Interactive Supercomputing's Star-P Platform: Parallel MATLAB and MPI Homework Classroom Study on High Level Language Productivity

Alan Edelman
Massachusetts Institute of Technology
and Interactive Supercomputing:
edelman@math.mit.edu

Parry Husbands
Interactive Supercomputing
and Lawrence Berkeley Lab
phusbands@interactivesupercomputing.com

Steve Leibman
Interactive Supercomputing
sleibman@interactivesupercomputing.com

Productivity through High Level Infrastructure

The thesis of this extended abstract is simple. High productivity comes from high level infrastructures. To measure this, we introduce a methodology that goes beyond the tradition of timing software in serial and tuned parallel modes. We perform a classroom productivity study involving 29 students who have written a homework exercise in a low level language (MPI message passing) and a high level language (Star-P with MATLAB client). Our conclusions indicate what perhaps should be of little surprise: 1) the high level language is always far easier on the students than the low level language. 2) The early versions of the high level language perform inadequately compared to the tuned low level language, but later versions substantially catch up. Asymptotically, the analogy must hold that message passing is to high level language parallel programming as assembler is to high level environments such as MATLAB, Mathematica, Maple, or even Python.

We follow the Kepner method [6] that correctly realizes that traditional speedup numbers without some discussion of the human cost of reaching these numbers can fail to reflect the true human productivity cost of high performance computing. Traditional data compares low level message passing with serial computation. With the benefit of a high level language system in place, in our case Star-P running with MATLAB client, and with the benefit of a large data pool: 29 students, each running the same code ten times on three evolutions of the same platform, we can methodically demonstrate the productivity gains. To date we are not aware of any high level system as extensive and interoperable as Star-P, nor are we aware of an experiment of this kind performed with this volume of data.

Star-P Architecture

The Star-P research project begun at MIT in 1998 [1,2,3] and is commercialized by Interactive Supercomputing, founded in 2004 (see [4]). Interactive Supercomputing's Star-P platform (architecture illustrated below) is designed to bring the first two author's dream of faster computing on larger data sets to the millions of scientists and engineers who wish to concentrate on their specialties rather than take the time and expense to learn how to write traditional parallel programs. In Star-P, MATLAB users insert the simple characters "*p" to tag large data sizes for data parallelism. Users identify the task parallelism when appropriate with a "ppeval" or parallel evaluate call reminiscent of feval for function evaluation. Their serial MATLAB code is transformed into parallel MATLAB code far more readily than traditional approaches

The Star-P 2.3 system appears to the user as a "parallel MATLAB" but Figure 1 below shows that architecturally Star-P is a language agnostic platform. In Star-P 2.3, users can write MATLAB codes and add serial and parallel extensions.

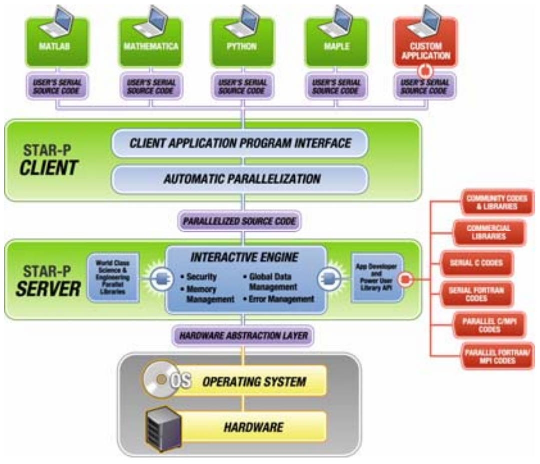
MATLAB
MATHEMATICA
PYTHON
MAPLE
CUSTOM APPLICATION
USER'S SERIAL SOURCE CODE
USER'S SERIAL SOURCE CODE
USER'S SERIAL SOURCE CODE
USER'S SERIAL SOURCE CODE
USER'S SERIAL SOURCE CODE
STAR-P CLIENT
CLIENT APPLICATION PROGRAM INTERFACE
AUTOMATIC PARALLELIZATION
PARALLELIZED SOURCE CODE
STAR-P SERVER
World Class Science & Engineering Parallel Libraries
INTERACTIVE ENGINE
• Security
• Memory Management
• Global Data Management
• Error Management
App Developer and Power User Library API
COMMUNITY CODES & LIBRARIES
COMMERCIAL LIBRARIES
SERIAL C CODES
SERIAL FORTRAN CODES
PARALLEL C/MPI CODES
PARALLEL FORTRAN/ MPI CODES
HARDWARE ABSTRACTION LAYER
OS
OPERATING SYSTEM
HARDWARE

