmark and any number of CPUs. Performance ratio clearly depends on the application. For FT, the gap is accentuated because this benchmark uses global communications, which are presently not highly optimized on the SP systems. The results of the comparison confirms the ones presented in [5] where the authors claim that a cluster of uniprocessors can be faster that a cluster of multiprocessors. The detailed comparison of these

two types of clusters is out of the scope of this paper, which aims on comparing programming models. More, comparing performance according to the number of CPUs is very debatable according to the cost issues and the current trends in parallel architectures. Because of the increasing gap between CPU performance and DRAM performance, the technological trends are towards larger SMP nodes (NH2 nodes of the IBM SP use 16-way nodes and the future SP4 systems will have 32-way nodes) built as clusters of smaller SMP nodes. This is why we only consider performance of SMP nodes in the rest of the paper. 4-way nodes are the largest nodes that can be used to compare MPI and MPI+OpenMP on the two SP systems that were used.

2.5 Methodology

We first applied the incremental approach to the original MPI NPB-2.3 benchmarks to get the hybrid version. The NAS benchmarks have not been tuned specifically to take advantage of the memory hierarchy of the Power3 for both models. So performance results do not give the highest possible value for these codes. However, we try to be fair in comparing codes with quite equivalent optimization levels. Then, the execution time have been decomposed into computation and communication times. For that, all benchmarks have been instrumented with time measurements. Timings are accumulated across all nodes and divided by the number of nodes to give mean values per processor. The communication time includes the communication calls and the synchronization procedures (Wait).

During the experiments on the IBM platforms and up to now, no software was available to access the hardware performance counters of the Power3 on the IBM SP. So all interpretations of the measured results are based on the timing measurements. We have taken care to limit the interpretation to the clearly apparent phenomena. A previous study has been done and published [3] for a cluster of 64 2-way Pentium-II nodes. On this platform, we had access to the hardware performance counters and we could provide a deeper analysis.

3 Overall comparison between MPI and MPI+OpenMP

Figure 5 presents the ratio between the MPI execution time and the MPI+OpenMP execution time according to the number of 4-way nodes for each class of benchmarks (A or B) and each type of SP nodes (WH2 and NH1). For BT and SP, which use a square number of processes, we only use the number of nodes (1, 4 and 16) that allows a direct comparison with the other benchmarks, omitting the 9-node configuration. A ratio less than 1 means that the unified MPI version is more efficient than the hybrid version.

The comparison results are clearly application-dependent. Whatever nodes and data sets are used, the unified MPI model is always better for LU, MG, BT and SP. The advantage is spectacular with LU, for which unified MPI is always more than 2 times more efficient than MPI+OpenMP. For CG and FT, the advantage of one model depends on the data set size, on the type of nodes and also on the number of nodes. For NH1 nodes and class A, MPI shows better performance for CG and FT for any number of nodes. On the opposite, the hybrid model is always better for FT with WH2 nodes (classes A and B) and for class B (WH2 and NH1 nodes). It is always better for CG with WH2 nodes and class B. The advantage of one model can depend on the number of nodes: for CG and the WH2-class A and NH1-class B configurations, MPI is better for a small number of nodes and MPI+OpenMP for a larger number of nodes. In both case, the threshold is low (respectively for 2 and 8 nodes).

The results are quite similar for the classes A and B, but they are slightly more favorable to the hybrid model for the class B. The results are also slightly more favorable to the hybrid model when using WH2 nodes instead of NH1 nodes.

These results show that the relative performance of each model depends on the application, the dataset size and the features of the different components of the architecture (CPU, Memory system, Interconnection system).

4 Understanding the performance results

To explain the results of the previous section, we have decomposed the execution time into computation and communication times. According to Figure 2, we have further decomposed the computation time into S+P and Pn, where S, P and Pn are respectively the sequential time, the MPI parallel time (that cannot be parallelized with OpenMP) and the computation time that can be parallelized with OpenMP.

4.1 Amdahl's law for the hybrid model

For constant size problems, when using the MPI unified model with N 4-way nodes, 4N CPUs will run concurrently all the code parts except the sequential and the communication parts. For the hybrid model with N 4-way nodes, only the Pn part of the program can be accelerated.

