Final Report for DE-FG02-04ER25623
Operating/Runtime Systems for Extreme Scale Scientific Computation
Program Notice 04-13
August 2009

SmartApps: Middle-ware for Adaptive Applications on Reconfigurable Platforms

Principal Investigator:

Lawrence Rauchwerger
Parasol Lab
Dept. of Computer Science
Texas A&M University
http://parasol.tamu.edu/people/rwerger
rwerger@cs.tamu.edu
tel: +1-979-845-8872
fax: +1-458-0718

Co-Investigators:

Texas A&M University

Marvin L. Adams
Dept. of Nuclear Engineering
Texas A&M University
mladams@tamu.edu

Nancy M. Amato
Parasol Lab
Dept. of Computer Science
Texas A&M University
amato@cs.tamu.edu

Bjarne Stroustrup
Parasol Lab
Dept. of Computer Science
Texas A&M University
bs@cs.tamu.edu

IBM T. J. Watson Research Center

Orran Y. Krieger*
Manager
Advanced Operating Systems
IBM T.J. Watson Research Center
okrieg@us.ibm.com

Jose M. Moreira
Manager
Modular System Software
IBM T.J. Watson Research Center
jmoreira@us.ibm.com

Vivek Sarkar*
Senior Manager
Programming Technologies
IBM T.J. Watson Research Cen
vsarkar@us.ibm.com

Lawrence Livermore National Laboratory

Dan Quinlan
Staff Scientist
Lawrence Livermore National Laboratory
dquinlan@llnl.gov

*Please note that Orran Krieger and Vivek Sarkar both left IBM after the project began.

1 Overview

There are several reasons why the performance of current distributed and heterogeneous systems is often disappointing. For example, the characteristics of the application may be input sensitive and evolve during execution causing dramatic changes in memory reference patterns, resource requirements, or degree of concurrency between different phases of the computation. Or, the system may change dynamically with nodes failing or appearing, some network links severed and other links established with different latencies and bandwidths.

Another important reason for poor performance is the fairly compartmentalized approach to optimization: applications, compilers, operating systems and hardware configurations are designed and optimized in isolation and without the knowledge of instance specific information and needs of a running application. There is too little information flow across these boundaries and no global optimization is even attempted. For example, most operating systems services like paging, virtual-to-physical page mapping, I/O, or data layout in disks, provide little or no application customization. Similarly, the off-the-shelf hardware used by most commercial systems is optimized to give best average-case performance.

To address this problem, we have proposed application-centric computing, or SMART APPLICATIONS (SMARTAPPS). In the SMARTAPPS executable, the compiler embeds most run-time system services, and a performance-optimizing feedback loop that monitors the application's performance and adaptively reconfigures the application and the OS/system platform. At run-time, after incorporating the code's input and determining the system's resources and state, the SMARTAPPS performs an *instance* specific optimization, which is more tractable than a global generic optimization between application, OS and system.

The overriding philosophy of SMARTAPPS is "measure, compare, and adapt if beneficial." That is, the application will continually monitor its performance and the available resources to determine if, and by how much, performance could be improved if the application was restructured. Then, if the potential performance benefit outweighs the projected overhead costs, the application will restructure itself and the underlying system accordingly. The SMARTAPPS framework includes performance monitoring and modeling components and mechanisms for performing the actual restructuring at various levels including: (i) algorithmic adaptation, (ii) run-time software optimization (e.g., input sensitivity analysis, etc.), (iii) tuning reconfigurable OS services (scheduling policy, page size, etc), and (iv) system configuration (e.g., selecting which, and how many, computational resources to use).

SmartApps is being developed in the STAPL infrastructure. STAPL (the Standard Template Adaptive Parallel Library) is a framework for developing highly-optimizable, adaptable, and portable parallel and distributed applications. It consists of a relatively new and still evolving collection of generic parallel algorithms and distributed containers and a run-time system (RTS) through which the application and compiler interact with the OS and hardware.

The overall architecture of our system shown in Figures 1 and 2, consists of a nested multi-level adaptive feedback loop that monitors the application's performance and, based on the magnitude of deviation from expected performance, compensates with various actions. Such actions may be run-time software adaptation, re-compilation, or operating system and system reconfiguration. The system shown in Figure 1 uses techniques from a TOOLBOX shown in Figure 2. The TOOLBOX contains application and system specific databases and algorithms for performance evaluation, prediction and system reconfiguration. The tools are supported by architectural and performance models. More details of this architecture can be found in [40].

In the following, we will report on the various research activities that were undertaken as part

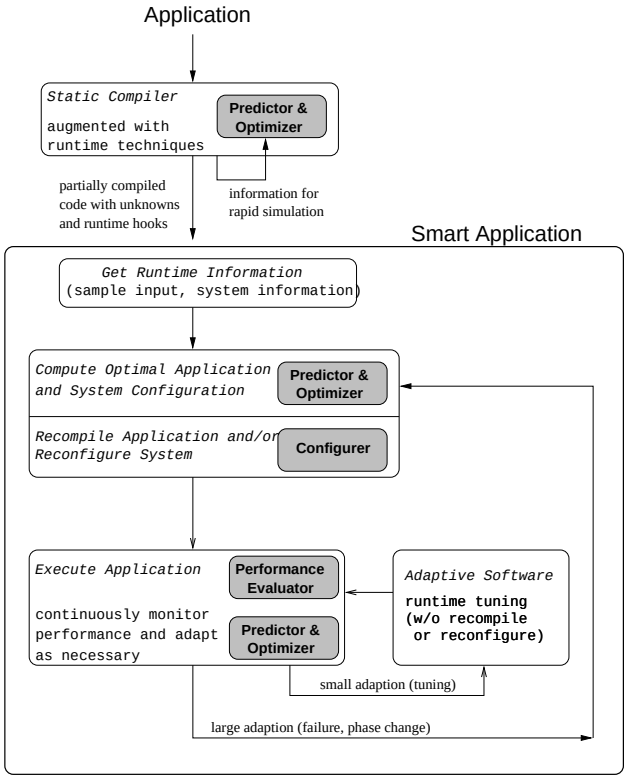


Figure 1: Smart Application.

