Software Tool Evaluation Methodology [1]

Salim Hariri[2] , Sung-Yong Park, Rajashekar Reddy,
Mahesh Subramanyan, Rajesh Yadav, and Geoffrey Fox

Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY 13244

Manish Parashar
Department of Computer Sciences & Center for Relativity
University of Texas, Austin, TX

(SCCS #643)

[1]This research is funded by Rome Laboratory (contract number F-30602-92-C-0063), Rome, NY

[2]Dr. Salim Hariri, Professor, ECE Department, Syracuse University, Syracuse, NY 13244, email: hariri@cat.syr.edu

Abstract

The recent development of parallel and distributed computing software has introduced a variety of software tools that support several programming paradigms and languages. This variety of tools makes the selection of the best tool to run a given class of applications on a parallel or distributed system a non-trivial task that requires some investigation. We expect tool evaluation to receive more attention as the deployment and usage of distributed systems increases. In this paper, we present a multi-level evaluation methodology for parallel/distributed tools in which tools are evaluated from different perspectives. We apply our evaluation methodology to three message passing tools viz Express, p4, and PVM. The approach covers several important distributed systems platforms consisting of different computers (e.g., IBM-SP1, Alpha cluster, SUN workstations) interconnected by different types of networks (e.g., Ethernet, FDDI, ATM).

1 Introduction

The recent decades have seen an increasing interest in parallel/distributed multi-computer systems (multiple independent computing units interconnected by local-area or custom networks) as a feasible and cost-effective means of achieving the high-performance computing capabilities demanded by existing and future applications. Consequently there has been a proliferation of (commercial as well as academic) software systems aimed at providing the communication infrastructure required to exploit such computing environments. Available software systems or parallel/distributed computing tools (PDC tools) vary significantly in terms of the application domain targeted and corresponding functionality provided, the computational & communication model supported, the underlying implementation philosophy, and the computing environments supported.

General purpose systems like MPI, PVM and P4 provide a wide class of basic communications primitives while dedicate systems like BLACS (Basic Linear Algebra Communication System) and TCGMSG (Theoretical Chemistry Group Message Passing System) are tailored to specific application domains. Furthermore, some systems provide higher level abstractions of application specific data-structures (e.g. VSG (Virtual Shared Grids), GRIDS, CANOPY). Existing systems also differ in the computational model they provide to the user; for example loosely synchronous data parallelism, functional parallelism, or shared memory. Different systems use different implementation philosophies such as remote procedure calls, interrupt handlers, active messages, or client-server based, which makes them more suited for a particular type of communication. Finally certain systems (such as CMMD and NX/2) are tied to a particular system; in contrast to portable systems like PVM and MPI.

Given the number and diversity of available systems, the selection of a particular system for an application development is non-trivial. Factors governing such a selection include application characteristics and system specifications as well as the usability of a system and the development interface it provides. It is critical therefore, that there exists a methodology for generating a normalized evaluation of available systems which can assist users in evaluating the suitability of any particular system to their needs. In this paper we define such an evaluation methodology. The proposed methodology provides a comprehensive characterization of PDC tools by defining their evaluation from three different perspectives:

1. A low-level performance perspective evaluates the basic communication primitives provided by the tools. These include point-to-point communications (send/receive), collective communications (bcast/mcast), ring communications (all nodes send and receive), and global reduction operations.

2. An application-level performance perspective evaluates the PDC tools from the application's point of view. Here we evaluate the performance of representative

applications developed using different tools. Applications are chosen so that they incorporate a wide set of basic algorithmic building blocks.

3. A development interface (or usability) perspective characterizes the tools in terms of the functionality provided, computational/communication models supported, the ease of application development (coding, testing, and debugging support), computing environment supported, portability, etc.

The proposed methodology has two key objectives:

1. To provide a means for evaluating, quantifying and comparing the suitability of PDC tools with regard to user requirements, thereby enabling the selection of the most appropriate PDC tools for a particular application class and system configuration.

2. To serve as a unified platform for PDC tool developers for identifying the deficiencies and bottlenecks in existing systems and for defining the requirements of future systems.

The application of the proposed evaluation methodology is illustrated using a selection of widely used academic and commercial PDC tools. The low-level and application-level performance metrics are obtained experimentally using a diverse set of parallel/distributed multi-computer systems (IBM SP-1 using custom crossbar switch & LAN; and workstation clusters (Dec Alpha & SUN Sparcstations) interconnected using Ethernet, FDDI and ATM networks). The application suite that can be used to evaluate PDC tools from application perspective, includes codes from four broad classes: numerical applications, signal and image processing, simulation, and system utilities (such as parallel make, spell checker compiler). Finally, a set of criteria are outlined for characterizing the usability of the tool and its development interface. The tools considered in this study are Express [10] (Parasoft Inc.), p4 [9](Argonne National Labs), and PVM [8] (Oak Ridge National Labs).

The rest of this paper is organized as follows: Section 2 describes the proposed evaluation methodology and details the three evaluation perspectives. In Section 3, we apply the methodology to evaluate three PDC tools. The corresponding experimental results are also presented. Finally Section 4 summarizes the evaluation methodology and outlines future research directions.

2 Multi-level Tool Evaluation Methodology

Currently, there are no general criteria to evaluate a parallel/distributed tool nor it is easy to lay down such criteria [5]. One of the main difficulties in obtaining such an evaluation criteria set is that the importance and relevance of each criterion depends on many factors which include the type of available computers, the typical set of user applications, and the type of computing environment (education, government, military, industry etc.). For example, a user would give the response time as the most important performance metric to evaluate an application execution. On the other hand, a system manager might consider the system utilization or throughput as the main evaluation criterion and attempts to push the utilization to saturation (100%). By doing so, the application response time increases and reaches infinity when the system is fully saturated. These two measures are contradicting each other. Consequently, one needs to decide first the point of view (user or system manager) that needs to be considered in evaluating the performance of a given tool.

