

# Research Statement

*Greg Bronevetsky*

Current trends in processor design are rapidly driving parallel computing to dominate the computer industry, with multicore, cluster and distributed computing taking up the vast bulk of our society's computing workload. Today, these systems help us tackle an enormous variety of problems, ranging from everyday desktop and graphics applications to business and web workloads to large-scale scientific simulations. However, their complex structure presents system, language, compiler and hardware designers with new challenges that must be met if parallel systems are to meet their potential.

My research is focused on parallel computing and uses techniques from various branch of computer science to improve the usability, performance and reliability of parallel systems. Examples of my work include operating systems research on scalable checkpointing algorithms and I/O systems, formalization of shared memory models, architecture work on debugging tools and study of compiler analyses of parallel applications, including checkpoint optimization. Although I expect to continue to collaborate across fields to solve problems in parallel computing, over the next several years I will focus on two specific projects that I believe will make significant contributions to both their respective fields and to parallel computing as a whole. The first focuses on developing compiler analyses for parallel applications that can take into account the application's parallel structure and communication topology. My second project addresses the problem of hardware failures, common in large parallel systems. It will concentrate on tools that characterize how errors arise in and propagate through parallel applications, thus enabling application developers to efficiently improve the reliability of their applications.

Since today parallel computing is an important component of many areas of computer science I expect my work to be funded by a wide variety of government agencies such as NSF, DOE, DARPA, NSA and corporate sponsors such as Intel, IBM and Microsoft. Some examples of current programs that align well with my research agenda are the NSF High End Computing University Research Activity (HECURA), NSF Multicore Chip Design and Architecture (MDCA), DOE Forum to Address Scalable Technology for runtime and Operating Systems (FAST-OS) and DARPA Architecture-Aware Compiler Environment (AACE). Furthermore, I plan to continue developing collaborations with researchers from a variety of fields to extend my recent and ongoing collaborations with Cornell University, Purdue University, University of Minnesota, University of Rochester, as well as Lawrence Livermore, Los Alamos and Oak Ridge National Laboratories.

## **Dataflow for Parallel Applications**

The growth of parallel computing has caused a dramatic rise in the number of languages and libraries that allow developers to express the parallelism in their applications. Unfortunately, it has been matched by little progress in compiler techniques for analyzing such explicitly parallel applications. Indeed, most current techniques for analyzing such applications are based on a fundamentally sequential understanding of the structure of parallel applications that does not take into account the application's communication topology or synchronization pattern.

While there have been advances in analyzing applications that use restricted communication primitives, such as those available in Erlang or  $\pi$ -calculus, work on more general communication models such as MPI and OpenMP has largely been limited to matching barriers on different threads and processes. This slow progress has bounded the ability of compilers to optimize application performance, enhance debugging tools and perform any of the other program analyses and transformations that are routinely employed with sequential applications.

To resolve this problem I am developing a new compiler analysis framework that extends traditional dataflow techniques to explicitly parallel applications, enabling dataflow analyses to propagate information along the application's communications and synchronizations. It works by extending

traditional control-flow graphs (CFGs) to extended parallel CFGs (pCFGs), where every node represents the state of the parallel application. Specifically, each node maps different sets of application processes to their current CFG node, with pCFG edges representing state transitions by different process sets. By allowing developers to provide separate abstractions for process sets, communication expressions and dataflow state, the framework makes it possible to tailor a specific combination of abstractions to a specific communication pattern and target application property.

My ongoing work has focused on defining and implementing the core framework, with initial results to be published at this year's Symposium on Code Generation and Optimization (CGO) 2008. Over the next several years I will focus on several key framework extensions:

- Developing a context-sensitive inter-procedural analysis to augment the current intra-procedural analysis,
- Developing support for general non-deterministic communication primitives such as semaphores and MPI\_ANY\_SOURCE to augment the currently supported deterministic primitives, and
- Developing and evaluating various compiler analyses that take advantage of the parallel dataflow framework to optimize application performance, improve debugging, among others.

As part of this project I am collaborating with researchers from Rochester University, IBM and Los Alamos National Laboratory.

The ultimate result of this work will be an extension of traditional compiler analyses to one of the most important innovations in modern software engineering: explicitly parallel applications. By revolutionizing the compiler's ability to understand such applications, this work will enable fundamental improvements in programmer productivity and application quality and performance.