Fig 1: Architecture of the Star-P Platform

MIT Graduate Class Experimental Data

The first author has been teaching a large cross section of graduate students at MIT since 1994 about the realities and myths of high performance computing (see [5]). He is proud that among his students have been the authors of FFTW, some of the authors of pMATLAB,[7,8] and of course many of the students who have worked on and tested Star-P (a project formerly known as MITMATLAB, pMATLAB itself, MATLABp, and MATLAB*p) most particularly the second and third authors.

This course has participated in performance studies as part of the development time study experiment of the HPEC program [6]. What has become increasingly clear from these studies is that a few very talented students who have the knack, can find ways to improve the performance of codes, but even the most talented and inclined still expend a great deal of time.

The students were given a by now standard programming assignment in parallel computing classes, the two dimensional Buffon needle problem. A typical parallel MATLAB solution in Star-P looked like:

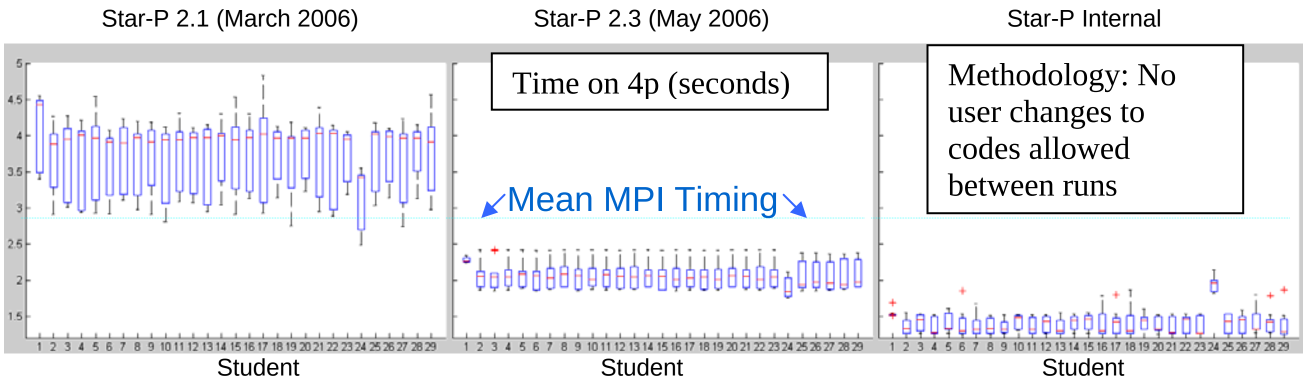

Fig 2: The Buffon Needle Problem executed by 29 students in three evolutionary versions of Star-P each executed ten times and compared with MPI runs written by the same students. The mean MPI timing was 2.8 seconds. We have not here normalized per student but we should report that a handful of students who worked hard achieved what might be considered the optimum of 1 sec on 4 processors in MPI. In a boxplot, the blue box ranges from the 25th to 75th percentiles of the ten data points. The red line is at the median. The whisker is the full extent of the data omitting outliers which are the red plusses. Writing message passing code was widely considered an unpleasant chore while the insertion of the two characters "*p" hardly seemed to be worthy of an MIT problem set.

```
function z=Buffon(a,b,l, trials)
r=rand(trials*p,3);
x=a*r(:,1)+l*cos(2*pi*r(:,3));
y=b*r(:,2)+l*sin(2*pi*r(:,3));
inside = (x >= 0) & (y>=0) & (x <= a) & (y <= b);
buffonpi=(2*l*(a+b) - l^2)/ (a*b*(1-
sum(inside)/trials));
```

The serial MATLAB code differs from the parallel one by the "*p" in red above. We ran each code ten times in three evolutions Star-P. Figure 2 plots the students timings on 4 processors (ten million trials).

