3.3 Computer Science

Group Leaders: Michael Heath and Laxmikant Kale

Faculty: Eric de Sturler, Michael Heath, Laxmikant Kale, Edgar Ramos, and Marianne Winslett

Research Scientists/Programmers: Phillip Alexander, Michael Campbell, Robert Fiedler, Damrong Guoy, Xiangmin (Jim) Jiao, Orion Lawlor, and John Norris

Post-Doctoral Associates: Jonghyun Lee

Graduate Research Assistants: Sayantan Chakravorty, William Cochran, Damrong Guoy, Rebecca Hartman-Baker, Vanessa Lopez, Xiaosong Ma, Soumyadeb Mitra, Michael Parks, Manoj Parmer, Kenneth Scheiwe, Joseph Schulz, Christopher Seifert, Evan Vanderzee, Hanna Vanderzee, Terry Wilmarth, Michael Wolf, Shengke Yu, and Gengbin Zheng

Overview

The computer science group continues to provide technologies and tools used by the rest of CSAR. Production runs of the integrated *Rocstar* code use our automatic tool *Makeflo* to partition the structured mesh for *Rocflo*, the tools provided by the Charm++ FEM Framework to partition the unstructured mesh for *Rocflu*, and our adaptive version of MPI, AMPI, to provide parallel communication along with enhanced latency tolerance, automated load balancing, and numerous other benefits. Various aspects of our research in computer science are summarized in the following sections.

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 Environments

The Computer Science Group has provided support for several key technologies used by the rest of the Center. Our unstructured mesh support library, the FEM Framework based on *Charm++*, is in full production use by the Fluids Group and in the integrated code. During the next year, we plan to improve the meshing, mesh partitioning, and remeshing support in the *Charm++* FEM framework.

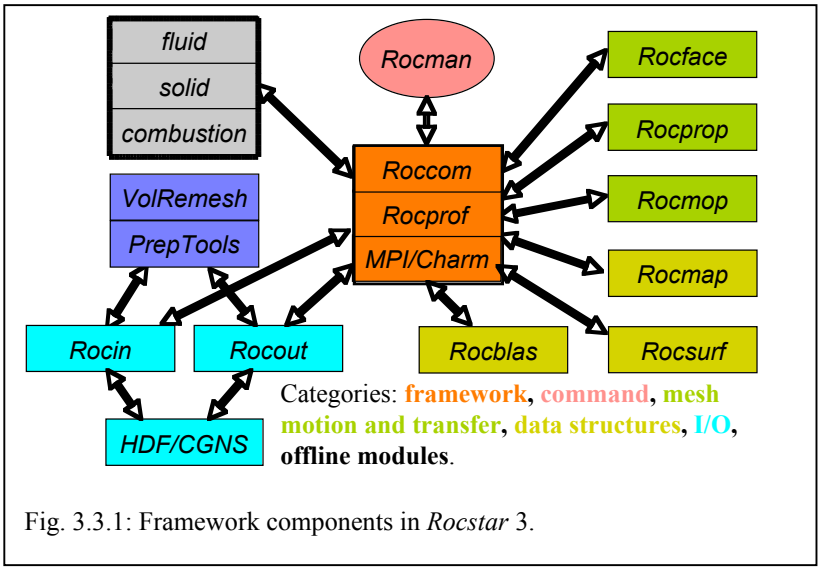

Fig. 3.3.1: Framework components in *Rocstar* 3.

System Integration Framework (Jiao)

Roccom is an innovative object-oriented, data-centric

integration framework developed at CSAR for large-scale numerical scientific simulations. Its window-and-pane data abstraction drastically simplifies inter-module communication, enhances modularity of computer-science service utilities, and eases their integration with application codes. The novel concept of partial inheritance of windows in *Roccom* has been critical in achieving a manageable implementation for *Rocstar*'s complex control module *Rocman*, and has used extensively to facilitate the implementation of service modules.

