

Creating Software Technology to Harness the Power of Leadership-class Computing Systems

John Mellor-Crummey¹, Peter Beckman², Jack Dongarra³, Barton Miller⁴, and Katherine Yelick⁵

Abstract:

As part of the SciDAC-2 program, science teams aim to tackle problems that require simulation and modeling at unprecedented scales. The DOE Office of Science has invested heavily in leadership-class computing platforms to support this effort. A grand challenge for computer science is to develop software technology that makes it possible to harness the power of these systems and to apply it productively to aid scientific discovery. To address this challenge, the Center for Scalable Application Development Software (CScADS) is pursuing an integrated set of activities that include community vision-building, collaborative research, and development of open-source software infrastructures to support efficient scientific computing on the emerging leadership-class platforms.

1 Introduction

Technical computing, loosely defined as computing based on mathematical models of physical phenomena, spans a wide range of computational requirements. At the low end are computations that may take only milliseconds when programmed using a scripting language, or even a spreadsheet. At the other extreme are complex problems in physics and biology that are not yet tractable. Applications may range from computational experiments drafted by an individual in a few hours to applications with millions of lines of code written over decades by hundreds of programmers.

Existing software for technical computing does not make it easy to write applications that span the space of application size and complexity, nor the space of system size and complexity. One major reason for this is the mismatch between problem formulation and programming models. Many algorithms can be expressed concisely in mathematical notation, e.g., on a single page of a scientific paper. When translated into a scripting language such as Matlab, they remain reasonably concise. Even when they are translated to simple loops in Fortran (or C), the expansion is minimal if calls to standard domain libraries are used.

Unfortunately, this situation changes dramatically when one attempts to tailor applications to achieve high performance. Without extensive tuning, data-intensive scientific applications typically achieve less than 10% efficiency on large-scale microprocessor-based parallel systems. Both the scarcity and scale of leadership-class systems make code efficiency an important concern. Boosting efficiency typically requires enormous effort by application developers. Tuning applications requires explicitly managing all aspects of the hardware—processor functional units, memory hierarchy, communication, and I/O—with great care. Today such restructuring is primarily the responsibility of application developers. For this reason, programming modern high-end computer systems is an enormous productivity sink. Furthermore, tailoring applications for high performance on a particular system leads to code that is hard to understand, verify, and maintain.

A final challenge is that of moving parallel applications to new high-end computing platforms. The experience of developers at the national laboratories is that re-tuning an application for new processor architectures, or even for new models of a given architecture, can require amounts of effort that equal or

¹ Department of Computer Science, Rice University, Houston, TX 77251, johnmc@cs.rice.edu.

² Mathematics & Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, beckman@mcs.anl.gov.

³ Computer Science Department, University of Tennessee, ORNL, and University of Manchester, dongarra@cs.utk.edu.

⁴ Computer Sciences, University of Wisconsin - Madison, Madison, WI 53706, bart@cs.wisc.edu.

⁵ Computer Science Division University of California at Berkeley Berkeley, CA 94720, yelick@cs.berkeley.edu.

exceed the effort to develop the application initially. Compounding this problem is the arrival of hybrid supercomputing systems such as LANL's Roadrunner with both Opteron and Cell processors [IBM 06]. On hybrid systems, applications must be partitioned and matched to the strengths of different processors to achieve the highest level of performance. If such systems are to be usable by a broad range of scientific users, we must provide automated strategies for mapping applications onto them.

1.1 Scalable Application Development Software

To increase the productivity of application developers for high-end systems, computational scientists need software tools that help automate, in full or in part, the process of scaling applications in three different dimensions:

- scaling from simple high-productivity languages on a laptop to efficient applications on high-end, single-processor workstations;
- scaling from small numbers of processors to full processor ensembles consisting of thousands of processors with minimal efficiency loss; and
- scaling from a single abstract program representation to tuned implementations for many different high-end machines and heterogeneous processors with minimal programming effort.

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0" + Indent at: 0.25"

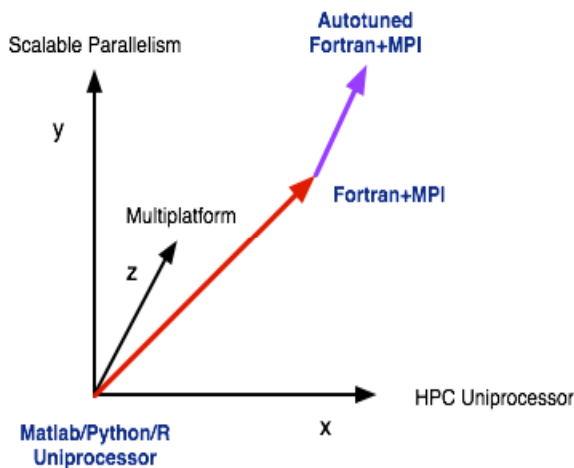


Figure 1: Scalable application development software.

We call software that supports scaling in these three dimensions *scalable application development software*. This idea is illustrated in Figure 1. In the figure, a base program is specified in a high-level language, such as Matlab, Python, or R. Using appropriate compiler technology, this specification might be translated directly to a high-performance scalable parallel implementation in Fortran or C plus MPI; then this implementation could be tuned independently to different platforms using automatic search strategies.

While scaling along any of these axes of scaling could be considered independently, we believe that there is significant leverage to be gained by considering them as aspects of a single process: that of moving from a simple high-level application specification to efficient, scalable parallel implementations on a variety of computing platforms. A benefit of this approach is that it encourages the maintenance of a

single source version for each application, with automatic or semi-automatic translation to high-performance platforms of different types and scales.

