3.3 Computer Science

Group Leaders: Michael Heath and Laxmikant Kale

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

Research Scientists/Programmers: Milind Bhandarkar, Michael Campbell, Robert Fiedler, Xiangmin (Jim) Jiao, Orion Lawlor, John Norris, and Kent Seamons

Post-Doctoral Associates: Damrong Guoy and Joerg Liesen

Graduate Research Assistants: Yelda Aydin, Phillip Alexander, William Cochran, Damrong Guoy, Rebecca Hartman-Baker, Jonghyun Lee, Vanessa Lopez, Xiaosong Ma, Greg Mackey, Jeffrey Naisbitt, Joseph Pepper, Kenneth Scheiwe, Hanna Vanderzee, Terry Wilmarth, Shengke Yu, and Jiajing Zhu

Consultant: Timothy Baker (Princeton University)

Overview

Two teams — Computational Environment, and Computational Mathematics and Geometry — address research in the Computer Science Group. 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, parallel input/output, and visualization. Work in Computational Mathematics and Geometry focuses on the areas of linear solvers, mesh generation and adaptation, and interpolation between diverse discretizations.

Computational Environment

Software Integration Framework (Jiao, Heath, and Lawlor)

We continued the development of *Roccom*, which is our component-based integration framework for *Roc** (Figure 3.3.1). The philosophy of *Roccom* is object-oriented, enabling concurrent development of components, minimizing user effort for integrating application modules, and providing seamless interoperability between C/C++ and Fortran. We introduced some new concepts into *Roccom*, including member functions of distributed window attributes, and inheritance of a subset of data from a window onto another. We also added new features, including memory management and profiling into *Roccom*. These new functionalities have been very helpful for quick development, debugging, and testing of the integrated code. We also extended *Roccom* to enable application codes to work seamlessly with the parallel programming infrastructure, *Charm++*.

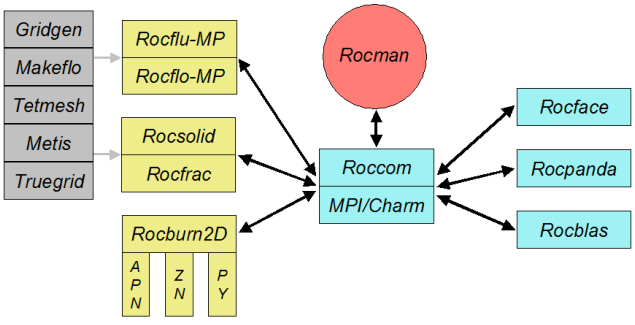

Fig. 3.3.1: *Roc** (GEN2.5) architecture.

Several new add-on components of *Roccom* were develop this year. An important one is *Rocman*, which encapsulates all the integration-specific codes, including manipulation of data for jump conditions, input/output for both restart and visualization, and predictor-corrector iterations. *Rocman* is implemented at a high-level using the *Roccom* API and leveraging the services provided by *Rocface*, *Rocpanda*, and *Rocblas,* the latter is a new component that provides basic algebraic subroutines for data attributes in *Roccom*. With *Rocman*, an application component needs to provide only its basic functionality and a few interface windows to be integrated into even our most complicated coupling algorithms.

Operations currently provided by *Rocblas* include component-wise addition, subtraction, multiplication, and division, and the vector operations dot product, 2-norm, swap, copy, and saxpy. These operations are defined for combinations of strided and contiguous data. One of the operands must be a nodal or elemental vector. The other operands may be a scalar pointer, a windowed/panel attribute (scalar or vector), or a nodal/elemental attribute (scalar or vector). The most common operations are provided for all of the types of data that require these operations.

Roccom has been demonstrated successfully for code integration. In the near future, we plan to improve and generalize its interface and apply it to other applications. We will continue to develop new add-on service modules for *Roccom*. For example, we expect to develop a set of new tools for error estimation, error control, and stability analysis. These tools will enable us to monitor individual applications and the integrated code, so that the accuracy and stability of a simulation can be established or violations of assumptions can be identified. This will be convenient for application developers and users who must perform code verification and debugging. Furthermore, it can also serve as a basis for solution-based mesh adaptation and time zooming.

Programming Environments (Kale, Lawlor, Wilmarth)

Research and development on programming environments at CSAR involved work on extending the parallel programming tools developed at Illinois, and incorporating them in CSAR codes. The feedback from this use has been instrumental in enhancing the utility and functionality of these tools, which include AMPI, the *Charm++* runtime infrastructure, and component frameworks.

AMPI: Feature enhancements continued on AMPI, our adaptive version of MPI, which provides automatic dynamic load balancing as well as other benefits for MPI codes. We have simplified the interface to AMPI to the point where AMPI-capable codes can be easily run on either AMPI or native MPI without change. A new capability, isomalloc heaps, was added that allows AMPI threads to migrate across processors without users having to write pack/unpack (pup) routines. The only requirement that remains is that all global variables (except read-only) must be encapsulated in dynamically allocated structures, available in F90. The programs can be automatically checkpointed, and restarted on a different number of processors. They can change the set of processors used on command.