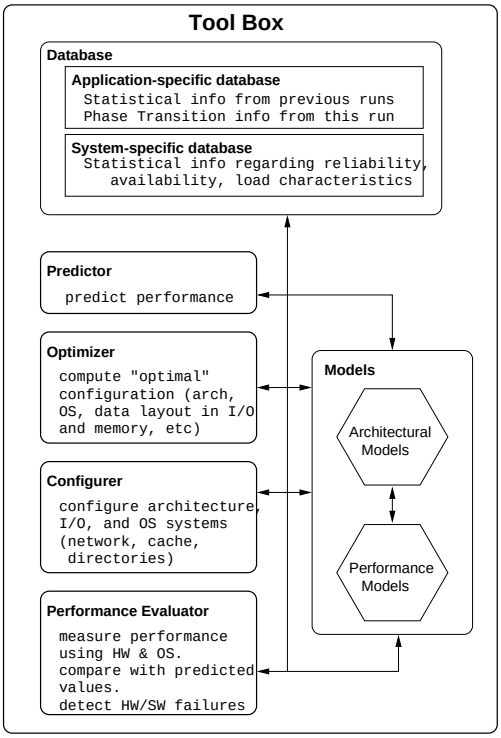


Figure 2: ToolBox.

of this project and note some of our accomplishments. The document is organized according to the major tasks of the project:

- the STAPL parallel programming infrastructure, including algorithmic adaptation and applications developed using STAPL;
- the ARMI run-time system, including its ability to exploit future operating systems such as K42; and
- the compiler infrastructure.

2 STAPL Development

STAPL (the Standard Template Adaptive Parallel Library) is a framework for parallel C++ code development [3, 2, 39, 48, 71]. Its core is a library of ISO Standard C++ components with interfaces similar to the (sequential) ISO C++ standard library [36]. STAPL offers the parallel system programmer a shared object view of the data space. The objects are distributed across the memory hierarchy which can be shared and/or distributed address spaces. Internal STAPL mechanisms assure an automatic translation from one space to another, presenting to the less experienced user a unified data space. For more experienced users the local/remote distinction of accesses can be exposed and performance enhanced. STAPL supports the SPMD model of parallelism with essentially the same consistency model as OpenMP. To exploit future petascale systems and current

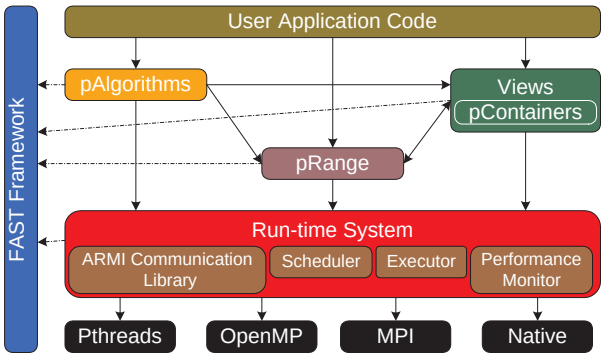


Figure 3: STAPL Overview

large systems such as IBM's BlueGene/L, STAPL allows for (recursive) nested parallelism (as in NESL [7]).

A key design goal of STAPL is to deliver so-called 'portable performance' – that is, the same program performs well on all systems without any modification. This is a major challenge in general, but is even more challenging on parallel systems because performance of parallel algorithms is sensitive to system architecture (latency, topology, etc.) and to application data (data type, distribution, density, etc.). We address this challenge in STAPL by providing the infrastructure to make STAPL applications SMARTAPPS. That is, the STAPL framework provides the tools and mechanisms that enable STAPL SMARTAPPS to continually adapt to the system and the data at all levels – from selecting the most appropriate algorithmic implementation to balancing communication granularity with latency by self-tuning the message aggregation factor, etc. In particular, the library must be capable of making intelligent and automated decisions to select concrete implementations for each of its generic components, customizing the implementation for each application that uses it. Although more experienced users may later choose to refine performance manually (explicitly within STAPL), it is imperative that good performance be provided for even the novice parallel programmer.

For users, STAPL provides three levels of abstraction appropriate to an *application developer* (level 1), a *library developer* (level 2), and a *run-time system developer* (level 3). At the highest level, STAPL offers the application developer an STL compatible interface to a generic parallel machine. Parallel programs can be composed by non-expert parallel programmers using building blocks from the core STAPL library. Users don't have to (but can) be aware of the distributed nature of the machine. At the intermediate level, STAPL exposes sufficient information to allow a library developer to implement new STAPL-like algorithms and containers, e.g., to expand the STAPL base or build a domain specific library. This is the lowest level at which the "usual" user of STAPL operates. The shared object view and abstract interface to the machine and RTS result in platform independent portable code. Nevertheless, remote and local accesses may be distinguished, and a new container class must specify the data types it stores to enable RMI to pack/unpack container objects. For new pAlgorithms, the developer must (provide a method to) define a DDG describing the algorithm's dependences. At the lowest level, the RTS developer has access to the implementation of the communication and synchronization library, the interaction between OS, STAPL thread scheduling, memory management and machine specific features such as topology and memory subsystem organization.