2 Center for Scalable Application Development Software (CScADS)

With support from the DOE's SciDAC-2 program, Rice University, Argonne National Laboratory, University of California – Berkeley, University of Tennessee – Knoxville, and University of Wisconsin – Madison, have established the Center for Scalable Application Development Software (CScADS). CScADS aims to explore strategies to support scalable application development with the goal of increasing the productivity of scientific application developers on high-end computer systems.

CScADS will support three basic activities: community outreach and vision-building, research on enabling technologies, and development of open-source software infrastructures to support the SciDAC-2 mission of making petascale computing practical. Achieving the goals of the SciDAC-2 program involves establishing partnerships with other academic institutions, DOE laboratories, leading-edge computing facilities, and industry.

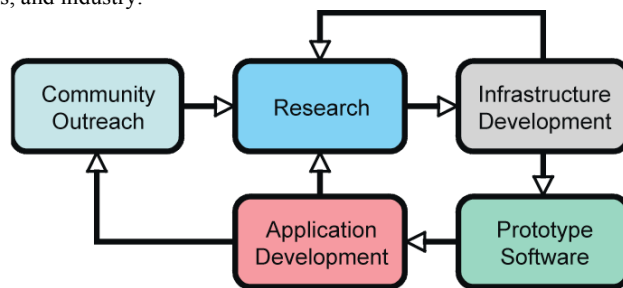


Figure 2: Relationship between CScADS activities.

Figure 2 illustrates the relationships between the Center's activities. The flow of ideas originates from two sources: the community outreach and vision-building workshops, and direct collaboration with application development. These activities focus research efforts on important problems. In turn, research drives the infrastructure development by identifying capabilities that are needed to support the long-range vision. Infrastructure feeds back into the research program, but also to prototype software tools that support further application development. Finally, experiences by developers using compilers, tools and libraries will spur the next cycle of research and development.

2.1 Community Outreach and Vision-Building

Achieving petascale performance with applications will require a close collaboration between scientists developing computational models and computer science teams developing enabling technologies. To engage the community in the challenges and foster inter-disciplinary collaborations, we have established the CScADS Summer Workshops – a series of one-week workshops that will focus on topics related to scalable software for the DOE's leadership-class systems. For summer 2007, we have scheduled a series of four workshops.

- *Automatic Tuning for Petascale Systems.* The goal of this workshop is to bring together researchers and practitioners to identify some of the opportunities and challenges of using automatic tuning on applications for petascale systems.
- *Performance Tools for Petascale Computing.* The goal of this workshop is to bring together tool researchers to foster collaboration on a community infrastructure for performance tools with the aim of accelerating development of tools for leadership-class platforms.

Formatted: Indent: Left: 0.15", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.25" + Indent at: 0.5", Tab stops: 0.15", List tab + Not at 0.25"

- *Petascale Architectures and Performance*. The goals of this workshop include familiarizing participants with the effective use of the DOE leadership-class systems.
- *Libraries and Algorithms for Petascale Applications*. The goal of this workshop to identify challenges for library and algorithm developers from the needs of the SciDAC applications, and to foster collaboration between the applications and library communities.

Formatted: Indent: Left: 0.15", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.25" + Indent at: 0.5", Tab stops: 0.15", List tab + Not at 0.25"

Formatted: Indent: Left: 0.15", Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.25" + Indent at: 0.5", Tab stops: 0.15", List tab + Not at 0.25"

The first two workshops are intended to engage researchers active in their focus areas. The last two workshops are intended to engage the broader community and to foster dialogue and collaboration between application teams and researchers developing enabling technologies. Each workshop will produce a report for the community that summarizes the challenges ahead and a path forward.

2.2 Research and Development

Several national reports (e.g., [PITAC 00]) have pointed out that open-source software represents an opportunity to address the shortage of software support for programming high-end systems. The power of this approach has been amply demonstrated by the success of Linux in fostering the development of operating systems for high-performance clusters.

The CScADS research program focuses on strategies for improving the productivity of application developers for developing high performance codes for leadership-class machines. Rather than attack a narrow range of problems within this space, we will explore a broad spectrum of issues because we believe that there is a high degree of synergy to be exploited.

Research on software support for high-end systems cannot be performed in a vacuum. Direct interaction between application developers and enabling technologies teams can clarify the problems that need to be addressed, yield insight into strategies for overcoming performance bottlenecks, identify how those strategies might be automated, and produce a vision for new tools and programming systems. To date, CScADS researchers have been interacting with developers of important DOE applications such as the Joule and INCITE codes, including S3D (see sidebar *Performance Analysis and Tuning of S3D*), GTC (a gyrokinetic code used to study plasma turbulence in toroidal fusion devices), and XGC1 (a code being developed by the SciDAC-2 Center for Plasma Edge Simulation).

2.3 Open-Source Software Infrastructure

To facilitate the research, both within CScADS and in the community at large, we are developing the *CScADS Open Software Suite* – an open-source software infrastructure to support compiler/programming-language research, development, and evaluation. This infrastructure, which is needed by our research as well as by a range of SciDAC projects and the wider HPC community, is not currently receiving development and deployment support elsewhere. Upon completion, the CScADS Open Software Suite will include

- *Compiler infrastructure* based on the Open64 compiler as well as Rice's D System compiler infrastructure to support high-level source-to-source optimization of programs, and
- *Performance tools infrastructure* to support binary analysis, instrumentation, data collection, and measurement interpretation that will draw from Rice's HPCToolkit and Wisconsin's Paradyn and Dyninst tools.

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0" + Indent at: 0.25"

3 CScADS Research Themes

In CScADS, we have begun a broad program of research on the software to support scalability in three dimensions: productivity, homogeneous scalability, and platform heterogeneity. We briefly outline the themes of this work in the next three sections.