Planned and ongoing work on AMPI includes support for asynchronous (split-phase) reductions, and optimized collective operations based on a library that learns from communication patterns observed in earlier iterations. We also plan to support and optimize some features of AMPI including processor-sections and processor-topologies.

***Charm*++ and Parallel Infrastructure:** Work on Charm++ completed last year includes ports to new machines (PSC Lemieux, NCSA Platinum cluster, etc.). We also extended *Charm*++ RTS to support "immediate" messages that are handled even while other user-code is running. New work on extending runtime support to large next-generation machines such as BlueGene/L has already begun. We have developed an emulator for BG/C as well as BG/L, which allows us to test *Charm*++ programs on emulated BG/L style machines with 100K+ processors, using only hundreds of conventional processors. A library for optimizing collective communication operation has been developed, and new algorithms will be added to it over time.

Component Frameworks: The work on frameworks aims at reusing code for commonly needed parallel data structures and operations. The current set of frameworks include an unstructured mesh framework, and *Mblock*, the structured grid framework. In addition, we developed preliminary versions of an AMR framework and a particles framework during last year.

Working with the Fluids Group, we have extended our unstructured-mesh parallel framework, originally developed to support FEM applications, to treat ghost regions and symmetry conditions in a generalized manner. This was motivated by the needs of the unstructured finite volume solver *Rocflu*, which has now been parallelized using the framework.

Performance Analysis (Campbell and Reed)

Over the past year, we have made considerable progress toward building a suite of performance analysis tools for CSAR. This was accomplished by enhancing existing tools as well as the developing new performance analysis software. Using existing *Pablo* tools, we completed a performance study on GEN2 and *Rocflo*. We identified and eliminated performance bottlenecks, resulting in a significant speedup and vastly improved scalability for *Rocflo* and thus GEN2, and enabling practical runs on more than 512 processors. We also studied the new CSAR fluids module, *Rocflo-MP*. We isolated sections of the code causing non-scalable behavior. We also learned new ways to utilize multiprocessor machines to maximize application performance.

Parallel Input/Output (Lee, Ma, Yu, and Winslett)

Following the successful integration of *Rocpanda*'s output facilities with the other GEN2 modules, we designed and implemented a restart function for GEN2 in *Rocpanda*. *Rocpanda*'s restart operation offers GEN2 developers a simple user interface through its collaboration with *Roccom*, and has been shown to provide better runtime performance than the GEN2 native restart operation.

As an outgrowth of our previous work on active buffering for GEN2, we proposed and implemented a novel architecture for the *Panda* parallel I/O library, which enables both active buffering and client-side compressed data migration. We showed that the integration of these two features can reduce both I/O and migration turnaround time of an application, and began the process of transferring that technology to the popular HDF/ROMIO parallel output libraries.

We also completed the first version of a platform-independent performance meta-model that can automatically and efficiently generate a performance model for parallel I/O for a particular installation, based on sampled data from I/O benchmarks on the target platform. Such a meta-model can be used to build a portable, cost-based parallel I/O optimizer.

In the next year, we plan to enhance *Rocpanda* by adding support for compression during long-distance data migration, and by adding load balancing facilities to compensate for skewed distributions of data across compute processors. We will also work on data declustering over heterogeneous disks, which will help *Rocketeer* in querying and visualizing data stored on inherently heterogeneous clusters. For automatic performance optimization, we plan to improve the current approach's prediction accuracy, work on the automatic identification of I/O performance parameters, and create a general performance model framework that combines all of our previous efforts in an extensible, modular, and plug-and-play fashion.

Computational Mathematics and Geometry

Data Transfer and Interface Propagation (Jiao and Heath)

We continued improving our method for data transfer across interface meshes. Our method constructs a common refinement, that is, a finer mesh whose polygons simultaneously subdivide the polygons of the given input meshes, and uses the common refinement as a reference to transfer data accurately, conservatively, and efficiently. To enhance the robustness and reliability of the common refinement, we investigated a new approach called url-thresholding for identifying geometric features in surface meshes. This approach is less sensitive to noise in the input data, and in turn improves the reliability and accuracy of data transfer. We implemented the data transfer algorithms in *Rocface*, which has been integrated into GEN2 and applied to problems with fairly complicated geometries. In collaboration with other Groups, including those of Tortorelli (Structures and Materials Group) and Eric Loth, we also applied our methods to applications beyond CSAR. We are currently optimizing *Rocface* and preparing to release it for more general use.

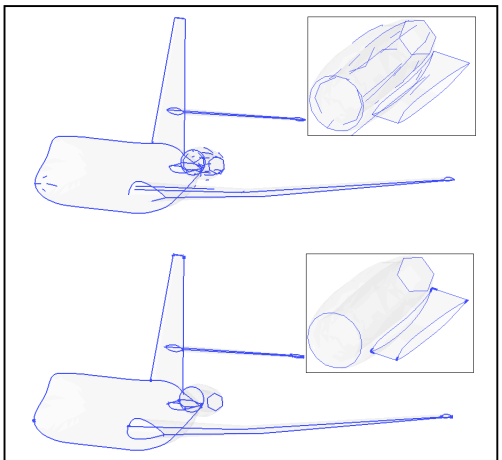

Fig. 3.3.2: Example features detected using traditional methods (top) and our new method (bottom).