Roccom's design has always been motivated by the sophisticated needs of the complex software integration of the highly heterogeneous *Rocstar* code suite. In the past year, we have taken further major efforts in enhancing the flexibility and usability of *Roccom* in various respects to help address *Rocstar*'s foremost challenge, the adaptivity that is crucial to cope with the dynamic changes in geometry over the course of a simulation. One notable new feature in *Roccom* is the support of dynamic arrays, which allows attributes to be resized dynamically, in a manner resembling the standard vector containers of C++. The dynamic arrays provide convenience, for example, for mesh repair and for dynamic creation of particles in fluids. Furthermore, *Roccom* now provides a more comprehensive and more unified application-programming interface (API) that allows application codes to register and retrieve data from *Roccom* with substantially more flexibility, and provides more extensive support for multiple connectivity tables in mixed meshes and parallel communications for partitioned meshes. *Roccom* also introduces some advanced features for even more seamless interoperability between C++ and Fortran 90, such as invocation of C++ member functions from Fortran codes and association of memory space allocated in C++ with Fortran 90 pointers.

Gradually, *Roccom* has become a comprehensive integration framework, with unique advantages for integration of parallel simulation codes using C++ and Fortran 90. In the coming year, we will continue the development of new service modules on top of *Roccom* to meet the challenges imposed by adaptivity in *Rocstar*. The applicability of *Roccom* and the demand for such a flexible framework goes far beyond the scope of rocket simulations. The focus of the framework efforts, however, will shift toward making *Roccom* into a more general-purpose, full-featured, industrial-strength system integration framework, so that it can serve a larger scientific community beyond CSAR. We have already begun efforts toward this goal, for example, utilizing *Roccom* in an ongoing interdisciplinary project on solidification, which requires interaction between data structures implemented in C++ and analysis codes implemented in Fortran 90. We will continue to explore such external applications, and also look into the interoperability of *Roccom* with other frameworks, such as the Common Component Architecture.

Abstraction of Input and Output (Norris and Jiao)

The new enhancements in *Roccom* allow physics modules to use windows as high-level abstractions for sending and receiving data when communicating with the outside world, independently of whether the other module may use files of different formats, or require a service utility, or may be located on remote machines. In particular, we have introduced two new I/O service utilities, *Rocin* and *Rocout*, which provide mappings between *Roccom* windows and scientific file formats (such as HDF and CGNS), so that the details of file formats are transparent to physics modules. A subset of the user interface of *Rocin*, in particular, obtaining data from a window, is generic and can be shared by other

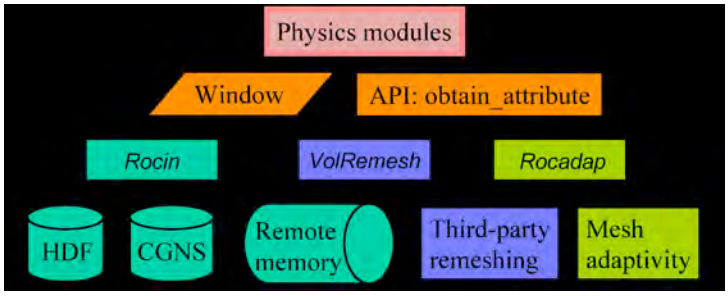

Fig. 3.3.2: Levels of abstraction of data communication.

service modules, such as on-line remeshing tools. This new design allows physics modules to use the same initialization routine for initial startup, restart, restart after off-line remeshing, or future on-line parallel remeshing tools. This abstraction has been implemented in *Rocstar* 3.

In the coming year, *Rocin* and *Rocout* will be improved for better flexibility and higher efficiency. Currently, one data file can reference the mesh data in a different file, reducing data redundancy. We will expand upon this idea so that a file can reference any attribute in another file, not just mesh data. We will also add full support for the CGNS format, which can deliver better performance than HDF4, and is also more widely used by mesh generation and visualization tools. In addition, we will add support for asynchronous operations in *Rocin* and *Rocout* to allow overlapping of I/O with computation.