Figure 6 presents the Pn execution time/total execution time ratio according to the number of 1-way nodes to evaluate the part of the execution time that can effectively benefit from the OpenMP parallelization on SMP nodes. The results correspond to Class A and Class B benchmarks.

Clearly, the time spent on the sequential sections and the MPI parallel sections that cannot be parallelized with OpenMP and the communication time limit the fraction of

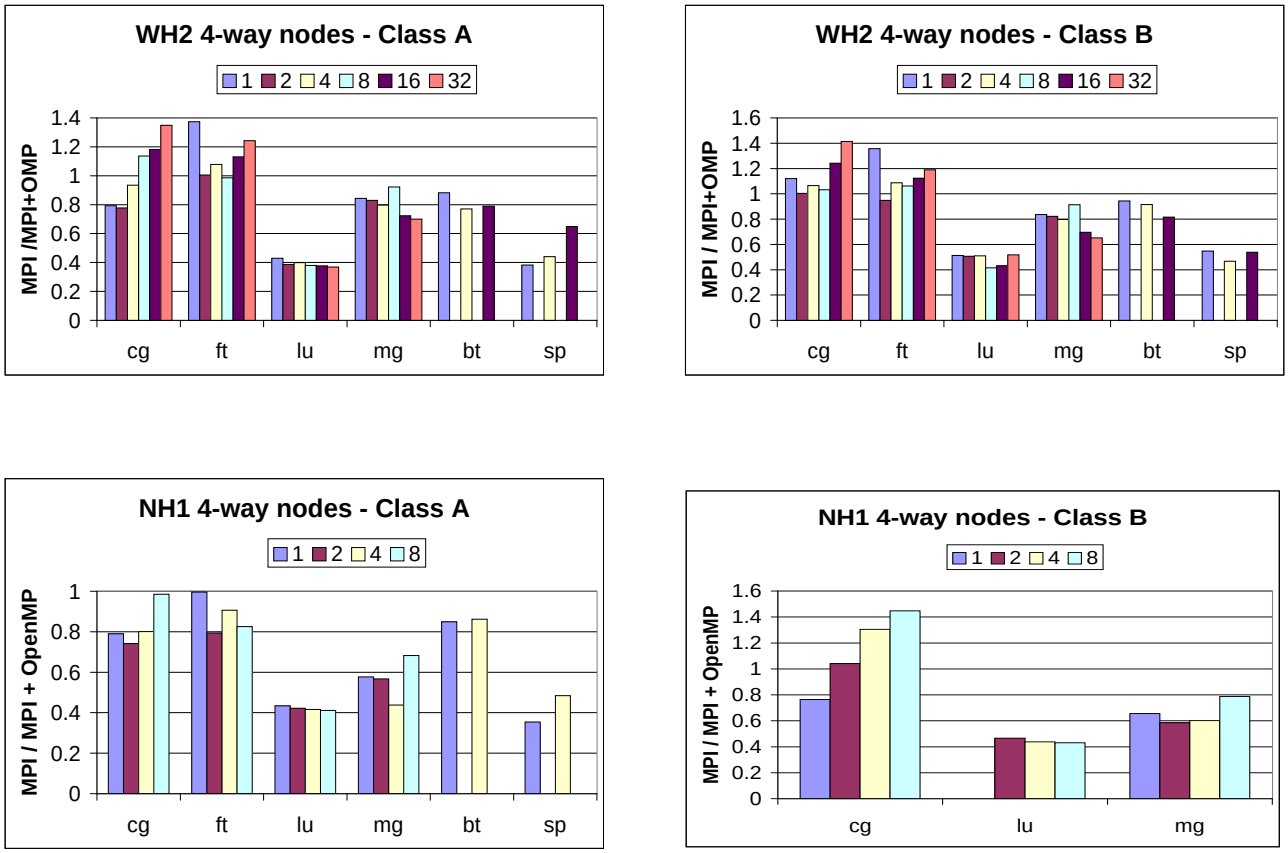

Figure 5. Ratio of the MPI execution time over the MPI+OpenMP execution time according to the number of 4-way nodes. The values are less than 1 when MPI is better. The top diagrams correspond to WH2 nodes, with class A (left) and class B (left). The bottom diagrams correspond to NH1 nodes. For the NH1 nodes and class B, some measures are missing.

