A Relative Development Time Productivity Metric for HPC Systems

Andrew Funk and Jeremy Kepner
MIT Lincoln Laboratory
{afunk, kepner} @ ll.mit.edu

Victor Basili and Lorin Hochstein
University of Maryland
{basili, lorin} @ cs.umd.edu

Abstract

Software is often the dominant cost associated with developing DoD High Performance Embedded Computing (HPEC) systems. Historically there has been no quantifiable methodology for comparing the difficulty of developing code on different HPEC systems and trading off ease of development vs. execution performance. The DARPA High Productivity Computing Systems (HPCS) program is developing methodologies for the High Performance Computing (HPC) community, which may also be applicable to the HPEC community. This paper presents early results of one approach for measuring the relative development time productivity of different parallel programming environments. This metric, defined as the ratio of relative execution performance to relative programmer effort, has been used to analyze several HPC benchmark codes and classroom programming assignments. The results of this analysis show consistent trends for various programming models. This approach enables a high-level evaluation of relative development time productivity for a given programming model, which is essential to the task of estimating software development cost for HPC and HPEC systems.

Introduction

Software is often the dominant cost associated with developing HPEC systems, but historically there has been no method of quantifying and comparing the difficulty of developing code on different systems, or for weighing ease of development against execution performance. The HPCS program [1] is developing methods of quantifying the productivity of HPC systems, and these methods may also apply to HPEC systems. In the HPCS program, overall HPC system productivity, Ψ, has been defined as utility over cost:

$$\Psi = \frac{U(T)}{C_S + C_O + C_M} \quad [2]$$

The utility of the solution, U(T), is a (generally decreasing) function of time, and the denominator of the formula is a sum of software (C_S), operation (C_O), and machine (C_M) costs. In the special case of a lone researcher or team developing small codes (possibly for HPEC systems), we measure utility in terms of how fast the code runs, and assume we are only concerned with the cost, or effort required, to develop the code (C_S). If we then normalize the parallel performance and cost with respect to those of a corresponding serial code, we arrive at a formula for relative development time productivity:

$$\frac{\text{Relative Development}}{\text{Time Productivity}} = \frac{\text{Speedup}}{\text{Relative Effort}}$$

For the analyses in this paper, relative effort is calculated as the ratio of parallel to serial code size. To test this metric, we have applied it to two HPC benchmark suites. The HPC Challenge suite [3] consists of several activity-based benchmarks designed to test various aspects of a computing platform. The four benchmarks used in this study were FFT (v0.6a), High Performance Linpack (HPL, v0.6a), RandomAccess (v0.5b), and Stream (v0.6a). The NAS Parallel Benchmark (NPB) [4] suite consists of five kernel benchmarks and three pseudo-applications from the field of computational fluid dynamics. In addition, we have applied this metric to data collected from a series of classroom parallel programming assignments.

Analysis and Results

The HPC Challenge benchmarks were run on 64 dual-processor nodes connected by Gigabit Ethernet [5]. To compare low- and high-level languages, each benchmark was implemented in serial C, C+MPI, Matlab, and pMatlab. The execution time of each implementation was measured and normalized with respect to serial C to compute speedup. Similarly, the relative effort required for each implementation was computed by normalizing its size, measured in Source Lines of Code (SLOC), with respect to serial C. The relative development time productivity for each implementation was then calculated by dividing speedup by relative effort.

The results for the HPC Challenge benchmarks are presented in the first column of Figure 1. (The implementations of Random Access require a great deal of inter-processor communication, and so actually run slower as more processors are involved in a network cluster.) With the exception of Random Access, the MPI implementations all fall into the upper-right quadrant of the top graph, indicating that they deliver some level of parallel speedup, while requiring larger code sizes relative to serial C. As expected, the serial Matlab implementations do not deliver any speedup, but do all have smaller relative code sizes. The pMatlab implementations (except Random Access) fall into the upper-left quadrant of the graph, delivering parallel speedup while requiring smaller relative code size. The

This work is sponsored by Defense Advanced Research Projects Agency, under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