Interface Data Transfer *(*Jiao*)*

Rocface is the interface code that transfers data between non-matching surface meshes in an accurate and conservative manner. To achieve these goals, *Rocface* constructs a common refinement, that is, a finer mesh whose polygons simultaneously subdivide the polygons of the given input meshes. Because the common refinement can be significantly larger than the input surface meshes (typically 7 to 10 times larger), the performance of *Rocface* can sometimes be an issue. In the past year, we optimized the implementation of *Rocface* and achieved a speedup of about 30% for large meshes. Along another front, in collaboration with Jaiman, Guebell, and Loth, we have performed a more systematic comparison of the common-refinement based method used in *Rocface* with some other methods in the literature for fluid-solid interactions, identified conclusively the source of large errors in those traditional methods, and demonstrated the superiority of *Rocface*'s methods.

Our future efforts on *Rocface* will focus on adaptive algorithms. In particular, we are extending *Rocface* to support partially overlapping meshes where the boundaries of the meshes do not coincide, including those occurring with sliding interfaces. In addition, improving feature-detection and relaxing feature-matching schemes are also becoming increasingly important. Finally, in the coming years, the construction of the common refinement will also be parallelized, in synchronization with parallel remeshing.

Lagrangian Interface Propagation (Jiao, Heath, and Lawlor)

The problem in Lagrangian interface propagation is to track the motion of an interface with Lagrangian interface meshes that evolve with the motion of the interface. Interface propagation is a complex process with a number of facets in numerics, geometry, and topology. In *Rocstar*, numerical complexities arise from the fact that the burning rate is not uniform and may even have discontinuities (for example, different burning rates between propellant and inhibitor); geometric complexities arise from sharp features (such as ridges and corners) of the surface; topological complexities arise from potential topological changes when the propellant nearly burns out. Our focus in the past year has been on developing robust schemes to handle numerical and geometrical discontinuities. The key idea of our method is to propagate the faces of the interface, and then position the nodes to approximate the offsets of the faces in an optimal manner, where the optimality may be application dependent. For simple transformation problems, we have found that an effective approach is to minimize the sum of squared distances (MSSD) to the face offsets, which can be solved by eigen-decomposition of a $d \times d$ matrix, where d is the dimension the space. For burning and other physical processes that satisfy Huygens' principle, where MSSD converges to the wrong solution at expansion fans, we are investigating a predictor-corrector approach that predicts the position using MSSD but rescales the nodal motion based on its local curvature. Along smooth regions, we regularize the objective function of MSSD to maintain good nodal spacing. Our new scheme has the advantages of being efficient, conservative, and easy to impose constraints (such as burning along the rocket case). Its parallel implementation requires the communication of ghost nodes

and cells, which will be provided as a service utility by *Rocmap*, leveraging some low-level implementations in the FEM framework.

Lagrangian interface propagation is relatively easy to implement, but its theoretical analysis, especially stability analysis, is very challenging. To date, we have been relying mostly on empirical evidence in investigating the properties of our algorithm. In the coming year, we will utilize more expertise in numerical analysis to study the properties of our Lagrangian methods more systematically. We also plan to work on surface mesh adaptation for handing further geometrical complexities associated with large deformation and substantial burns (such as burn-out of fins), and for handling topological changes.