We do believe it is a challenging task to identify a meaningful criterion that takes into consideration all these factors. Hence our approach to evaluate tools is based on multi-levels where each level is representing one perspective for tool evaluation. By using weight factors, an overall tool evaluation can be tailored to take into account the most relevant factors associated with certain types of users. In this paper, we present a three level approach to evaluate parallel/distributed software tools. These levels are as follows:

1. Tool Performance Level (TPL): In this level, we evaluate the performance of tool primitives when they run on distributed systems that utilize different computer architectures and networks.
2. Application Performance Level (APL): In this level, we evaluate the performance of parallel/distributed applications that are implemented using these tools and run on different platforms.
3. Application Development Level (ADL): In this level, we evaluate the tool capability to support and facilitate the development of parallel/distributed applications.

In this paper, we evaluate three tools viz, Express, p4, and PVM with respect to each level. However, other levels can be added if necessary to take into consideration any additional set of criteria that has not been considered in these three levels. In what follows, we discuss the set of criteria to be used at each level.

2.1 Tool Performance Level (TPL)

In this level, we evaluate the performance of the primitives supported by a given tool. The primitives of any parallel/distributed software tool can be broadly characterized into four groups: 1) Communication primitives 2) Synchronization primitives 3) Management/Control primitives and 4) Exception Handling primitives.

1. **Communication Primitives:** These primitives can be divided into two types: point-to-point and group communication primitives.
   (a) **Point-to-Point Communication:** It is the basic message passing primitive for any parallel/distributed programming tool. To provide efficient point-to-point communication, most systems provide a set of function calls rather than the simplest *send* and *receive* primitives. The main primitives include synchronous and asynchronous send/receive, synchronous and asynchronous data exchange, non-contiguous or vector data.
   (b) **Group Communication:** These primitives can be divided into three categories: one-to-many, many-to-one, and many-to-many.

2. **Synchronization Primitives:** A parallel/distributed program can be divided into several different computational phases. To prevent asynchronous messages from different phases interfering with one another, it is important to synchronize all processes or a group of processes. Usually, a simple command without any parameter, such as, *exsync* in Express can provide a transparent mechanism to synchronize all the processes. But, there are several options that can be adopted to synchronize a group of processes. In PVM, *pvm_barrier*, which requires two parameters *group_name* and *num*, blocks the caller until a certain number of calls with the same group name are made.

3. **System Management:** The tasks of configuration, control, and management of a system are quite different from system to system. Most of the configuration, control and management primitives supported by the studied software tools include primitives to allocate and deallocate one processor or a group of processors to load, start, terminate, or abort programs, for dynamic reconfiguration, process concurrent or asynchronous file I/O, and query the status of environment.

4. **Exception Handling:** In a parallel/distributed environment, it is important that the network hardware and software failures must be reported to the user's application or system kernel. In traditional operating systems such as UNIX, exception handling is processed by an event-based approach, where a signal is used to notify a process that an event has occurred and after that, a signal handler is invoked to take care of the event. Basically, an event could be a hardware condition (e.g., bus error) or software condition (e.g., arithmetic exception). Express supports tools for debugging and performance evaluation.

The experimental results presented later evaluate the performance of send/receive, broadcast/multicast, ring communication and global summation primitives of the studied software tools (see Table 1).

These communication primitives play an important role in determining the performance of a large class of parallel/distributed applications. Hence, the tool that provides the best performance in executing its communication primitives will also give

Primitive	Express	p4	PVM
Send/Receive	exsend exreceive	p4_send p4_recv	pvm_send pvm_recv
Broadcast/Multicast	exbroadcast	p4_broadcast	pvm_mcast
Ring	exsend exreceive	p4_send p4_recv	pvm_send pvm_recv
Global Sum	excombine	p4_global_op	Not Available

Table 1: Communications primitives for evaluating tools at TPL

#	Numerical Algorithms	Signal/Image Processing	Simulation/Optimization	Utilities
1.	Fast Fourier Transform	JPEG Compression	N-body Simulation	ADA Compiler
2.	LU Decomposition	Hough Transform	Monte Carlo Integration	Parallel Sorting
3.	Linear Equation Solver	Ray Tracing	Traveling Salesman	Parallel Search
4.	Matrix Multiplication	Data Compression	Branch and Bound	Distributed Spell Checker
5.		Cryptology		Distributed Make

Table 2: SU_PDABS

the best performance results for a large number of distributed applications as will be shown later in section 4.

2.2 Application Performance Level (APL)

Low level benchmark tests such as communication primitive performance can some time be misleading by suggesting performance advantages for one tool over another that may not be relevant in actual applications. So in this level, we evaluate the tools from application performance perspective. We have used different classes of applications from parallel/distributed applications benchmark suit (SU_PDABS) that is currently being developed at NPAC (Northeast Parallel Architectures Center) of Syracuse University.

We have divided the applications into four classes namely, Numerical algorithms, Signal/Image Processing applications, Simulation/Optimization applications, and Utilities. Applications under different classes are shown in Table 2. We have chosen applications to include simple, medium, and complex problems, to represent a broad spectrum of applications. Even though it covers a broad spectrum of applications, it is not comprehensive. All applications in this suit are written in C using different distributed/parallel tools viz. Express, p4, and PVM.

From this benchmark suit, we have chosen JPEG Compression, Fast Fourier Transform (FFT), Monte Carlo Integration and Parallel sorting applications for benchmarking the software tools in this paper.

