Communication Policies Performance: A Case Study

Daniele Tessera
Dipartimento di Informatica e Sistemistica
Università degli Studi di Pavia
Via Ferrata, 1
I-27100 Pavia, Italy
tessera@gilda.unipv.it

Anshu Dubey
Astronomy & Astrophysics
University of Chicago
5640 S. Ellis Ave
Chicago, IL 60637
dubey@tagore.uchicago.edu

Abstract

Communication activities are one of the most influential factors for the performance of parallel applications, and usually limit the number of processors that can be profitably allocated. Two components usually determine the communication cost of a parallel algorithm. One is the volume and range of data transfer, which is inherent to a specific algorithm. The other is the choice of communication strategy, e.g., point-to-point versus collective exchanges, blocking versus non blocking protocols, which has impact on setup costs, overheads due to buffering and/or contentions. Knowledge of comparative performance of different strategies can be very useful for a user if several choices are available. In this article we present the results of a study to determine the best approach to high volume, long range communications within the frame work of multidimensional FFT algorithm. We have investigated five widely used communication strategies, available in the MPI standard, which have identical data volumes and range of communications. We also present a systematic analysis of the causes of performance differences, with analytical models supporting the experimental evidence.

1 Introduction

Real life applications, simulating complex chemical/physical phenomena, demand performance levels that only parallel machines can provide. Good overall performance is crucial for these applications. Performance optimization of parallel applications is a tough job since it is influenced by a large number of factors (e.g., parallelization strategies, communication policies, interaction of application code with hardware and software components of the parallel machines). Programmers have to choose the most appropriate parallelization and data distribution strategies (e.g., data parallel techniques versus functional parallelism) together with the most appropriate communication policies to design and implement their applications. They must consider the cost of communication in parallel applications, and the ways of minimizing it [4, 5, 12]. In addition there is the issue of portability of the code performance if it is likely to be used on more than one platform. In depth knowledge of a specific parallel machine allows programmers to fine tune their applications for it, but that does not guarantee their performance on different architectures or when allocating different numbers of processors. Hence performance diagnosis and debugging is more an art than a science. A systematic approach towards the analysis of the performance achieved by parallel applications, relating the performance to specific code regions of the applications themselves, is therefore very important.

In what follows we describe a performance analysis study aimed at evaluating the performance of a widely used computation kernel, i.e., a multidimensional Fast Fourier Transform code. We have analyzed the FFT kernel of a parallel three dimensional magneto hydrodynamics code [7] written at the University of Chicago for the NASA HPCC initiative. The FFT kernel accounted for 99% of the parallelism and 90% of the computation in the code, hence its performance was critical to the overall performance of the code. The other reason for choosing the FFT in this study is that it can be designed with a variety of different communication models while keeping the total volume of data transferred and the range of transfer constant. Thus the difference in performance comes from the inherent characteristics of the communication model used. To determine the best approach, we created five different versions of parallel FFT. Two of them use point-to-point blocking communication, one uses point-to-point non-blocking communication, one has a set of collective communications and the last one uses a single collective operation. MPI is used for all the communications, and the experimentation was done on IBM Sp2. The performances were analyzed using Medea [6], a tool for workload characterization and perfor-

mance diagnosis of parallel applications developed at the University of Pavia.

The outcome of this experiment has some surprising deviations from theoretical expectations. The non-blocking version turns out to be the worst performing, despite the fact the IBM-SP2 has separate communication processors. The version with single collective operation is the fastest, where normally one would have expected a lot of contention and hence degradation in performance. An in-depth analysis of the performance helps pin-point the causes for this unexpected behavior. We believe that the results of this work should also be relevant for applications that have large volume of long range structured data transfer.

2 Redistribution Strategies

This section describes the five kernels used in this work. We define a three dimensional problem N^3 distributed over P processors with slabwise domain decomposition. In this distribution, one dimension of the FFT is parallel at a time. The local dimensions are transformed first, followed by a distributed transpose, followed by a transform in the third dimension. The initial data distribution, the final data distribution and the volume and range of data transfer are identical for all the five kernels. The distributed transpose is essentially a complete exchange algorithm [3], where every processor sends data to all the processors. It can be done with a single complete exchange step, where all planes are processed together for communication, or in steps, where each plane initiates its own complete exchange.