2.1 STAPL Components

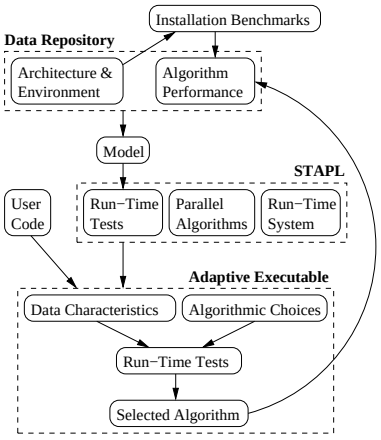
The STAPL infrastructure consists of platform independent and platform dependent components that are revealed to the programmer at an appropriate level of detail through a hierarchy of abstract interfaces (see Figure 3). The platform independent components include the core parallel library, a view of a generic parallel/distributed machine, and an abstract interface to the communication library and run-time system. The core STAPL library consists of `pAlgorithms` (parallel algorithms) and `pContainers` (distributed data structures). Important aspects of all STAPL components are *extendibility* and *composability*, e.g., the `pContainers` implemented within the framework allow users to extend and specialize them, and to operate on `pContainers` of `pContainers`, and `pAlgorithms` may themselves call `pAlgorithms`.

`pContainers`, the distributed counterpart of STL containers, are thread-safe, concurrent objects, i.e., shared objects that provide parallel methods that can be invoked concurrently. They are composable and extendible by users via inheritance. Currently, STAPL provides counterparts of all STL containers (e.g., `pArray`, `pVector`, `pList`, `pHashMap`, etc.), and two `pContainers` that do not have STL equivalents: parallel matrix (`pMatrix`) and parallel graph (`pGraph`). Analogous to STL iterators, `views` provide a generic access interface to `pContainer` data for `pAlgorithms`. While a `pContainer`'s data may be distributed, `pContainers` offer the programmer a *shared object view*, i.e., they are shared data structures with a global address space. This is provided by an internal object translation method which can locate, transparently, both local and remote elements. The physical distribution of `pContainer` data can be computed automatically or user-specified. A `pAlgorithm` is the parallel equivalent of an STL algorithm. STAPL currently includes a large collection of parallel algorithms, including parallel counterparts of all STL algorithms, `pAlgorithms` for important parallel algorithmic techniques (e.g., prefix sums, the Euler tour technique), and some for use with STAPL extensions to STL (i.e., graph traversals for `pGraph`).

Parallel computations in STAPL (e.g., `pAlgorithms`) are represented by `pRanges`. Briefly, a `pRange` consists of a set of tasks and the dependences, if any, between them. A task includes both *work* (represented by work functions) and *data* (from `pContainers`, generically accessed through `views`). The `executor`, itself a distributed shared object, is responsible for the parallel execution of computations represented by `pRanges`; as tasks complete, the `executor` updates dependences, identifies tasks that are ready for execution, and works with the `scheduler` to determine which tasks to execute. Nested parallelism is created by composing `pRanges`, e.g., work functions that themselves call `pAlgorithms`. Some additional papers describing some of the features of STAPL can be found in [10, 57, 58].

2.2 Algorithmic Adaptation in STAPL

For many important operations there exist multiple, functionally equivalent, algorithms that could be used to perform them. Common examples include sorting (insertion sort, quicksort, mergesort, etc.) or computing minimum spanning trees (Kruskal's and Prim's algorithms). When there exist many functionally equivalent algorithmic options, the decision of which to use can depend on many factors such as type and size of input, ease of implementation and verification, etc. These factors make it difficult even for an experienced programmer to determine which algorithm should be used in a given situation. Moreover, and perhaps more importantly, they make it very difficult to write portable programs that perform well on multiple platforms or for varying input data sizes and types, and this problem will only increase in difficulty and importance for petascale systems. Ideally, the programmer should simply specify the desired operation (e.g., sort) and the decision



(a)

if $p \leq 8$ **then**
  $sort =$ "sample"
else
  if $presortedness \leq 0.117188$ **then**
    $sort =$ "sample"
  else
    if $presortedness \leq 0.370483$ **then**
      $sort =$ "column"
    else
      $sort =$ "sample"
    end if
  end if
end if

(b)

Figure 4: (a) Adaptive Algorithm Selection and Tuning framework and (b) p_sort decision tree.

of which algorithm to use should be made later, possibly even at run-time, once the environment and input characteristics are known.

Toward this goal, we have developed a general framework for adaptive algorithm selection for use in STAPL. Figure 4 shows the major components and flow of information in the adaptive framework. During STAPL installation, the framework collects statically available information from standard header files and vendor system calls about the architecture and environment, such as available memory, cache size, and number of processors. Algorithm performance data is also gathered from the installation benchmarks for the algorithmic options available in STAPL. Next, machine learning techniques are used to analyze the data in the repository and to determine tests that will be used at run-time to select the appropriate algorithm from algorithmic options in STAPL. At run-time, any necessary characteristics are measured and then a decision about which algorithmic option to use is made.

We applied a prototype implementation of our framework in STAPL to select among different parallel algorithms for sorting and matrix multiplication [71]. In our validation tests on multiple platforms, our framework provided run-time tests that correctly selected the best performing algorithm from among several competing algorithmic options in 93-100% of the cases studied for sorting and in 86-92% of the cases studied for matrix multiplication, depending on the system. (See [71] for more detail.)

The adaptive algorithm selection framework in STAPL is a generalization of previous work done by our group for the adaptive selection among multiple algorithmic options for parallel sorting [49, 2], for selecting among different motion planning methods [33, 35], and for performing parallel reduction operations [83, 84]. To the best of our knowledge, our framework provides the first general methodology for automatically developing a model for selecting among multiple algorithmic options. All other work of which we are aware is limited to parameter tuning of a particular algorithm (e.g., ATLAS [80]), relies upon some level of manual modeling (e.g., [9, 25, 80] or is domain specific (e.g., [38]). More details can be found in [71].

3 Adaptive Run-Time System (RTS)

The platform dependent STAPL components are mainly contained in the STAPL runtime system (RTS), which provides the API to the OS and several important functions. The RTS includes the ARMI (Adaptive Remote Method Invocation) communication library that abstracts inter-processor communication for the higher level STAPL components, the `executor` and `scheduler` modules that are responsible for allocating resources for the computation and for executing it, and the performance monitor.