3.1 Rapid Construction of High-Performance Applications

An application specification is high level if (1) it is written in a programming system that supports rapid prototyping; (2) aside from algorithm choice, it does not include any hardware-specific programming strategies (e.g., loop tiling); and (3) it is possible to generate code for the entire spectrum of different computing platforms from a single source version. The goal of CScADS productivity research is to explore how we can transform such high-level specifications into high performance implementations for leadership-class systems.

For higher productivity, we believe that developers should construct high-performance applications by using scripting languages to integrate domain-specific component libraries. At Rice we have been exploring a strategy, called *telescoping languages*, to generate high-performance compilers for scientific scripting languages [Kennedy 05]. The fundamental idea is to preprocess a library of components to produce a compiler that understands and optimizes component invocations as if they were language primitives. As part of this effort, we have been exploring analysis and optimization based on inference about generalized types [McCosh 02]. A goal of CScADS research is to explore how we can adapt these ideas to optimize programs based on the Common Component Architecture (CCA) [Armstrong 99].

3.2 Scaling to Homogeneous Parallel Systems

Achieving high performance on a modern microprocessor, though challenging, is not by itself enough for SciDAC applications: in addition, applications must be able to scale to the thousands or even hundreds of thousands of processors that make up a petascale computing platform. Two general classes of software systems are needed to make this feasible: (1) tools that analyze scalable performance and help the developer overcome bottlenecks, and (2) compiler support that can take higher-level languages and map them efficiently to large numbers of processors.

3.2.1 Tools for Scalable Parallel Performance Analysis and Improvement

Effectively harnessing leadership-class systems for capability computing is a grand challenge for computer science. Running codes that are poorly tuned on such systems would waste these precious resources. To help users tune codes for leadership-class systems, we are conducting research on performance tools that addresses the following challenges:

Analyzing integrated measurements. Understanding application performance requires capturing detailed information about parallel application behavior, including the interplay of computation, data movement, synchronization, and I/O. We are focusing on analysis techniques that help understand the interplay of these activities.

Taming the complexity of scale. Analysis and presentation techniques must support top-down analysis to cope with the complexity of large codes running on thousands of processors. To understand executions on thousands of processors, it is not practical to inspect them individually. We are exploring statistical techniques for classifying behaviors into equivalence classes and differential performance analysis techniques for identifying scalability bottlenecks.

Coping with dynamic parallelism. The arrival of multicore processors is will give rise to more dynamic threading models on processor nodes. Strategies to analyze the effectiveness of dynamic parallelism will be important in understanding performance on emerging processors.

This work on performance tools extends and complements activities in the Performance Engineering Research Institute (PERI). The CScADS tools research and development will build upon work at Rice on HPCToolkit [Mellor-Crummey 07] and work at Wisconsin on Dyninst and other tools for analysis and instrumentation of application binaries [Buck 00, Harris 05]. An outcome of this effort will be shared interoperable components that will accelerate development of better tools for analyzing the performance of applications running on leadership class systems.

3.2.2 Compiler Technology for Parallel Languages

The principal stumbling block to using parallel computers productively is that parallel programming models in wide use today place most of the burden of managing parallelism and optimizing parallel performance on application developers. We face a looming productivity crisis if we continue programming parallel systems at such a low level of abstraction as these parallel systems increase in scale and architectural complexity. As a component of CScADS research, we are exploring a range of compiler technologies for parallel systems ranging from technologies with near term impact to technologies for higher level programming models that we expect to pay off further in the future. This work is being done in conjunction with the DOE-funded Center for Programming Models for Scalable Parallel Computing. Technologies that we are exploring include

Partitioned global address space (PGAS) languages. Communication optimization will be critical to the performance of PGAS languages on large-scale systems. As part of CScADS, we intend to enhance the Open64 compiler infrastructure to support compile-time communication analysis of Co-Array Fortran and UPC. The Berkeley group will also develop communication optimizations for UPC [Chen 05a, Chen 05b].

Global array languages. High-level languages that support data-parallel programming using a global view offer a dramatically simpler alternative for programming parallel systems. Programming in such languages is simpler: one simply reads and writes shared variables without worrying about synchronization and data movement. An application programmer merely specifies how to partition the data and leaves the details of partitioning the computation and choreographing communication to a parallelizing compiler. Having an HPF program achieve over 10 TFLOPS on Japan's Earth Simulator has rekindled interest in high-level programming models within the US. Research challenges include improving the expressiveness, performance, and portability of high-level programming models.

Parallel scripting languages. Matlab and other scripting languages boost developer productivity both by providing a rich set of library primitives as well as by abstracting away mundane details of programming. Ongoing work at Rice is exploring compiler technology for Matlab. Work at Tennessee involves parallel implementations of scripting languages such as Matlab, Python, and Mathematica. As a part of this project, we are exploring compiler and run-time techniques that will enable such high level programming systems scale to much larger computation configurations while retaining support for most languages features.

3.2.3 Support for Multicore Platforms

Multicore chips will force at least two dimensions of parallelism into scalable architectures: (1) on-chip, shared memory parallelism and (2) cross-chip distributed-memory parallelism. Many architects predict that with processor speed improvements slowing, the number of cores per chip is likely to double every two years. In addition, many of the contemplated architectures will incorporate multithreading on each of the cores, adding a third dimension of parallelism. Based on this increased complexity, we see three principal challenges in dealing with scalable parallel systems constructed from multicore chips.

