

Research Plans

A Research Statement focused on the future

Laxmikant Kale

My research continues to be aimed at improving performance and productivity in parallel programming. To that end, during the past ten years, I have extensively developed the idea of migratable objects, and embodied it in parallel programming systems called Charm++ and Adaptive MPI. In the coming years, enabled by the core idea of migratable objects, I think we can significantly improve the state of art in parallel programming with research on adaptive runtime systems, new parallel programming abstractions derived from years of experience in parallel CSE applications, domain-specific frameworks, and performance analysis. Further, specialized runtime strategies based on migratable objects will also enable effective exploitation of resources across the wide-area computational grid.

One of the tenets of my research has been “application oriented yet computer science centered research”. I believe that I can do computer science research with long lasting impact only if I base it on direct experience with CSE applications. Without such direct experience, which provides a fertile ground for generation as well as validation of ideas, research in parallel programming will likely remain isolated from real issues faced by parallel programmers. At the same time, being centered on a single application will not generate CS research that is widely applicable. I favor an approach where I work with multiple applications, directly and facing the nitty-gritty needs of applications that are used by scientists and engineers, as well a deeper parallelization challenges, and where I develop common abstractions that are useful across multiple applications, layering them so that they can be used by an even wider class of applications.

This path led me first to the idea of migratable objects: if a program is expressed as a collection of migratable objects that interact with each other by asynchronous method invocations, the runtime system is empowered to optimize execution by optimally moving objects among processors, and mediating communication. This approach, I think, presents a methodology where the parallel programmers do what they can do best (namely, problem decomposition) while leaving to the system what it can, at least potentially, do best (namely automated resource management, including dynamic load balancing). Of course, to realize the potential, one must develop increasingly sophisticated runtime strategies, which has defined a significant part of my past and current research agenda (see Runtime Strategies below).

I believe that my approach has been proved successful by now, even though it (by nature) has been slow and tedious. Charm++ and AMPI (Adaptive MPI) are now established systems, with over 10,000 installations; The runtime techniques have proved their utility in parallelizing complex applications including NAMD, the parallel Molecular Dynamics application used by biophysicists for which our paper co-won the Gordon Bell award in 2002.

Consistent with the above approach, my future research includes runtime strategies (including fault tolerance, load balancing and communication optimizations), programming abstractions, domain-specific frameworks, resource management on computational grids, and applications within computer science as well as outside.

1 Adaptive Runtime Systems

We have attained excellent research results with adaptive runtime systems, but I consider most of of that low-hanging fruit, with much that remains to be done. Migratable objects, and a runtime system that mediates in their communication, provides good mechanisms for automatic

adaptation. The *principle of persistence* that we enunciated (although it is intuitively obvious once stated) creates potential for measurement-based adaptation, since the RTS can automatically collect object-level instrumentation.

(The principle of persistence states that once an application has been expressed in terms of its natural objects and their interactions, their computational loads, and communication characteristics *tend to persist* over time, allowing recent past to be a good predictor of near future).

Although we have written first generation of load balancers and communication optimizers based on this idea, a more sophisticated set of strategies are possible and necessary to tackle issues such as handling dependence structure among the objects, dealing with multiple time-stepping and multi-phase load balancing in general, handling emerging topologies of real machines effectively, and handling communication-latency-vs-overhead trade-offs. Another promising direction is to provide “control-points” (distributed call-backs) whereby application components can be controlled/tuned by the RTS.

The ARTS are currently used via Charm++ and AMPI (Adaptive MPI, a full implementation of MPI based on Charm++). A related objective I have is to establish AMPI as a competing state-of-art implementation used by programmers routinely.

2 Languages and Abstractions

After years of experience with applications using Charm++ and AMPI, I finally think I have a handle on the ultimate (or at least “the next”) set of abstractions — call it “languages” if you wish — for parallel programming. Essential ingredients for such a language are: migratable work-units and data units (objects), a highly restricted/disciplined global address space for a subset of data, ability to provide a global view of control, and compositionality without sacrificing performance. Along with my students, have made some preliminary progress on these directions: MSA (multiphase shared arrays) provide migratable threads that can access shared data arrays segmented into migratable pages; However, each array can be accessed only in read, write-by-one-thread, or accumulate mode in each phase of its life. For imparting a global view of control to message-drive-objects programs, we are developing an orchestration language for Charm++. Both of these have shown excellent promise in enhancing expressiveness and productivity for an appropriate class of applications. Extending them, possibly via compiler support, and integrating them with compositionality and modules remain future challenges. I am excited by the promise of this research, in spite of the bad name “language research” has gotten — of course, our language will be pragmatic, and compatible with C/C++ and Fortran for sequential components.