ARMI abstracts communication of data and work across the distributed memory machine by providing a common remote method invocation (RMI) interface to all other STAPL components [48, 70, 71]. There is also support in ARMI for collective operations common in parallel programming, such as broadcasts and reductions. Its implementation is machine dependent and it can generate synchronous or asynchronous messages (e.g., MPI messages) or synchronizations, e.g., OpenMP synchronizations.

The `executor` and `scheduler` provide support for an assortment of common scheduling policies and load balancing strategies, and also provide mechanisms through which users can add their own versions of these services.

We designed and implemented a multi-threaded version of ARMI and our RTS. The STAPL runtime system will support nested parallelism if the underlying architecture allows nested parallelism via a hierarchical native runtime system. Otherwise, the runtime system will serialize the nested parallelism.

3.1 Application-specific Customization of the RTS

When the project was initiated, it was believed that K42 would be a living project that could be integrated into our work. For the first few years of the project, we worked actively towards this goal. However, towards the end of the project, it became clear this would not be the case and hence development using K42 slowed. In the following, we briefly report on the progress that was made.

We installed the K42 operating system on a dual processor Apple G5 and worked to port our applications to this system. After several problems were solved we could run the full STAPL test suite on this machine. We also worked to install and tested Mambo, the IBM simulator on a PC. The objective was to give us more insight into the behavior of applications on K42. However, we did not run Mambo for our experiments because it was too slow.

We have also developed several custom memory allocators: *Defero* [19] and *TP* [20]. The ultimate objective was to interface to K42'named paging system. *Defero* allows each object to customize its memory allocation policy. We have established an interesting relationship between our "closest neighbor" allocation policy and page size. *TP* exploits this allocation policy: It allocates "close" to a hint provided by the user. When using STL then these hints are provided automatically. We believe that we will be able to improve even more memory management performance when when operating systems will provide customization capabilities like those that were present in K42.

4 Compiler for STAPL/C++

Our ultimate aim is to support high-level parallelism expressed in terms of C++ libraries by static program analysis, program transformation, information to be used at run-time extracted from source code, and semantic information embedded in the program text. For that, we have built

a static analysis and transformation system, the Pivot, which can handle all of C++ (eventually all of C++0x). We place particular emphasis on the higher levels of abstraction, such as systematically use of templates. We were major players in the ISO C++ standardization effort to provide direct language support for generic programming (concepts) incl. a notation for semantic properties of types (axioms).
Accomplishments:
(1) Design and implementation of the static analysis framework, the Pivot. This is now being used for student projects related to code validation, code simplification, code rejuvenation, and transformation to use new (C++0x) facilities. We found that the initial Pivot (relying on the visitor pattern for traversal) was hard for students to use and have been working of facilities for simple traversal and lowering of semantic levels (see Luke Wagner's thesis). We plan further work to simplify the use of the Pivot. (2) We took a major role in the design and implementation of concepts for C++0x (See papers and technical reports by Stroustrup). Unfortunately, the ISO C++ standards committee decided to postpone further work on concepts (to a large extent because of Bjarne Stroustrup's analysis of usability problems. This leaves us with a major design effort done, a fair bit of practical experience (incl. a complete rewrite of the C++ standard library using concepts - remember STAPL is based on the C++ standard library), and no implementation to act as front-end to the Pivot. We are now looking at ways to get an improved concepts design implemented and interfaced with the Pivot. The design and implementation of the Pivot - our high-level analysis and source code transformation system for C++ is progressing as planned. In particular, the design of the IPR representation of *concepts* a type system for types aimed is complete. The IPR provides a foundations for semantics-based optimization of high-level libraries using templates, such as STAPL. The IPR is a fully typed abstract syntax tree representation of ISO Standard C++. The IPR is designed with an eye on C++0x, the next revision of the ISO C++ standard. Implementations based on GCC and EDG compiler front-ends are being tested. We use two compiler front-ends to avoid vendor lock-in. Some papers describing in more detail the Pivot infrastructure can be found here: [41, 56, 55].

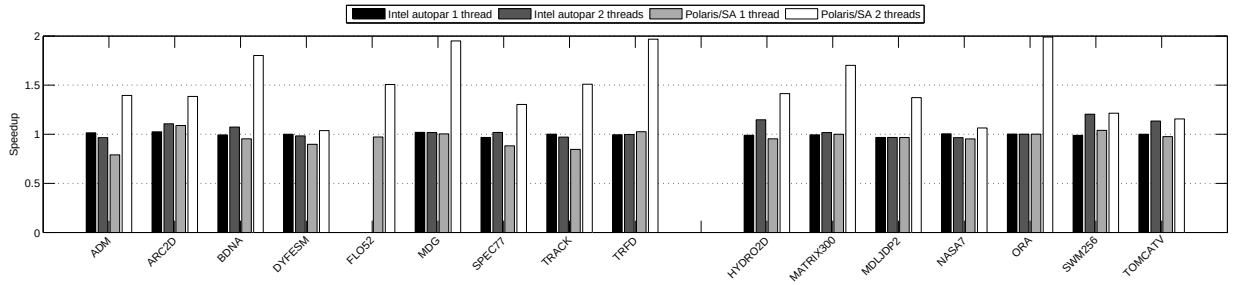
To better adapt to input data we have developed a compiler infrastructure for F77 compilation that uses bridges static and dynamic compilation in a seamless manner. The Hybrid Analysis developed during these years allows the compiler to perform the most sophisticated analysis currently available and, in case of inconclusive results, to extract the minimum necessary information that needs to (and can be) provided at run-time to perfrom aggressive optimization. We have used Hybrid Analysis to automatically parallelize many Perfect and SPECfp codes and obtained excellent results. In Fig. 5,6 we show some of tehse results.