- *Decomposing available parallelism and mapping it well to available resources.* For a given loop nest, we will need to find instruction-level parallelism to exploit short-vector operations, multi-threaded parallelism to map across multiple cores and outer-loop parallelism to exploit an entire scalable system.
- *Keeping multiple cores busy requires that more data be transferred from off-chip memory.* In the near term, given the limitations on sockets, the aggregate off-chip bandwidth will not scale linearly with the number of cores. For this reason, it will be critical to transform applications to achieve high levels of cache reuse.
- *Choreographing parallelism and data movement.* Rather than having cores compute independently, coordinating their computation with synchronization can improve reuse.

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0" + Indent at: 0.25"

In CScADS, we are pursuing two approaches to cope with the challenges for efficient multi-core computing. First, Tennessee is exploring the design of algorithms and component libraries for systems employing multicore chips. (See sidebar *The Impact of Multicore on Math Software*.) This work seeks to achieve the highest possible performance, produce useful libraries, and drive the research on compilation strategies and automatic tuning for multicore chips. Second, Rice University is exploring compiler transformations to exploit multicore processors effectively by carefully partitioning and scheduling computation to enhance inter-core data reuse.

3.3 Portability and Support for Heterogeneous Platforms

The third dimension of scalability is mapping an application to different sequential and parallel computing platforms. Over the lifetime of an application, the effort spent in porting and retuning for new platforms can often exceed the original implementation effort. In support of portability, we are initially focusing on obtaining the highest possible performance on leadership-class machines. In addition, we will explore compilation and optimization of applications to permit them to run efficiently on computer systems that incorporate different kinds of computational elements, such as vector/SIMD and scalar processors.

3.3.1 Automatic Tuning to New Platforms

The success of libraries such as Atlas [Whaley 01] and FFTW [Frigo 98] has increased interest in automatic tuning of components and applications. The goal of research in this area is to develop compiler and run-time technology to identify which loop nests in a program are critical for high performance and restructure them appropriately to achieve the highest performance on a target platform.

The search space for alternative implementations of loop nests is too big to explore exhaustively. We have been exploring several strategies to reduce the cost of searching for the best loop structure. By leveraging capabilities of Rice's HPCToolkit, we can pinpoint sources of inefficiency at the loop level, which can guide exploration of transformation parameters. We have been employing model guidance along with search to dramatically reduce the size of the search required needed for good performance [Qasem 05].

As part of CScADS, the Rice and Tennessee groups are continuing their efforts based on LoopTool, HPCToolkit, and Atlas 2, with a focus on pre-tuning component libraries for various platforms. This work will provide variants of arbitrary component libraries optimized for different platforms and different application contexts, much as Atlas does today. A second group at Rice, led by Keith Cooper, is extending adaptive code optimization strategies [Almagor 04] to tune components. This work will explore adaptive transformations and aggressive interprocedural optimization.

3.3.2 Compiling to Heterogeneous Computing Platforms

Emerging high-end computing architectures are beginning to have heterogeneous computational components within a single system. Exploiting these features (or even coping with them) will be a challenge. We believe that new techniques must be incorporated into compilers and tools to support portable high-performance programming. To date, our work has explored compilation for chips with attached vector units such SSE on Intel chips and AltiVec on the IBM G5 [Zhao 05]. We are building upon this work to develop compiler techniques for partitioning and mapping computations onto the resources to which they are best suited. These techniques will be critical for effective use of systems that incorporate both vector and scalar elements in the same machine, such as those outlined in Cray's strategy for "adaptive supercomputing."

4 Conclusions

Our early experiences with S3D demonstrate the value of the CScADS approach of tightly coupling computer science research with application development and tuning. Performance challenges identified in S3D with HPCToolkit led directly to development of support for scalarization and loop unswitching in

LoopTool. In turn, LoopTool-optimized code is being incorporated into the reference version of S3D in preparation for large-scale simulation runs on ORNL's Cray XT3/XT4.

As CScADS moves forward with development of components in the Open Software Suite, maintaining a close connection with applications will ensure that the center's research will continue to address the fundamental challenges facing application teams developing codes for DOE leadership-class platforms.

----- Sidebar 1 starts here -----

The Impact of Multicore on Math Software

Multicore architectures are a disruptive technology for math software. The *Parallel Linear Algebra for Scalable Multi-Core Architectures (PLASMA)* project at the University of Tennessee and Oak Ridge National Laboratory aims to create software frameworks that help programmers achieve both high performance and portability. However, the current pace of change in architectures makes it premature to attack this goal directly. More experimentation is needed with new designs to understand how prior techniques can be adapted and to discover where novel approaches are needed to make programming frameworks sufficiently flexible for this new class of targets.

