Performance of the NAS Benchmarks on a Cluster of SMP PCs Using a Parallelization of the MPI Programs with OpenMP

Franck Cappello, Olivier Richard and Daniel Etiemble

LRI, Université Paris-Sud, 91405 Orsay, France
Email fci@lri.fr

Abstract. The availability of multiprocessors and high performance networks offer the opportunity to build CLUMP's (Cluster of Multiprocessors) and use them as parallel computing platforms. The main distinctive feature of the CLUMP architecture over the usual parallel computers is its hybrid memory model (message passing between the nodes and shared memory inside the nodes). To be largely used, the CLUMPs must be able to execute the existing programs with few modifications. We investigate the performance of a programming approach based on the MPI for inter-multiprocessor communications and OpenMP standards for intra-multiprocessor exchanges. The approach consists in the intra-node parallelization of the MPI programs with an OpenMP directive based parallel compiler. The paper details the approach in the context of the biprocessor PC CLUMPs and presents a performance evaluation for the NAS parallel benchmarks.

1 Introduction

Many computing centers are now equipped with parallel platforms using PCs as computing nodes and a high performance network like Myrinet as the interconnection network. Today, most of the microprocessors (and especially the Pentium II family) and their chip-sets are designed to easily build multiprocessors. Two-way multiprocessor PCs are now available as workstation PCs. They are about 1.5 times more expensive than uniprocessor PCs with the same microprocessor and memory size. Four-way PCs are also available for servers, but they are far more expensive. In this paper, we only consider 2-way multiprocessor PCs.

The main potential interest of multiprocessor nodes lies in the reduced number of the network connections for a given number of processors in a parallel platform. The speed-up of biprocessor nodes over single processor nodes promise identical performance for the parallel platforms composed of n single processor nodes or of n/2 biprocessor nodes. Since the cost of the network connection is a large portion of the cost of a PC node in a parallel platform, it becomes appealing to use multiprocessor PCs as the nodes for a parallel platform. Instead of reducing the global cost of a parallel platform, the multiprocessor nodes could be used
to increase the global platform performance for a constant number of nodes. This approach increases the parallel platform cost. In both cases (constant number of processors or constant number of nodes) an intra-node speed-up study must be done to evaluate the actual speed-up of multiprocessor under uniprocessor nodes.

1.1 Programming and Testing the CLUMPs

Several ways exist to program the CLUMPs and measure their intra-node speed-up. Networks of multiprocessor PCs present an hybrid hardware memory model: message passing between nodes and shared memory inside each node and conform to the term of CLUMP (CLUster of MultiProcessors). In [?] and [?] the authors present respectively a taxonomy and a classification of the programming models for the CLUMPs. We may classify two main approaches by distinguishing the programmer view of the memory in a CLUMP:

- a single memory model (SMM) or
- an hydride memory model (HMM)

Message Passing SMM In [?], [?] and [?] the authors have developed a version of their message passing libraries to work in SMP platforms. A model to program the CLUMP through a small kernel of collective communication and computation primitives is described in [?]. The communication primitives are implemented on top of a message passing library and a SMP Library. Algorithms implemented with this model are programmed as a succession of collective computations and collective communications.

Shared Memory SMM Shared Virtual Memory environments provide the opportunity to program the CLUMPs with the shared memory model. Several projects have already published some design and performance results about the CLUMPs: [?], Shasta [?], Cashmere-2L [?] and SoftFLASH [?]. As for the uniprocessor platforms, the performance of DVSM mainly relies on the protocol efficiency. Recently, OpenMP [?] has been implemented on a network of workstations on top of the Treadmark DSM system. It provides a very convenient way to program the distributed memory architectures. Its performance relies on the shared virtual memory software.