the overall code that can be accelerated with an OpenMP parallelization. This application of Amdahl's law is more significant for the Class A than for the Class B, for which the communication impact is less significant. Its impact is more significant with the most powerful WH2 nodes. For class A and WH2 nodes, CG has the worst behavior, the Pn fraction going down to 25%. Pn fraction decreases from 100% (CG, FT) or 80% (LU, MG) down to 50 or 70% according to the benchmark and the class.

These results are relevant considering the original assumption that we use OpenMP to parallelize existing MPI applications. Also, we should state that a better parallelization could be achieved for some benchmarks like LU using a stronger parallelization effort, but this no longer corresponds to the incremental approach of parallelizing loop nests, which is the context of this paper.

4.2 Communication and Computation times

To explain why the hybrid model performs better on some benchmarks and SP configurations, we have decomposed the execution time into computation and communication times. Summing separately the computation times and communication times across all CPUs make easier the presentation of the results. With this assumption, a perfect speedup would give the same value whatever number of CPUs is used.

Class A results with WH2 nodes Figure 7 shows the result for CG, FT, LU and MG in class A with WH2 nodes. Comparing the left and right scales for each benchmark indicates the relative contribution of the communication time on the overall execution time.

For all benchmarks except LU, MPI communication

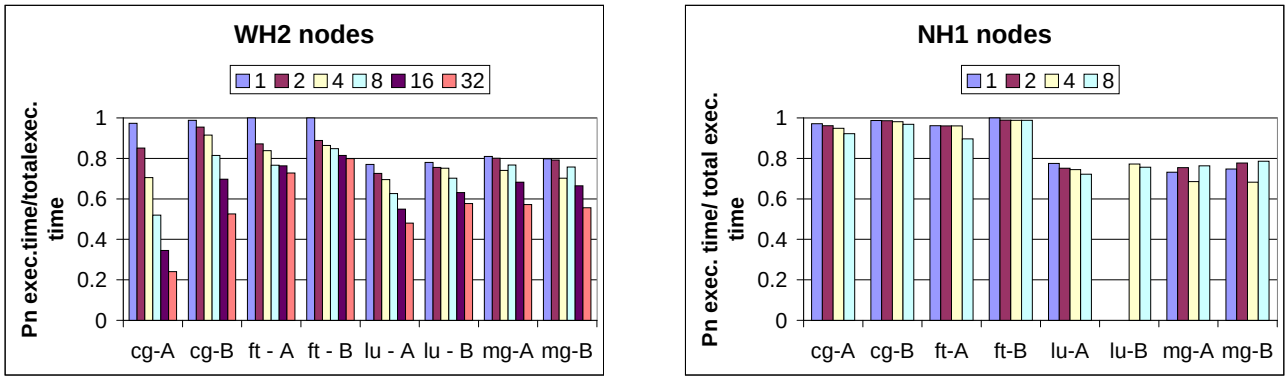

Figure 6. Fraction of the execution time that can be accelerated with OpenMP on clusters of 1-way nodes according to the number of nodes for Class A and B benchmarks.

