

HPC Productivity: An Overarching View

Dr. Jeremy Kepner[†]
kepner@ll.mit.edu
MIT Lincoln Laboratory
244 Wood St., Lexington, MA 02138

March, 2003

Abstract

The DARPA High Productivity Computing Systems (HPCS) program is focused on providing a new generation of economically viable high productivity computing systems for national security and for the industrial user community. The value of a High Performance Computing (HPC) system to a user includes many factors, such as execution time on a particular problem, software development time, direct hardware costs and indirect administrative and maintenance costs. This special issue focused on HPC Productivity brings together—for the first time—a series of novel articles written by several distinguished authors who share their views on this topic. The topic of productivity in HPC is very new and the authors have been encouraged to speculate. The goal of this first article is to present an overarching context and framework for the other articles and define some common ideas that have emerged in considering the problem of HPC productivity. In addition, this article defines several characteristic HPC Workflows that are useful for understanding how users exploit HPC systems, and discusses the role of activity and purpose benchmarks in establishing an empirical basis for HPC productivity.

1 Introduction

The DARPA High Productivity Computing Systems (HPCS) program is focused on providing a new generation of economically viable high productivity computing systems for national security and for the industrial user community. HPCS researchers have initiated a fundamental reassessment of how we define and measure performance, programmability, portability, robustness and ultimately, productivity in the HPC domain. The value of a High Performance Computing (HPC) system to a user includes many factors, such as execution time on a particular problem, software development time, direct hardware costs and indirect administrative and maintenance costs. The HPCS program is developing systems that deliver increased value to users at a rate commensurate with the rate of improvement in the underlying technology. For example, if the relevant technology was silicon, the goal of such a system would be to double in productivity (or value) every 18 months, following Moore's Law. The key question is how do we define and measure productivity, and what are the underlying technologies that affect productivity. Addressing this question is the focus of this Special Issue and

[†]This work is sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

will lead us to a fundamental reassessment of how we define and measure performance, programmability, portability, and robustness in the HPC domain.

This special issue on HPC Productivity brings together—for the first time—a series of novel articles written by several distinguished authors who will share with you their views on this topic. The ten articles are organized as follows. This first article aims to present an overarching context and framework for the other articles and defines some common ideas that have emerged in considering the problem of HPC productivity. The second article by Doug Post looks at several very large HPC codes and articulates some of the key productivity lessons learned from those experiences. The third article by Marc Snir and David Bader develops a model of productivity based on the economic concept of time dependent utility; they go on to discuss empirical approaches for measuring productivity. The fourth article by Thomas Sterling presents a conceptual productivity framework and a rigorous mathematical formulation of productivity based on the rate of work performed and overall system cost. The fifth article by Ken Kennedy, Charles Koelbel and Rob Schreiber looks specifically at characterizing productivity of programming environments in terms of their inherent expressiveness and their execution efficiency. The sixth article by Bob Numrich focuses on an approach for optimizing overall time to solution using the physics inspired concept of least action. The seventh article by Larry Votta looks at the question of productivity from an empirical software engineering perspective. The eighth article by John Gustafson looks more closely at how one can benchmark productivity and describes purpose based benchmarking. The ninth article by David Kuck examines productivity from the perspective of various actors in the marketplace. Finally, in the last article, I attempt to synthesize and unify much of the work presented by the other authors and begin the process of putting it on an empirical foundation.

2 Productivity Framework

The measurement of productivity for a particular user on a particular system with a particular application is a difficult question that must encompass a variety of concepts:

Productivity which quantitatively defines utility and costs

Workflows that describe the time spent by both the programmer and program in different stages of activity

Productivity Metrics that characterize system specify capabilities

Execution Time of an application or benchmark

Development Time of an application or benchmark

Execution Interface between a program and the hardware

Development Interface between the programmer and the system

Benchmarks for measuring performance

System Model for predicting performance (including reliability and portability)

Modeling Interface for performance prediction tools

System Parameters that define the capability and can be used to predict productivity

Figure 1 diagrams one way these concepts might be put together. Specifically, we envision productivity as the utility (as a function of cost) that the system provides to particular users (or set of users) on a particular application (or set of applications). Productivity is strongly influenced by the workflow of the user (e.g. time spent running vs. time spent programming). Independent of the user, we would like to define productivity metrics that can characterize the execution time (e.g. clock time on activity benchmarks) and development time (e.g. total effort to implement purpose benchmarks) of as system. Furthermore, we would like to model the system so we can predict its capabilities. The details of the models will be complex functions of the architecture, but several interfaces should be readily definable: the execution interface (operating system, memory model, thread model, ...), the development interface (languages, libraries and tools), and the modeling interface which encompasses system parameters such as number of cpus, clock rates and bandwidth.

