Research Statement

October 20, 2006

During the past six years my research activities have focused primarily on the development of the parallel programming language Unified Parallel C. UPC is one of the so-called partitioned shared memory programming languages. These languages are an important part of the current national effort to develop high productivity languages for high performance parallel platforms. With seed funding and technical support from Compaq and then HP, I led the group that developed the first public domain implementation of UPC. MTU was licensed to modify the Edison Design Group compiler front end to convert UPC programs to C code containing calls to our MuPC runtime system. MuPC is a set of procedures that perform remote memory operations at runtime to support UPC's partitioned shared memory model. MuPC is based on POSIX threads and MPI so the MTU UPC compiler can run on most clusters and SMPs.

MuPC is used at MTU to study research issues in the design, implementation, and performance of UPC. Recent work has produced a performance model for UPC programs. Performance models for parallel programming systems facilitate the design of algorithms and the development of scalable applications. Constructing a performance model for UPC is more difficult than it is for a message passing system because the remote memory operations in UPC are usually implicit. Compiler and runtime optimizations hidden under the covers add to these difficulties. The performance model that was developed consists of a set of microbenchmarks that measure the performance of the platform under a variety of remote memory access patterns. Based on an analysis of the access patterns in a UPC application program, the model yields a synthetic characterization of the performance of the program. The model requires no information about the internals of the compiler or runtime system being modeled. At this point the performance of simple kernels can be accurately predicted. Work is now underway to model the effects of a runtime cache and other performance features.

Another ongoing project is the development of collective operations for UPC. A well-designed set of collective operations is an aid to productivity and performance. As the moderator of the UPC collective operations working group, I helped guide the effort to develop a specification. The first collectives specification was approved at the end of 2003 and it has since been added to the UPC language standard. During that period I was on sabbatical at Hewlett-Packard and contributed to the collectives library for HP's UPC compiler and produced an open source reference implementation of the collective functions for the user community.

Soon after the collective operations standard was approved the user community called for extensions to the standard. Our MTU group produced two proposals in 2005, one of which was selected for further discussion. At the same time some of us expressed concerned about the level of complexity that the extensions introduced. Acknowledging this concern, the UPC community tabled the proposed extensions at the most recent UPC workshop.

Current work at MTU and Berkeley is now considering simplified forms of the standard set of collectives. This change of direction marks the first time that collective operations are being described in the language of the shared memory programming model rather than in terms of message passing operations.

While at HP I also studied the optimizations provided by HP's UPC runtime system. This work led to the redesign of several aspects of the UPC runtime cache. Some of these improvements are included in the current release of the HP UPC compiler and others are in the version to be released in the near future. The UPC performance model will be used to quantify the effects of some of these changes.

Prior to 2000 my efforts were directed at a much different target. After working with Pat Worley and his group at Oak Ridge National Laboratory while on sabbatical in 1992-93, it became clear that Michigan Tech could benefit from a computational science and engineering research program. From 1996 to 2002 I worked to develop the existing non-departmental Computational Science and Engineering Ph.D. program, which then served as the Computer Science Department's Ph.D. program, into a stand-alone program to serve computational scientists across the university. The Computer Science Department was on track to start its own Ph.D. program, which it did in 2000, and the CS&E Ph.D. program was likely to fade if it was not given continuing attention. This effort culminated in the creation of the Computational Science and Engineering Research Institute in 2002 at which point Phillip Merkey undertook its direction. With funding and equipment donations from NSF, NASA, and DoD the CSERI now provides medium-scale computational resources for CS&E researchers and for instruction. During the past four years the CS&E program has awarded one Ph.D. degree per year. The home departments of these graduates were physics, mathematics, biology, and geology, so the goal of providing a university-wide computational science Ph.D. program is being achieved.

Near term future work includes the study of a set of one-sided collective operations for UPC. Others in our group are developing parallel implementations of nonregular, graph-theoretic applications and investigating atomic memory operations for UPC. Other topics of interest include optimizations suggested by performance model studies and the consideration of other optimization techniques that may be applicable to UPC. The long term goal is to nudge architectural designs in the direction of application developers' needs for high productivity languages. At the same time, compilers and runtime systems must continue to develop optimizations that can take advantage of available architectures.

Steven R. Seidel
Department of Computer Science
Michigan Technological University
http://www.cs.mtu.edu/~steve