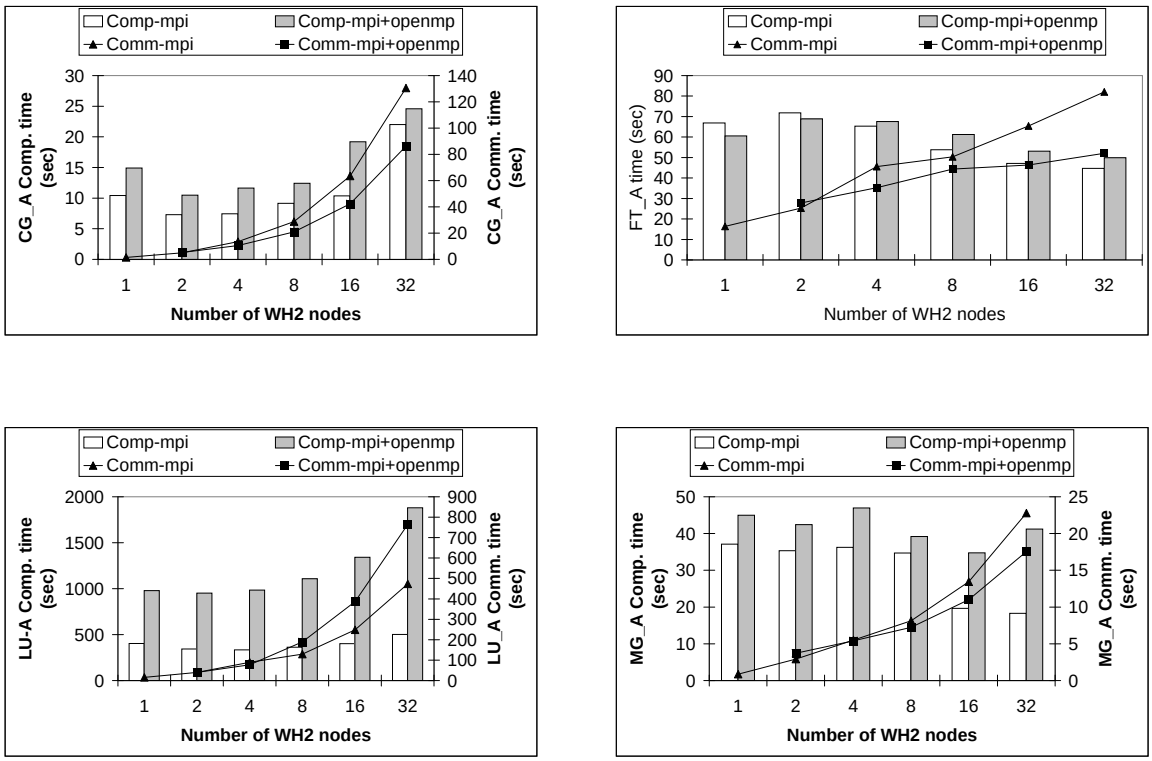

Figure 7. Time Breakdown for MG, CG, LU and FT MPI and MPI+OpenMP version according to the number of 4-way WH2 nodes. The figures breakdown the total execution time, summed across all the processors, into computation times (shown as bars with the left scale) and communication times (lines with the right scale).

times are larger than for MPI+OpenMP. The difference increases significantly from 4 nodes. To explain this difference we should state that the two models use different communication patterns for the same number of CPUs. With N 4-way nodes, there are communications between 4N processes for the MPI model opposed to communications between N processes for the MPI+OpenMP model. For constant size problems, Won et al. [2] have shown for the NAS benchmarks that message size generally decreases when the number of nodes increases, but the number of messages sent and received by each node is likely to increase as more nodes lead to more complicated communication patterns.

These communication patterns and performance (latency and bandwidth) explain the behavior of CG. Figure 8 presents the main communication patterns for CG when using 2, 4, 8 and 16 processors. The small white boxes are for the processors. The grey rectangles correspond to the mapping of communications on 4-way nodes. The external communications are between the grey boxes. In each part of the figure, communications are decomposed into small and large messages, with the number of each type (number * 64-bit words). For CG, the long messages dominates the communication time. When increasing the number of processes, CG exchanges more long messages but the messages are shorter. As shown in table 1, the bandwidth when using a single MPI process per node is similar to the aggregate bandwidth when using 4 MPI processes per node. So, only the number of long messages gives advantage to the MPI+OpenMP.

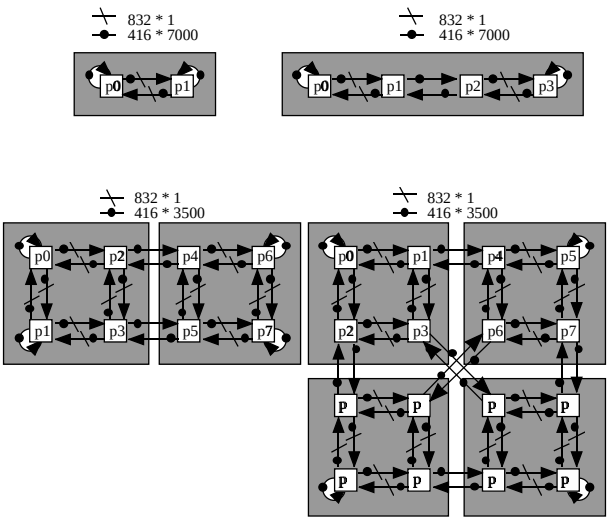

Figure 8. Communication patterns for CG using 2, 4, 8 and 16 processors.

