Experimentally Assessing the Usability of Parallel Programming Systems

Duane Szafron [*] Jonathan Schaeffer [†]

Abstract

This paper discusses an experiment to compare the usability of two parallel programming systems (PPS). In this experiment, half of the students in a graduate parallel and distributed computing course solved a problem using the Enterprise PPS while the other half solved the same problem using a PVM-like library of message-passing routines. The feedback from such experiments is necessary to help narrow the gap between what parallel programmers want, and what current PPSs provide.

1 Introduction

A large number of software systems have been developed to simplify the task of developing parallel software. At one extreme, some of these systems support specialized programming models that allow programmers to quickly achieve high performance for selected applications. Unfortunately, this high performance cannot be matched across all classes of applications. Other systems provide a set of low-level primitives that allow the programmer to achieve high performance for many applications, but at the expense of drastically increased software development time.

There are many considerations that affect the assessment of parallel programming systems (PPSs), but the majority fall into three categories: performance, applicability and usability [7]. Since the performance and applicability issues are addressed in other papers, we do not elaborate on it further. Usability may be the most important since it influences the productivity of programmers. Given the extra complexity of debugging and testing parallel and distributed software, it is essential that a PPS eliminate, simplify, or at least mask the complexity.

Although there have been many human-factors studies of the productivity of sequential programmers [1], other than [4] we know of no comparable studies for programmers developing parallel software. In [7], we proposed two experiments to assess the productivity of programmers using PPSs, one for novices and one for experts. This paper describes an experiment to measure the ease with which novices can learn the programming system and produce correct, but not necessarily efficient, programs.

[*] Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1. Email: duane@cs.ualberta.ca.

[†] Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1. Email: jonathan@cs.ualberta.ca. Visiting professor, University of Limburg, Netherlands.

A controlled experiment was conducted where half of the graduate students in a parallel/distributed computing class solved a problem using the Enterprise PPS [5] while the rest used a PPS consisting of a PVM-like library of message passing calls called NMP [3] [1]. The specific PPSs used in this experiment are not the focus of this paper. Instead, we argue that controlled experiments must be conducted so that PPS developers can determine which features should be included in PPSs.

2 The Programming Task

The problem chosen for the experiment was the computation of a transitive closure that iterates until all values in a set have been assigned a value. Each iteration must traverse a graph using the information from the previous iteration to resolve additional data values. This problem has an obvious solution, where each processor is responsible for a sub-graph, and the processes synchronize at the end of each iteration. It is possible to create a chaotic solution, where the processes do not synchronize, but this requires careful consideration of the termination conditions.

In Enterprise the interactions of processes in a parallel computation are described using an analogy based on the parallelism in a business organization [5]. Every sequential procedure that will execute concurrently is assigned an asset type (individual, line, department, etc.) that determines its parallel behavior. The user code for each of these procedures is sequential C, but a procedure call to such an asset is automatically translated into a message send by Enterprise. Consider the following user C code, assuming that `func` is an asset in the program:

```
result = func( x, y );
/* other C code */
a = result;
```

When Enterprise translates this code to run on a network of workstations, the parameters `x` and `y` are packed into a message and sent to the process that executes the asset `func`. The caller continues executing and only blocks and waits for the function result when it accesses the result (`a = result`). Allowing concurrent actions until the result of a previous computation is required has been called a *future*.

Enterprise has three components: an object-oriented graphical interface, a pre-compiler, and a run-time executive. The user specifies the application parallelism by drawing a hierarchical enterprise that consists of assets. At run-time, each asset corresponds to one or more processes. Sequential procedure calls in C are translated by the pre-compiler into message send/receives across a network. The execution of the program (process/processor assignment, establishing communication links, monitoring network load) is done by the run-time executive.