We have also directed our attention to dynamic compilation because we believe that it will improve the performance of SMARTAPPS for dynamic, input dependent codes. We have developed a compilation system that can stop the execution of a code, for example after it has read all its inputs, recompile with high levels of specialization and continue executing the remainder of the code. The program can be stopped any time during execution and recompiled.

4.1 Applications Developed using STAPL

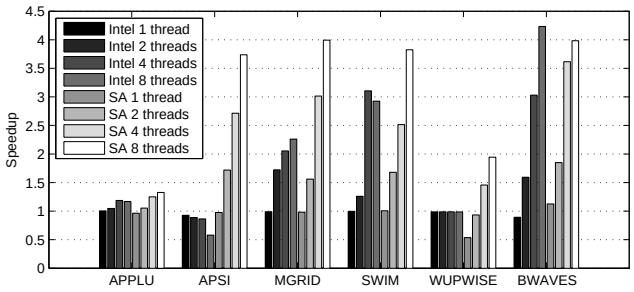
STAPL has been co-developed with domain specific libraries for some high-performance scientific applications:

- PDT - Parallel Deterministic Transport: a parallel code built in STAPL implementing parallel discrete ordinates particle transport.



PERFECT SPEC 89-92

Figure 5: Speedups relative to the sequential version of automatically parallelized benchmark applications on 1, 2 processors of a Core Duo Intel system. CT = speedups obtained using only compile-time methods. The applications are from the PERFECT and previous SPEC 89-92 benchmark suites respectively.



SPEC 2000, SPEC 2006

Figure 6: Speedups relative to the sequential version of automatically parallelized benchmark applications on 1, 2, and 4 processors on a SUN quad socket, AMD dual core system. CT = speedups obtained using only compile-time methods. The applications are from the SPEC2000 and SPEC2006 benchmark suites.

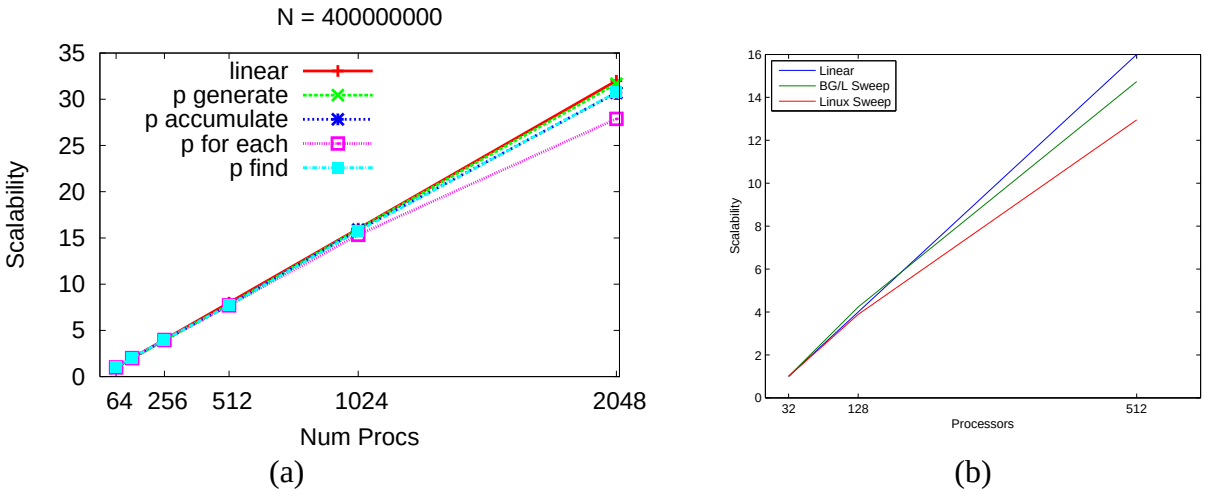


Figure 7: Strong scaling studies (fixed problem size). (a) `pAlgorithm` performance using `pArray` containers on a IBM SP RS/6000. (b) STAPL particle transport computation on an IBM BlueGene/L and a linux cluster. Note that the scaling is computed relative to 64 and 32 processors, respectively; linear scaling is shown for reference.

- a new computational method based on motion planning for protein and RNA folding [74], and
- Seismic Ray Tracing: a parallel code built in STAPL implementing a wavefront approach to seismic ray tracing.

The PDT code is an ongoing project at Texas A&M. It is a software infrastructure developed in STAPL for investigating *discrete-ordinates* [21, 24, 30] methods for deterministic particle transport in irregular problems with complex geometries. Each iteration of a discrete-ordinates method involves multiple, but independent, sweeps through the spatial domain, one for each direction of particle travel. Although each directional sweep is sequential in nature, all spatial cells in a wavefront are independent. The efficiency and scalability of the parallelization depends on many factors including the decomposition and distribution of the spatial domain, the scheduling of the sweeps, the granularity of the parallelization, and the computational method applied to each cell. Each of these steps can be implemented in several ways. Our STAPL particle transport library PDT provides a testbed for exploring these options.

The PDT project has been supported by the DOE and the NSF and the PDT code is currently being utilized and further developed in the DOE PSAAP CRASH center (Michigan, with Texas A&M collaborating, 'Center for Radiative Shock Hydrodynamics', DE-FC52-08NA28616). PDT is also being utilized and further developed as part of the DNDO-NSF Academic Research Initiative SHIELD project (Texas A&M, 'A Framework for Developing Novel Detection Systems Focused on Interdicting Shielded HEU', 2008-DN-077-ARI018-02). A strong scaling study for the discrete-ordinates transport code PDT are shown in Figure 7. Note that the scaling is computed relative to 64 and 32 processors, respectively; linear scaling is also shown on the figure for reference. Here are some relevant papers [76, 74]

References

[1] M. L. Adams. 'I Have An Idea!' an appreciation of Edward W. Larsen's contributions to particle transport. *Annals of Nuclear Energy*, 31(17):1963–1986, 2004.