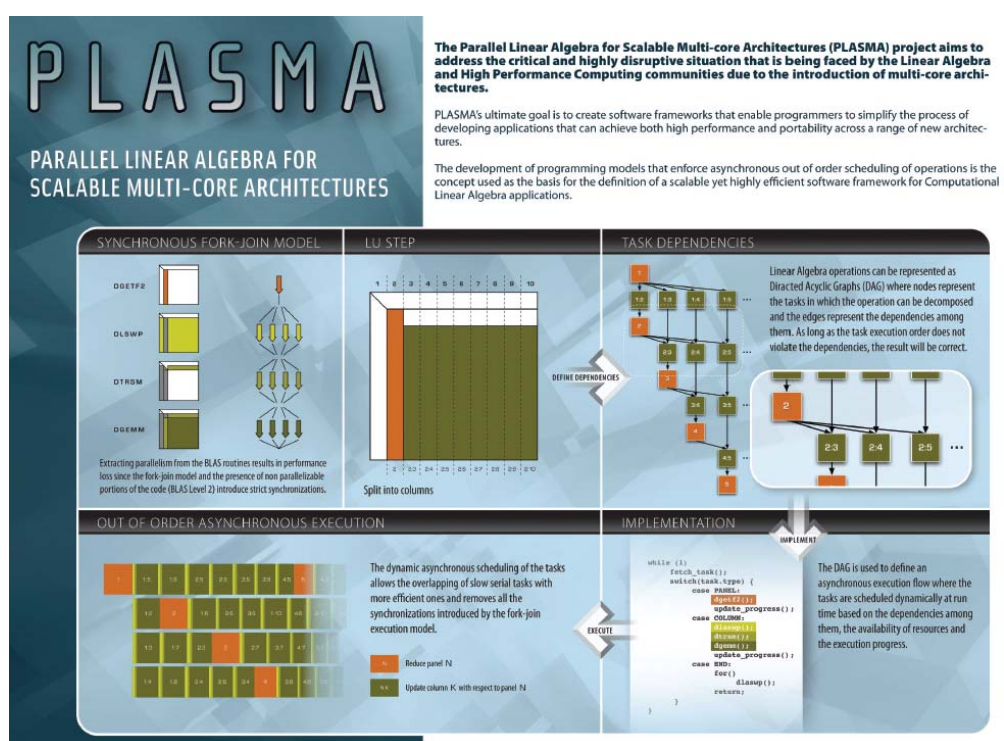


Figure 3: The PLASMA framework - Parallel Linear Algebra for Scalable Multi-core Architectures.

Our preliminary work on IBM's Cell processor shows that techniques for increasing parallelism and exploiting heterogeneity can dramatically accelerate application performance on these types of systems. In the PLASMA project, we are pursuing a three-pronged strategy:

- *Experiment with techniques.* We are exploring an execution model based on coarse-grain data flow to exploit dynamic and adaptive out-of-order execution patterns on multicore processors. Early experiences with matrix factorization techniques have led us to the idea of dynamic look-ahead, and our preliminary experiments show that this technique substantially improves performance.

Formatted: Bulleted + Level: 1 + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5"

- *Develop prototypes.* We are testing the most promising techniques to study their limits and gain insight into potential problems (e.g. portability). These prototypes enable us to assess how amenable these approaches are to dynamic adaptation and automated tuning.
- *Provide a design draft for the PLASMA framework.* We have begun developing an initial design of PLASMA frameworks for multi-core and hybrid architectures. We will distribute this design and software prototypes for community feedback.

We believe that in developing a programming framework for multicore processors there are clear advantages to our initial focus on Linear Algebra (LA) in general and Dense Linear Algebra (DLA) in particular. For one thing, DLA libraries are critically important to Computational Science across an enormous spectrum of disciplines and applications, so a framework of the type we envision for PLASMA will be indispensable and is urgently needed. However, DLA also has strategic advantages as a research vehicle, because the methods and algorithms that underlie it have been so thoroughly studied. Our long experience with this domain will enable us to devise techniques that maximally exploit emerging multicore processors.

----- Sidebar 1 ends here -----

----- Sidebar 2 starts here -----

LoopTool: Transforming Fortran Loop Nests for High Performance

The gap between memory speed and processor speed is increasing with each new generation of processors. As a result, lack of sufficient bandwidth between the processor and various levels of the memory hierarchy has become a principal obstacle to achieving high performance with data-intensive applications. When applications programs are written in a clear style that facilitates code development and maintainability, they often fail to exploit opportunities for data reuse. Although significant performance gains can be achieved by hand optimization to exploit reuse, automation is needed for improving productivity, portability, and maintainability.

Ideally, compilers would automatically tune loop nests. In practice, compilers often fail to achieve the desired result automatically. For this reason, Rice University is enhancing LoopTool – a compiler-based tool that assists expert human programmers by transforming Fortran loop nests for high performance [Qasem 03]. LoopTool enables application developers to apply a complex set of well-known loop transformations to improve data reuse at various levels of the memory hierarchy. Here we provide an overview of some of LoopTool’s key transformations. An example of LoopTool’s use and utility is described in the sidebar *Performance Analysis and Tuning of S3D*.

Controlled multi-level loop fusion. LoopTool performs multi-level loop fusion by adjusting the alignment of statements in different loops relative to a common iteration space. Loop fusion improves cache reuse by reducing the reuse distance between accesses to data within a cache line. Guided by user directives that specify exactly which loops to fuse, LoopTool verifies the legality of fusion before performing the transformation.

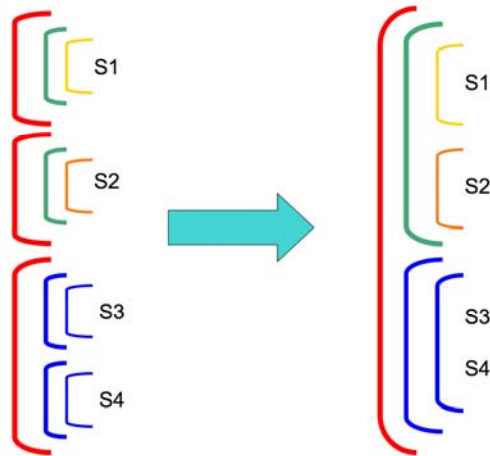


Figure 4: Multi-level fusion in which selected corresponding loops are fused and others are not.

Unroll-and-jam. Applying the unroll-and-jam transformation to a loop nest unrolls an outer loop and then fuses resulting copies of the loop it encloses. This transformation is useful for exploiting temporal reuse across iterations of an outer loop. Applying this transformation to a loop nest brings the reuse in the outer loops closer together, which can improve cache and sometimes register reuse.