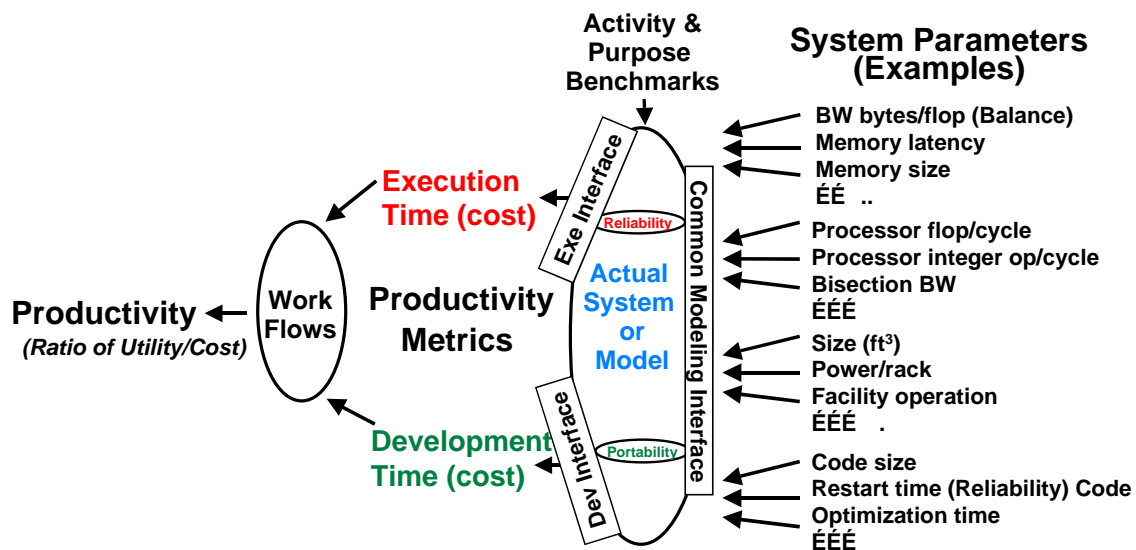


Figure 1: Productivity Assessment Framework. The goal of the framework is to provide a mechanism for integrating system specific capabilities with user specific needs to assess the value of a particular machine for a particular mission.

The ultimate goal of the framework is to provide a means for reasoning about systems. Specifically, the framework should be helpful to several important classes of users. Below we describe these users, their goals and a possible approach they might employ:

HPC Acquisition Community

Goal: Compare vendor products

Approach: Run benchmarks and/or execution models, select development model, select workflows, compute “productivity”

HPC End User

Goal: predict effort required to achieve performance

Approach: select workflows, development model, compute “productivity”, compare with baseline data

HPC Vendor/Designer

Goal: maximize user base

Approach: select system parameters, development environments, compute “productivity” for different workflows

HPC Technology Researcher

Goal: measure technology impact

Approach: apply benchmarks, apply development time experiment methodology, compute change in “productivity”, compare with baseline

3 Workflows

How a system is used is clearly one of the most important pieces of information for assessing value and productivity. Figure 2 shows three canonical workflows that encompass many HPC users: individual researchers, enterprise, and production/operations. Each workflow is broken up into two iterative cycles. The overall cycle describes the stages a mission goes through to meet its goal. The development cycle refers to the process of actually writing the software to be run on an HPC system. These workflows are not specific to HPC, but they do contain a common element which is unique to HPC: the process of porting to an HPC system, testing and validating at scale, and optimizing for performance.

