# 3.5 Computer Science

Group Leaders: Michael Heath, David Padua, and Daniel Reed

Faculty: Eric de Sturler, Herbert Edelsbrunner (Duke University), Michael Heath, Laxmikant (Sanjay) Kale, David Padua, Daniel Reed, Shang-Hua Teng, Paul Saylor, and Marianne Winslett

Research Scientists/Programmers: Milind Bhandarkar, Jay Hoeflinger, John Norris, and Kent Seamons

Post-Doctoral Associates: Joerg Liesen, Alla Sheffer, and Mario Pantano

Graduate Research Assistants: Gheorge Almasi, Cinda Heeren, Xiangmin (Jim) Jiao, Xiangyang Li, Xiaosong Ma, Ali Pinar, Sameer Paranjpye, Alper Ungor, Terry Wilmarth, Peng Wu, Shengke Yu, Zhe (Robert) Zhang, Jie Zheng, and Jiajing Zhu

Consultant: Timothy Baker (Princeton University)

## Overview

Two teams, Computational Mathematics and Geometry, and Computational Environment, address research in Computer Science. Work in Computational Mathematics and Geometry focuses on the areas of linear solvers, mesh generation and refinement, and interpolation between diverse discretizations. The target of our Computational Environment team is to provide the broad computational infrastructure necessary to support complex, large-scale simulations in general, and for rocket simulation in particular. Areas of research include parallel programming environments, compiler optimization and parallelization, parallel performance analysis and tools, and parallel input/output. The Computer Science activity will be discussed under three headings covering evaluation of parallel programming paradigms, tools that support development and tuning of simulation codes, and research on algorithms for mesh generation.

## Parallel Programming Paradigms

During the last year, center personnel conducted several studies on parallel programming paradigms. Three of these studies considered the shared memory paradigm and the interaction with message passing paradigms. For the first project, a sequential version of a combus-

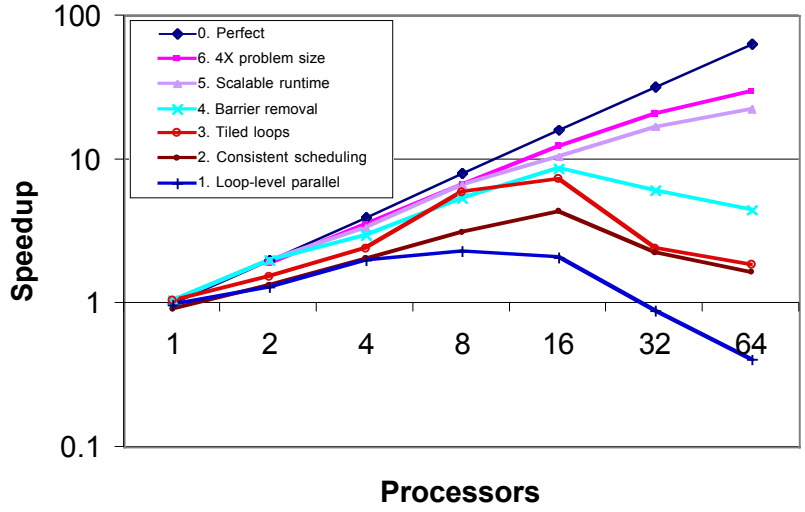


Fig. 3.5.1: Speedups for combustion code on SGI Origin 2000. Each line represents effect of manual optimization applied to OpenMP program.

tion code was converted to parallel form using *OpenMP*. The main conclusion of this study is that by judicious use of the primitives in *OpenMP,* it possible to get reasonable scaling performance (see Figure 3.5.1). The second project studied the combination of MPI with *OpenMP* in the context of *Rocflo*, running on 128 processors of a 256-processor Origin 2000 at NCSA, using what we learned about efficient use of *OpenMP*. It was concluded that for the case of 128 processors, the best performance is attained with a mixed approach involving eight MPI processes, each involving 16 *OpenMP* threads (see Figure 3.5.2). The third study combined MPI with shared memory parallelism in a non-hierarchical fashion by converting regular accesses to MPI while leaving irregular accesses to be handled by a software distributed shared memory system called *TreadMarks*. This combination proved advantageous in hand-experiments, producing good speedups in several benchmarks (see Figure 3.5.3).