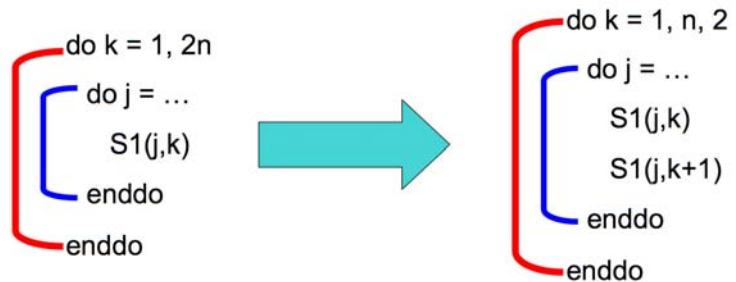


Figure 5: Unroll and jam of the outer loop into an inner loop; code to handle $\text{MOD}(n,2) = 1$ omitted.

Array contraction. If both the definitions and all uses of a temporary array fall in the same loop nest after fusion, often LoopTool can automatically reduce the storage footprint by applying array contraction if it can prove that only a subset of the values need to be live simultaneously. Reducing a computation's storage footprint enhances data reuse and can reduce the memory bandwidth required if the reduced-size array fits into cache at some level.

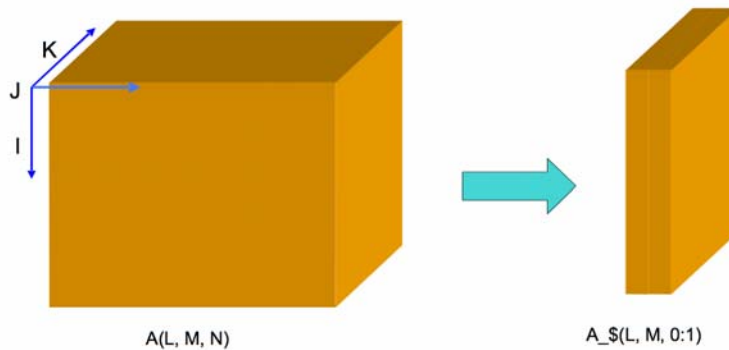


Figure 6: Contracting the third dimension of a 3D array down to 2 planes.

Loop unswitching. Unswitching a loop means hoisting a conditional within a loop nest out of one or more levels of enclosing loops and creating a custom version of the loop nest for the true and false branches of the conditional. By creating condition-free loop bodies, unswitching enables instructions to be scheduled more effectively.

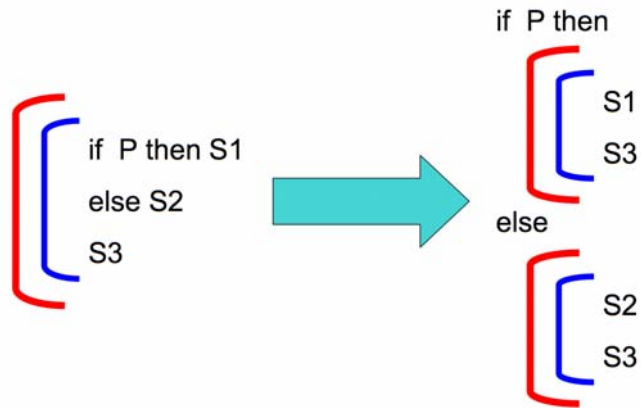


Figure 7: Unswitching a conditional out of two enclosing loops.

Guard-free core generation. Using a symbolic set manipulation package, we compute the iteration space for a guard-free core loop nest from a fused computation. Pieces of iteration spaces are clipped off along each dimension of the fused loop nest to uncover a guard-free rectangular core.

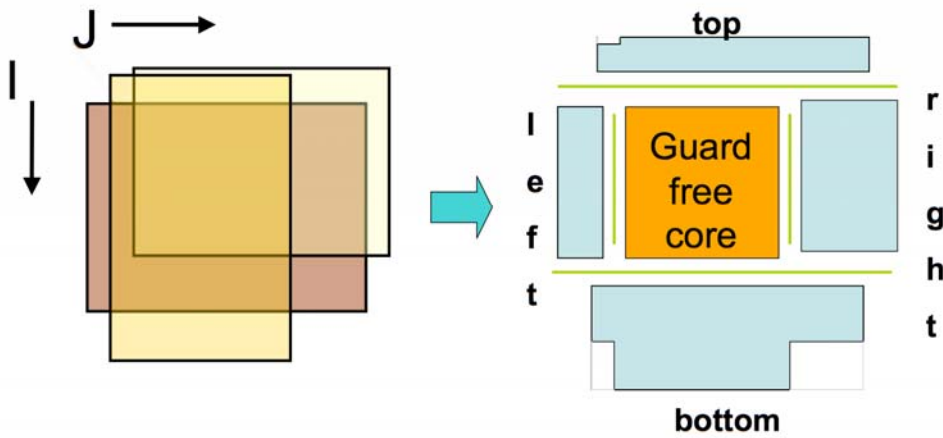


Figure 8: Generating a guard-free core for overlapping 2D iteration spaces.

----- Sidebar 2 ends here -----

----- Sidebar 3 starts here -----

Performance Analysis and Tuning of S3D

The S3D code being developed at Sandia National Laboratory is a massively parallel solver for turbulent reacting flows [Monroe 02]. The code includes multiple physical and chemical aspects, such as detailed chemistry and molecular transport. S3D is a focus of analysis and optimization by a PERI performance “Tiger Team” to help optimize it for large-scale simulation runs on the leadership-class Cray XT3/XT4 at Oak Ridge National Laboratory.