### **Application Vulnerability Analysis**

As the feature sizes of modern electronics shrink and computer chips are built from increasing numbers of lower-power transistors, electronics become increasingly vulnerable to low-level physical errors. These include hard or intermittent errors due to manufacturing process variation as well as radiation-induced soft errors. While infrequent on individual desktops or servers, such errors become important for large parallel systems such as supercomputers or computing cloud server farms. Indeed, large systems such as ASCI Q at Los Alamos National Lab and BlueGene/L at Lawrence Livermore National Lab, suffered from 26 and 48 cosmic radiation-induced failures per week, respectively, due to radiation-induced memory errors. Other large parallel systems see similar types of errors in their CPUs, memories and hard disks.

Given the high cost of eliminating such errors on large systems, it is imperative for applications to be explicitly designed to minimize the effect of hardware errors on application output. This research project is focused on providing application developers with tools they can use to understand the vulnerability of their applications to errors and adapt them accordingly. Specifically, these tools will

- identify application code and data regions that are vulnerable to errors, enabling developers to replace them with more reliable variants and
- discover application data paths that amplify errors, which will support developer efforts to restructure their applications to improve application stability.

Such tools will enable application developers to focus their reliability enhancement efforts to ensure maximal effect on application reliability at a minimal cost in performance and developer time.

Because application reliability has been important for a long time, there exists significant work on this topic. However, most of this work has focused on fault injection, a technique where the application is executed many times (thousands to tens of thousands), with each execution being injected with a random error. Although generally effective, fault injection is very expensive. Further, having to run the application multiple times means that realistically large input sizes and running times cannot be used during fault injection. Since large-scale, long-running applications are most vulnerable to hardware failures, this limits

fault injection to making weak inferences about the error vulnerability of such applications instead of direct predictions.

My work in this area will address the poor scalability of traditional fault injection by investigating modular fault injection, where fault injection is used to build error profiles of individual application modules, which are then connected into a whole-application profile. To this end I am currently investigating applications primarily composed of calls to library routines. Such applications can be represented as a dependence graph with individual routines as nodes and their input/output dependences as edges. If a random error affects an individual routine (since errors are rare it is accurate to assume that only one error happens at time), its output becomes corrupted. This corrupted output is then fed into the input of some other routine, causing its output to become corrupted and so on until errors reach the application outputs. It is possible to model this process by focusing on the individual routines. Specifically, we need to

- Use fault injection on each routine to determine its output error pattern and
- Find an accurate representation of how each routine maps its input errors to output errors by training a predictor from a large number of samples.

These routine-specific error profiles, which can be computed by the library vendor, make it possible to simulate the effect of errors in any part of the application. The resulting vulnerability profile identifies the error injection locations most likely to produce errors in the application output and the paths such critical errors take through the application. This can then be used to finely target the developer's efforts to improve application reliability.

I have published preliminary work on this analysis at the International Conference on Supercomputing (ICS) 2008 and am currently finalizing my modular fault injection work for submission to a top-tier conference. My ongoing and future work will extend the basic approach described above to create a fully generic application vulnerability analysis tool. First, I am developing spectral decomposition-based techniques to find the error patterns of loops without propagating errors through all loop iterations. I will then extend this work by exploring compiler techniques that build error profiles for individual functions and propagate error patterns through application source code. Finally, I will develop vulnerability profiles of parallel applications by (i) computing modular vulnerability profiles of smaller sub-sets of processes and (ii) using my compiler framework for parallel applications in conjunction with the above compiler-based vulnerability analysis.

The ultimate goal of my work will be a general-purpose, highly-scalable tool that will enable application developers to understand how their applications are affected by hardware errors. This will enable them to bound the probability of application errors with a minimal effect on developer time and application performance, ensuring that applications execute reliably on unreliable hardware.

## **Summary**

We are in a period of time when our fundamental systems, architectures and languages are being challenged by the complexity of parallel systems and applications. The reward of fulfilling this promise is a variety of exciting applications ranging from intuitive desktops to global knowledge aggregation. The price of failure is a dramatic slowdown of the computer industry as newer hardware is unable to provide better performance. My work focuses on employing techniques from all branches of computer science to address these challenges and make parallel computing realize its true potential. As part of this work I will develop new inter-disciplinary connections and collaborations and focus on two major projects. One project will focus on developing compiler analyses for explicitly parallel applications that are sensitive to their parallel structure. The other will address the decreasing reliability of parallel systems by developing tools to make applications less vulnerable to system errors. I believe that my work in this direction will make a significant contribution to computer science in general as well as the desktop, enterprise and scientific computing communities that will continue to rely on parallel computing.