Mesh Adaptivity (Alexander, Guoy, Jiao, Lawlor, and Parmar)

An outstanding issue with *Rocstar*'s integrated simulations is the degradation of mesh quality due to changing geometry. This degradation leads to small time steps and even failure when an element becomes poorly shaped or inverted. Simple smoothing algorithms, such as Laplacian smoothing, are built into the physics modules, but they are inadequate when the mesh undergoes moderately large deformations. We have developed a three-level strategy for adapting the mesh as the geometry changes: mesh smoothing, local mesh repair, and global remeshing. Mesh smoothing is suitable for gradual evolution of geometry. Local mesh repair is appropriate when poor elements are localized in specific areas such as burning cracks. Global remeshing is the most drastic operation, replacing the entire mesh when the other two operations prove inadequate. In the past year, we have continued our efforts in collaboration with CAE industry, national laboratories, and other universities in order to accomplish our goals.

Global remeshing and local mesh repair are effective, but too expensive to be performed frequently, and are currently available only for tetrahedral meshes in 3-D. To complement other mesh adaptivity efforts, we recently introduced a new module, *Rocmop*, to provide mesh optimization as a separate service with high robustness and sophistication, through a combination of in-house tools and integration with external packages. To date, *Rocmop* provides two major functionalities: feature-aware surface mesh smoothing using in-house tools, and volume mesh smoothing using MESQUITE (from Sandia National Laboratories), both of which are sequential but are currently being parallelized. To assess the effectiveness of various mesh improvement techniques, as well as to determine when they are needed, we have also implemented in *Rocmop* a set of mesh quality metrics that quantify specific geometric and algebraic elemental properties. These quality metrics have also been integrated

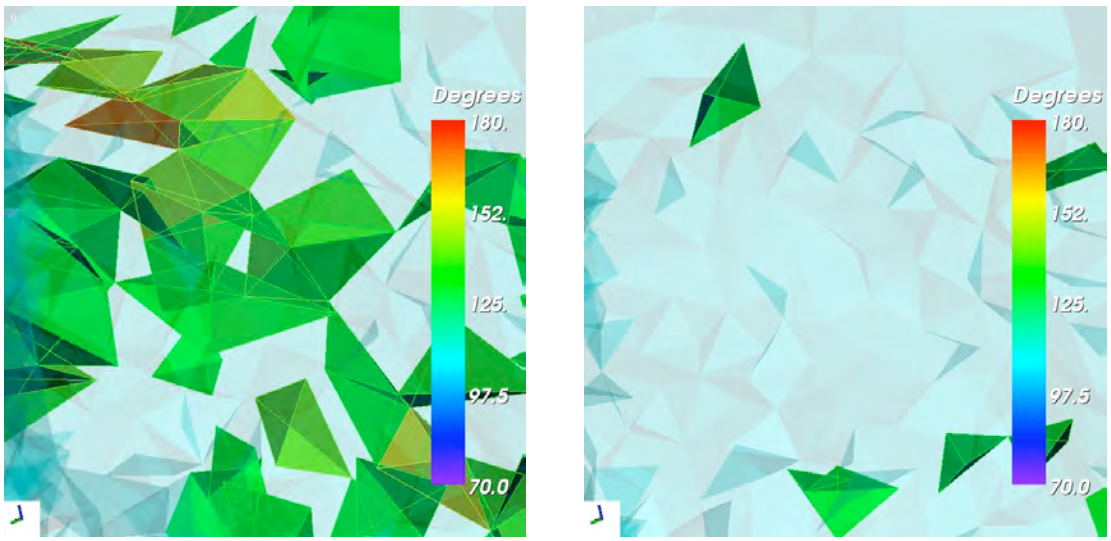

Fig. 3.3.3: Maximal dihedral angles before and after mesh smoothing. Elements with dihedral angles greater than 130° are highlighted.

into *Rocketeer* to enable visual inspection of mesh quality. Figure 3.3.3 highlights a portion of volume mesh, before and after optimization by *Rocmop* using MESQUITE. The colors show the maximum elemental dihedral angle, computed and visualized using *Rocketeer*.