The comparison of the communication time for LU gives the opposite result. The unified MPI approach provides a lower communication time than the hybrid approach. Note that the communication time also encompasses the synchronization of the running processes. In the original code, there is no way to separate the communication and the synchronization times. LU uses many more small ($<$ 1 KB) messages with blocking communication calls than large messages with asynchronous communication calls. Only considering the extended echo test (for cluster of multiprocessors) for synchronous communications cannot explain the opposite behavior of LU compared to CG. Since 1) the communication latency for the four processes inside a SMP node (unified approach) is two times higher (71 μs for 64 processes) than the communication latency (30 μs for 16 processes) for a single process node (hybrid approach) and 2) the communication time for the hybrid approach (30 μs) stays close to the communication time (22 μs) for a void message (the latency is the dominant factor) although the hybrid approach uses larger messages (2 times) than the unified approach, the communication pattern should give the advantage to the hybrid approach. The opposite result suggests that the synchronization time is higher for the hybrid approach than for the unified one.

For all benchmarks, the MPI computation times are less than the MPI+OpenMP ones. The difference is relatively small for FT and CG. For MG, it becomes significant for a large number of nodes. For LU, the MPI computation time is more than two times smaller than the MPI+OpenMP one. Three factors contribute to the difference in computation time between the two versions. First one is the Amdahl's law on the OpenMP parallel part that was mentioned in the previous section. Second one is the data layout, which may be different when a process is split into 4 threads by an OpenMP parallelization or when there are 4 MPI processes within a node. This difference impacts on the cache behaviour, specially when MPI programming allows to express multi-dimensional blocking. A third factor is the overhead of thread management in the OpenMP version. Next section on "memory access patterns" and parallel efficiency on the OpenMP parallel part will give more information to understand the large difference on LU computation times between the two versions.

Combining computation and communication values explain the overall results. The MPI approach with LU, which has both better computation and communication performance, outperforms the hybrid approach. For MG, the computation advantage of MPI always compensates the communication disadvantage. For CG, the computation advantage of MPI compensates the communication disadvantage only for a small number of nodes.

For FT, communication disadvantage of the MPI approach associated with the more complicated "all to all" communication patterns leads to better performance with the hybrid approach. As previously mentioned, the global

communications on SP3 systems are not optimized with SMP nodes.

Other configurations For LU and MG, the decomposition of the execution time gives very similar results for the 3 other configurations: WH2-class B, NH1-class A and NH1-class B.

We present now for each configuration the time decomposition of CG (figure 9). Like FT, CG is a benchmark for which the best programming model depends on the configuration, the dataset size and the number of nodes. For class A with WH2 nodes, as previously explained, the computation advantage of MPI compensates the communication disadvantage up to 4 nodes, but not for a larger number of nodes. With NH1 nodes, the computation for both models are larger than with WH2 nodes (according to respective clock frequency) and the communication time are also slightly larger: the situation is similar, except that MPI keeps advantage until 8 nodes, but measures with 16 NH1 nodes would give advantage to the hybrid model. For class B, the clock frequency advantage of WH2 nodes only appears for 4 or 8 nodes, probably because some cache effects become significant. For less nodes, the frequency advantage is counterbalanced by the worse performance of the memory system (bus versus crossbar). With NH1 nodes, the gap between the MPI communication time and the MPI+OpenMP communication time increases quickly and only the configuration with one node shows an advantage for the MPI model. Remember that the communication time includes the synchronization time. We have no clear explanation for the significant difference in MPI communication times between the two hardware systems for the same number of nodes. One reason might be that the communication time encompasses a CPU part and a Network Interface Communication part. In that case, slower CPUs would lead to larger communication times. Unfortunately, no hardware monitoring counters are available on the SP systems. They would be needed to precisely examine the behaviour of the memory hierarchy for each hardware configuration and benchmark class.

