Robert F. Lucas
Computational Sciences Division
Information Sciences Institute
University of Southern California
rflucas@isi.edu

Musings on the Path Forward to Exascale

In the hey-day of supercomputing, when Cray Research was the darling of Wall Street, scientists and engineers in both the public sector, Universities and National Laboratories, as well as those in industry used the same computer systems. Sometimes, they used the same codes, such as NASTRAN, which was initially developed by NASA GSFC and later distributed World-wide by independent software vendors (ISVs), like today's MSC Software. In other fields, such as nuclear weapons design, for which there is no commercial market, government-funded scientists could still leverage the same rich software ecosystem of operating systems, compilers, numerical libraries, and debuggers as was available to their colleagues in industry.

The advent of distributed memory, message-passing systems dramatically changed the above status quo. The codes that consumed the most supercomputer cycle time were often highly specialized to the Cray architecture, and it was not practical, if even feasible to port them. In Labs and academe, a new generation of capability codes was developed, often from scratch, i.e., designed form the beginning to exploit the new systems. This also required the development of a new software ecosystem with numerical libraries, debuggers, etc. Passing the burden of orchestrating data distribution and communication to the user (i.e., MPI) at least allowed most us to continue to use standard languages and compilers on individual processing nodes.

While they were much slower to do so, industrial users have now also adopted distributed memory systems. More and more of today's mainstream commercial software exploits thread level and even message-passing concurrency, though scaling of these codes is usually quite limited. Thus, an automaker with thousands of CPUs will not launch a handful of large capability computations, designed to explore some novel design, but rather will launch a large ensemble of modest jobs (~32 processors), each evaluating a small perturbation in their design space.

The divergence of public and industrial use of supercomputers had a deleterious impact for all involved. The market for high end systems stopped growing, and many system vendors left the market. Many users found that they could lower the cost of computation over the course of the last decade, but could not increase the scale and fidelity of those same computations [ref. Vince Scarafino, Ford Motor Company].

As we look forward to Exascale, there are reasons to believe that we will face a transformation similar to that experienced in the early 1990s, when distributed memory stopped being a curiosity, and went mainstream. The rate at which users and their tools must expose additional concurrency is actually increasing, and by the dawn of the

Exscale era could exceed 10^9. Meanwhile, the ratio of Bytes to Flops could drop by orders of magnitude as DRAM sees the end of its Moore's Law growth earlier than logic circuits. This in turn will almost certainly lead to a new memory hierarchy as technology like Flash fills the void. Heterogeneous systems like RoadRunner may become prevalent.

An obvious question that arises is how do we learn from our past, and manage this next transformation so that it is not as disruptive as the last one? The thesis of this white paper is that we need to do so in multiple ways. First, we must evolve our systems and software, whenever possible, in a manner that is predictable by users and developers. There is more value in application software today than there is computing systems. There are applications in use today that are forty or more years old (e.g., NASTRAN), and these applications can be expected to grow over the course of the next decade. The developers of these codes must be provided with a path to the future that allows them to incrementally add new features and anticipate changes in computing systems. Note, the transition from scalar to vector circa 1980 was evolutionary for most developers.

Secondly, we must remember that computer systems exist to solve problems for their human users. Thus Exascale systems must be co-designed with the applications that they will ultimately run. Building message passing systems was expedient for the system architects, punting to application developers the hard problems of distributing and coordinating the computation. As the level of concurrency approaches 10^9, this will no longer be feasible. We will not be able to tolerate unnecessary overheads in communication and synchronization, lest Amdahl fractions preclude users from making practical use of such systems. This author believes we should start an Exascale system program with the scientific and engineering challenges it will be expected to solve. In such a design, we must consider existing software to be an important boundary condition.

Finally, there is concern that the reliability of systems will begin to decline as we approach Exascale. The scale of the systems and the number of components involved is increasing. Worse, as VLSI geometries continue to shrink, the long-term reliability of integrated circuits will decrease and they may become increasing vulnerable to transient failures. Unfortunately, in mainstream science and engineering, the programming model has always been that the system is reliable, and simple measures like checkpoint/restart would be adequate for the rare exceptions. I firmly believe that every attempt must be made by computer hardware and system software to continue to isolate software developers and end users from any reduction in component reliability. Otherwise, we will poison the ecosystem and can expect to see fewer and fewer users of capability systems.

.