The first four kernels process one plane at a time. The first one, *The Replace Kernel*, uses **send_receive_replace**, a point-to-point blocking protocol that has the same buffer for both send and receive. The contents of the buffer are replaced by the received data after the send is completed. The second kernel, *The Standard Kernel* uses the standard **send** and **receive**, also a point-to-point blocking protocol, where send and receive can be scheduled separately and can use different buffers. The number of communication calls issued is twice that of the first kernel. The third kernel, *The Overlap Kernel*, uses point-to-point non-blocking protocol. By scheduling them carefully it is possible to overlap communication with computation. The fourth kernel *The Oneplane Kernel* uses collective operation **all_to_all** for each plane separately. The last kernel, the *Allplanes Kernel*, uses the collective communication for all the planes at once. We have not included point-to-point communication kernels that process all planes together, since they sometimes cause deadlocks for large data sizes.

3 Experimental Environment

The FFT kernels, mentioned in section 2, have been instrumented and executed on the IBM Sp2 [2] of the Maui High Performance Computing Center (machine details can be browsed at http://www.mhpcc.edu). These kernels have been instrumented by inserting statements into its source code which when executed record some information (e.g., a unique id, the wallclock time, and the processor id) [11]. This information is related to specific activities performed by some processors like communication statements, execution of either specific functions or specific portions of application code. Hence, during the execution of the application, measurements about its behavior are collected for post-mortem analysis. Our FFT kernels have been instrumented by means of both the monitoring facilities provided by mpich [9] and ad hoc monitoring software. Since the monitoring code is executed concurrently with the application code, it is important to minimize its influence on the application performance. The influence can come in many ways, such as use of CPU cycles and memory spaces or perturbing the application performance with some side effects (e.g., cache misses). All this makes it crucial to carefully choose the interesting behaviors/activities of the application which have to be investigated. In our study we have monitored the communication activities and a few routines (i.e., the main FFT solver together with some pure computations functions from the ESSL [1] library). Special care was taken to identify the initial costs (e.g., initialization of trigonometric tables and working arrays, synchronizations of the allocated processors) which have been discarded in the performance analysis. After instrumenting the kernels we ran a set of experiments, varying both the number of allocated processors and the problem size. The measurements from these experiments were then analyzed with Medea, tool for workload characterization and performance diagnosis of parallel applications. We also derived the analytical models from these performance figures, that could relate the performance to the application parameters.

4 Performance Results

In this section we discuss the performance of the various FFT kernels on an IBM Sp2 machine. As testbed problem we have chosen to transform a grid of $128 \times 128 \times 128$ points (real values) varying the number of processors from 1 to 64. Since the FFT computing kernel resembles iterative algorithms, we have collected the timing accounted for each forward and one inverse transforms. When analyzing the behavior of such kernels, startup and synchronization costs, which are negligible in production runs, may become significant. For this reason we have taken special care to discard all the startup costs due to both initialization of trigonomet-

ric tables (for computing local one-two dimensional FFT) and synchronization delays among the allocated processors. As an overview of the performance, figure 1 shows the time required for performing a complete step as a function of the number of allocated processors. All the kernels scale pretty well up to 32 processors. To understand the relative per-

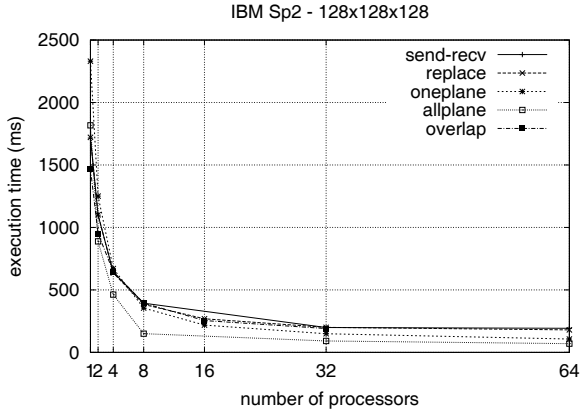


Figure 1. Execution times, as a function of the number of allocated processors, for each FFT kernel.