2.3 Application Development (Usability) Perspective

The application development perspective characterizes PDC tools on the basis of their usability (ease of use), their functionality, and the development overheads incurred in using them. In what follows we outline a set of criterion that can be used in this characterization.

Programming Models Supported: The development of any parallel or distributed application is based on an underlying programming model which determines its implementation. A number of parallel/distributed programming models have been defined to meet varied requirements; the choice of the appropriate programming model being dictated jointly by the characteristics the application and the specifications of the target computing environment. The Data Parallel programming model achieves parallelism by identifying data elements that can be operated on in parallel; while Functional Parallelism decomposes the application into tasks that can be performed concurrently. A Shared Memory programming model assumes a common memory space and achieves cooperation via shared data elements. A Message Passing programming model, on the other hand, uses explicit messages for communication. Other models include Synchronous (processing agents proceed in lock step), Loosely Synchronous (processing agents are constrained to communicate at regular intervals) and asynchronous.

The PDC tools studied in this paper support either one or both of the following programming models:

- **Host-Node Model:** The host-node programming model consists of a single host process that coordinates the execution of one or more node processes. The host is typically responsible for input/output and administrative operations while the node processes concurrently perform computations. Node process can communicate among themselves or with the host.
- **SPMD or Cubix Model:** The SPMD (single program multiple data) or Cubix model is a loosely-synchronous data-parallel programming model wherein the computing nodes execute the same program stream on different data elements.

Language Interface: The programming languages supported by PDC tools have a key impact on its usability. Supporting popular languages not only enables the developer to work with a familiar environment but also enables the reuse of existing program components. Tools supporting multiple languages allow different parts of the application to be implemented using different languages, which may be beneficial for certain applications. The PDC tools evaluated in this paper support C and FORTRAN.

Development Interface: The development interface criteria evaluates the support provided during application development. It includes the following four sub-criteria:

Ease of Programming: Ease of programming measures the effort required on the user part to interact with the tool. If the user spends more time thinking about how to use the tool or making the tool works, the tool is hindering and not helping with the programming task. Measures of this criterion include the learning curve for new as well as experienced developers, and the amount of re-engineering of re-development required.

Debugging Support: Given the complexity of parallel/distributed applications development and non-determinism that is typical of such an environment, suitable debugging supports is desirable of the PDC tool used. Possible debugging support includes:

- The ability to trace the execution of the parallel application on the PDC system.
- The ability to define break points in the application program and to stop execution at these points.
- The ability to view application data-structures at defined break points and during execution of the application.

Customization: The ability to customize a PDC tool and its interface to a developer needs provides a more comfortable development environment. Customization support includes:

- The ability to define new commands and macros for frequently used command sequences.
- Re-configuration of the tool according to desired tradeoffs for such parameters as response speed and memory utilization.
- Re-definition of tool input and output formats.

Error Handling: A PDC tool should be able to gracefully exit when an non-retrievable error occurs. In other cases, the error message should be a pointer to the type of error that has occurred. Protection from costly errors should be provided. For example, when the application requires more memory than what is available, it is an error condition. In this case, the tool should give an appropriate error message, delete all allocated memory, and exit the program without causing the terminal to hang. All the tools that we used in this paper do not have a mature error/exception handling feature and hence will not be evaluated favorably at this level.

Run-Time Interface: The run-time interface handles (among others) issues such as parallel I/O, data redistribution, and dynamic load-balancing. The ability to perform I/O concurrently across processors is becoming increasing important, especially for I/O bound application where sequential I/O can be a significant bottleneck. Run-time data redistribution is necessary when the communication patterns of the applications change from one phase to another. Finally, dynamic load-balancing is critical for application with widely varying run-time load distributions.

Integration with other Software Systems: Applications often require the services of other software systems for functionality such as visualization, profiling, data input, etc. Hence, the ability to effectively interface with other software system is an important criterion to facilitate the development of parallel/distributed applications and is used at this level of tool evaluation.

Portability: Given the number and diversity of existing parallel/distributed systems, it is critical that PDC tools and the applications developed based on them are portable. Portability also dictates that the tool provide an architecture independent programming interface. For example, Express provides the user with a virtual processor topology which is independent of the actual physical topology.

3 Experimental Results

In this section, we apply our evaluation methodology to three PDC tools (Express, p4, and PVM) and evaluate them from three different perspectives: tool performance, application performance, and tool usability. The results of our evaluation can be used to assist in determining the best platform, network technology, and PDC tool to run a given class of applications.