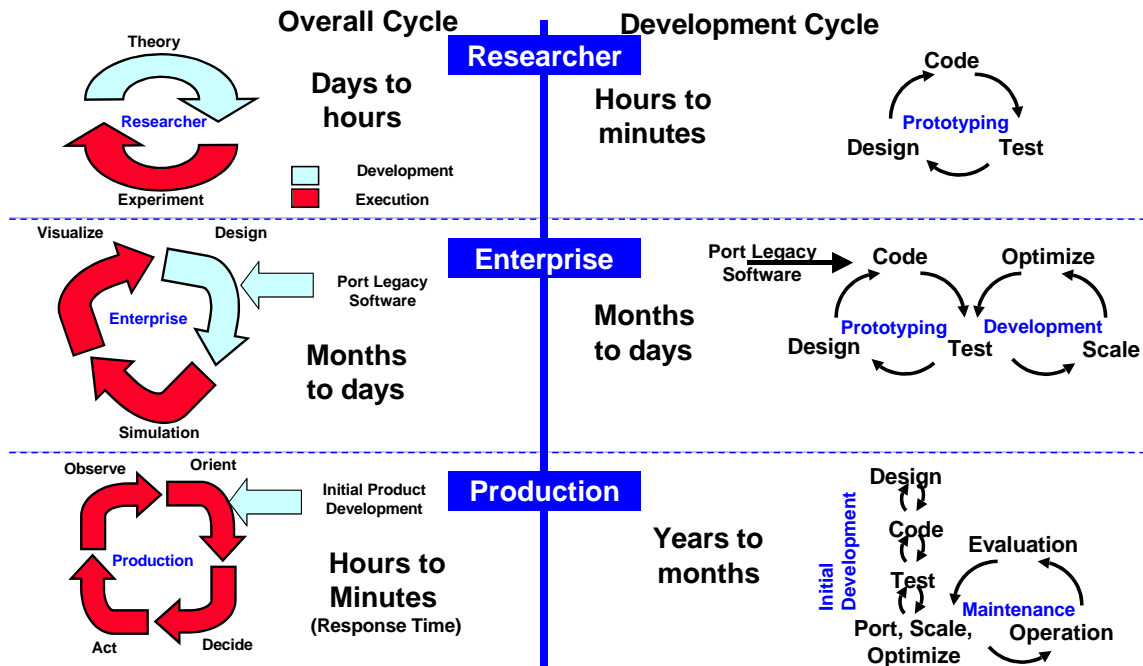


Figure 2: HPC Workflows. The three canonical workflows that are typically used in the HPC community. Each workflow has a mission cycle and a development cycle.

The researcher workflow addresses the case of an individual researcher who is principally focused on a knowledge discovery process. Their HPC computations are often viewed as experiments to gain scientific insight, and they often iterate between development and execution on relatively short time scale (days to hours). The individual researcher software development process most closely resembles that found in rapid prototyping and is akin to agile programming methods.

The enterprise workflow applies to organizations focused on developing and integrating very large codes that are simulating multiple coupled processes. New modules are designed in a rapid prototyping manner, but then must be integrated into larger legacy codes. The overall cycle often involves simulations that may run for weeks and require significant resources to visualize and understand. The software development process is similar to those used in iterative programming.

The production workflows applies to the case where the mission goal is to process data to support an OODA loop (Observe, Orient, Decide, Act), which in the field may be executed on a very short time scale (hours to minutes). However, since this is a deployed system, the development process may take years and follows a more traditional waterfall development process that takes the code through design, code, test, and on into the maintenance phase.

4 Benchmarks

Benchmarks are very important part of HPC productivity. Benchmarks and the associated metrics used to evaluate them can have a large impact on the design process. The HPC community has enormous experience with execution time benchmarks that look solely at the compute performance of a system (e.g. Top500, STREAM, and NAS parallel benchmarks). However, current HPC execution time benchmarks favor large caches and deep pipelines and produce systems ill-suited to many applications with low spatial locality (i.e., data accesses are not localized) and low temporal locality (a small number of operations are performed on each data value). The HPC community has considerably less experience in the area of development time, where generally no metrics are used, which results in least common denominator standards that are often difficult to use and difficult to optimize.