formances and their causes, the speedup saturation region, where there the communication starts dominating, is likely to provide more insights. Figure 2 shows the time spent in communication activities, as a function of the number of allocated processors, for each of the kernels. Note that by communication time we do not mean time to physically deliver the message to the destination, but the time spent by the processors to handle the communications (that is, performing MPI functions).

The figure clearly shows an anomaly between expected behavior and exhibited behavior. The overlap kernel, which was expected to be the fastest due to communication and computation overlap, consistently performs the worst. This is despite there being a separate communication processors to minimize the communication time. The CPU is required only to setup the communication, leaving its management to the dedicated coprocessor. The best performing kernel turns out to be the one with a single `MPI_Alltoall` call. Given the amount of flexibility available in scheduling the data transfers of the other ones, the outcome was expected to be different.

The message passing paradigm requires several tasks to be performed by both the sender/receiver processors. The sender has to copy the message from the program memory space to system buffers. The issuing process is blocked until the message is either buffered or sent. This can result in delay if the system buffers are full. For small messages eager

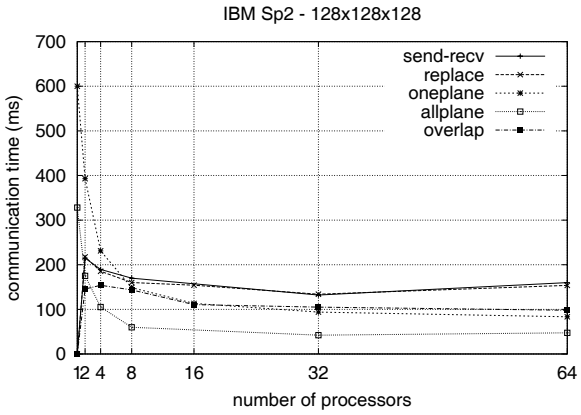
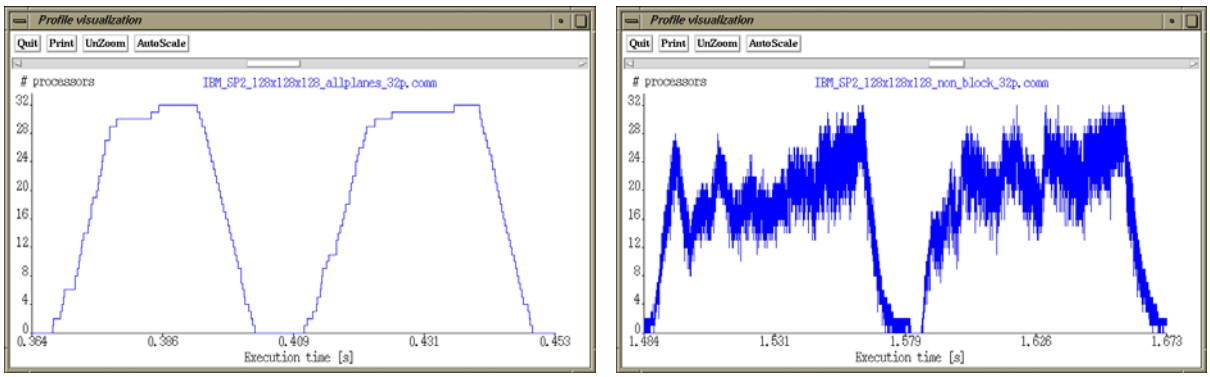


Figure 2. Communication times, as a function of the number of allocated processors, for each FFT kernels.