3.1 Experimentation Environment

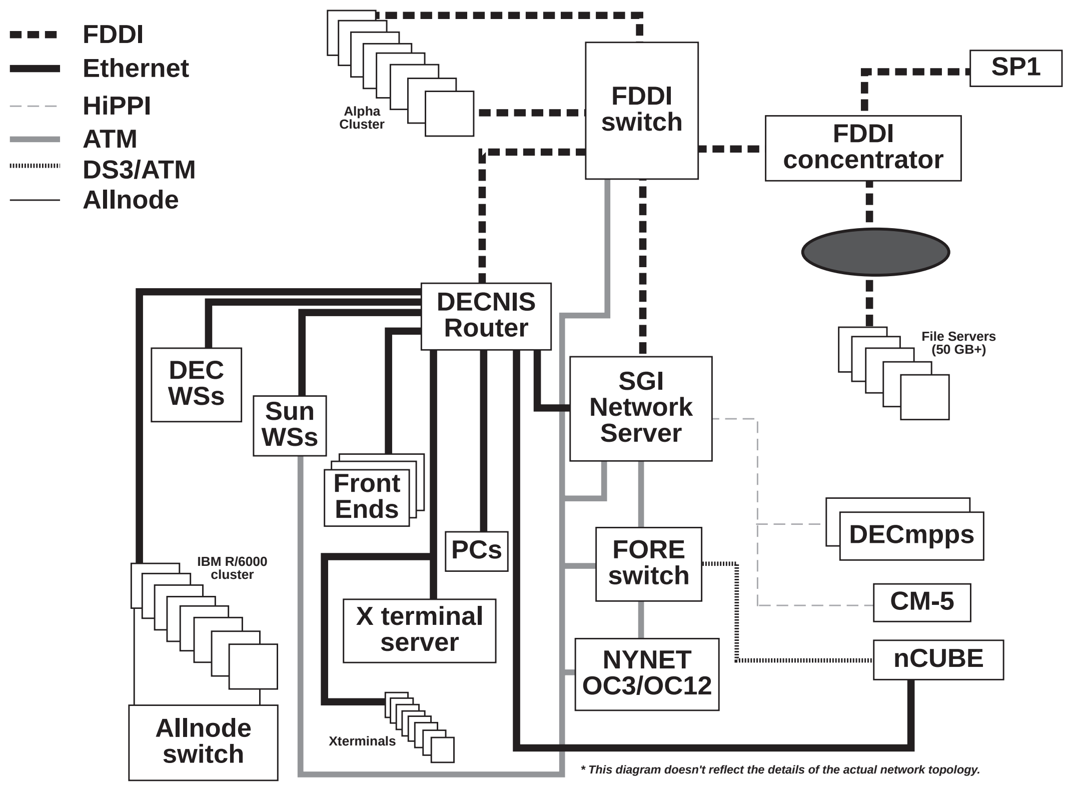

Figure 1: Computing Environment at NPAC

The evaluation presented in this section was performed on a wide set of *state-of-the-art* multi-computer systems which are a part of the high performance computing environment at the Northeast Parallel Architectures Center, Syracuse University (see Figure 1). The platforms used are briefly described below:

IBM SP-1: The SP-1 consists of a cluster 16 RISC/6000 370 nodes interconnected by a crossbar switch (Allnode) and a dedicated Ethernet. Each node runs at a clock rate of 62.5 MHz. The evaluation presented in this section is performed on the Allnode switch and the dedicated Ethernet.

ALPHA/FDDI: The ALPHA/FDDI configurations consisted of 8 DEC ALPHA workstations interconnected by a high performance (100 Mbps) backbone composed of dedicated, switched FDDI segments. The ALPHA nodes have a clock rate of 150 MHz.

SUN/ATM WAN: This configuration consists of SUN SPARCstation IPXs communicating over the NYNET. NYNET is an ATM wide area network (WAN) that covers all New York State and Part of Massachusetts State. Most of the wide area portion of the NYNET operates at speed OC 48 (2.4 Giga bits per second) while each site is connected with two OC 3 links (155 Million bits per second). In this paper, we evaluate the performance of PDC tools on the NYNET connection between Syracuse University and Rome Laboratories, Rome, NY.

SUN/ATM LAN: This configuration consists of SUN SPARCstation IPXs interconnected by an ATM LAN using an ATM FORE switch. TAXI interface is provided between the workstations and the ATM switch. The network bandwidth is 140 Mbps. SUN IPX nodes operate on a 40MHz clock.

SUN/Ethernet: This configuration consists of SUN SPARCstation ELCs interconnected by an Ethernet LAN. The ELCs operate at a clock rate of 33 MHz.

3.2 Tool Performance Level (TPL)

In what follows, we benchmark the point-to-point and group communication primitives of the three tools on different distributed computing platforms.

3.2.1 Send/Receive primitives

Table 3 shows the execution time of snd/rcv primitives when implemented in Express, p4, and PVM and for different message sizes up to 64 Kbytes. For example, for message size of 16 Kbytes, snd/rcv primitive takes approximately 111, 44, and 61 milliseconds when it is implemented using Express, p4, and PVM, respectively over Ethernet. It is clear from this table that the p4 implementation of point-to-point communications on SUN Workstations has the best performance when compared to the other tool implementations.

Table 3 shows the snd/recv time for these tools on SUN SPARCstations over ATM LAN and ATM WAN (NYNET). Similarly to the Ethernet results, p4 implementation of the send/receive primitives outperformed the other tool implementations. Express performs a little better than PVM for small message sizes (upto 1 Kbytes) but PVM outperforms Express for large messages. This table shows the significant improvement in throughput when ATM networks are used as the underlying communication network of high performance distributed systems. Furthermore, this table shows that ATM WAN performance of send/receive primitives is similar.to those of ATM LAN.

Hence, it is feasible to build distributed computing systems across an ATM WAN and their performance is comparable to those based on LANs.

Mesg Size (Kbytes)	PVM			p4			Express	
	Ethernet	ATM (LAN)	ATM (WAN)	Ethernet	ATM (LAN)	ATM (WAN)	Ethernet	ATM (LAN)
0	9.655	7.991	7.764	3.199	2.966	3.636	4.807	4.152
1	11.693	8.678	8.878	3.599	3.393	4.168	10.375	7.240
2	14.306	9.896	10.105	4.399	3.748	4.822	18.362	11.061
4	25.537	13.673	14.665	9.332	4.404	5.069	32.669	16.990
8	44.392	18.574	19.526	24.165	6.482	7.459	59.166	27.047
16	61.096	27.365	28.679	44.164	11.191	13.573	111.411	46.003
32	109.844	48.028	53.320	98.996	19.104	22.254	189.760	82.566
64	189.120	88.176	91.353	173.158	35.899	41.725	311.700	153.970

Table 3: snd/recv timing for SUN SPARCstations (in milliseconds)

3.2.2 Broadcast Primitives

Figure 2 shows the execution time for broadcasting messages of different message sizes up to 64 Kbytes among 4 Sun Workstations over Ethernet and ATM wide area network. For this group communication primitive, p4 has the best performance while Express has the worst performance. It is worth noting that the tool with better snd/rcv performance does not necessarily imply the better performance for broadcast/multicast primitives. This is because of the fact that broadcast/multicast performance greatly depends on the algorithm used for its implementation. We observe similar results on NYNET network.

