1 Introduction

Programming models are at the very center of our effort to address the exascale challenge. They are the key interface that will allow the separation of the programmers’ concerns from those of system designers, potentially at different levels of granularity. Any such model must meet the extensive needs of application developers and be supported by the entire software stack. Considerable research is needed to define and implement the programming and execution models for such systems. Whereas evolutionary approaches may best support the migration of existing application software, revolutionary models may be best suited to providing extreme-scale performance for new applications on emerging architectures. Both approaches should be explored.

Desirable properties of exascale programming models include the following:

- They should provide the highest levels of performance. Most HPC programs are written for performance. Exascale programming languages should be performance-aware: they should provide an adequate abstraction of high-performance parallel hardware platforms to enable the exploitation of their features, and some means to tune performance. The failure of automatically parallelizing compilers and HPF was caused not only by technical immaturity but also by a lack of an interface in the programming language for performance improvement. When the programmer finds a performance bug, he or she should have some means to improve performance by modifying the program.
- They should provide the necessary means to allow tools (especially performance tools) to obtain information on the application’s execution behavior. Programming environments should be designed to take the performance tuning process into account by having an interface to performance tools.
- Expressiveness is a key requirement. Exascale programming languages should provide a model and an interface to expose and express the parallelism inside programs. In functional programming languages and “old” dataflow languages, parallelism is implicit since the model of computation itself exploits the parallelism. In imperative languages, new constructs and mechanisms should be introduced to express the parallelism. For the application’s point of view, task parallelism must be able to

1UNIVERSITY OF HOUSTON, HOUSTON, TEXAS, USA
2UNIVERSITAT POLITÈCNICA DE CATALUNYA, BARCELONA, SPAIN
3RICE UNIVERSITY, HOUSTON, TEXAS, USA (VSARKAR@RICE.EDU)
4UNIVERSITY OF TSUKUBA, TSUKUBA, JAPAN

DOI: 10.1177/1094342009347488
© The Author(s), 2009. Reprints and permissions: http://www.sagepub.co.uk/journalsPermissions.nav
support coupled multiphysics simulations at several levels for exascale systems. Applications will need to express massive amounts of potentially fine-grain parallelism, of asynchrony and locality. Dynamic application behavior will need to be supported. It should be possible to express hierarchical parallelism within the application. Latency hiding needs to be facilitated.

- They should enable composability. Composability is essential to support productive programming on exascale systems. Libraries and object-oriented approach help accomplish this in conventional sequential programming, but they do not always work in parallel programming. For example, it is difficult to use parallel libraries with current OpenMP. Parallel object-oriented programming is sometimes useful, but has some problems.
- They should support fault tolerance and error handling. Fault tolerance is one of the most difficult issues faced on exascale systems. If faults are exposed to programmers, then some programming language support will be required to handle them. It must, moreover, be possible for an application to respond to faults and program errors gracefully rather than simply crashing.
- They need to support massively parallel I/O. An abstraction of I/O, including the file system, may help programmers handle the huge amounts of data that will have to be read and written.

Approaches to programming exascale systems should take the following into account:

- There is a need to provide a smooth transition path from existing practices and codes to future approaches.
- Approaches should provide portability (functional and performance) across platforms such that the porting effort can be amortized over the foreseeable variation of systems to appear from now to the exaflop era and beyond.
- Incremental parallelization/tuning of applications is a desirable property closely related to the above two issues.
- Initial approaches should address the device-, node-, and system-level programming. Proposals for hybrid programming should ensure clean interaction between the different levels and ensure that the synchronization semantics and scheduling decisions at one level do not imply restrictions on other levels.

Topics for detailed study include:

- Address space structure: Identify abstract levels of such a structure that is simple enough for use by a programmer to express objects/ideas yet provides the run time with flexibility regarding its mapping to the potentially varied physical structure.
- Flexible work generation (parallelism/task specification) and synchronization structures beyond pure fork-join approaches in order to support flexible parallelism and high levels of asynchrony. Ideas from data flow or functional programming may be revisited and smoothly integrated into current practices.
- Latency tolerance: being able to specify required data accesses with large lookaheads such that implementations (compiler or run time) can anticipate the required data transfers and schedule them appropriately.
- The issue of hierarchy and heterogeneity: providing mechanisms for modular designs with interchangeable implementations of tasks.
- Separation of functionality and performance: providing mechanisms for the programmer to provide hints that may help satisfy performance or power requirements, but are not required to provide functionality of the algorithms.
- Malleability: the ability of applications to dynamically adapt to the available resources which may vary during a job run. Programming and execution models should support/promote malleable programming practices by separating (virtualizing) the algorithmic structure of a program from the resources where it is executed.
- Error handling and fault tolerance: providing the appropriate hooks for resilient applications.
- Application development environments that facilitate the migration of current codes and/or the development of new ones from scratch.

In all the above issues, the programming and execution model interfaces are key to allowing programmers to focus on their algorithms while providing the mechanisms that will enable the compilers and run times to infer the information they need to optimize, automatically and dynamically, the use of system resources (cores, memory, bandwidth, power).