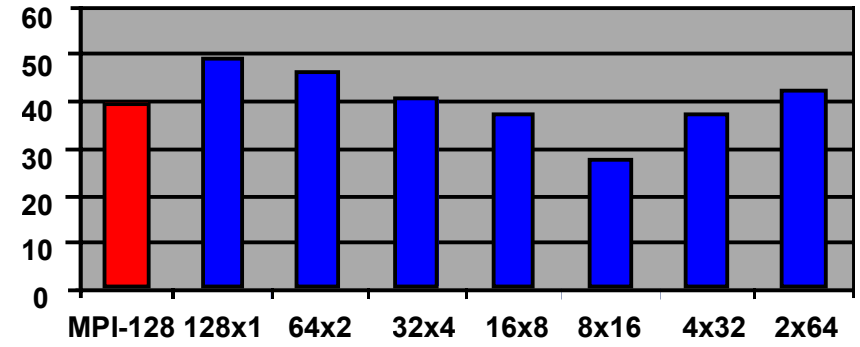

Fig. 3.5.2: Running times for ROCFLO code on 128 processors of 256 processor Origin, using only MPI (left-most bar) and various combinations of MPI and OpenMP threads. NxM means "N" MPI processes and "M" OpenMP thread within each MPI process.

The fourth study focused on an object-oriented parallel programming paradigm based on the *Converse* runtime system and the *Charm++* load-balancing framework. During Y3, this framework was adapted to procedural languages such as Fortran 90, and message passing programming paradigms. Detailed performance analysis of *ArrayMPI*, an implementation of MPI standard on top of *Charm++* has been done and has been optimized to have minimal overhead as compared to native MPI on several machines. *Rocflo*, *Rocsolid*, and *Rocface* have been ported to *ArrayMPI* with minimal changes and with little effort, providing them with automatic load balancing capabilities. A Fortran 90 front-end for the *Polaris* compiler is now been implemented to support the automatic conversion of future versions of *Rocflo*, *Rocsolid*, and *Rocface* to the *Charm++* framework.

The *ArrayMPI* library was also used to develop a prototype finite element method (FEM) framework. Using this FEM framework, code developers write sequential programs that implement the node and element-specific computations of the FEM solver while the details of parallelism, such as partitioning, communication, and load balancing are taken care of by the framework. A 2-D pressure-driven crack propagation application (originally developed by Geubelle, et al.) has been ported to use this framework. Performance results demonstrate the advantages of the multi-partitioning approach employed in the *Charm++* framework.

The multi-partitioned object-based approach has several other benefits besides automatic load balancing and efficient cache reuse. We have developed prototype software components that allow parallel programs to shrink and expand at runtime in order to use effectively inherently dynamic computational resources, such as clusters of workstations, which

are becoming increasingly popular parallel architectures. An efficient checkpointing framework has been developed that allows a parallel program to checkpoint and restart on a possibly different number of processors.

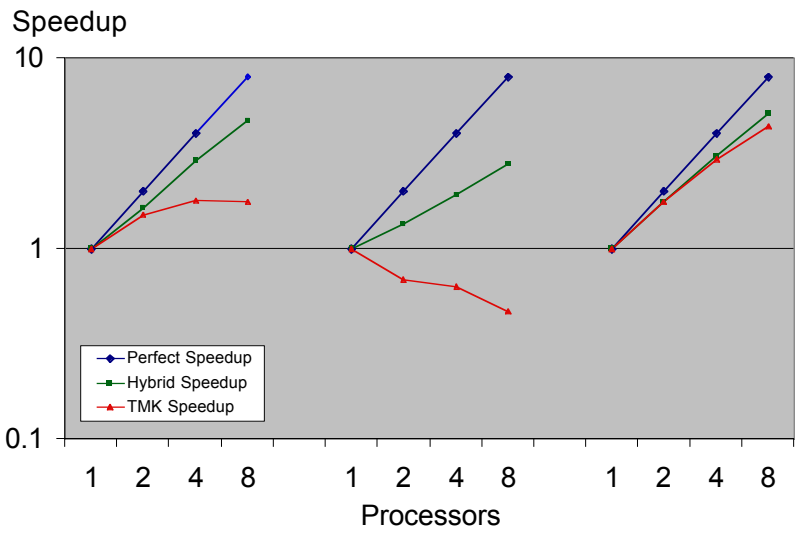


Fig. 3.5.3: Speedups for three benchmark applications for MPI/*TreadMarks* hybrid approach and pure *TreadMarks* on network of Sun workstations. Left-most benchmark is Jacobi solver, middle benchmark is SWIM from SPEC, and right-most benchmark is *TreadMarks* version of *Gauss*.