4.3 Memory access patterns

The Amdahl's law is not sufficient to explain the difference of computation times between the two models. As for the communication patterns, the memory access patterns are different for the two models. To go further, we examine the parallel efficiency, which is defined as the measured speedup divided by the number of CPUs, on the Pn part of the computation (part that can be parallelized with OpenMP). Note that Pn part corresponds to the same code for both models. However loop nests are executed differently by each model. Figure 10 shows the parallel efficiency on the Pn part for the class A and class B benchmarks with WH2 nodes. FT and MG exhibits similar behavior: the parallel efficiency is approximately the same for the two approaches. For FT, it increases with the number of nodes for small dataset sizes, which correspond to a better use of caches or remains constant if the dataset size is too large for the cache hierarchy. For MG, the parallel efficiency increases when the number of nodes increases above a threshold. For CG and LU, the MPI version has better parallel efficiency. For CG, the parallel efficiency increases with the number of nodes for large dataset sizes and reaches a maximum with 4 nodes and then decreases: the dataset size is too small for a full exploitation of the cache hierarchy. This last situation exists for LU for class A and class B. The decomposition of the computation time for the hybrid version shows that two parameters govern its performance: 1) the time spent on the core of the loop nests and 2) the overhead of thread management. Although, the hybrid version reaches lower computation time on loop cores for some benchmarks, it has higher computation time for the whole loop nests due to the overhead of the thread management.

At this point, we should state that there is no reason why the hybrid approach performs better on some loop nests except because loop nest optimizations used in the original MPI version do not match well the Power3 memory hierarchy.

Conversely, we should note that the OpenMP parallelization has a severe limitation compared to the MPI version: OpenMP cannot express multi-dimensional blocking when using the easiest parallelization approach or simple loop optimizations. Such kinds of patterns require a complete rewriting of the loop nest (loop fusion), which means a time consuming programming effort.

5 Related works

The performance of the IBM SP with Winter Hawk II nodes is presented in [4]. The author compares this architecture to a AlphaServer SC for several kernels and applications that do not include the NAS benchmarks. The paper does not compare unified and hybrid versions.

The performance issues of the NAS benchmarks have been presented in [2]. Especially, Wong et al. have examined the architectural requirements and scalability. This work is restricted to uniprocessor nodes and compares performance of a cluster of workstations and the SGI Origin 2000 machine.

The performance of Message Passing has been detailed in [5] for a cluster of SUN SMPs. Authors note that the memory hierarchy and especially the population of the memory banks have a significant impact on the performance. The paper does not address the issue of hybrid programming models.

A comparison of shared memory and message passing

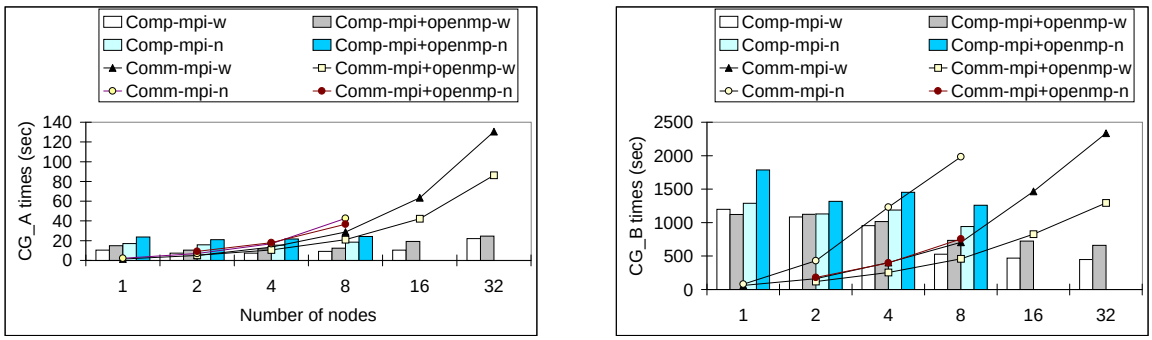

Figure 9. The execution time breakdown for CG according to the number of 4-way nodes for the MPI and MPI+OpenMP versions. The left part corresponds to the Class A and the right part to the class B. w (resp. n) stands for WH2 (resp. NH1) nodes. The figures breakdown the total execution time, summed across all the processors, into computation times (shown as bars) and communication time (lines). The left scale is the same for computation and communication times.