We are investigating methods for interface propagation, which is the problem of tracking the moving interface between two geometric domains given the speed at points on the interface. Previously, we developed a general approach based on an entropy-satisfying Huygens' principle. Our approach is adaptive and can resolve singularities and topological changes in the interface. We have implemented and verified the methodology in two dimensions. For three dimensions, we are currently developing a new face-centered scheme with the constraint of fixed connectivity. This constraint is necessary for its integration into *Roc** before the physical components can adapt their volume meshes according to connectivity changes in the surface. This restricted version of our propagation approach also approximates the entropy-satisfying Huygens' principle reasonably well, to the extent allowed by the given connectivity.

Future work on data transfer includes the parallelization and adaptation of the common refinement algorithm. Currently the common refinement is constructed serially during preprocessing, which is reasonable if the mesh connectivity of the interface does not change dynamically. When mesh adaptation is integrated into *Roc*,* the common refinement must be gener-

ated and adapted in parallel. Another future work is to transfer data between volume meshes, for example after volume remeshing. For interface propagation, we will continue improve the algorithms and implement the fully adaptive algorithm without the constraint of fixed connectivity in three dimensions.

Mesh Generation and Adaptation (Guoy, Heath, Lawlor, and Ramos)

We improved our process of initial mesh generation to be more systematic in order to avoid nonmatching geometry between solid and fluid domains. Our coupled simulations allow different discretizations for solid and fluid domains, but their underlying geometries should match well at the interface between them. The meshing process begins with modeling of clean CAD geometry using *Pro/ENGINEER*. The resulting CAD model is exported to a standard IGES (Initial Graphics Exchange Specifications) format. The IGES file is then imported into *Patran/TetMesh* and *Gridgen* for generating the solid and fluid meshes, respectively. Because we emphasize clean CAD geometry, none of the meshing software needs to perform geometric simplification, which had previously been a source of problems with nonmatching fluid-solid interfaces.

We are currently exploring potential strategies for parallel mesh repair for tetrahedral meshes. Basic technology for repairing tetrahedral meshes on serial computers is available commercially, and we have developed several new techniques for sequential mesh repair. We cannot afford to use sequential mesh repair within a parallel analysis code, however, due to the cost of gathering and redistributing data. Our planned strategy is to apply existing sequential technology on each processor separately to repair or remesh individual portions of a partitioned mesh simultaneously. One major challenge is to maintain conformity at partition boundaries between processors. We are in the process of establishing collaborative efforts on these issues with national laboratories, industry, and other universities.

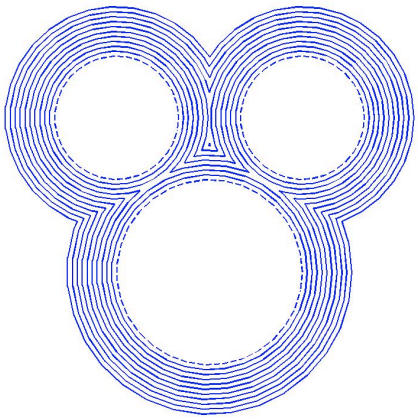

Fig. 3.3.3: Traces of outward propagation of three (dashed) circles with four topological changes over time.

We have developed a new algorithm for parameterizing a tessellated surface that significantly advances the state of the art in functionality and robustness. This algorithm is important for 3-D meshing. In the past year, we have significantly improved the numerical algorithms involved. This project, which has included collaboration with Steve Owen of Sandia National Laboratories, also has applications in graphics.

Makeflo, our tool for automatically and efficiently partitioning structured 3-D meshes, is now in full production use for both fluids simulations and for integrated simulations.

Linear Solvers and Preconditioners (de Sturler and Mackey)

Together with several collaborators, we extended the theory of preconditioners for symmetric and nonsymmetric indefinite linear systems. This resulted in powerful new preconditioners for problems in constrained optimization and systems of partial differential equations with conservation laws. We have also adapted the optimally truncated GMRES method, which we developed a few years ago, for efficiently solving a sequence of changing linear systems arising in

problems such as crack propagation and other time dependent problems. Although further testing and development are required, initial tests with Geubelle's Structures subgroup look very promising.

We will continue our work on preconditioners and solvers for sequences of linear systems arising in the 3-D simulation of the constitutive and failure behavior of solid propellant. This involves the linear solver research mentioned above and work on domain decomposition preconditioners such as FETI and balancing methods. Another interesting class of preconditioners derives from recent promising work by Hendrickson and Boman of Sandia. We will also consider preconditioners based on approximate inverses. Our basic research on solvers and preconditioners for very large and difficult problems will continue and include the indefinite preconditioners mentioned above. We anticipate improving robustness and convergence of Lanczos methods for nonsymmetric linear systems.