[2] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger. STAPL: A standard template adaptive parallel C++ library. In *Proc. of the International Workshop on Advanced Compiler Technology for High Performance and Embedded Processors (IWACT)*, Bucharest, Romania, Jul 2001.

[3] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger. STAPL: An adaptive, generic parallel programming library for C++. In *Proceedings 14th Annual Workshop on Programming Languages and Compilers for Parallel Computing*, Cumberland Falls, Kentucky, Aug 2001.

[4] T. E. Bailey, M. L. Adams, B. Yang, and M. R. Zika. A piecewise linear finite element discretization of the diffusion equation for arbitrary polyhedral grids. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[5] O. B. Bayazit, Jyh-Ming Lien, and N. M. Amato. Swarming behavior using probabilistic roadmap techniques. *Lecture Notes in Computer Science*, 2005(3342):112–125, January 2005.

[6] O. B. Bayazit, Dawen Xie, and N. M. Amato. Iterative relaxation of constraints: A framework for improving automated motion planning. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 3433–3440, Edmonton, Alberta, Canada, 2005.

[7] Guy E. Blelloch, Siddhartha Chatterjee, Jonathan C. Hardwick, Jay Sipelstein, and Marco Zagha. Implementation of a portable nested data-parallel language. In *PPOPP*, pages 102–111, 1993.

[8] C. Boyle, P. I. E. de Oliveira, C. R. E. de Oliveira, M. L. Adams, and J. M. Galan. GERALD: a general environment for radiation analysis and design. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[9] Eric A. Brewer. High-level optimization via automated statistical modeling. In *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPoPP)*, pages 80–91, 1995.

[10] "A. Buss, T. Smith, G. Tanase, N. Thomas, M. Bianco, N. Amato, and L. Rauchwerger". Design for interoperability in stapl. In *Proceedings 21th Annual Workshop on Programming Languages and Compilers for Parallel Computing*, Edmonton, Canada, Aug., 2008.

[11] Antal A. Buss, Timmie Smith, Gabriel Tanase, Nathan Thomas, Mauro Bianco, Nancy M. Amato, and Lawrence Rauchwerger. Design for interoperability in STAPL: pMatrices and linear algebra algorithms. In *International Workshop on Languages and Compilers for Parallel Computing (LCPC), published in Lecture Notes in Computer Science (LNCS)*, volume 5335, pages 304–315, Edmonton, Alberta, Canada, July 2008.

[12] J. H. Chang and M. L. Adams. Effectiveness of various transport synthetic accelerationmethods with and without GMRES. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[13] K. T. Clarno and M. L. Adams. Capturing the effects of unlike neighbors in single-assembly calculations. *Nucl. Sci. Eng.*, 149:182–196, 2005.

[14] M. Garzaran, M. Prvulovic, J. Llaberia, V. Vinals, L. Rauchwerger, and J. Torrellas. Tradeoffs in buffering speculative memory state for thread-level speculation in multiprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2006.

[15] Michael Gibbs and Bjarne Stroustrup. Fast dynamic casting. *Software - Practice & Experience*, 35(12), 2005.

[16] W. D. Hawkins and M. L. Adams. Consistent stretched transport synthetic acceleration of one-dimensional Sn problems. *Trans. Amer. Nucl. Soc.*, 91, 2004.

[17] H. Hiruta, D. Y. Anistratov, and M. L. Adams. Splitting method for solving the coarse-mesh discretized low-order quasidiffusion equations. *Nucl. Sci. Eng.*, 149:162–181, 2005.

[18] R. Iyer, J. Perdue, L. Rauchwerger, N. M. Amato, and L. Bhuyan. An experimental evaluation of the HP V-Class and SGI Origin 2000 multiprocessors using microbenchmarks and scientific applications. *Int. J. Par. Prog.*, 33(4):307–350, 2005.

[19] A. Jula and L. Rauchwerger. Custom memory allocation for free: Improving data locality with container-centric memory allocation. In *Proceedings 19th Annual Workshop on Programming Languages and Compilers for Parallel Computing*, New Orleans, Louisiana, November 2006.

[20] A. Jula and L. Rauchwerger. Two memory allocators that use hints to improve locality. In ACM SIGPLAN Int. Symposium on Memory Management(ISMM'09), *Dublin, Ireland, June*, 2009.

[21] K. R. Koch, R. S. Baker, and R. E. Alcouffe. Solution of the first-order form of the 3D discrete ordinates equation on a massively parallel processor. *Transactions of the American Nuclear Society*, 65:198–199, 1992.

[22] B. D. Lansrud and M. L. Adams. A spatial multigrid iterative method for one-dimensional discrete-ordinates transport problems. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[23] B. D. Lansrud and M. L. Adams. A spatial multigrid iterative method for two-dimensional discrete-ordinates transport problems. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[24] E. E. Lewis and W. F. Miller. *Computational Methods of Neutron Transport*. American Nuclear Society, LaGrange Park, IL, 1993.

[25] X. Li, M. J. Garzaran, and D. Padua. A dynamically tuned sorting library. In *Proc. of the International Symposium on Code Generation and Optimization*, pages 111–124, March 2004.

[26] J.-M. Lien and N. M. Amato. Approximate convex decomposition of polygons. 2006.

[27] Jyh-Ming Lien, Samuel Rodriguez, Jean-Philippe Malric, and Nancy M. Amato. Shepherding behaviors with multiple shepherds. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3413–3418, April 2005.

[28] A. E. Maslowski and M. L. Adams. Behavior of continuous finite element discretizations of the slab-geometry transport equation. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[29] A. E. Maslowski and M. L. Adams. A new approach to the iterative solution of transport problems. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[30] M. M. Mathis, N. M. Amato, and M. L. Adams. A general performance model for parallel sweeps on orthogonal grids for particle transport calculations. In *Proc. ACM Int. Conf. Supercomputing (ICS)*, pages 255–263, 2000.