The Network Multiprocessor Package (NMP) is a PVM-like message passing library [3]. Essentially it is a friendly interface to TCP and UDP. NMP provides the same basic facilities as PVM [2], except support for using heterogeneous processors and dynamic reconfiguration of processes and their communications paths. NMP was chosen over PVM for three reasons: 1) it is a subset of PVM and has less than 20 different library calls

[1] The experiment is described in more detail in [6].

to learn, 2) the documentation for NMP is less than 20 pages, and 3) NMP has been used in a graduate course for the past 5 years.

3 Experiment Design

There are a number of considerations that must be taken into account in the design of a fair experiment to measure usability.

1. Prelude: We consulted a cognitive psychologist with expertise in designing experiments that involve human subjects. To eliminate biases, it was important that the students not know the exact nature of what was being measured.
2. Subjects: The students in a parallel programming graduate course were used as subjects. None of them had any previous parallel programming experience prior to taking this course. The 15 students were randomly divided into 2 groups: NMP (8 students) and Enterprise (7 students).
3. Instruction: For both Enterprise and NMP, the students were given a 50 minute classroom lecture, a 20 minute lab demonstration and documentation. In addition to the instructor, a teaching assistant who was familiar with both NMP and Enterprise was available to answer student questions.
4. Environment: Each student account was provided with a modified *zsh* shell that logged all commands executed by the students. The students were not told about the instrumentation. This is an important point since subjects who know about instrumentation may consciously or subconsciously modify their behavior. We had to be wary of ethical issues, and made sure the information gathered was comparable to that provided by the UNIX *lastcomm* facility. All programming was done on a network of 20 SUN 4s.
5. Epilogue: At the experiment's conclusion, students were asked to submit a two-page write-up commenting on their respective PPS.

4 Experiment Results

Our experiment measured five factors that seem to be indirect measures of usability as well as one factor (run-time performance) that may be sacrificed for increased usability. Figure 1 shows the six statistics that were analyzed:

1. Number of hours a student was logged in actively working on the assignment.
2. Number of lines of code in the solution program [2].
3. Number of editing sessions.
4. Number of compiles that attempted to link the program together (i.e., compiles which failed because of syntax errors were not included).
5. Number of times the students tested their parallel program by running it.
6. Execution times of their program.

[2] Count included blank lines and comments. Students were given the sequential program (128 lines of code) and were expected to parallelize it. They were also given a library containing the parts of the program that did not have to be altered (over 1000 lines of code) during parallelization. The figure shows the parallel code written less the 128 lines of sequential code.

In the first five cases, a lower number indicates higher usability, while in the sixth case, a lower number indicates better run-time performance. In each figure, the hollow circles represent NMP data points and the solid circles represent Enterprise data points. Each student is given a number, so the reader can compare an individual's performance across graphs. These graphs are ordered with the best performer on the left and the worst on the right. The right-hand side of each graph shows the average of the NMP students (dashed line) and the Enterprise students (solid line).

The statistics support our initial expectations that students would do less work (higher usability) with Enterprise, but get better run-time performance with NMP. Enterprise students did 14% fewer edits, wrote 66% fewer lines of code, did 34% fewer compiles and 13% fewer program test runs. However, perhaps surprisingly, they used 26% more login time. Why does this apparent anomaly exist? There are several reasons:

1. Enterprise compiles take roughly 5 times as long as NMP compiles. Enterprise must preprocess the user's code by making several passes over the input file before it produces a file that is compiled by the C compiler. From Figures 1a) and 1c), the average NMP user compiled 7.2 times per hour, while the average Enterprise user compiled only 3.5 times per hour.
2. Enterprise includes an option to replay a computation using animation. The user can see (and inspect) the messages being sent and monitor the status of each process. If the user watches an animation to completion using the default settings, it could take as long as 10 minutes. Each Enterprise user, on average, used this feature 25 times.
3. The students uncovered nine bugs in Enterprise; two of them serious errors that affected the student's progress. Although turnaround on bug fixes was rapid, most students assumed that the bug was in their program and not in Enterprise. We do not know how much time they devoted to solving these problems before reporting them.
4. Since the NMP performance was better, Enterprise students spent more time doing performance tuning to try to obtain better speed-ups.

