White paper for the ASCR workshop on software productivity for extreme-scale science

Robert J. Harrison

Center for Scientific Computing, Brookhaven National Laboratory
and
Institute for Advanced Computational Science, Stony Brook University

Productivity should be viewed holistically and include overall system throughput and costs

Presently, the configuration, performance, productivity, and cost of ownership of leadership computer systems are only analyzed as if 100% of the time they are running a single application, but this is far from the case. Sharing is presently performed by partitioning the machine and dedicating sub-groups of nodes to individual applications, which in turn are written according to this assumption using programming models that poorly, if at all, support any sense that resources could be shared. Looking forward, and probably even now, it is clear that this is far from optimal – by not fully sharing resources we are letting vast amounts of memory sit idle, wasting memory/communication/IO bandwidth, and squandering CPU cycles and functional units. How much money is spent on machine features that individually are only fully utilized 5% or less of the entire lifetime of the machine?

Time and space sharing of nodes/cores, interleaved execution, shared low-level task queues, and intelligent resource management would enable, for instance, a memory hungry and bandwidth-limited application such as a large astrophysics simulation to co-exist with a compute intensive and low-memory application such as quantum Monte Carlo. Massively-threaded node architectures are actually designed with sharing in mind – lots of thread units to hide memory latency and maximize use of memory bandwidth, lots of individual caches to reduce contention, and hyperthreading to interleave instruction streams. Only memory capacity (local and global) seems to absolutely constrain an application's ability to share the machine, and the performance variability that inevitably arises from both energy/power management and space sharing of the system, will already force applications to use dynamic scheduling. Naive sharing or simple hyperthreading could easily cause more problems than benefits, but appropriate programming models with resource-aware intelligent runtimes and O/S support could credibly bring huge gains in utilization and system throughput – these are many of the same concepts being discussed in the context of programmer productivity. And applications that demand it (and can pay for it) could still be run in dedicated mode.

Big data = big compute

Integrating data-centric and performance-centric computing, including streaming data between extreme-scale experiments and extreme-scale computers, is still an open research topic but potential use cases and benefits are quite apparent. Example use cases might be integrated simulation for control of an operating fusion experiment, or intelligent data reduction from high bandwidth experiments on next generation detectors at NSLS-II or LHC. Increasingly, experiments simply cannot keep the vast quantities of data they produce and coupling high-end computing is essential to reduce the data, and then to compute with the reduced forms. Absent established concepts, frameworks, and tools, this is a both a research and a productivity issue. How to compose these applications? What are the reusable or transferrable concepts?

Agile codes that can adapt to disruptive architecture and science trends over time

Nearly all science codes have been caught flat footed by the many-/multi-core era along with the renewed emphasis on vectorization for high floating-point performance and the increasing complexity of the memory hierarchy. It is not just the technology but its pace of change, and this pace is anticipated to continue at least until the early exascale era. Some unifying concepts have emerged, but these are insufficient to help all but a small

set of kernels, and, moreover, the focus is still unfortunately mostly on current rather than probable future architectures. The big picture is that we must express all available concurrency in a manner that can be transformed by compilers and other tools to match the underlying hardware. This seems to be a nearly intractably hard problem, and we are left looking beyond compiler technology for solutions. Autotuning is too limited. DSLs are a powerful solution for sufficiently narrow and structured problems (e.g., the Tensor Contraction Engine), but are by no means a solution for all. APGAS seem to powerfully address some of the coarser-grain concurrency issues by neatly synthesizing task and data parallel programming, but fail to advance fine grain concurrency and end up more a tool for the library writer than the application scientist. It is also not just computer architecture that is in flux – science itself is constantly advancing along with the underlying numerical representations and algorithms. We need to rewrite that 1M LOC not just to exploit the next technology, but because it will open new science frontiers. Thus, the mantra should be "agile" in the face of opportunity and change.

We must examine the relevance of old concepts such as telescoping languages and other hierarchical program representations, that seem to have been set aside in the constant push for new research topics, and the success of high-level programming from specification exemplified by SPIRAL. SPIRAL's developers have a quote along the lines of "as soon as you start writing code there is an immediate expiration date stamped upon it, whereas the math lives forever." It takes enormous effort, circa a graduate student career, to specify everything about a kernel and an architecture that SPIRAL needs to generate code within 1% of the very best attainable performance – but what about 90% or just 25%? 90% would be fantastic for most computational kernels, and 25% or even worse adequate for most non-performance critical code. This is much more than DSLs – it is about encoding and exploiting *knowledge*. It is about a near complete separation of concerns that is very poorly done by current tools.

A sustainable software ecosystem

How could we significantly improve productivity across the board, drive standards to more rapidly adopt new concepts, and remove some of the barriers to adopting products of C/S research? A standard software ecosystem for extreme scale computing that spans from the laptop to the exaflop computer.

What is the ecosystem? How should we deploy it? We need a few, small, agile, coordinated pilot projects rather than a bloated multi-lab project. E.g., in the spirit of co-design, develop a lightweight software distribution to nucleate a broader community-based effort to develop a standard extreme-scale, scientific software environment for a few standard Linux distributions, Apple laptops, the leadership computing centers, and NERSC. Not only will this drive a lot of inherent complexity and waste out of the entire software and science lifecycle, given confidence that tools are everywhere people will start to exploit things like source- to source translation for DSLs, autotuning, etc., and the ecosystem can become the place for new tools to grow.

Productivity and performance is not just about the software – what about the math?

Fast algorithms, in the math sense, are a visibly missing element in the exascale program. They are optimal in several senses, and although sometimes slower for small systems (due to a sometimes large prefactor) their superior computational complexity and numerical robustness make them preferred for large systems. Typically they are tree-based with computationally intense kernels, so are well suited to shepherding data up and down the memory hierarchy with good reuse. They often replace spectral methods that have expensive global communications. But they are complex to implement (accurately and with optimal single node speed), and their non-uniform adaptive refinement and sometimes irregular communication can represent scaling and programming challenges. We can broaden this observation to other representations and algorithms for which superior versions exist but are not broadly adopted for lack of awareness or standard implementations (e.g., replacing smooth particle dynamics with the much superior Lagrangian particles). Faster, more robust, higher-order methods will enhance the science as well as the performance, and go far beyond the concept of "solvers."