We can only report anecdotal evidence about the human time for all 29 students, but overwhelmingly the students preferred adding the two characters "*p" to their code as compared to writing the MPI code. The mean time was 2.8 seconds on four processors. A handful of the students who were determined to performance tune their MPI code reached times close to 1 second. Thus the Star-P system brings users to within 40% of the hand coded optimum. The Star-P design allows for even this overhead to be shaved down further in future releases.

To understand scalability, the following times are the mean run times on the internal version of Star-P. (We note that the other versions of Star-P indicate similar scalability characteristics:) Each number is the average of 290 runs, 10 runs for each of 29 student codes.

Processors	1	2	4	8
Avg Seconds	5.7	2.9	1.4	0.7

Our view of this experiment is best illustrated as in the cartoon in Figure 3 which follows the productivity methodology introduced by Kepner and colleagues.

Conclusion

High level systems such as Star-P can allow users to write in high level languages such as MATLAB thereby providing the look and feel of a "parallel MATLAB." In much the same way that productivity has been obtained from underneath by faster cpu speeds, users of Star-P need not change codes between releases, and yet obtain faster execution as the infrastructure continues to squeeze out the best performance possible.

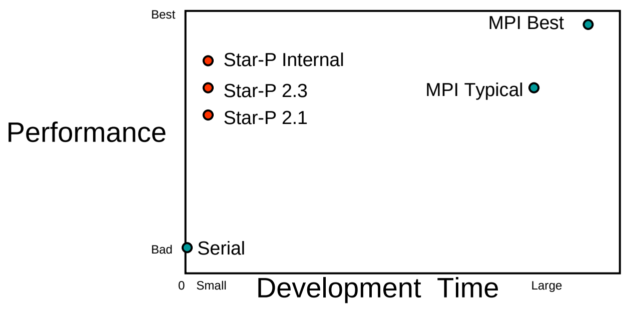

Fig 3: Kepner diagram illustrating the main point of this study. Productivity may be thought of as best slope on line to the origin. The vertical rise in performance of Star-P may be thought of as riding the technology curve as students expended no additional effort. Typical methodologies only report MPI vs serial on the vertical axis. The Kepner methodology provides the means of seeing productivity on a two dimensional scatter plot.

We thank Lorin Hochstein for his assistance in setting up the classroom studies. Funding for the studies was generously provided to the first author as part of the HPEC Productivity Study in the DOE Petascale Application Development Program. We particularly thank Jeremy Kepner for numerous interesting conversations and his leadership in the productivity area.

References.

[1] R. Choy and A. Edelman, "Parallel MATLAB doing it right," Proceedings of the IEEE, Vol.93, No.2, Feb 2005, pages 331-341.
[2] P. Husbands and C. Isbell, "The Parallel Problems Server: A Client-Server Model for Large Scale Scientific Computation." Proceedings of the Third International Conference on Vector and Parallel Processing. Portugal, 1998.
[3] P. Husbands, Interactive Supercomputing, PhD Thesis, Massachusetts Institute of Technology, Cambridge, 1999.
[4] Interactive Supercomputing: http://www.interactivesupercomputing.com.
[5] A Edelman, MIT Course 18.337: http://beowulf.csail.mit.edu.
[6] J. Kepner, http://www.highproductivity.org
[7] J. Kepner and S. Ahalt, "MatlabMPI," Journal of Parallel and Distributed Computing (JPDC), 64(8): 997-1005 (2004). (http://www.ll.mit.ede/MatlabMPI).
[8] N. Travinin and J. Kepner, pMatlab Parallel Matlab Library IJHPCA 2006. (http://www.ll.mit.edu/pMatlab).