[31] W. McLendon III, B. Hendrickson, S. Plimpton, and L. Rauchwerger. Finding strongly connected components in distributed graphs. *J. Par. Dist. Comp.*, 65(8):901–910, March 2005.

[32] Marco Morales, Roger Pearce, and Nancy M. Amato. Analysis of the evolution of C-Space models built through incremental exploration. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1029–1034, April 2007.

[33] Marco Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M. Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, Berlin/Heidelberg, 2005. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Utrecht/Zeist, The Netherlands, 2004.

[34] Marco A. Morales A., Roger Pearce, and Nancy M. Amato. Metrics for analyzing the evolution of C-Space models. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1268–1273, May 2006.

[35] Marco A. Morales A., Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M. Amato. C-space subdivision and integration in feature-sensitive motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3114–3119, April 2005.

[36] David Musser, Gillmer Derge, and Atul Saini. *STL Tutorial and Reference Guide, Second Edition*. Addison-Wesley, 2001.

[37] S. J. Plimpton, B. Hendrickson, S. Burns, W. McLendon III, and L. Rauchwerger. Parallel algorithms for s_n transport on unstructured grids. *J. Nucl. Sci. Eng.*, 150(7):1–17, 2005.

[38] Of Signal Processing. SPIRAL: A generator for platform-adapted libraries.

[39] L. Rauchwerger, F. Arzu, and K. Ouchi. Standard Templates Adaptive Parallel Library. In *Proc. of the 4th International Workshop on Languages, Compilers and Run-Time Systems for Scalable Computers (LCR)*, Pittsburgh, PA, May 1998.

[40] Lawrence Rauchwerger and Nancy Amato. Smartapps: Middle-ware for adaptive applications on reconfigurable platforms. ACM SIGOPS Operating Systems Reviews, *Special Issue on Operating and Runtime Systems for High-End Computing Systems*, **40***(2):73–82*, 2006.

[41] Gabriel Dos Reis and Bjarne Stroustrup. Specifying C++ concepts. In *Proc. ACM Symp. on Princ. of Prog. Lan. (POPL)*, 2006.

[42] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato. An obstacle-based rapidly-exploring random tree. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2006.

[43] Samuel Rodriguez, Jyh-Ming Lien, and Nancy M. Amato. Planning motion in completely deformable environments. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2466–2471, May 2006.

[44] Samuel Rodriguez, Jyh-Ming Lien, and Nancy M. Amato. A framework for planning motion in environments with moving obstacles. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2007.

[45] Samuel Rodriguez, Shawna Thomas, Roger Pearce, and Nancy M. Amato. (RESAMPL): A region-sensitive adaptive motion planner. In *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, Berlin/Heidelberg, 2008. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), New York City, 2006.

[46] S. Rus, G. He, and L. Rauchwerger. Scalable array SSA and array data flow analysis. In *Proceedings Annual Workshop on Programming Languages and Compilers for Parallel Computing*, 2005.

[47] Silvius Rus, Dongmin Zhang, and Lawrence Rauchwerger. The value evolution graph and its use in memory reference analysis. In *Proceedings of the 13-th International Conference on Parallel Architectures and Compilation Techniques, Antibes Juan-les-Pins, France*, October 2004.

[48] Steven Saunders and Lawrence Rauchwerger. Armi: an adaptive, platform independent communication library. In *Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 230–241, San Diego, California, USA, 2003. ACM.

[49] Steven Saunders, Nathan Thomas, Nancy Amato, and Lawrence Rauchwerger. Adaptive parallel sorting in the STAPL library. Technical Report TR01-005, Parasol Laboratory, Texas A&M University, November 2001.

[50] H. G. Stone and M. L. Adams. New spatial discretization methods for transporton unstructured grids. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[51] J. C. Stone and M. L. Adams. Adaptive discrete-ordinates algorithms and strategies. In *Proc. Conf. Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications*, 2005.

[52] B. Stroustrup. Abstraction and the C++ machine model. In *Internat. Conf. on Embedded Software and Systems (ICESS)*, December 2004.

[53] Bjarne Stroustrup. A brief look at C++0x. In *Modern C++ design and programming*, Shanghai, China, November 2005.

[54] Bjarne Stroustrup. The design of C++0x. *C/C++ Users Journal*, May 2005.

[55] Bjarne Stroustrup. A rationale for semantically enhanced library languages. In *Workshop on Library-Centric Software Design (LCSD)*, 2005.

[56] Bjarne Stroustrup and Gabriel Dos Reis. Supporting sell for high-performance computing. In *Workshop on Languages and Compilers for Parallel Computing (LCPC)*, October 2005.

[57] G. Tanase, M. Bianco, N. Amato, and L. Rauchwerger. The stapl parray. In Proceedings of the 2007 Workshop on Memory Performance: Dealing with Applications, Systems and Architecture (MEDEA'07) (Brasov, Romania). ACM, New York, NY, 73-80. DOI= http://doi.acm.org/10.1145/1327171.1327180, 2007.

[58] G. Tanase, C. Raman, M. Bianco, N. Amato, and L. Rauchwerger. Associative parallel containers in stapl. In *Proceedings 20th Annual Workshop on Programming Languages and Compilers for Parallel Computing*, Urbana-Chapmaign, IL, Oct, 2007.

[59] Gabriel Tanase, Mauro Bianco, Nancy M. Amato, and Lawrence Rauchwerger. The STAPL pArray. In *Proceedings of the 2007 Workshop on Memory Performance (MEDEA)*, pages 73–80, Brasov, Romania, 2007.