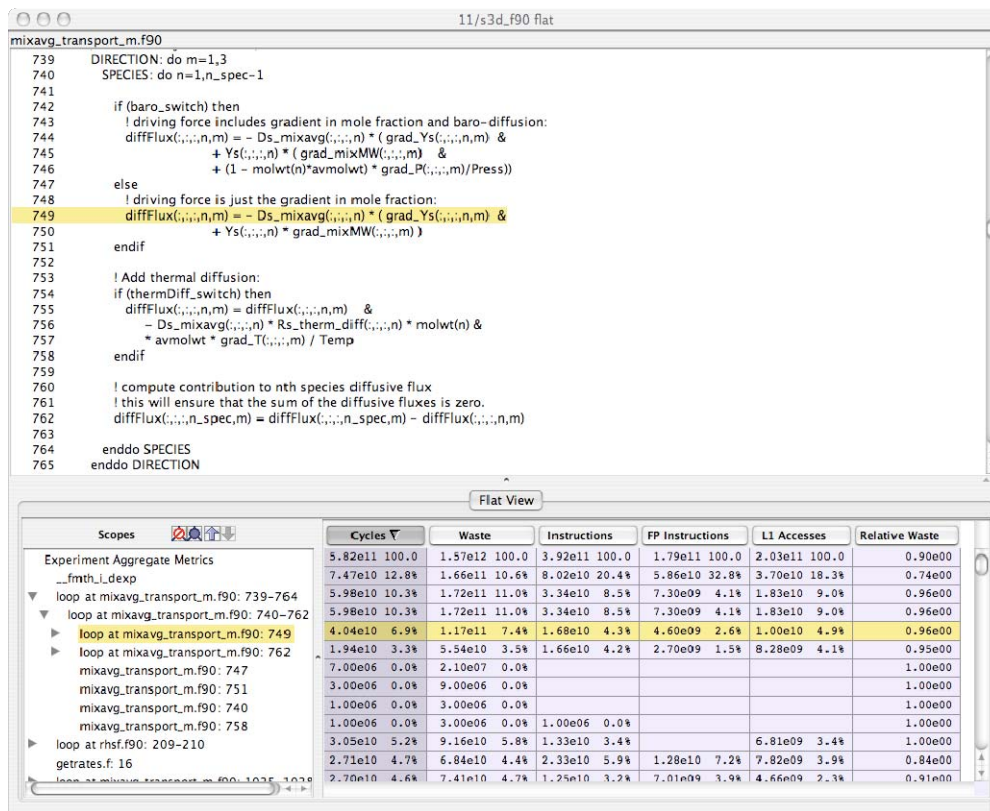


Figure 9: HPCToolkit's hpcviewer displaying a loop-level profile for S3D.

Figure 9 shows a flat profile of a single-processor execution of S3D on a 2.2 GHz dual-core Opteron 275 collected with Rice's HPCToolkit – a performance tool being developed with support from both PERI and CScADS. The tool provides a navigation pane that contains a rank-ordered display of program constructs (procedures, loops, and lines), a metric pane that shows both measured metrics (*cycles*, *instructions*, *FP instructions*, and *L1 cache accesses*) and derived metrics (*waste* and *relative waste*), and a source pane that displays code for the selected scope. Here, program scopes are rank ordered by cycles – the column selected in the metric pane. The source code pane shows the corresponding computation over a 5D data structure. Loops over the DIRECTION and SPECIES are explicit in the code; other 3D loops are implicit in the Fortran 90 array operations. The *waste* column in the metric pane represents the difference between the theoretical peak number of FLOPs possible on an

Opteron 275 core and the number of actual FLOPs executed, i.e., (2 x CYCLES) – FLOPs. This metric tells us how much unrealized opportunity for floating point computation is associated with each context. The *relative waste* metric shows the fraction of theoretical peak performance we are wasting in each context. With this metric, we can see that this loop nest achieves only 4% of the theoretical peak.

```

!dir$ uj 3
do m=1,3 ! DIRECTION
!dir$ uj 2
do n=1,n_spec-1 !SPECIES

!dir$ unswitch 2
if (baro_switch) then
! driving force includes gradient in mole fraction and baro-diffusion:
!dir$ fuse 1 1 1
diffFlux(:,:,:,n,m) = - Ds_mixavg(:,:,:,n) * ( grad_Ys(:,:,:,n,m) &
+ Ys(:,:,:,n) * ( grad_mixMW(:,:,:,m) &
+ (1 - molwt(n)*avmolwt) * grad_P(:,:,:,m)/Press))
else
! driving force is just the gradient in mole fraction:
!dir$ fuse 1 1 1
diffFlux(:,:,:,n,m) = - Ds_mixavg(:,:,:,n) * ( grad_Ys(:,:,:,n,m) &
+ Ys(:,:,:,n) * grad_mixMW(:,:,:,m) )
endif

! Add thermal diffusion:
!dir$ unswitch 2
if (thermDiff_switch) then
!dir$ fuse 1 1 1
diffFlux(:,:,:,n,m) = diffFlux(:,:,:,n,m) - Ds_mixavg(:,:,:,n) *
Rs_therm_diff(:,:,:,n) * molwt(n) * avmolwt * grad_T(:,:,:,m) / Temp
endif

! compute contribution to nth species diffusive flux
! this will ensure that the sum of the diffusive fluxes is zero.
!dir$ fuse 1 1 1
diffFlux(:,:,:,n_spec,m) = diffFlux(:,:,:,n_spec,m) - diffFlux(:,:,:,n,m)

enddo ! SPECIES
enddo ! DIRECTION

```

Figure 10: An S3D loop nest annotated with a transformation recipe for CScADS's LoopTool.