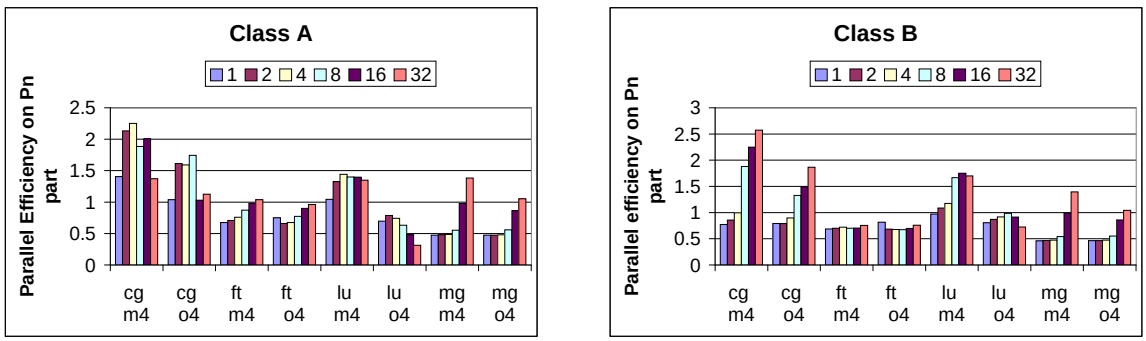

Figure 10. Parallel efficiency on the Pn part according to the number of nodes for the Class A (left part) and the Class B benchmarks (right part). m4 tag corresponds to the unified MPI version with 4-way WH2 nodes. o4 tag corresponds to the hybrid version with 4-way WH2 nodes.

memory models has been previously presented in [6]. The paper focuses on classical parallel architectures and does not examine the CLUMP issues.

Several papers in the literature present the implementation of applications using a hybrid model like [7]. None of them provides a deep understanding of why this model is better or worst than a unified one.

Other programming models have been designed for the CLUMPs. For example, the shared virtual memory environments (DSVM) provide another alternative to unify the memory model, as presented in [8] [9] [10][11]. Recently, OpenMP [12] has been implemented on a cluster of SMPs on top of the Treadmark DSM system. As far as we know, there is no comparison between these unified approaches and some hybrid ones.

6 Concluding remarks

Based on the analysis of the NAS benchmarks running on two different IBM SP systems, we have demonstrated that the choice between unified or hybrid programming models is not trivial from the MPI existing codes.

Several parameters must be considered. The first one is the level of shared memory parallelization achievable for

the loop nests of the original MPI program. From the original MPI program, it is difficult to evaluate if the incremental approach that parallelizes the loop nests with OpenMP (including specific optimizations) is sufficient to improve performance or if it is needed to reconsider the parallelization of the application from scratch.

The second parameter concerns the communication cost. For the same number of processors, this cost depends on how the communication patterns of the application match the communication architecture of the cluster. Generally, clusters of multiprocessors perform better with unified MPI approach for latency sensitive programs and worse for bandwidth sensitive programs.

A third parameter is the memory access patterns used by each approach. Whereas MPI programming allows to express multi-dimensional blocking, it is not natural for OpenMP to do so. To perform the same memory patterns, loop nests must be rewritten which may be complex for long loop core or deep loop nests. Efforts for improving OpenMP may provide a solution to this problem. Other programming languages like Co-array [13] may also be considered as alternative to OpenMP for the hybrid programming or even for the unified programming.

The last parameter is the performance balance of the main components (CPUs, Memory, Network) of the CLUMP. For the same network performance, fast processors may reduce the computation time compared to slow processors up to the point that communication performance becomes significant. In that case the communication performance allows to select the right model. In the other case, MPI seems to be always the best.