HMM HMM has been the first approach used to program the CLUMPs. Original works include programming the PVPs (Parallel Vector Processor) where Vector Supercomputers are interconnected with a high speed network to form a parallel architecture. Recently the NAS parallel benchmarks repository has published the performance of a Power Challenge Array on a selection of the NPB2 benchmark programs [?]. A hybrid shared memory/distributed memory programming model for the CLUMPs is presented in [?]. Intra-node computation utilizes a multi-threaded programming style. Inter-node programming is based on message passing and remote memory operations.
Portability Requirement  Both for SMM and HMM, the portability of code is a main issue. Moving from traditional supercomputers (vector machine) to shared memory or message passing parallel computers has already forced the users to reconsider their application programs. Moving from single processor nodes to multiprocessor nodes in the parallel architectures may also need some effort. Thus, a methodology to program the CLUMPs should seriously consider the portability and provides an approach compliant with a wide variety of CLUMP configurations.

1.2 A Method Based on MPI and OpenMP

In this paper, we investigate the performance of an approach which primary aims to provide portable codes with a reduced effort in the context of the HMM approach. The approach uses OpenMP for shared memory parallelism inside the nodes and MPI for message passing between nodes.

MPI is one of the most popular libraries for message passing for multi-PC parallel platforms. A lot of applications have been written or ported for the message passing paradigm. A methodology proposed to program the CLUMPs from MPI should provide a way to execute the MPI existing programs written for uniprocessor nodes.

OpenMP derives from the ANSI X3H5 standard effort. It is a set of compiler directives and runtime library routines that extend a sequential programming language to express shared memory parallelism. The language is extended by a collection of compiler directives, library routines, and environment variables. OpenMP conforms to the SPMD programming paradigm. The OpenMP API uses the fork-join model of parallel execution.

2 OpenMP Parallelization of the NAS NPB 2.3 MPI Programs

2.1 The Basic Methodology

Parallelizing an application for the message passing following the SPMD paradigm often produces a program with the typical layout presented in figure 1. The program starts by initializing the communication system. Then it performs some local computations and calls some communication subroutines to split the data sets among the nodes participating to the application. The program continues executing a main block typically containing a loop nest. The main block is designed for parallel execution. The body of the loop nest can be described as a succession of three sections: local computations, communications, synchronization. The final part of the program gathers individual partial results and computes the final result.

The applications written from MPI programs come as one executable file running on each node of the parallel platform. Within each node, the program is executed inside a process. Parallelizing the program executed on each node
Program CG
-- call initialize_mpi
-- call setup_proc_info(num_procs, ...)
call setup_submatrix_info(12npcols, ...)
-- do it = 1, niter
-- call conj_grad(colidx, ...)
-- do i = 1, 12npcols
-- call mpi_recv(norm_temp2, ...)
call mpi_send(norm_temp1, ...)
call mpi_wait(request, ...)
-- enddo
--
call mpi_finalize(ierr)
--
end

Subroutine conj_grad(colidx, ...)
do i = 1, 12npcols
-- call mpi_recv(rho, ...)
call mpi_send(sum, ...)
call mpi_wait(request, ...)
enddo

!$OMP PARALLEL PRIVATE(k, sum)
!$OMP DO
-- do j = 1, lastrow-firstrow+1
-- sum = 0.0d0
-- do k = rowstr(j), rowstr(j+1)-1
-- sum = sum + a(k)'*p(colidx(k))
-- enddo
-- w(j) = sum
-- enddo
!$OMP END DO
!$OMP END PARALLEL
-- do i = 12npcols, 1, -1
-- call mpi_recv(!)
call mpi_send(!)
call mpi_wait(request, ...)
enddo
-- return
end

---

Fig. 1. Parallelizing the MPI Code. The main loop nest of the CG code calls the
conj-grad subroutine and contains some communication calls. The computation loop nest of
conj-grad is parallelized for intra-node execution using the shared memory paradigm.
leads to parallelize the main block. This block often encompasses an iterative
calculus with inter-iteration dependencies. So intra-process parallelization could
not be generally attempted at this level. For most of the NAS programs, we
have parallelized the computational section of the main block. Figure 1 presents
this hierarchy of parallelism: message passing between MPI processes and intra-
ode parallelism within each MPI process. The intra-node parallel execution is
performed by a group of threads within the same process. The parallelization
directives are simply applied to parallelizable loop nests of the original MPI code.
From other nodes point of view, a biprocessor node with an intra node multi-
threaded execution of a fraction of the MPI code behaves like a uniprocessor node.