As expected the NMP solutions (excluding the anomalous NMP-2 data point) had better run-time performance (27%). For this problem, the Enterprise communication time could be as high as 30% of the execution time depending on how the problem was solved. Since Enterprise has hidden manager processes that forward messages to replicated assets, there could be twice as many messages as in a hand-coded NMP solution. In addition, at least two of the Enterprise solutions had bugs in them whereby two futures overlapped, forcing sequential execution where concurrent execution was intended.

5 Conclusions

This paper has identified an area where the parallel/distributed computing community has been negligent in providing quantitative data. Hardware vendors are quick to cite measures that flatter the performance of their machines, but neglect to quantify the usability of their software. The growing base of parallel computing users could significantly benefit from an objective assessment of the usability of PPSs.

This experiment had four major results:

1. It demonstrated that the usability of PPSs can be measured.

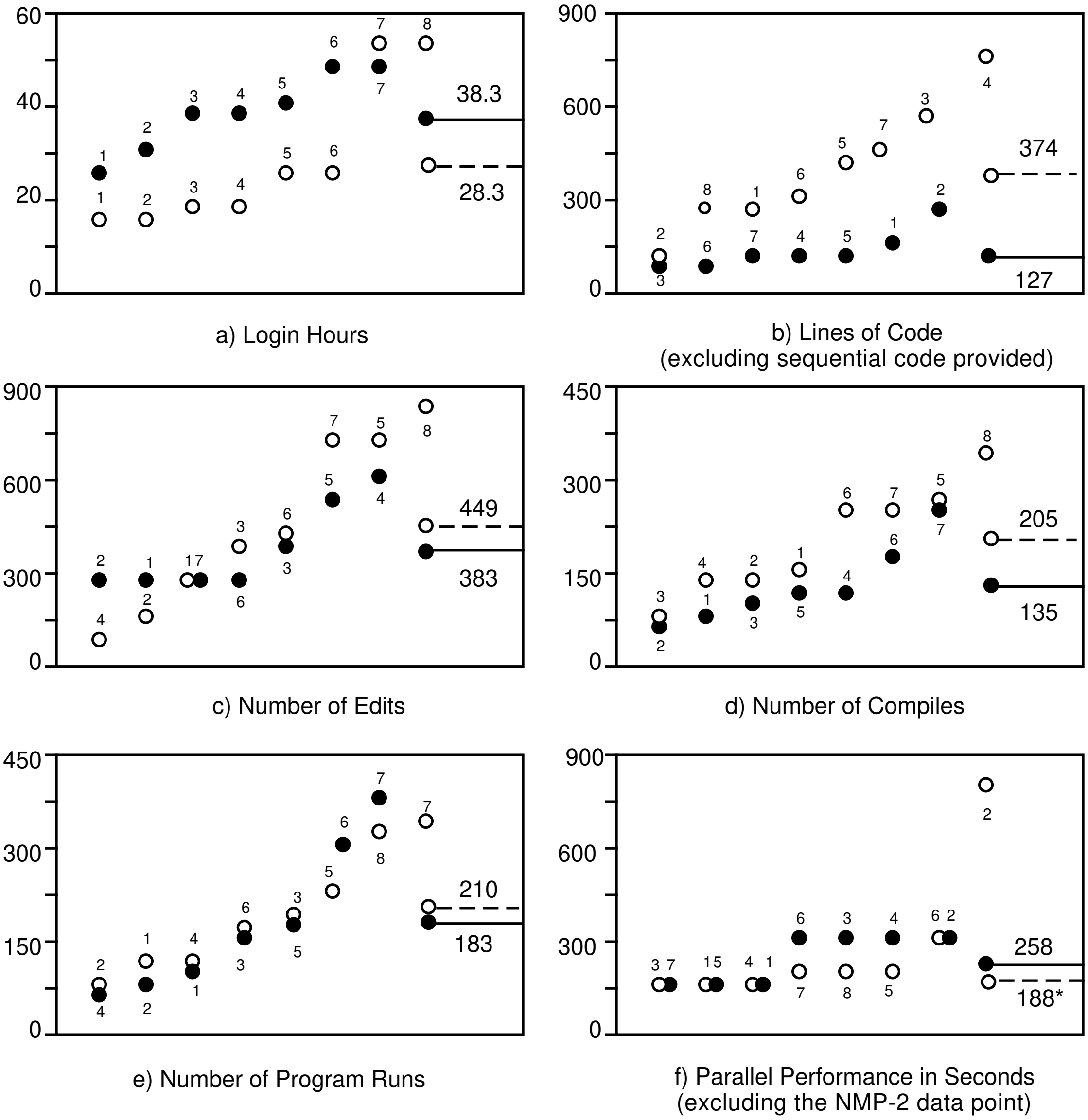

Figure 1: Experimental Results

2. It supported the claim that Enterprise is more usable than a message passing library.
3. It produced several direct benefits to the Enterprise PPS, including bug fixes, user-interface enhancements, extensions to the programming model and identified the need for faster compilation, more debugging tools and better documentation.
4. It identified some fundamental PPS independent concepts that are difficult for most novice parallel programmers to understand and indicated that these concepts should be stressed in the documentation for all PPSs. These include process startup, process termination and passing pointers between processes.