## Integrated GEN1 Implementation

Implementation of the integrated GEN1 code using *Charm++* with multi-partitioning is underway. We will make the existing partitioning schemes more flexible by allowing partitions of *Rocflo* and *Rocsolid* to be load-balanced independently. We plan to integrate our load balancing framework and the various application and system-level frameworks (e.g. *ArrayMPI*, FEM framework, shrink-expand, checkpoint) with other CS research activities within the Center, such as computational steering and a framework for coupling multi-physics modules. Also, we will enhance the FEM framework by adding the ability to handle multi-grid FEM and adaptive refinement. Scalability testing of the component codes of GEN1 software is planned in Y4.

## Software Integration Framework

The Software Integration Framework Team (SWIFT) is developing a programming environment for combining multiple parallel applications for multi-physics simulations with minimal changes to the individual component applications. This environment is mesh-aware, numerics-aware, and physics-aware. It handles the complications of interfacing simulation codes, while the internal parallelization, communication, and numerical solution methods of the applications remain intact. This is a major difference from the traditional framework approach. The emphasis in this work is on enabling and maintaining independence of application developers.

The new framework uses the parallel runtime system *Charm++* to run multiple parallel applications simultaneously. *Charm++* was extended to run multiple MPI programs. Each processor can run several chunks, each an application with a subset of its data, from different applications. This allows application developers to write stand-alone SPMD MPI programs, while the system effectively runs all applications simultaneously, allowing arbitrary distributions. Moreover, the system can dynamically start codes without extensive changes to the other codes. *Charm++* also provides load-balancing support. GEN1 was ported to *Charm++* as a first test of the viability of this approach. This port required minimal changes to the ap-

plication codes and the performance results were excellent. A few tests, especially of very large configurations, still must be completed.

Two application programming interfaces (APIs) have been developed for coupling independent applications. One is based on two MPI-based libraries. These provide calls for the applications (F90) and the interface code (C++) that transparently manage communication details and set up the required data structures. The other is based on *Autopilot*. *Autopilot* uses "sensors" and "actuators" to provide read and write access respectively to remote processes at runtime.

*Autopilot* is also used to provide import and export of data and method access to remote applications as part of a framework for coupling multiple stand-alone applications. As proof of concept, a version of GEN1 implemented using this approach. In this implementation the components that make up GEN1—*Rocface*, *Rocflo*, and *Rocsolid*—are stand-alone parallel codes. Application changes required for integration with *Autopilot* were minimal. Other capabilities provided by *Autopilot* include runtime data adaptation, computational steering, runtime visualization, and computational grid (heterogeneous) functionality.

## Tools

There are four main tool development efforts at CSAR: The *Rocketeer* visualization system, an interpolation module (*Rocface*) that serves as interface between the different components of our rocket simulation system, the *Panda* I/O library, and the *SVPablo* performance visualization system. The first two were initiated by CSAR to support its software development efforts and the latter two were developing systems when CSAR started and have provided useful support to CSAR's development and tuning effort.

### Rocketeer

*Rocketeer*, a 3-D scientific visualization tool developed at CSAR by Fiedler and Norris, was rewritten using *wxWindows* for cross-platform portability to run on Sun and Linux systems. An enhanced version (1.1) enables drawing spheres at a set of points (useful for showing the packing of particles in solid propellants and for showing aluminum droplets in the flow). Another recent enhancement is the ability to plot structural mechanics data in the undeformed coordinates, the deformed coordinates, or both, on the same image. The displacements can be exaggerated by a user-selectable factor. Support for discontinuous Galerkin finite element meshes and 2-D triangular meshes has also been added. It is now possible to specify external geometry files that can save a considerable amount of disk space when dealing with several dumps of large datasets when the grid's geometry never (or rarely) changes.

Work has progressed well in developing a client/server version of *Rocketeer*, called *Apollo* and *Houston*, respectively. This product should greatly improve response time when visualizing large datasets. Also, work has started on a parallel "companion" program for *Rocketeer*, called *Voyager*. It is a command-line program written with MPI that will render frames of a *Rocketeer* movie in parallel, greatly reducing the amount of time this takes, as well as freeing up the researcher's desktop workstation for other tasks.