The application may exhibit a large number of loop nests and subroutine
calls. It can be difficult to discover manually (reading the text source) which
loop nests worth to be parallelized. Table 1 gives the cost of the main parallel
operations for several nodes of our platform.

<table>
<thead>
<tr>
<th></th>
<th>Pentium Pro 200</th>
<th>Pentium II 300</th>
<th>Pentium II 400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fork-Join (Parallel DO)</td>
<td>5 us</td>
<td>3.5 us</td>
<td>3 us</td>
</tr>
<tr>
<td>Lock (Critical)</td>
<td>1.66 us</td>
<td>1.45 us</td>
<td>1.4 us</td>
</tr>
<tr>
<td>Barrier</td>
<td>1.36 us</td>
<td>1.33 us</td>
<td>1 us</td>
</tr>
</tbody>
</table>

Table 1. The Cost of the Parallel Operations of OpenMP on our Platform

Figure 2 presents a framework for selecting loop nests to parallelize.

Fig. 2. The Parallelization Framework
The framework begins with the MPI source file. The profiled execution allows to discover the most expensive loop nests. The next step inserts OpenMP directives in the code in order to obtain a candidate MPI code for SMP nodes. Then the program is compiled and run on the platform. At the end of the execution, the result correctness and the speed-up must be checked. If the result is wrong or if the speed-up is less than one, the parallelization must be refined or removed. This process may require several iterations in order to provide an efficient and correct code.

A conceptual limit of the approach comes from the way the shared memory parallelization is applied to the message passing programs. With this method, we should not expect a local speed-up close to the speedup that can be obtained by directly parallelizing a sequential program. In contrast with a shared memory program directly derived from a sequential program, there is a lot of substantial work that cannot be parallelized. More precisely, the speed-up is not only bounded by the local sequential part contribution to the local execution time (Amdahl's law) but also by the communication and synchronization contributions to the local execution time.

Another limit of the approach comes from the way the OpenMP compiler distributes loop iterations among threads (e.g. processors). Our PGI compiler splits loop iterations in sections where \( n \) is the number of threads. Each thread computes one section. We cannot control the loop splitting into sections (sections are always made of contiguous iterations) or the distribution of section among threads (processors). Uncontrolled distribution of sections may lead to higher execution time due to inappropriate cache fill at the interfaces between parallel and sequential parts.

### 2.2 Example: Intra Node Parallelization of MG Benchmark

MG uses a multi-grid algorithm to obtain an approximate solution of a three-dimensional scalar Poisson equation. Figure 3 presents the calling hierarchy of parallelized subroutines. The local cost doesn't include the cost of subroutines called by the current subroutine. The total cost includes the cost of routines called by the current subroutine. Values (\% of the total execution time) must be understood by considering the very long initialization procedure. The initialization time is approximately 66\% of the total execution time. Some time consuming subroutines cannot be parallelized (zran, comm1p, norm2a3, comm3) because they contain communication calls or inter-iteration dependencies. Their combined execution time exceeds 30\% of the total execution time.

`mg3p` is not parallelizable because it calls some communication subroutines: zero3, rpr3, interp, resid and psinv contain loop nests that are parallelizable at the outermost loop.
3 Performance

3.1 Platform Hardware and Software

Our platform contains a Myrinet network with four ports. We use two types of biprocessor nodes: Pentium II 300 MHz and Pentium II 400 MHz. Each Myrinet PCI interface has a 1MB local memory.

The software environment includes Linux 2.0.33, BIP 0.94c version of MPI library, F77 PGI 1.7 programming environment and Linux Pthread library. With BIP on Myrinet connected PCs, we have a latency of 5us and a bandwidth of 1 Gbit/s. MPI-BIP reaches 20 us (latency) and 1 Gb/s (bandwidth). All benchmarks have been compiled with the c2, unroll and P6 options.