Note that that our results and analysis only concerns a particular approach of hybrid programming, which consists in loop level parallelization with a reasonable programming effort after profiling to select the loop nests to parallelize. Other hybrid approaches may lead to different results and analysis conclusions.

We believe that selecting the appropriate model is a general problem that is not specific to the applications analyzed in the paper. First, NAS benchmarks exhibit a large set of communication, memory reference and computation patterns. Except some irregular ones, real applications are likely to encompass similar patterns. Second, we have demonstrated that several important parameters distinguish the performance of both approaches. Although the relative impact of these parameters is application dependent, they exist in any parallel application.

Future work will include performing the same analysis for other clusters of multiprocessors like Compaq SC, comparing with the other hybrid approaches and especially the coarse grain parallelization and deriving a general performance model that may help choosing between unified and hybrid models according to the characteristics of the architecture and the application.

7 Acknowledgments

Alain Quere and Eric Boyer (CINES in Montpellier) allowed the first set of measurements on the SP3 with NH1 nodes as a single user with ”user space” communications. Olivier Hess, from IBM Montpellier, was very helpful for all technical problems. John Levesque and its group in the IBM Advanced Computing Technology Center allowed the set of measurements on the SP3 with WH2 nodes.

References

[1] F. Cappello and O. Richard. Intra-node parallelization of MPI programs with OpenMP *Report TR-CAP-9901*, URL: http://www.lri.fr/~fci/goinfreWWW/1196.ps.gz

[2] F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, D. T. Wu, and D. E. Culler. Architectural requirements and scalability of the NAS parallel benchmarks. In *Proceedings of Supercomputing'99*, 1999.

[3] F. Cappello, O. Richard O and D. Etiemble. Investigating the performance of two programming models for clusters of SMP PCs In *Proc. of the 6th IEEE Symp. on High-Performance Computer Architecture (HPCA-6)*. January 2000.

[4] P. H. Worley. Performance Evaluation of the IBM SP and Compaq AlphaServer SC. In *Proc. of the International Conference on Supercomputing*, ICS, May 2000

[5] Steven S. Lumetta, Alan Mainwaring, and David E. Culler. Multi-protocol active messages on a cluster of SMPs. In *SC'97: High Performance Networking and Computing: Proceedings of the 1997 ACM/IEEE SC97 Conference: November 15–21, 1997, San Jose, California, USA.*, 1997.

[6] H. Shan and J. P. Singh. A Comparison of MPI, SHMEM, and Cache-Coherent Shared Address Space Programming Models on the SGI Origin2000. In *Proc. of the International Conference on Supercomputing*, ICS, June 1999

[7] S. W. Bova, C. P. Breshears, C. Cuicchi, Z. Demirbilek, H. A. Gapp. Dual-level Parallel Analysis of Harbor Wave Response Using MPI and OpenMP In *The International Journal of High Performance Computing Applications*, Spring 2000.

[8] D. J. Scales, K. Gharachorloo, and A. Aggarwal. Fine-grain software distributed shared memory on

SMP clusters. In *Proc. of the 4th IEEE Symp. on High-Performance Computer Architecture (HPCA-4)*, February 1998.

[9] R. Stets, S. Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and Michael Scott. Cashmere-2L: Software coherent shared memory on a clustered remote-write network. In *Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16)*, October 1997.

[10] R. Samanta, A. Bilas, L. Iftode, and J. P. Singh. Home-based SVM protocols for SMP clusters: Design and performance. In *Proc. of the 4th IEEE Symp. on High-Performance Computer Architecture (HPCA-4)*, February 1998.

[11] Andrew Erlichson, Neal Nuckolls, Greg Chesson, and John Hennessy. SoftFLASH: Analyzing the performance of clustered distributed virtual shared memory. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 210–220, 1996.

[12] Charlie Hu Honghui Lu Alan Cox and Willy Zwaenepoel. OpenMP for Networks of SMPs. In *Proc. of the Second Merged Symp. IPPS/SPDP 99*, 1999.

[13] R. W. Numrich and J. K. Reid. Co-Array Fortran for parallel programming *Research report RAL-TR-1998-060*, Department for Computation and Information, Rutherford Appleton Laboratory, Oxon, UK, August 1998