In the near future, we will implement a tiered scheme in *Rocmop* to perform mesh smoothing in parallel using MESQUITE's sequential kernels. During our work on surface smoothing, we have developed schemes to project points onto surfaces, which will be coupled with MESQUITE to invoke its surface smoothing capabilities. We will extend the use of quality metrics to trigger mesh improvement, through mesh optimization, mesh adaptivity, or remeshing. Our in-house mesh optimization efforts will continue to focus on surface meshes, including further investigation of projection schemes, utilizing least squares techniques to filter out high-frequency oscillations in surfaces, and solving point projection and node relocation for surface smoothing simultaneously as constrained optimization problems.

Recently we have established a collaboration with the Scientific Computation Research Center (SCOREC) of Rensselaer Polytechnic Institute (RPI). The mesh modification tools from RPI, or their commercial counterparts from Simmetrix, will be used for local mesh repair. The RPI tools apply local refinement, local coarsening, and local swaps to improve tetrahedral elements in local areas. The ability to modify a mesh only in a specific area is crucial for crack propagation problems. We expect to integrate the RPI tools into *Rocstar* within the next year.

The software from our collaborators was designed to work sequentially. Our near-term strategy is to re-assemble the partitioned mesh and apply the software sequentially. Longer term, we plan to develop the capability to apply the software in parallel. A potential strategy is to use the medial axis for partitioning of the domain geometry. We have studied this strategy in the specific case of an automatic blocking scheme for 2-D structured meshes in multiphase flow problems.

In the past year we demonstrated a prototype of a high-quality remeshing and data transfer capability. The steps involved in remeshing are as follows:

- The old mesh and solution data are read, in parallel, from ordinary output dumps of the integrated code using the new *Rocin* and *Roccom* interface.
- Remeshing is performed by reassembling the serial interface mesh, including boundary conditions, and calling the commercial meshing tools YAMS and TetMesh from Simulog.
- The new mesh and boundary conditions are partitioned using the *Charm*++ FEM Framework.
- Solution data are transferred from the old mesh to the new mesh in parallel, using the *Charm*++ Parallel Collision Detection Library to match up the old and new meshes.
- The integrated code is restarted using the new mesh and solution data using *Roccom*.

Working with the physics developers we have defined a common *Roccom* representation for mesh boundary conditions, which allows face-centered and node-centered boundary conditions to be preserved through the entire remeshing process. The solution transfer process also now fully supports both cell-centered and node-centered solution data. Solution transfer at the mesh boundary, which may be slightly different in the old and new meshes, is accomplished by the novel technique of extruding the surface elements of the old mesh beyond the surface of the new mesh.

In the coming year we plan to extend our remeshing prototype to include propagation of the mesh size field from the initial mesh, accurate and conservative node-centered data transfer, support for additional type of elements, and integration of the prototype remeshing capability into the *Rocstar* code.

Mesh Partitioning and Mesh-Based Solvers (Cochran and Heath)

The goal of this work is to create intelligent meshes that can partition themselves across multiple processors and solve related systems of equations using only the distributed mesh data structure, without having to form matrices explicitly. In the past year, we have generalized our hierarchical data structure for meshes to allow arbitrary inclusion relationships between cells (nodes, edges, or elements) of the mesh. We have also focused on improving the heuristics used for mesh partitioning, and have begun implementation of both direct and iterative solvers based on the new mesh data structure. During the coming year we plan to investigate hybrid geometric/topological partitioning strategies based on the medial surface transform to address some of the limitations of purely geometric partitioning while retaining its positive features.

Linear Solvers and Preconditioners (de Sturler and Siefert)

We have continued and extended our work on solving long sequences of linear systems where both the left and right sides change gradually . Such sequences arise in problems such as crack propagation and in nonlinear solvers and optimization problems. Explicitly taking into account the slow changes in the problem, we recycle the search space of the current linear system solve subsequent systems significantly faster. In addition, we have also started to adapt the recycling of selected subspaces to the underlying physics and the optimization routine chosen. This leads to further improvements in the rate of convergence. We plan to implement some of these methods in the Trilinos package from Sandia National Laboratories.