Although this is only a first attempt at measuring the usability of PPSs, the experiment nevertheless highlights the human factors issues that have been neglected to date. We propose that the above experiment (or variations on it) should be an integral part of the development cycle for parallel software tools. Given the diversity of programming systems available, researchers need more feedback as to what works well and why. We recognize that the cost of performing such quantitative measurements will be large. However, the cost of not performing them, as borne by a group which selects a low-usability PPS, will certainly be much larger.

Acknowledgements

Renee Elio, Randal Kornelson, Ian Parsons, Paul Iglinski, Robert Lake, Carol Smith and Bob Beck helped make this experiment possible. Many of the ideas in this paper originated from discussions with Greg Wilson. We would like to thank the CMPUT 507 class for their participation. This research has been funded in part, by NSERC grant OGP-8173 and a grant from IBM Canada Limited.

References

[1] R. Brooks. Studying Programmer Behavior Experimentally: The Problems of Proper Methodology. *CACM*, vol. 23, no. 4, pp. 207-213, 1980.

[2] G. Geist and V. Sunderam. Network-Based Concurrent Computing on the PVM System. *Concurrency: Practice and Experience*, vol. 4, no. 4, pp. 293-311, 1992.

[3] T. Marsland, T. Breitkreutz and S. Sutphen. A Network Multiprocessor for Experiments in Parallelism. *Concurrency: Practice and Experience*, vol. 3, no. 1, pp. 203-219, 1991.

[4] M. Rao, Z. Segall and D. Vrsalovic. Implementation Machine Paradigm for Parallel Processing. Supercomputing '90, ACM, New York, pp. 594-603, 1990.

[5] J. Schaeffer, D. Szafron, G. Lobe and I. Parsons. The Enterprise Model for Developing Distributed Applications. *IEEE Parallel and Distributed Technology*, vol. 1, no. 3, pp. 85-96, 1993.

[6] D. Szafron and J. Schaeffer. A Usability Study Comparing Two Parallel Programming Systems Technical Report 94-03, Department of Computing Science, University of Alberta. Available via anonymous ftp from ftp.cs.ualberta.ca.

[7] G. Wilson, J. Schaeffer and D. Szafron. Enterprise in Context: Assessing the Usability of Parallel Programming Environments. IBM CASCON, Toronto, pp. 999-1010, 1993.