Development of these products will continue during Y4. *Voyager* will be made more full-featured and easier to use. A feature will be added to *Rocketeer* and *Apollo* to generate a *Voyager* data file, which will be used to render movie frames. New visualization techniques,

such as vector fields, will be added. *Rocketeer* and *Apollo* will be ported to non-Unix platforms.

### Interface Mesh Association, Interpolation, and Propagation

Earlier work by Jiao and Heath on mesh association and conservative interpolation at CSAR has been extended during the last year. In the GEN1 integrated code, Farhat's conservative interpolation method was used in *Rocface* to transfer load and displacement between fluid and solid meshes, and efficient mesh association algorithms were developed for supporting this interpolation method. However, Farhat's method has some restrictions that limit its generality. For example, it requires that the fluid mesh is at least as fine as the solid mesh and it does not generalize easily to interpolating other types of physical data. To remove these constraints, a more general interpolation method is being investigated. The basic idea is to construct another mesh, each of whose elements is nested in an element of both input meshes. The data are then interpolated through this intermediate mesh and local conservation is enforced on the elements of the intermediate mesh. To support this scheme, an optimal algorithm for computing the intermediate mesh that runs in linear time is been designed and implement.

A related effort by Jiao and Heath for *Rocface* is to develop efficient methods for interface propagation. Interface propagation is concerned with moving the interface between two meshes given the speed at points on the interface, such as the burning rates at points on the surface of a solid propellant. The level set method is a standard method for interface propagation, but it has some drawbacks that limit its usefulness in our setting: it is very expensive in three dimensions, and because it defines the interface surface only implicitly, it is difficult to track the motion of the grid points of an interface mesh. A new approach for interface propagation based on the same entropy condition satisfying Huygens' principle as in the level set method is being designed. However, the new method works directly on the interface mesh instead of a volume mesh as in the level set method. New techniques have been designed for resolving local self-intersection and for detecting topological changes in the interface by introducing the concept of the null set of an interface, which allows "expendable" data to be identified and eliminated before self-intersection occurs. Implementation of the new method has been completed in two dimensions is under development for three dimensions.

### *Panda* I/O Library

*Panda*, a high-performance parallel I/O library from Winslett's group, has been adapted to work with both *Rocflo* and *Rocsolid* in GEN1 on the SGI Origin2000. A new file format has been added to *Panda* to enable one I/O node to gather data from multiple compute nodes and write multi-block snapshot files. *Panda* now supports HDF4 files so that the snapshots can be visualized by *Rocketeer*: data items from the same block are put together by random accessing of datasets although they are written in separate collective I/O calls. The current version of *Panda* was ported to Blue Horizon at UCSD. *Panda* has been used for parallel I/O in a new version of *Rocsolid*. This new version will be integrated with the *Panda* version of *Rocflo* to complete a version of the GEN1 code with parallel I/O (HDF format) and background file transfer capabilities.

An enhanced data migration facility was built into *Panda* to move efficiently data between different platforms. Data are transferred over multiple streams to get the best throughput. Recent experiments with selected applications showed good turnaround time. A global I/O performance optimization method has been adapted in the current *Panda* optimizer and improved optimization speed significantly.

During the coming year, new algorithms for I/O node placement and other issues will be ported to Blue Horizon, an IBM SP with eight-way SMPs, to improve I/O performance. Self-tuning algorithms for data migration will be studied. The *Panda* algorithms currently used for ordinary database query optimizers will be adapted to the needs of GEN1 and other simulation codes to be developed by CSAR, and the optimizer will be made more portable and modular.

### *SvPablo*

CSAR has continued to use and benefit from performance analysis tools developed by Reed's Pablo research group. Among these tools are *SvPablo,* a graphical user interface for browsing source code and runtime performance data, *Autopilot,* an infrastructure for real-time adaptive control, and *Virtue,* a collaborative virtual environment for data visualization. *SvPablo* and *Autopilot* have been installed on the CSE Intel Linux cluster, where *SvPablo* has been used to instrument the GEN1 code with *Autopilot* sensors and actuators. Using this instrumentation, it has been possible to measure and visualize the behavior of the GEN1 code using the Pablo *Virtue* performance visualization toolkit. In addition, *Autopilot* sensors and actuators have enabled implementation of a runtime computational steering capability.