Saddle point problems and their generalizations occur in constrained optimization and physical problems involving conservation laws. They typically lead to very poor convergence for linear solvers. Good preconditioners yield tightly clustered eigenvalues, which leads to rapid convergence. We have extended our work on preconditioners for saddle point problems to preconditioners that address more general problems (stabilized finite elements and small compressibility) and cheaper preconditioners (using approximations to the Schur complements arising in these preconditioners). We have developed an extensive theoretical framework to analyze problems and preconditioners. Convergence results for model problems are very good. We are currently implementing these preconditioners for realistic practical problems.

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

During the past year, our *Rocpanda* module continued to be used as the main I/O module for both debugging and production runs of the integrated *Rocstar* code. Our GODIVA library for data-format-independent prefetching and caching of scientific data was included in the production releases of both the batch and interactive versions of the *Rocketeeer* visualization tools. The attached figure shows the performance benefits that accrue from the use of GODIVA in a typical batch run of *Rocketeer*. We began work on extending GODIVA to support more complex data structures, such as tree structures, that we anticipate CSAR will employ in the future. We have incorporated the extended version of GODIVA into a demonstration version of the Salsa visualization toolkit for tree-structured cosmology data from N-body simulations, developed at the University of Washington.

As a continuation of our technology transfer effort, we built a remote file access module called RFS for the ROMIO parallel I/O library in collaboration with Argonne National Laboratory. When appropriate, RFS uses active buffering (which we developed at CSAR in previous years) to hide the cost of migrating data across wide area networks. We proposed several optimization approaches for use in this integration, and our performance results showed that without special backbone networks, RFS can reduce the apparent cost of data migration so that it approaches the cost of sending messages locally. In joint work with ANL, we also began the process of integrating active buffering into the new parallel version of netCDF (PnetCDF) developed at ANL. This new prototype is now in the testing phase.

Our future research efforts in parallel I/O will be focused on three areas. First, we will continue to support *Rocpanda* as an I/O module for *Rocstar* and maintain it to be compatible with *Rocstar* as the *Rocstar* code and its choices of underlying I/O libraries and formats evolve.

Second, we will continue our work on extending and improving the GODIVA library for data-format-independent buffering, caching, and prefetching, so that will continue to meet the needs of the *Rocketeer* visualization tool. In particular, we are working on support for data structures that include pointers, and will extend GODIVA to support planned future changes in the *Rocketeer* data formats (Figure 3.3.4). We expect to be able to begin technology transfer work for GODIVA within the next two years, but at this point it is too early for us to have a particular technology transfer target in mind for GODIVA.

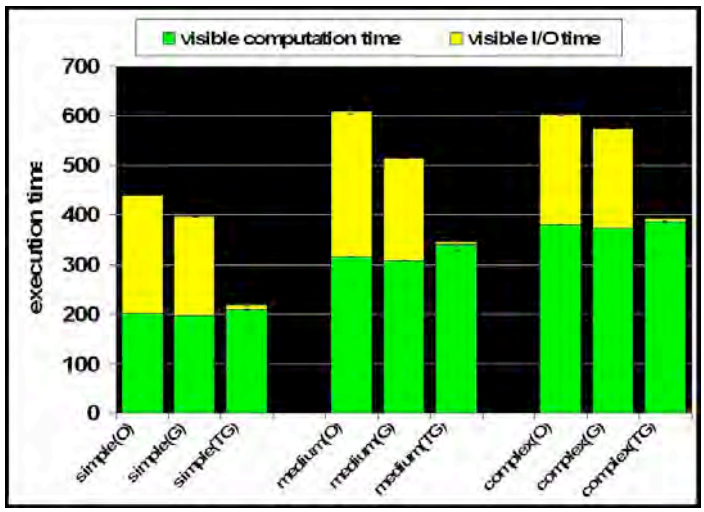

Fig. 3.3.4: Performance benefits that accrue from use of GODIVA in typical batch run of *Rocketeer*