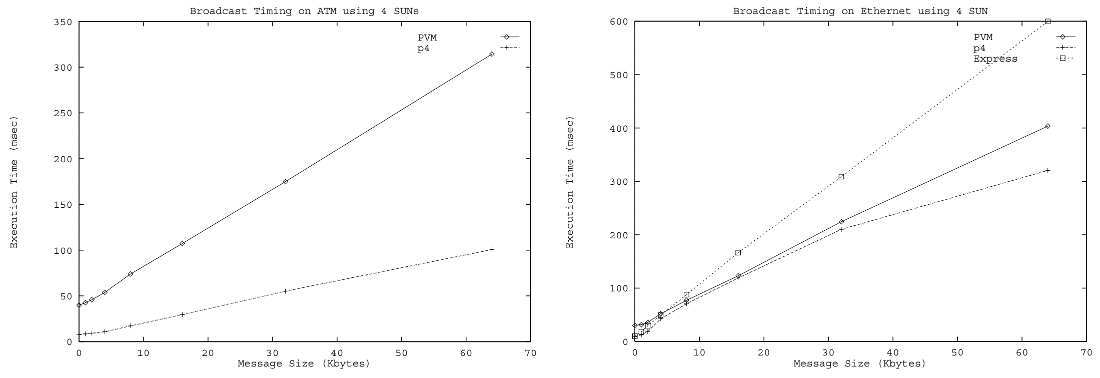

Figure 2: Broadcast on SUN SPARCstations over Ethernet and ATM WAN

3.2.3 Ring Communication

Results of the ring communication for different message sizes are given in Figure 3. Ring communication was implemented using snd/recv primitive in all three tools. As with other communication primitives p4 performs best among all other tools. One interesting point to note is that even though PVM performs better than Express in snd/recv primitive, Express outperforms PVM for ring communication and this indicates that Express is better suited for continuous flow of incoming and outgoing data when compared to PVM. However, p4 is the best among the three for this type of applications.

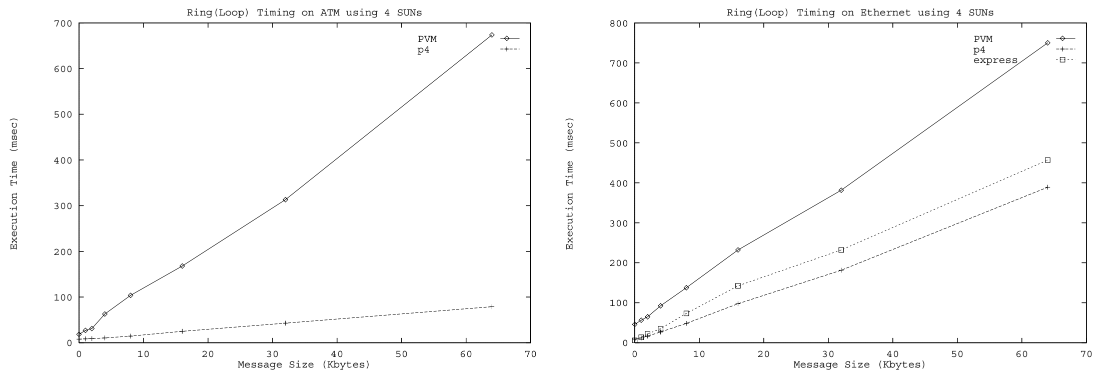

Figure 3: Ring communication on SUN SPARCstations over Ethernet and ATM WAN

3.2.4 Global Summation

Global operations are very important in measuring performance of PDC tools. We selected global summation for our performance measurement as this is the most commonly used global operation. PVM does not support any global operation and thus it is not evaluated for this operation. The performance results of p4 and Express implementation of this global summation on Ethernet are shown in Figure 4. This figure shows the performance on NYNET as well. P4 implementation is also better than Express for this operation.

Table 4 summarizes the results of our evaluation of these tools with respect to their communication primitives. From this table we can see that p4 outperforms Express and PVM in all classes of communication primitives. This can be attributed to the efficient implementation of p4 communication primitives which add very small amount of overhead to the underlying transport layer.

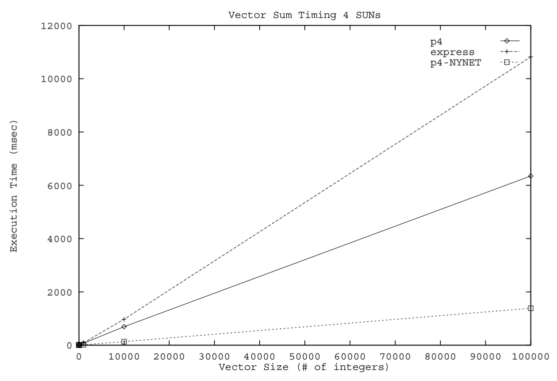

Figure 4: Global summation on SUN SPARCstations over Ethernet and ATM WAN

SUN/Ethernet				SUN/ATM		
snd/rcv	broadcast	ring	global sum	snd/rcv	broadcast	ring
p4 PVM Express	p4 PVM Express	p4 Express PVM	p4 Express	p4 PVM Express	p4 PVM	p4 PVM

Table 4: Summary of Tool Performance on different Platforms

3.3 Application Performance Level (APL)

In this section we evaluate the PDC tools by comparing the execution times of four applications that are commonly used in distributed systems. A brief description of these applications and their parallel implementations are highlighted below:

1. **JPEG Compression**
   The main problem with digital imaging applications is that a vast amount of data required to represent a digital image directly. Thus, the use of digital images is limited in distributed systems because of the high storage requirement and the long transmission times to transfer images from one site to another. Image compression technology can compress images by 1/10-1/50 of their original size without affecting image quality. JPEG (Joint Photographic Experts Group) is a standard image compression method which enables interoperability of equipments from different manufacturers. JPEG standards are based on DCT (Discrete Cosine Transform). This application involves simulation of JPEG image compression that requires substantial processing and storage. In this application, parallelism is achieved by data parallel model and thus the image to be compressed or decompressed is divided into N equal parts (where N denotes the number of processors), except for the one portion which can be slightly larger than the rest. We use host-node programming model in which the master process distributes the image among all nodes and then collects the results from all

nodes. It also processes its portion of the image. The parallel implementation of JPEG application consists of three phases: Distribution, computation, and collection phases. During distribution and collection phases, the computers exchange large volume of data while no communication is performed during the computation phase.

2. **Two-Dimensional Fast Fourier Transform (2D-FFT)**
Two-Dimensional FFT is a useful transformation and has many applications in image enhancement, data compression, and image reconstruction. To compute the FFT in two dimensions (e.g., a screen of video data), one has to compute a one dimensional FFT for each of the rows and each of the columns. This algorithm involves intensive computations. Although the processing in 2D-FFT can be easily distributed, a distributed 2D-FFT involves transfer of large amount of data between processors. Thus, it is a good application to benchmark the performance of communication primitives.

3. **Monte Carlo Integration**
Monte Carlo integration is an efficient method for evaluating definite integrals. The idea behind the Monte Carlo integration is to generate random points between the integration interval and calculate the function values at these points and the mean of these function values gives the value of the definite integral. Since this involves generating random samples, this is an approximate method and thus more samples lead to a better approximation. This application is compute intensive and communicate only short messages. Hence this can benchmark the computing capacity of parallel/distributed platforms and latency impact of different tool implementations on the performance of this type of applications.

4. **Sorting by Regular Sampling**
Sorting is one of the most studied problems in Computer Science because of its theoretical interest and practical importance. If huge amount of data needs to be sorted, sequential sorting will be quite slow necessitating parallel sorting. *Parallel Sorting by Regular Sampling (PSRS)* involves partitioning the data into smaller subsets such that all the elements in one subset not greater than any element in a later subset and sorting each subset independently. PSRS partitions the data into ordered subsets of approximately equal size. This algorithm represents a class of applications in which the computation and communication requirements are data dependent.

We have benchmarked these applications on all the platforms discussed in Section 3.1 when they are implemented using p4, PVM, and Express tools.

Figure 5 shows the benchmark results of these applications on ALPHA cluster. The p4 implementation of JPEG compression and 2D-FFT performed the best, whereas PVM and Express implementations were best for sorting and Monte Carlo integration,