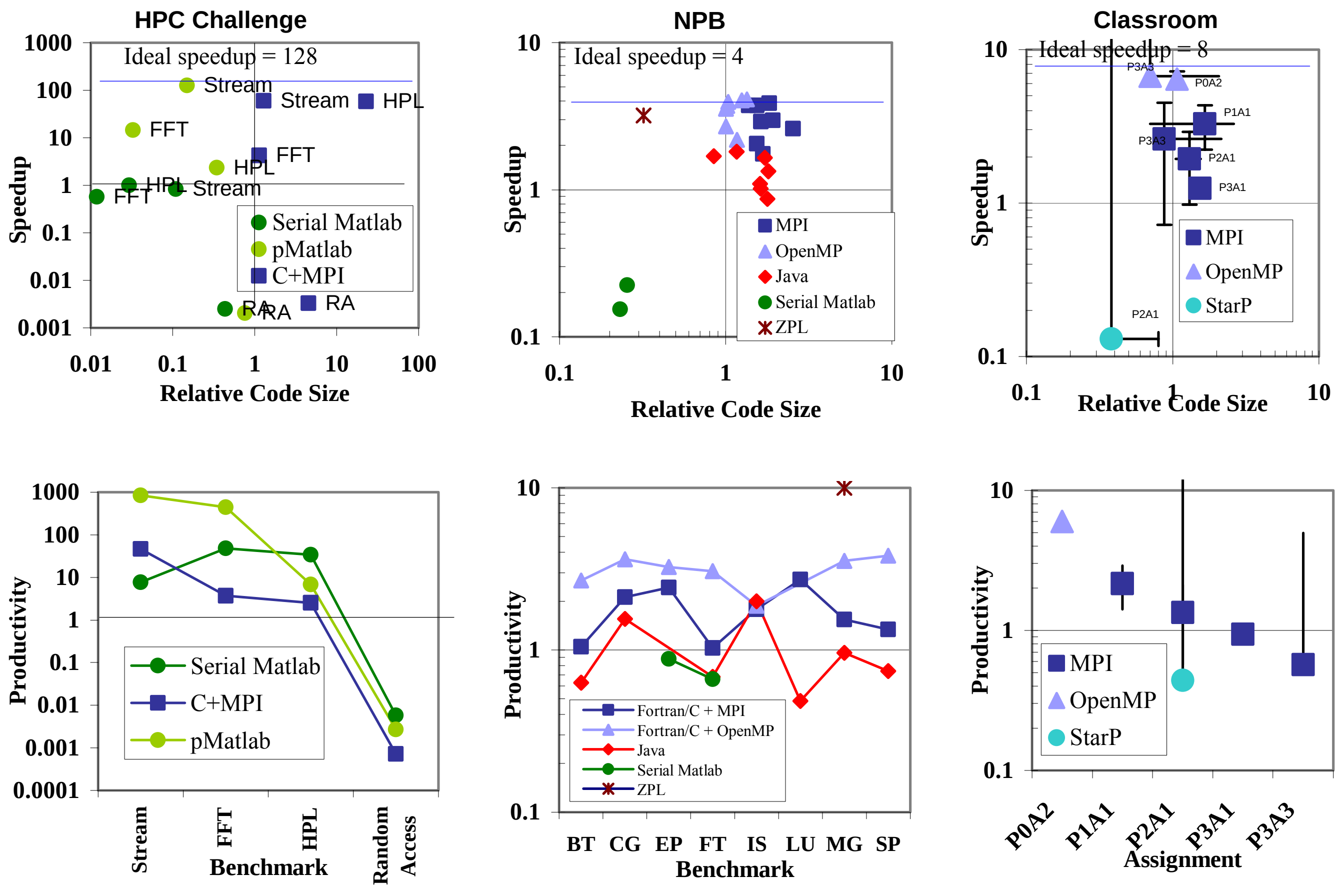

Figure 1: Top row: Speedup vs. Relative Code Size. Bottom row: Relative Development Time Productivity

combination of parallel speedup and reduced relative code size means that the pMatlab implementations generally have the highest relative development time productivity values of the three (Figure 1, bottom left).

The NPB codes were run on an IBM p655 multiprocessor computer using the Class A problem size and four processors (in the parallel case). The speedup, relative code size, and relative development time productivity were calculated in the same manner as for HPC Challenge. The results for NPB are shown in the middle column of Figure 1. These results show that OpenMP tends to have speedup comparable to MPI, with a smaller relative code size (Figure 1, top center). This is reflected in the higher relative development time productivity values for OpenMP (Figure 1, bottom center). As a general rule, we expect to see traditional parallel languages and libraries such as MPI and OpenMP fall in the upper-right quadrant of the graph. This reinforces our intuition that parallel performance is achieved at the cost of additional effort (over serial implementation).

A series of classroom experiments was conducted for the HPCS program, in which students from several different classes were asked to produce parallel programming solutions to a variety of textbook problems. In most cases the students first created a serial program to solve the problem, and this was used as the baseline for comparison with their parallel solution. The students used C, Fortran, and Matlab for their serial codes, and created parallel versions using MPI, OpenMP, and Matlab*P (aka StarP, a parallel extension to Matlab) [6]. The students ran their programs on a variety of computing platforms, and reported their own timings. All speedups were calculated using eight processors for the parallel case. Relative code size and relative development time productivity were calculated in the same manner as with the benchmark codes.

The results for the classroom assignments are presented in the right column of Figure 1. The speedup and relative code size were collected for each student, and the median values for each assignment are plotted on the graph. The ideal speedup in this case is eight. Some outlier data is not shown, and error bars are one standard deviation from the median. The MPI data points for the most part fall in the upper-right quadrant of the graph, resulting in development time productivity values at or above one (Figure 1, top right). The OpenMP data points indicate a higher achieved speedup compared to MPI, while also requiring fewer lines of code. This yields higher relative development time productivity values for OpenMP (Figure 1, bottom right).

Conclusions

We have introduced a common metric for measuring relative development time productivity of HPC software. This metric has been applied to data from benchmark codes and classroom experiments, with consistent results. In general the data supports the belief that MPI implementations yield good speedup but have larger relative code sizes than other implementations. OpenMP generally provides speedup comparable to MPI, but with smaller relative code size. This yields higher relative development time productivity values. The pMatlab implementations of HPC Challenge provide an example of a language that can yield good speedup for some problems, while requiring smaller relative code size, again leading to higher values of the relative development time productivity metric.

References

[1] High Productivity Computer Systems http://www.HighProductivity.org

[2] Kepner, J. "HPC Productivity Model Synthesis." *IJHPCA Special Issue on HPC Productivity*, Vol. 18, No. 4, SAGE 2004

[3] HPC Challenge. http://icl.cs.utk.edu/hpcc/

[4] NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/

[5] Haney, R. et. al. "pMatlab Takes the HPC Challenge." Poster presented at High Performance Embedded Computing (HPEC) workshop, Lexington, MA. 28-30 Sept. 2004

[6] Choy, R. and Edelman, A. *MATLAB*P 2.0: A unified parallel MATLAB*. MIT DSpace, Computer Science collection, Jan. 2003. http://hdl.handle.net/1721.1/3687

[7] Funk, A., Basili, V., Hochstein, L., Kepner, J. "Application of a Development Time Productivity Metric to Parallel Software Development." Second International Workshop on Software Engineering for High Performance Computing System Applications (SE-HPCS). St. Louis, MO. 15 May 2005.