3.2 Uniprocessor PC Versus Biprocessor PC

In this section, we compare a parallel platform based on uniprocessor nodes with a parallel platform based on biprocessor nodes. We compare these platforms for four different configurations: 1 node, 2 nodes, 4 nodes and 8 nodes. All the measurements are made with 400 MHz Pentium II nodes, except for figure 5 that compare performance between 300 MHz Pentium II nodes and 400 MHz Pentium II nodes.
Figure 4 presents the speedup of the bprocessor based CLUMPs over the single processor based platform for a constant number of 400 MHz Pentium II nodes.

\begin{center}
\begin{figure}
\centering
\includegraphics[width=\textwidth]{speedup.png}
\caption{Intra-Node Speed-up of the Bprocessors over Uniprocessors for the NPB 2.3 Benchmarks and for the Same Number of Nodes}
\end{figure}
\end{center}

The speed-up evolves with the number of nodes in the CLUMPs following one of two trends: it remains constant or it decreases. We have examined the breakdowns of the execution time of some key NAS programs. The analysis shows that the communications, the unparallelized computation loops and the cost of the local parallel execution (mainly bus conflicts) are the most significant bottlenecks.

Figure 5 compares the speed-up between 400 MHz nodes and 300 MHz nodes for the same number of nodes. It shows that Pentium II 400 bprocessors provide a more constant speed-up across the different benchmarks. They are also more sensitive to the communication cost.

Figure 6 presents the performance of the bprocessor based CLUMP against the single processor based platform for a constant number of 400 MHz processors.

We must consider the cost/ performance ratio of the bprocessor nodes against the uniprocessor nodes. A bprocessor based platform requires half the connections of a uniprocessor based platform. In a typical uniprocessor PC based par-
4 CLUMPs Versus Proprietary Supercomputers

The previous sections compared the relative performance of 2-way multiprocessor PC based CLUMPs with uniprocessor PC based CLUMPs. Users are mainly interested by the absolute performance. In this section, we compare the performance of uniprocessor and biprocessor CLUMPs with the performance of proprietary parallel supercomputers for the same number of nodes.
Figure 6. *Global Speed-up of the Biprocessors over Uniprocessors for the NPB 2.3 Benchmarks and for the Same Number of Processors*

Figure 7 presents the performance 8 nodes platforms including the uniprocessor based CLUMP (called NOW), the biprocessor based CLUMP (called CLUMP) and the following parallel supercomputers for 8 nodes platforms:

- the SGI/CRAY T3E 900
- the SGI/CRAY T3E 1200
- the IBM SP2 with 66 MHz Power 2 processors
- the SGI Origin 2000 with 195 MHz processors
- the HP/Convex Exemplar SPF 2000
- the SUN Ultra Enterprise 4000

8 nodes CLUMPs use 400 MHz Pentium II processors. The figures for proprietary parallel computers come from the NAS NPB2.3 repository [?].

Figure 7 shows that performance of 8 nodes CLUMPS and performance of 8 nodes supercomputers have the same order of magnitude. In any case, no platform outperforms the other ones.

According to our measures, it is obvious that clusters of biprocessor PCs are a very cost effective solution for parallel computers, at least for small configurations. The results must be confirmed on actual applications.
5 Conclusion

In this paper, we have investigated the performance of a method based on HMM (hybrid memory model) for programming clusters of multiprocessors. This method requires the programmer to deal both with the message passing and the shared memory paradigms. It consists in the intra-node parallelization of the MPI programs by using an OpenMP directives based parallel compiler. We have presented a framework to select the loop to parallelize.

The intra-node speed-up for the NAS parallel benchmark is significantly lower than 2 (between 1.2 and 1.8 depending of the program for 400 MHz Pentium II nodes) except for EP. Moreover the speed-up can decrease with the number of nodes depending of the benchmark program features.

Although the method provides variable local speed-ups, it is much more practical than the manual parallelization approach to program the CLUMPs.

Finally, from the cost/performance point of view, multiprocessors are a competitive alternative to uniprocessors as nodes of a PC based platform. This PC based platform is also a very cost effective alternative to proprietary parallel supercomputers, at least for small configurations.

Fig. 7. Performance of the CLUMP's and Some Parallel Supercomputer for the NAS NPB 2.3 Benchmark Suite with 8 Nodes