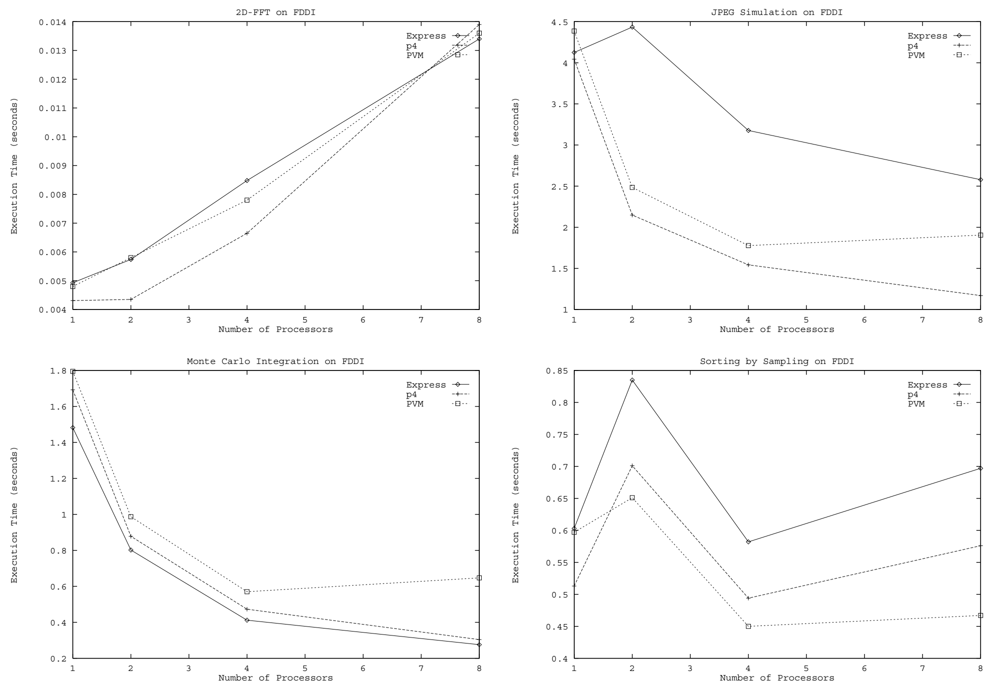

Figure 5: Application Performances on ALPHA/FDDI

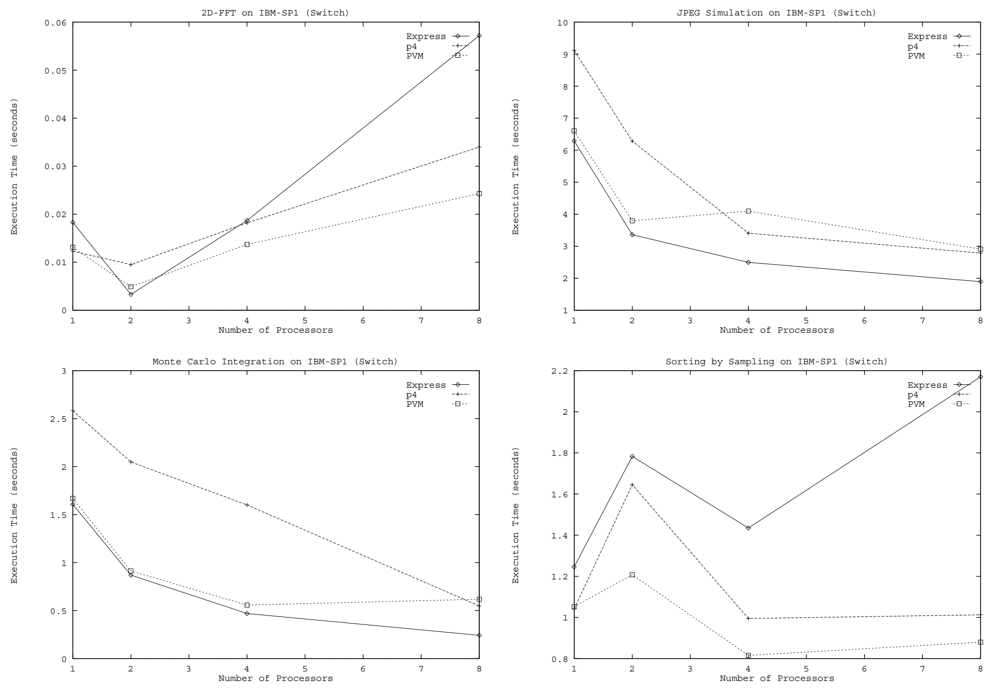

Figure 6: Application Performances on IBM-SP1 with crossbar switch

respectively. Since JPEG compression involves heavy communication, p4 implementation of JPEG compression is understandably performs best, since it involves least communication overhead among all three tools as shown in the previous subsection.

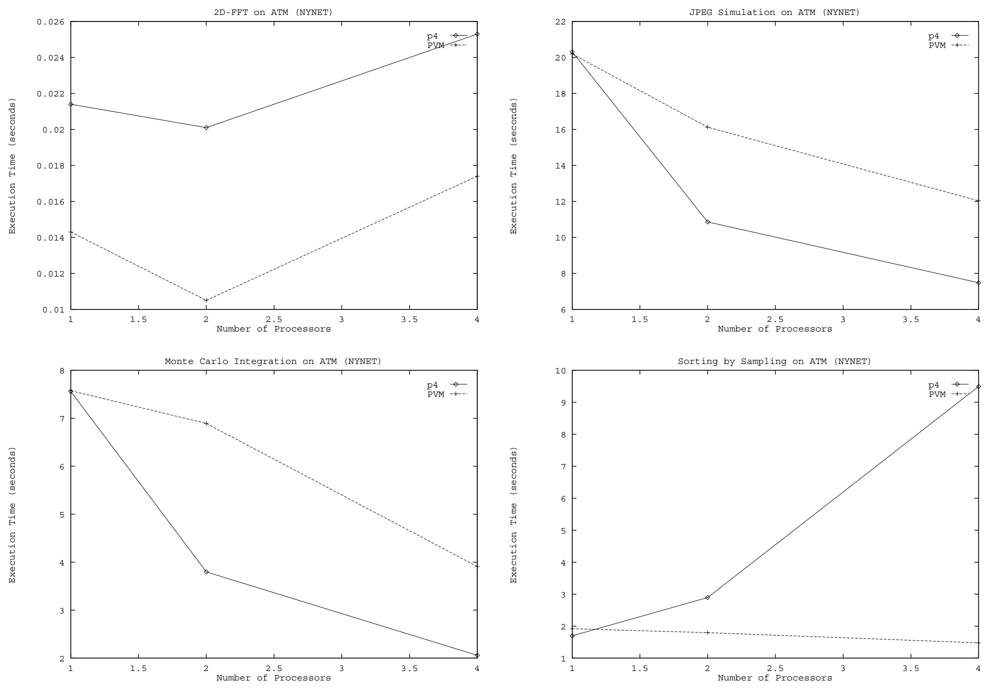

Figure 7: Application Performances on SUN/ATM-WAN(NYNET)

Figure 6 shows the benchmark results when the four applications run on IBM-SP1. The results of this figure are consistent with those obtained on the ALPHA cluster. However, the execution times are significantly higher on IBM-SP1 compare to ALPHA cluster because SP1 uses slower processing nodes and interconnect network.

Figure 7 and Figure 8 shows the timings on SUN IPXs connected by Ethernet and ATM WAN. Comparing the applications performance when they are implemented on NYNET (ATM WAN) and on Ethernet LAN shows that distributed computing is feasible across wide area networks and can outperform LANs if higher speed network technology such as ATM is used.