protocols, which do not require special care from the sender, are usually used. However, real applications have too large volumes of data to allow communication policies based on eager protocols. They typically use rendezvous protocols which need special handling of outcoming messages. The sender has to ensure that there is enough buffer space in the receiver processor before sending the data. Hence it first sends a system message (usually named ”envelope”) to the destination for reserving the buffer. The receiver can make the reservation or defer the sending until it has enough space. When the sender receives the acknowledgement, it sends the message itself (usually referred as the payload). The receiver has to move data from system to application buffer after the message arrives. In such a scenario, when a lot of messages come in to a processor, it quickly consumes all the buffering space and starts to defer communications. It is for this reason that the cost of communication is linked to the number of MPI calls. Even in the ideal case with no system buffer contentions, the communication software has to schedule few system tasks for each MPI call which manage the communication. These are executed concurrently with the application and their impact on the overall performance may not be negligible. The situation gets worse with non blocking calls which are costlier in their setup time and need additional buffering.

Figure 3 gives further insights into the kernel behavior by representing the communication profiles, i.e., the number of processors performing communication activities as a function of the execution times, of the Allplanes and Overlap kernels (the best and the worst performing ones). The figure shows that processors are somewhat synchronized in the collective communication, whereas in the Overlap kernel there are a lot of fluctuations. This is because collective



(a) *(b)*

Figure 3. Communication profiles for Allplane *(a)* and Overlap *(b)* kernels solving a grid of 128×128×128 points with 32 processors.

calls, by their very nature cause all processors to catch up with each other. After first collective call the processors are likely to stay in greater sync, and therefore reduce wait for subsequent calls. There is no such side effect in the non-blocking calls. They actually make the situation worse because of greater overheads they carry. Even when sophisticated hardware allows physical overlap of computation and communication activities, there is very little real benefit. For example, on the IBM Sp2 the communication processor is able to address the operating system buffers and to move data from/to them directly by using DMA techniques. Hence, theoretically the main processor may execute the application code concurrently with the communication activities. In truth, memory contentions limit the real benefits in terms of overall execution time. Up to 70% of the memory bus bandwidth may be used up by the communication processor, leaving very limited scope for concurrent processing by the main processor. Some curious results come up because of this complexity. For instance, if we compare the overall performance in Fig. 1 with the corresponding communication times in Fig. 2, we find that the difference in the overall performance of the Overlap and Standard kernels cannot be explained by the communication time alone. That difference is due to the hidden cost of non blocking communication, where the control is returned to the application before the actual physical data transfer is complete. This data transfer steals memory bandwidth from the main processor, thus the time of actual data transfer gets added to the computation rather than communication. As a result the time taken by the ESSL [1] routines to compute the local FFT's is increased from $1.8ms$ (scalar FFT) and $2.9ms$ (two dimensional FFT) to $2.4ms$ and $4.1ms$ respectively. There is also the additional cost of data packing to and from the extra buffers used to manage the communications. The impact of the hidden cost of communication on computation time

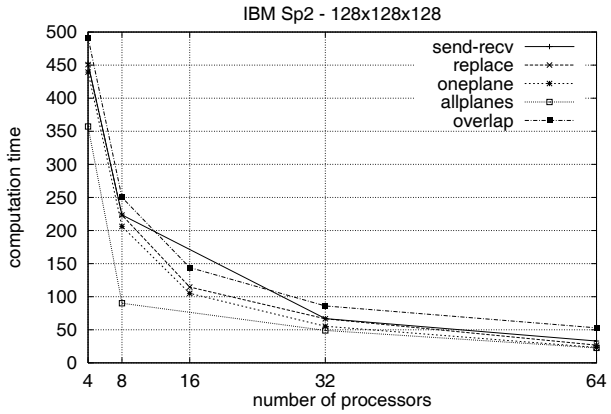


Figure 4. Computation times, as a function of both communication strategy and number of allocated processors.

is shown in Fig. 4 for all the kernels. This figure clearly shows that there is a direct correlation between the number of MPI calls and computational performance. Allplanes kernel, which requires only two MPI_Alltoall calls for each step (one for the forward FFT and the other for its backward counterpart) shows the best computation time. Moreover transferring all the data with a single communication call reduces the impact of setup/latency times allowing the communication hardware to achieve a good sustained bandwidth, as shown in Fig. 2.