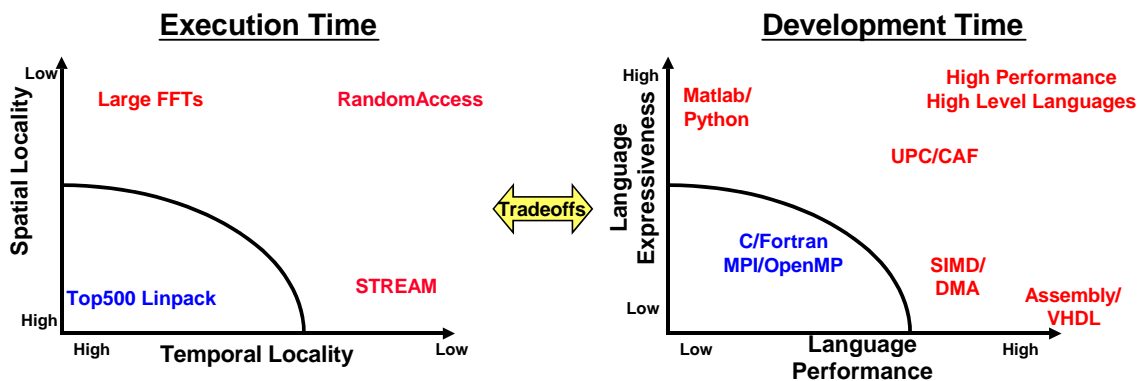
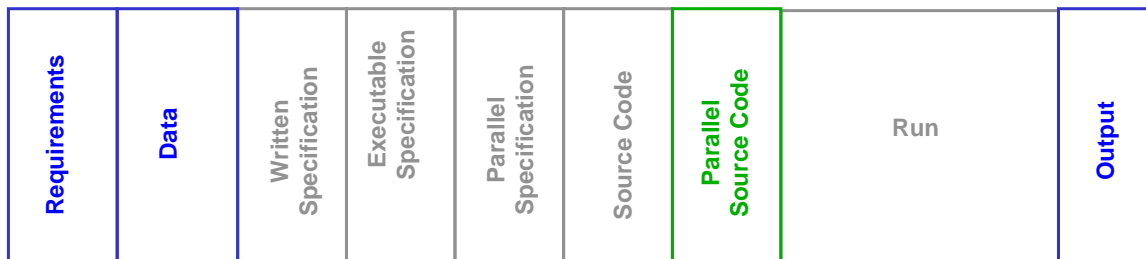


Figure 3: Benchmarks Drive Design. The goal of future benchmarks is to allow quantitative tradeoffs between execution time and development time.

Figure 3 illustrates just two of the many possible dimensions that can be explored with benchmarks. In the case of execution time, we can construct benchmarks that more fully span the dimensions of spatial and temporal locality by looking at streaming and random access type benchmarks. In the development time area, the dimensions of language expressiveness and language performance are shown, with high level languages (e.g. Matlab and Python) being the most expressive and low level languages (assembly and VHDL) giving the best performance. Ultimately, we would like to have benchmarks that allow us to look at both execution time and development time simultaneously so that tradeoffs can be made between ease of programming and performance.

One approach to gaining a deeper insight into both execution time and development time is through the use of activity and purpose benchmarks (see Figure 4). Activity benchmarks are traditional benchmarks that typically consist of a specific code that is to be run and timed on a system. Activity benchmarks provide no opportunity for different implementations to be tried and provide little insight into the development process. Purpose benchmarks are higher level descriptions of a problem to be solved and may consist of just goals, requirements, inputs and outputs. The NAS parallel benchmarks are an example of benchmarks that lie somewhere in between and are both very specific with reference implementations, but have sufficient high level descriptions to have been implemented in many different programming environments.

Activity and Purpose Benchmarks



Measure Different Parts of the Workflow

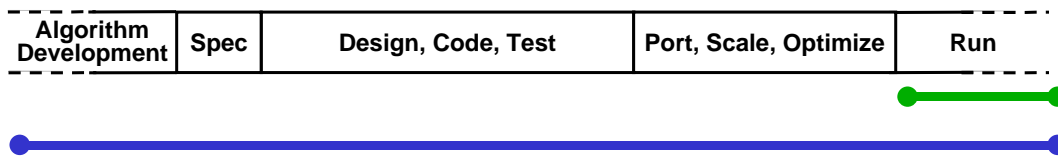


Figure 4: Activity and Purpose Benchmarks. Activity Benchmarks define a set of instructions (i.e., source code) to be executed. Purpose Benchmarks define requirements, inputs and output. Together they address the entire development workflow

5 Conclusions

The value of a High Performance Computing (HPC) system to a user includes many factors, such as execution time on a particular problem, software development time, direct hardware costs and indirect administrative and maintenance costs. This special issue focused on HPC Productivity brings together—for the first time—a series of novel articles written by several distinguished authors who share their views on this topic. The

goal of this first article is to present an overarching context and framework for the other articles and define some common ideas that have emerged in considering the problem of HPC productivity. In addition, this article defines several characteristic HPC Workflows that are useful for understanding how users exploit HPC systems, and discusses the role of activity and purpose benchmarks in establishing an empirical basis for HPC productivity.