The evolutionary path aims to adapt existing programming models to the needs of exascale computing, and facilitate the task of creating and tuning potentially hybrid application codes. This could include work to enhance MPI, OpenMP, CUDA/OpenCL, or other approaches to programming accelerators and SIMD units, as well as work to improve their interoperability. It might also include more effort to deploy the PGAS languages and ensure that they may interoperate with other programming interfaces. A revolutionary path might be based upon HPCS languages or might be a completely new path. It may build upon prior experience. It might be worthwhile to revisit old parallel programming models and languages to obtain new insights from the past, as is being done in the architecture community. It could be interesting to evaluate functional programming models used for programming the dataflow machines, such as
Id, SISAL, etc. HPF was a great effort to develop a standard parallel programming language and is also worthy of re-examination. It is important to take their experience of failure into account for better future developments.

In such an open field, it is advisable to pursue a few alternatives and ensure there is sufficient sharing of experiences as well as comparative studies between them. These should be in terms of complexity/readability of the code and programming effort as well as performance (both actual measurements on common platforms as well as predictions for different potential targets). Although these types of studies are often difficult to perform, special efforts should be devoted to them. Common sets of algorithms should be used for evaluation by all the proposed models.

Finally, we should promote the standard development effort of APIs between several levels and between existing software components in the IESP community. For programmers and end-users, candidates for standardization will include:

- PGAS and APGAS languages (UPC, CAF, X10).
- Global view models such as Chapel and HPF.

For the system developer, the candidates are:

- One-sided communication APIs.
- Fault tolerant models and APIs.
- APIs for I/O on massively parallel system.
- APIs for accelerators.
- Performance profile APIs and data formats such as OTF.
- APIs for thread scheduling.

Standards development efforts are key to “evolution” which develops the community. They will form the basis for the next “revolution” of rich diversity for exascale computing.

**Author Biographies**

*Barbara Chapman* is a professor of computer science at the University of Houston, where she teaches and performs research into languages and compilers for parallel computing. Her HPCTools Lab has produced the open source OpenMP compiler that provides a free implementation of OpenMP with all of its base languages. Dr. Chapman is the CEO of Compunity, an organization of researchers that contributes actively to the work of the OpenMP Architecture Review Board. She has co-authored two books in the field and many papers related to compiler technology, parallel programming languages and their implementation, and high-performance computing.

*Jesús Labarta* has been a full professor of computer architecture at the Technical University of Catalonia (UPC) since 1990. Since 1981 he has been lecturing on computer architecture, operating systems, computer networks and performance evaluation. His research interest has been centered on parallel computing, covering areas from multiprocessor architecture, memory hierarchy, parallelizing compilers, operating systems, parallelization of numerical kernels, performance analysis, and prediction tools. From 1995 to 2004 he was director of CEPBA (European Center of Parallelism of Barcelona) where he was highly involved in the promotion of parallel computing into industrial practice, and especially within SMEs. He led the technical work of UPC in more than 15 industrial R+D projects supported by the European Commission Research programs. Many of those projects dealt with the parallelization of numerical codes in areas such as structural analysis, CFD, production planning, or nuclear power plant simulation. Since 2005 he has been responsible for the Computer Science Research Department within the Barcelona Supercomputing Center (BSC), His major directions of current work relate to performance analysis tools (developing and distributing the CEPBA tools Paraver and Dimemas), programming models, in particular developing the StarSs model (with implementations for different multicore and accelerator nodes such as Cells, GPUs, SMP, etc.), interconnection networks, node architecture, and resource management. All these topics are being researched in the holistic framework of the MareIncognito project, a BSC R&D cooperation with IBM.

*Vivek Sarkar* conducts research in programming languages, program analysis, compiler optimizations, and virtual machines for parallel and high-performance computer systems, and currently leads the Habanero Multicore Software Research project at Rice University. Prior to joining Rice, he was Senior Manager of Programming Technologies at IBM Research. His responsibilities at IBM included leading IBM’s research efforts in programming model, tools, and productivity in the PERCS project during 2002–2007 as part of the DARPA High Productivity Computing System program. His past projects include the X10 programming language, the Jikes Research Virtual Machine for the Java language, the ASTI optimizer used in IBM’s XL Fortran product compilers, the PTRAN automatic parallelization system, and profile-directed partitioning and scheduling of Sisal programs. Vivek became a member of the IBM Academy of Technology in 1995, the E.D. Butcher Professor of Computer Science at Rice University in 2007, and was inducted as an ACM Fellow in 2008. He holds a B.Tech. degree from the Indian Institute of Technology, Kanpur, an M.S. degree from University of Wisconsin-Madison, and a Ph.D. from Stanford.
In 1997, he was on sabbatical as a visiting associate professor at MIT, where he was a founding member of the MIT RAW multicore project.

Mitsuhisa Sato received his M.S. and Ph.D. degrees in information science from the University of Tokyo in 1984 and 1990. He was a senior researcher at Electrotechnical Laboratory from 1991 to 1996, and a chief of the Parallel and Distributed System Performance Laboratory of the Real World Computing Partnership, Japan, from 1996 to 2001, leading the Omni OpenMP compiler project. Currently, he is a professor at the Graduate School of Systems and Information Engineering, University of Tsukuba. He has been a director of Center for Computational Sciences, University of Tsukuba since 2007. His research interests include computer architecture, compilers and performance evaluation for parallel computer systems, OpenMP and parallel programming, and grid computing. Dr. Sato is a member of IEEE CS and IPSJ (the Information Processing Society of Japan), JSIAM.