The Pablo tools have also been used create a software integration framework that can connect multiple, remote applications. This newly developed interfacing toolkit has been used to construct the GEN1 simulation from its standalone constituents, *Rocface*, *Rocflo*, and *Rocsolid*. Each component uses MPI for internal communication and *Autopilot* sensors and actuators for inter-component communication.

During the coming year, we plan to continue investigating *Autopilot* as an integration framework mechanism while also exploring the coupling of *Autopilot* with *Charm++* and automation of component integration. We intend to evaluate *Virtue* as a tool for visualizing the remote application interactions of the *Autopilot*-coupled GEN1 code. We will continue to work closely with the Pablo research group and benefit from the further development of Pablo performance analysis tools.

## Computational Geometry and Mathematics

### Research on Algorithms for Mesh Generation

There have been major advances in algorithms for generating meshes for the space-time discontinuous Galerkin (DG) analysis by Haber, Sheffer, and Ungor. To solve a DG system using an element-by-element procedure, the space-time mesh must satisfy a cone constraint, specifically, the faces of the mesh cannot be steeper in the time direction than a specified angle function, $\alpha$. Whenever there is a face that violates the cone constraint, the elements at the face must be coupled in the solution.

Several new algorithms have been developed to address this problem. Two layer-based methods were suggested that are suitable when the cone constraint is uniform. An intersection-based method has been introduced for generating 1-D X time meshes which satisfy a non-uniform $\alpha$. Since this approach does not extend to higher dimensions, the problem requirements were relaxed to obtain a solution for higher dimensions. We considered the problem of generating a simplicial space-time mesh where the size of each group of elements that need to be coupled is bounded by a constant number, *k*. An algorithm for generating such meshes that is valid for any n-D X time domain has been developed. Future areas of research include integration with real space-time DG analysis applications, providing a parallel implementation of the algorithm, and developing mesh adaptivity techniques.

Other work in this area includes Sheffer and de Sturler's new method for parameterization of tessellated 3-D surfaces. The method uses only the necessary and sufficient constraints for a valid two-dimensional triangulation. As a result the existence of a theoretical solution to the minimization procedure is guaranteed. Further research will concentrate on efficient solution procedures for the derived optimization problem.

Finally, Sheffer has introduced a new method for CAD model simplification that uses a local analysis approach based on a clustering procedure. The method is especially suited for simplifications performed in preparation for meshing a model.

### Krylov Subspace Methods for Linear Systems and Eigenvalue Problems

Work of de Sturler focused on developing new robust and efficient (parallel) iterative methods and preconditioners for linear systems and eigenvalue problems. The emphasis in (non-Hermitian) linear solvers is on two approaches. One is to develop truncation strategies for optimal, but expensive, methods that give almost optimal convergence while maintaining orthogonality to only a small part of the Krylov subspace, which makes these methods more efficient. The other is to extend the non-optimal, and generally not very robust but very efficient, methods based on the nonsymmetric Lanczos process to include additional subspace information. This will make them more robust. Although the two classes of methods are quite different, these approaches share the same, still incomplete, theoretical underpinning based on the principal angles between the projection spaces of oblique and orthogonal projections. The emphasis in eigenvalue solvers is to improve the robustness and speed of convergence of the Jacobi-Davidson method. These solvers and appropriate preconditioners are implemented in collaboration with CSAR's Structures and Materials group and tested in the nonlinear ALE structures code. There is also extensive collaboration in this area with several people at SNL and LLNL to implement and test the linear solvers developed on several very large and difficult ASCI problems.

Saylor has focused on formulation and implementation of approximate inverse preconditioners for multi-group Boltzmann transport and their testing in a variant of Zeus, a transport code formulated by Norman and his colleagues. The near term objective is pure radiation transport, but in the longer term material heating and hydrodynamic components will be included as well. This approach is being compared to source iteration and full-system solutions, and also with the ARDRA code from LLNL. This work is in collaboration with Doub Swesty of SUNY Stony Brook and Dennis Smolarski of Santa Clara University. Saylor has also collaborated with Lehoucq, Heroux, and Hendrickson of SNL, and Mahiels of LLNL

on constraint satisfaction in the solution of the Navier-Stokes equations. For a nonsymmetric matrix, for example, conservation of mass can be enforced by orthogonal projection, in contrast to the traditional saddle-point matrix.