5 Performance Modeling

To substantiate the observed behavior of different FFT kernels, we use analytical models of their communication and computation times. These models can help in determin-

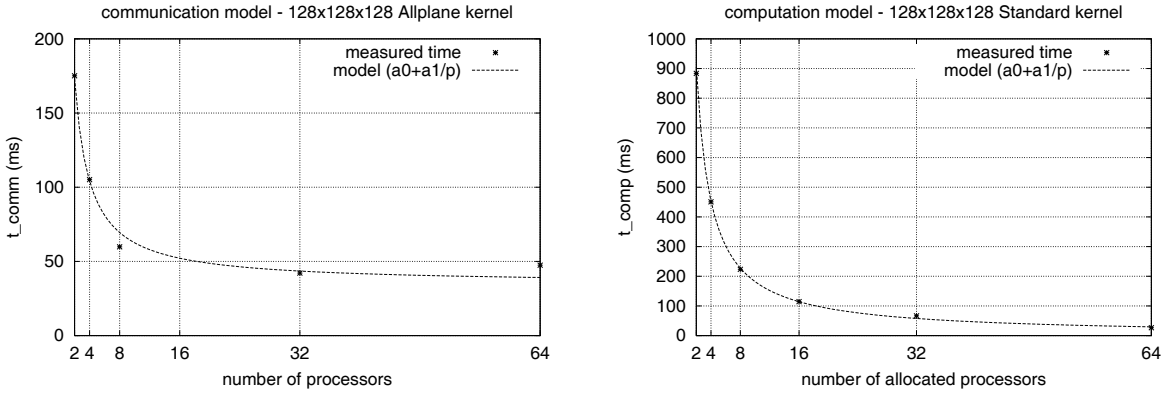


Figure 5. Computation model for Standard kernel *(a)* and communication model for Allplane kernel *(b)* together with the measured times.

ing performance and scalability of different kernels. We use the following two models:

$$t(p) = a_0 + \frac{a1}{p} \tag{1}$$

$$t(p) = a_0 + \frac{a1}{p} + a_2 p \tag{2}$$

where:

t represents the (communication/computation) time;

p represents the number of processors allocated;

a_0, a_1, and a_2 are the parameters of the models.

Model 1 is the Amdahl model applied to computation and communication times while model 2 is its extension which takes into account the overheads due to the management of the allocated processors. Note that even if the analytical expression of model 1 is similar to the Hockney model [10], shown in equation 3, it has a different meaning.

$$t = t_0 + \frac{m}{r^*_\infty} = t_0 + \frac{m}{p r_\infty} = t_0 + \frac{a}{p} \tag{3}$$

where:

$$\begin{aligned} r^*_\infty &= p r_\infty \\ a &= \frac{m}{r_\infty} \end{aligned}$$

In the Hockeny model r^*_∞ is the aggregate communication bandwidth of p processors and m is the amount of data to be transmitted. The difference between model 1 and 3 is in the meaning of their terms. The Hockney model expresses the communication time as the wallclock time required to physical deliver the message from the source to its destination processor, while in models 1 and 2 we consider the time spent by the allocated processors to manage the communication.

These models have been successfully applied to the measured times in order to describe the behavior of each data redistribution strategy.

The physical interpretation of the parameters is:

a_0 represents the amount of non parallelizable work (in computation model) or the setup time (communication model);

a_1 represents the parallelizable work (computation model) or the amount of data to be exchanged by the p processors;