- | Study of the loop nest in [Figure 9](#)[Figure 8](#) reveals several unexploited opportunities for data reuse. The `diffFlux` array slice being computed in the first vector statement is reused in subsequent statements. Within individual vector statements there are opportunities for reuse across different iterations of the SPECIES and DIRECTION loops since many array references lack `m` and `n` subscripts. By applying a sophisticated set of loop transformations, these opportunities for data reuse can be exploited.
- | [Figure 10](#)[Figure 9](#) shows the loop nest decorated with a LoopTool code transformation recipe. The recipe indicates that (a) the two “if” conditions should be unswitched out of all loops to create four customized loop kernels (one for each switch setting), (b) loops for all of the 3D vector computations should be fused, (c) the DIRECTION loop should be unrolled completely and jammed inside the innermost loop of the 5D loop nest, and (d) the SPECIES loop should be unrolled by two, with pairs of its iterations jammed into the innermost loop.

Based on these directives, LoopTool produces four customized loop nests. The LoopTool-generated code for the loop nest shown in [Figure 10](#) runs 2.94 times faster than the original, which cuts the entire program's execution time on a 50 x 50 x 50 problem by 6.8%.

----- Sidebar 3 ends here -----

Bibliography

- [Almagor 04] L. Almagor, Keith D. Cooper, Alexander Grosul, Timothy J. Harvey, Steven W. Reeves, Devika Subramanian, Linda Torczon, and Todd Waterman. Finding effective compilation sequences. *Proceedings of the 2004 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, Washington, D.C., USA, June 2004, pages 231-239.
- [Armstrong 99] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. Mcinnes, S. Parker, and B. Smolinsky. Toward a common component architecture for high performance scientific computing. *Proceedings of the High Performance Distributed Computing Conference 1999*.
- [Buck 00] Bryan R. Buck and Jeffrey K. Hollingsworth. An API for runtime code patching. *Journal of High Performance Computing Applications* 14 (4), Winter 2000.
- [Chen 05a] C. Iancu, P. Husbands, P. Hargrove. HUNTING the overlap. *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2005.
- [Chen 05b] W. Chen, C. Iancu, and K. Yelick. Communication optimizations for fine-grained UPC applications. *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2005.
- [Collins 05] Eli D. Collins and Barton P. Miller. A loop-aware search strategy for automated performance analysis. *High Performance Computing and Communications (HPCC-05)*, Sorrento, Italy, September 2005. Appears in Springer-Verlag LNCS #3726.
- [Frigo 98] M. Frigo and S. Johnson. FFTW: An adaptive software architecture for the FFT. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Seattle, WA, May 1998.
- [Harris 05] Laune C. Harris and Barton P. Miller. Practical analysis of stripped binary code. *Workshop on Binary Instrumentation and Applications (WBIA-05)*, St. Louis, Missouri, September 2005.
- [IBM 06] IBM, IBM to Build World's First Cell Broadband Engine Based Supercomputer. Sept. 2006. <http://www-03.ibm.com/press/us/en/pressrelease/20210.wss>.
- [Kennedy 05] K. Kennedy, B. Broom, A. Chauhan, R. Fowler, J. Garvin, C. Koelbel, C. McCosh, and J. Mellor-Crummey. Telescoping languages: A system for automatic generation of domain languages. *Proceedings of the IEEE* 93(3):387-408 (2005).
- [McCosh 02] Cheryl McCosh. Type-Based Specialization in a Telescoping Compiler for Matlab. Master's Thesis, Department of Computer Science, Rice University, 2002.
- [Mellor-Crummey 07] John Mellor-Crummey, Nathan Tallent, Michael Fagan, and Jan Odegard. Application performance profiling on the Cray XD1 using HPCToolkit. Cray User's group meeting, Seattle, WA, May 2007. Available as <http://www.cs.rice.edu/~johnmc/papers/hpctoolkit-cug-2007.pdf>.
- [Monroe 02] Don Monroe. ENERGY Science with DIGITAL Combustors. *SciDAC Review*, June, 2002. <http://www.scidacreview.org/0602/html/combustion.html>.
- [PITAC 00] President's Information Technology Advisory Committee (PITAC). *Developing Open Source Software to Advance High End Computing*, October 2000.

[Qasem 03] Improving Performance with Integrated Program Transformations, Apan Qasem, Guohua Jin, John Mellor-Crummey, Department of Computer Science, Rice University. Technical Report TR03-419, October 2003.

[Qasem 05] A Qasem and K. Kennedy. A cache-conscious profitability model for empirical tuning of loop fusion. *Proceedings of the 2005 International Workshop on Languages and Compilers for Parallel Computing*, Hawthorne, NY, October 20-22, 2005.

[Whaley 01] C. Whaley, A. Petitet and J. Dongarra. Automated empirical optimizations of software and the ATLAS Project. *Parallel Computing*, vol. 27 (2001), no. 1, pg. 3-25.

[Zhao 05] Y. Zhao and K. Kennedy. Scalarization on short vector machines. *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, March 2005.

Figure Credits:

1. Ken Kennedy, Rice University
2. Ken Kennedy, Rice University
3. David S. Rogers, University of Tennessee
4. Apan Qasem, Rice University
5. John Mellor-Crummey, Rice University
6. Apan Qasem, Rice University
7. John Mellor-Crummey, Rice University
8. Apan Qasem, Rice University
9. John Mellor-Crummey, Rice University
10. John Mellor-Crummey, Rice University

Formatted: Numbered + Level: 1 +
Numbering Style: 1, 2, 3, ... + Start at: 1 +
Alignment: Left + Aligned at: 0.25" + Tab
after: 0.5" + Indent at: 0.5"