3 Domain Specific Frameworks:

Another direction for improving productivity is to recognize that only a small set of distributed data structures account for a large fraction of parallel applications. These include: structured and unstructured grids, trees that decompose space adaptively, and particles. Developing frameworks/libraries for these will allow considerable reuse of common algorithms on them. However, a challenge is that the application developers won’t use frameworks that impose too heavy a structure on them. My approach is to design it in such a way that the parallel program feels like a small extension of their sequential program — where they have good control over their data structures, and their code can be in the language of their choice.

So far, our success has been with ParFUM, an unstructured mesh framework with limited support for mesh adaptivity, which is used in multiple applications (rocket simulation, crack propagation,

etc.). In future I expect to develop a framework for spatial-decomposition-trees based on our experience with molecular dynamics and computational cosmology (gravity). A Structured-grid framework that we already had developed, that supported AMR, can be either fully developed via interactions/collaboration with specific applications, or I hope to persuade other framework writers to use Charm++/AMPI as a base for their system.

3.1 Parallel Computational Geometry

For supporting adaptivity for unstructured meshes, I plan to extend ParFUM. Although this is an essential area of work, it requires (a) appreciation of the needs of application (b) sophisticated parallel programming expertise, in part because of the race conditions involved and (c) a good knowledge of theoretical computational geometry algorithm. Because of the reward structure of communities, this integration has received little attention (except by application developers at DOE etc.). Depending on appropriate collaboration, and building on the the new ParFUM structure, I plan to make a strong impact on this area.

4 Fault Tolerance

As machines grow large, this issue has become important again. I believe migratable objects present a great mechanism for handling faults effectively, even when the MTBF falls below effective checkpointing period. I have started a long term project to this end based on sender-side message logging, and parallel restart for all the failed objects. This will be integrated with our traditional schemes for automatic detection of faults, and support for checkpoints in NFS disks, other processor's memory of their local disks. In addition, we have a preliminary scheme for proactive fault handling that can exploit warnings of impending faults, when available. Since checkpointing is similar to object migration, the same mechanisms of our RTS can be leveraged for these schemes.

5 Resource Management Strategies for the Computational Grid

I think a lot of early research on the Grid focused on mechanisms, and possibly early standardization of those. The promise of the Grid, though, rests on the *strategies* for resource management across the grid, in addition to within individual clusters. These will ensure timely response for parallel applications, and good throughput for computer systems. I believe that adaptive runtime systems within individual jobs, within cluster schedulers, and meta-schedulers can be co-developed to make such "autonomous computing" a reality. We have made a small beginning with our "Faucets" project in that direction, which I hope to pursue further. Interestingly, all of these can be seen as being enabled by the "migratable object" paradigm, and so it doesn't detract from my focus. For example, jobs can run on multiple clusters simultaneously can be optimized effectively with the overdecomposition and specialized load balancers within the Charm++/AMPI framework.

6 Applications Research

Collaborative applications research has been the well-spring of ideas that have fueled my research. So, I plan to continue such collaborations. In addition to cross fertilization of ideas, and the ability of bringing new abstractions back in parallel programming, these collaborations are satisfying

because of the sense of contributing to the solution of important scientific, engineering, and ultimately societal problems.

My current “portfolio” of active collaborations include the longstanding biomolecular simulation (NAMD), quantum chemistry, computational astronomy, structural dynamics, rocket simulation, and space-time meshes, Some of these may conclude in near future. In spite of their large number, I am eager to participate in such collaborations, and with the right staff and students, I think I can contribute significantly in spite of distributing myself thin.

7 PDES and State-Space Search

Two applications within computer science are on my current list of interests. I hope to revisit my past research on state-space search, when I think I had made some early (circa 1990) and strong contributions (esp. for first-solution searches, and branch-and-bound searches). These need to be revisited in the context of the new generation of machines — compared with 1,000 processors used then, we can now use 100k processors — and newer problems in AI, genomics, etc.

Secondly, parallel discrete event simulations remain a tough nut to crack, in general. We have recently shown some progress on optimistic concurrency control that I’d like to push further. It also happens to be a key ingredient needed for efficient parallel performance prediction, especially when networks are simulation in detail.