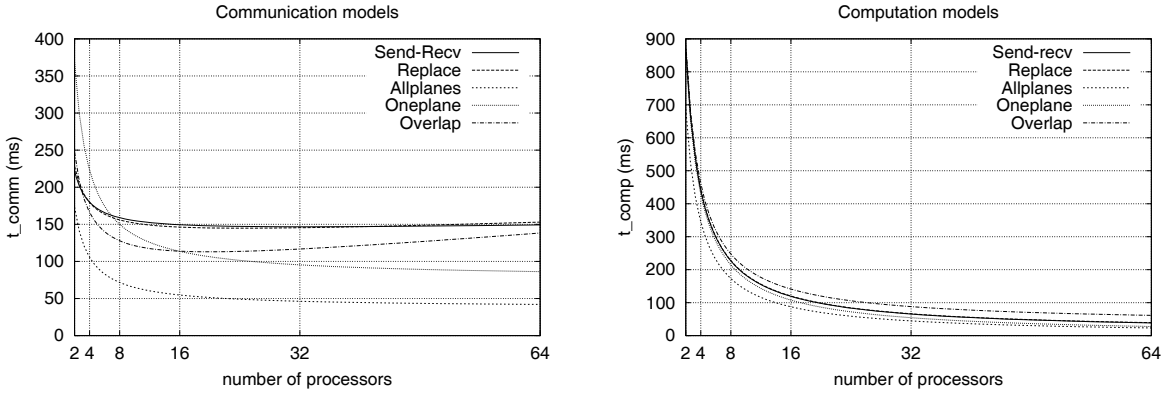
a_2 represents the costs required to manage the parallelism when p processors are considered.

The values of these parameters were estimated by means of fitting techniques available within the Medea tool, and the resulting models were validated against the measured data to ensure that they are accurate description of the phenomena under investigation. For example, fig. 5 shows the measured computation/communication times as well as the modeled ones.

Note that, the sequential runs (i.e., when only one processor is allocated) have been discarded when modeling the communication time because it is meaningless.

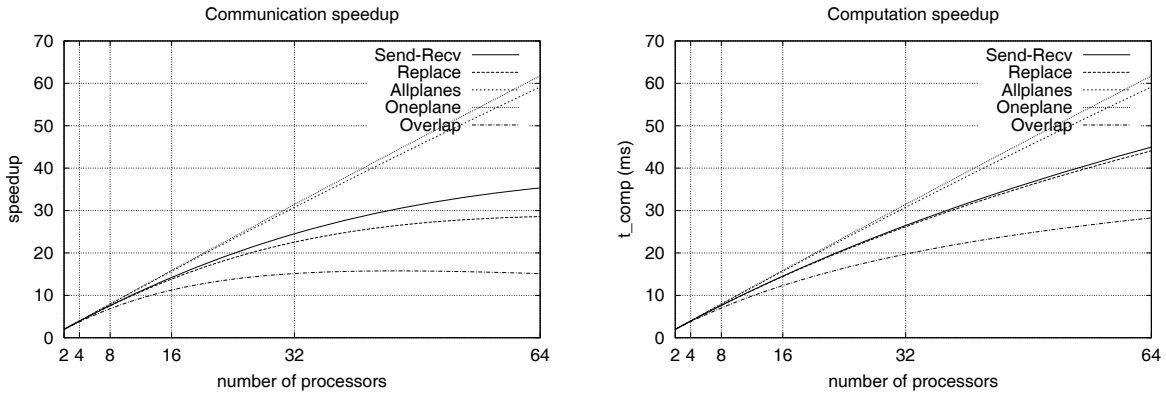
Both communication (see fig. 6 (a)) and computation (see fig. 6 (b)) models have been used to compare the performance of the data redistribution strategies.

Scalability can be quantitatively studied by analyzing the models of each kernel. Speedup figures can be easily derived from the models as the ratio between the performance



(a) *(b)*

Figure 6. Communication and computation models, respectively in figure *(a)* and *(b)*, for all the kernels.