3.3.1 Application Development (Usability) Perspective

In this section, we evaluate the tools from their programability and their support to developing efficient distributed computing applications. For each tool, we show whether or not a usability criterion is supported and if it does how well it is covered in such a tool. However, more research is needed to quantify and validate this assessment and we are investigating techniques to address these issues. Table 3.3.1 shows

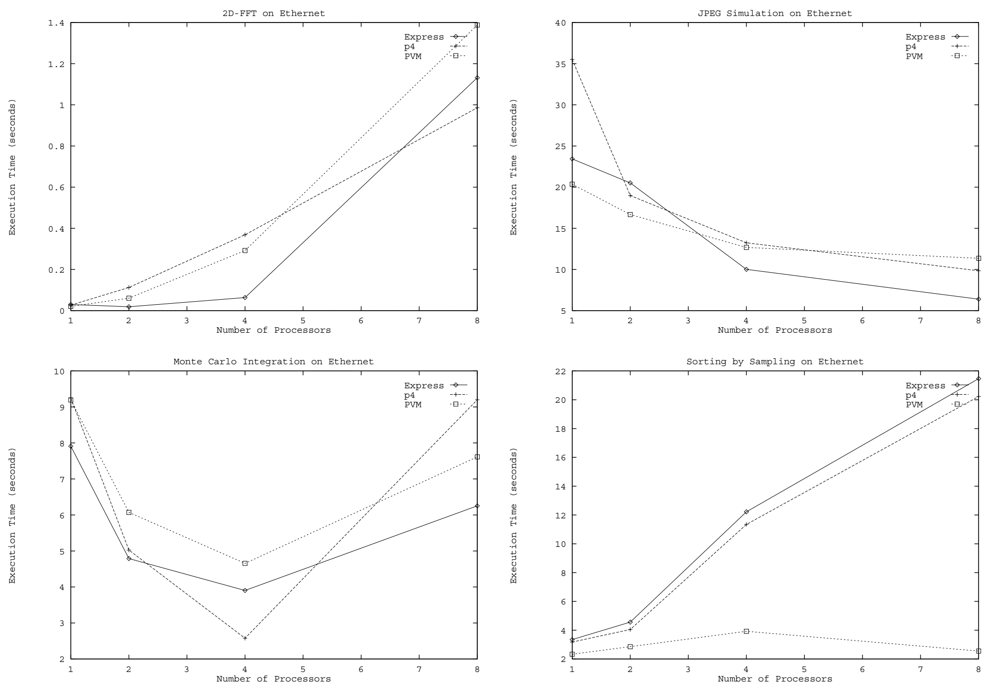

Figure 8: Application Performances on SUN/Ethernet

our assessment of these tools in terms of their support to the criteria mentioned in section 2.3

Criterion	P4	PVM	Express
Programming Models Supported	WS	WS	WS
Language Interface	WS	WS	WS
Development Interface			
Ease of Programming	PS	WS	PS
Debugging Support	PS	PS	WS
Customization	PS	NS	PS
Error Handling	PS	PS	PS
Run-Time Interface	PS	WS	WS
Integration with other Software Systems	PS	WS	NS
Portability	WS	WS	WS

4 Summary and Conclusion

Current trends in parallel/distributed computing indicate that the future of parallel computing lies in the integration of existing computers into a single heterogeneous high performance computing environment that allows them to cooperate in solving complex problems. Software development for that environment is a non-trivial process and requires a through understanding of the application and architecture. Another important aspect of high performance distributed computing is the availability of suitable message passing tools. The recent development of parallel/distributed computing software has introduced a variety of message passing tools. In this paper, we proposed a hierarchical approach for evaluating message passing tools. This scheme evaluates tools from different perspectives viz. tool performance, application performance, and application development. In evaluating tool performance, we used four different types of communication primitives (send/receive, broadcast, ring operation, and global summation) to evaluate tools performance. We also presented a benchmark suite with four classes of algorithms to evaluate PDC tools from application performance perspective. We also presented the performance of these tools on four applications. Furthermore, we presented a set of criteria to evaluate these tools from programmability perspective and their effectiveness to develop distributed applications. We then used this set to evaluate the PDC tools studied in this paper. Although the tool criteria presented in this paper cover a broad spectrum of requirements, they do not form an exhaustive list of requirements. A criterion can be added or deleted according to the user requirements. Our objective is to present an outline for a general multi-level evaluation methodology, which can be used to evaluate any parallel/distributed tool from different perspectives. Further research is needed to quantify and validate accurately the tools capability to support the development of parallel/distributed applications.

References

[1] Paul Messina., Arnold Alagar., Clive Ballie., Edward Felten., Paul Hipes., ANke Kamrath., Robert Leary., Wayne Pfeiffer., Jack Rogers., David Walker., Roy Williams., "Benchmarking Advanced Architecture Computers", Caltech Supercomputing Facility, San Diego Super Computing Center, Department of Mathematics, University of South Carolina, Caltech Report, C3P712.

[2] G. C. Fox, W. Furmanski, "Communications Algorithms for Regular Convolutions on the Hypercube", Caltech report C^3P-329(1986).

[3] Geoffrey C. Fox., Mark A. Johnson., Gregory A. lyzenga., Steve W. Otto., John K. Salmon., David W. Walker., "Solving Problems on Concurrent Processors", New Jersey : Prentice Hall, November 1988.

[4] Doveen Y. Cheng and D. M. Pase, "An Evaluation of Automatic and Interactive Parallel Programming Tools", Proceedings of Supercomputing, 1991.

[5] Salim Hariri, Geoffrey C. Fox, Balaji Thiagarajan, Manish Parashar, "Parallel Software Benchmark for BMC^3/IS Systems", Northeast Parallel Architectures Center, 111 College Place, Syracuse University, Syracuse, NY 13244-4100.

[6] R.Olson.,"Parallel Processing in a Message Based Operating System", IEEE Software, July 1985.

[7] D.Reed and D.Grunwald, "The performance of multicomputer interconnection network", IEEE Computer, June 1987.

[8] Adam Beguelin, Jack Dongara, Al Geist, Robert Manchek , and Vaidy Sunderam, "User Guide to PVM", Oak Ridge National Laboratory, Oak Ridge TN 378 31-6367 and Department of Mathematics and Computer Science, Emory University, February 1993.

[9] Ralph Butler, and Ewing Lusk, "User's Guide to the p4 Programming System", Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4801

[10] Parasoft Corporation, "Express 3.0 Documentation", Parasoft Corporation, 2500, E.Foothill Blvd. Pasadena, CA 91107.