[60] Gabriel Tanase, Chidambareswaran Raman, Mauro Bianco, Nancy M. Amato, and Lawrence Rauchwerger. Associative parallel containers in STAPL. In *International Workshop on Languages and Compilers for Parallel Computing (LCPC), published in Lecture Notes in Computer Science (LNCS)*, volume 5234, pages 156–171, Urbana-Champaign, 2008.

[61] X. Tang, B. Kirkpatrick, S. Thomas, G. Song, and N. M. Amato. Using motion planning to study RNA folding kinetics. *J. Comput. Biol.*, 12(6):862–881, 2005. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2004.

[62] X. Tang, S. Thomas, and N. M. Amato. Planning with reachable distances: Fast enforcement of closure constraints. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2694–2699, Roma, Italy, 2007.

[63] X. Tang, S. Thomas, and N. M. Amato. Planning with reachable distances. In *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*, Guanajuato, México, 2008.

[64] X. Tang, S. Thomas, L. Tapia, and N. M. Amato. Tools for simulating and analyzing RNA folding kinetics. In *Proc. Int. Conf. Comput. Molecular Biology (RECOMB)*, pages 268–282, 2007.

[65] Xinyu Tang, Shawna Thomas, Lydia Tapia, and Nancy M. Amato. Tools for simulating and analyzing RNA folding kinetics. *J. Comput. Biol.*, 2008. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2007. Submitted.

[66] Xinyu Tang, Shawna Thomas, Lydia Tapia, David P. Giedroc, and Nancy M. Amato. Simulating RNA folding kinetics on approximated energy landscapes. *J. Mol. Biol.*, 381:1055–1067, 2008.

[67] Lydia Tapia, Xinyu Tang, Shawna Thomas, and Nancy M. Amato. Kinetics analysis methods for approximate folding landscapes. In *Int. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, pages 539–548, 2007.

[68] Lydia Tapia, Xinyu Tang, Shawna Thomas, and Nancy M. Amato. Kinetics analysis methods for approximate folding landscapes. *Bioinformatics*, 23(13):539–548, 2007. Special issue of Int. Conf. on Intelligent Systems for Molecular Biology (ISMB) & European Conf. on Computational Biology (ECCB) 2007.

[69] Lydia Tapia, Shawna Thomas, Bryan Boyd, and Nancy M. Amato. An unsupervised adaptive strategy for constructing probabilistic roadmaps. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2009.

[70] N. Thomas, S. Saunders, T. Smith, G. Tanase, and L. Rauchwerger. Armi: A high level communication library for stapl. *Parallel Processing Letters*, **16***(2):261-280*, 2005.

[71] Nathan Thomas, Gabriel Tanase, Olga Tkachyshyn, Jack Perdue, Nancy M. Amato, and Lawrence Rauchwerger. "a framework for adaptive algorithm selection in STAPL". In *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog. (PPoPP)*, pages 277–288, 2005.

[72] S. Thomas, G. Song, and N. Amato. Protein folding by motion planning. *Physical Biology*, 2:S148–S155, 2005.

[73] S. Thomas, G. Tanase, L. Dale, J. Moreira, L.Rauchwerger, and N. Amato. Parallel protein folding with stapl. *Concurrency and Computation: Practice and Exprience*, 2005.

[74] S. Thomas, G. Tanase, L. K. Dale, J. M. Moreira, L.Rauchwerger, and N. M. Amato. Parallel protein folding with STAPL. *Concurrency and Computation: Practice and Experience*, 2005.

[75] Shawna Thomas, Marco Morales, Xinyu Tang, and Nancy M. Amato. Biasing samplers to improve performance. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1625–1630, April 2007.

[76] Shawna Thomas, Gabriel Tanase, Lucia K. Dale, Lawrence Rauchwerger Jose E. Moreira, and Nancy M. Amato. Parallel protein folding with stapl. Concurrency and Computation: Practice and Experience, **17***(14), pp. 1643–1656*, 2005.

[77] Shawna Thomas, Xinyu Tang, Lydia Tapia, and Nancy M. Amato. Simulating protein motions with rigidity analysis. In *Proc. Int. Conf. Comput. Molecular Biology (RECOMB)*, pages 394–409, 2006.

[78] Shawna Thomas, Xinyu Tang, Lydia Tapia, and Nancy M. Amato. Simulating protein motions with rigidity analysis. *J. Comput. Biol.*, 14(6):839–855, 2007. Special issue of Int. Conf. Comput. Molecular Biology (RECOMB) 2006.

[79] Aimée Vargas Estrada, Jyh-Ming Lien, and Nancy M. Amato. Vizmo++: a visualization, authoring, and educational tool for motion planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 727–732, May 2006.

[80] R. Clint Whaley, Antoine Petitet, and J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, January 2001.

[81] Dawen Xie, Marco Morales, Roger Pearce, Shawna Thomas, Jyh-Ming Lien, and Nancy M. Amato. Incremental map generation (IMG). In *Algorithmic Foundation of Robotics VII*, pages 53–68. Springer, Berlin/Heidelberg, 2008. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), New York City, 2006.

[82] H. Yu and L. Rauchwerger. An adaptive algorithm selection framework. *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[83] Hao Yu and Lawrence Rauchwerger. Adaptive reduction parallelization techniques. In *ICS '00: Proceedings of the 14th International Conference on Supercomputing*, pages 66–77, New York, NY, USA, 2000. ACM Press.

[84] Hao Yu, Dongmin Zhang, and Lawrence Rauchwerger. An adaptive algorithm selection framework. In *Proc. Intern. Conf. Parallel Architecture and Compilation Techniques (PACT)*, pages 278–289. IEEE Computer Society, 2004.

[85] Hao Yu, Dongmin Zhang, and Lawrence Rauchwerger. An adaptive algorithm selection framework. In *Proceedings of the 13-th International Conference on Parallel Architectures and Compilation Techniques, Antibes Juan-les-Pins, France*, October 2004.