(a) *(b)*

Figure 7. Communication and computation speedups, respectively in figure *(a)* and *(b)*, for all the kernels.

obtained with one processor over the performance when p processors are considered. Figure 7 shows the speedup curves for both communication (a) and computation (b) for all the kernels. The communication times of the $Overlap$ kernel Computation activities, whose execution times for all the kernels are expressed by model 1, exhibit a monotonic behavior and hence there is not a maximum number of processors that can be allocated. On the other hand, communication activities, which are expressed by model 2 for all the kernels but the $Allplane$ and $Oneplane$ ones, have a maximum when $p_{max} = \sqrt{\frac{a_1}{a_2}}$ processors are allocated. Further increase in the number of processors is likely to result in performance degradation. For examples, as shown in fig. 7 (a), $p_{max} = 49$ for the $Overlap$ communication times. By combining the models for communication and computation times it is possible to derive the expression for the overall speedup. In this case, for *Send-Recv*, *Replace*, and *Overlap* kernels result:

$$p_{max} = \sqrt{\frac{a_1 + b_1}{b_2}}$$

while the other two kernels do not have any constrains on the number of processors to be allocated.
Detailed analysis of the speedup may be also focused at evaluating the optimal number of processors which can be profitably allocated [8].

6 Conclusions

In this paper we have looked at various different ways of scheduling data communication between processors for the multidimensional FFT algorithm, while keeping the overall

data volume and range of transfer the same. Some strategies are clearly superior to others. Contrary to expectation, the more sophisticated strategies (like overlapping communication and computation) do not perform as well as expected. The overheads associated with individual calls, such as setup, buffering, lack of synchronization etc, play a more decisive role. Hence the kernel with the fewest individual call performs the best. Amidst the kernels with comparable number of calls, the setup and buffering overheads determine the performance. The experimental results were also substantiated by analytical models for all the kernels.

Acknowledgements

This research, in part conducted at the Maui High Performance Computing Center, was sponsored in part by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under cooperative agreement number UNIVY-0282-U00. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory, the U.S. Government, The University of New Mexico, or the Maui High Performance Computing Center.

This research was also supported in part by the European Commission under the ESPRIT IV Working Group APART.

Authors would like to thank Piero Colli Franzone for his precious help in the fitting analysis.

References

[1] *Engineering and Scientific Subroutine Library*. IBM Inc., 1999. #GC23-0529.

[2] T. Agerwale, J. Martin, J. Mirza, . D. Sadler, D. Dias, and M. Snir. SP2 system architecture. *IBM System Journal*, 34(2):152–184, 1995.

[3] S. Bokhari. Multiphase Complete Exchange: A Theoretical Analisys. *IEEE Transactions on Computer*, 45(2):220–229, 1996.

[4] J. Brehm, P. Worley, and M. Madhukar. Performance modeling for SPMD message-passing programs. *Concurrency: Practice and Experience*, 10(5):333–357, 1998.

[5] P. Calland, J. Dongarra, and R. Yves. Tiling on systems with communication/computation overlap. *Concurrency: Practice and Experience*, 11(3):139–153, 1999.

[6] M. Calzarossa, L. Massari, A. Merlo, M. Pantano, and D. Tessera. MEDEA – A Tool for Workload Characterization of Parallel Systems. *IEEE Parallel and Distributed Technology*, 2(4):72–80, 1995.

[7] A. Dubey and T. Clune. Optimization of a parallel pseudospectral MHD code. In *Proceedings of Frontiers '99: The 7th Symposium on the Frontiers of Massively Parallel Computation*, pages 208–212. IEEE Computer Society, 1999.

[8] D. Ghosal, G. Serazzi, and S. Tripathi. The Processor Working Set and its Use in Scheduling Multiprocessor Systems. *IEEE Trans. on Software Engineering*, SE-17(5):443–453, 1991.

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[10] R. Hockney and M. Berry. Public International Benchmarks for Parallel Computers: PARKBENCH Committee Report-1. *Scientific Computing*, 3(2):101–146, 1994.

[11] R. Hofmann, R. Klar, B. Mohr, A. Quick, and M. Siegle. Distributed Performance Monitoring: Methods, Tools, and Applications. *IEEE Trans. on Parallel and Distributed Systems*, 5(6):585–598, 1994.

[12] C. Scheiman and K. Schauser. Evaluating the Benefits of Communication Coprocessors. *Journal of Parallel and Distributed Computing*, 57(2):236–256, 1999.