Third, we will continue our technology transfer efforts for data migration and active buffering. The RFS remote I/O module for the ROMIO parallel I/O library will be augmented with efficient prefetching and caching facilities for remote read optimizations, benefiting read-intensive applications in distributed settings such as remote visualization. The use of RFS and active buffering will be evaluated with scientific data formats including HDF5 and Parallel netCDF, using different data transfer mechanisms such as TCP/IP and FOBS. This will be a lasting legacy of CSAR that will benefit the thousands of scientists who use ROMIO and netCDF.

Performance Tools and Tuning (Campbell)

We have continued our embedded performance monitoring strategy with the development of a new *Rocstar* profiling module, *Rocprof*. *Rocprof* replaces *Roccom's* automatic high-level profiling with an automatic, configurable, multi-level profiling utility. With *Rocprof*, we can obtain timing and hardware performance metrics (where available) on any automatically profiled or developer-defined code construct. The physics code developers are not required to add any additional code or instrumentation to take advantage of the automatic high-level profiling provided by *Rocprof*. In addition to automatic high-level profiling, *Rocprof* provides a mechanism for profiling developer defined sections of sub-module code. *Rocprof* provides this service through native, *Roccom*, and MPI_PCONTROL API's so that the physics codes will continue to work in stand-alone mode with or without profiling. Like many other modules, *Rocprof* is also a stand-alone profiling tool. Use of *Rocprof* in stand-alone mode requires developer instrumentation. The MPI_PCONTROL interface is preferred for stand-alone use as it allows profiling to be turned on or off at link time, and requires no instrumentation modifications in order to profile during integrated *Rocstar* runs.

Efforts to improve the performance of current and previous versions of *Rocstar* have been very successful. CSAR has demonstrated very scalable application behavior with nearly every version of *Rocstar* to date. Currently, *Rocstar* is approaching 96 percent efficiency on up to 1024 SP3 processors. Development of robust and detailed performance data collection mechanisms has allowed CSAR to begin focusing on improving the absolute performance of sections of code at the sub-

module level as well as overall scalability. To accomplish this, we use our profiling tools to identify the performance-critical sections of code and, making use of machine-dependent models, we predict the theoretical peak performance of those critical sections. We attempt to approach the theoretical peak of these sections through hand and compiler assisted tuning. Over the course of the past year, *Rocstar* has seen a dramatic performance boost as a result of these efforts. Currently, *Rocstar* runs at about 18 percent of theoretical peak on the SP3, compared with about 10 percent a year ago. With the release of *Rocstar* 3.0 and *Rocprof*, we intend to continue to demonstrate outstanding performance and scalability.

Fig. 3.3.5: Initial 64 nodes of new Apple G5 cluster. Complete cluster is 640 dual-processor nodes (1280 processors).

In the coming year, we plan many enhancements to the CSAR performance tools. Work is currently underway to process and visualize *Rocprof* profiling data with the excellent CHARM++ performance visualization tool, Projections. In order to increase the robustness and platform support of *Rocprof*, a version-independent implementation of PAPI hardware counter access is also in progress. We also intend to spend time improving the overhead associated with using *Rocprof*. Currently, the overhead of each instrumented section is about 6μs, which we hope to reduce by a factor of two with a new interface and function registration mechanism. We will soon roll out a new cluster consisting of 640 dual-processor Apple G5 Xserves (Figure 3.3.5 is the test cluster – 64 dual-processor nodes). We are investigating the use of hardware performance access API's on the Mac OS X platform in order to provide the same profiling services available on other platforms.

Parallel Programming Environment (Zheng, Wilmarth, Lawlor, and Kale)

The Adaptive MPI (AMPI) system was enhanced during the past year in many ways, including enhanced support for checkpointing and preliminary fault tolerance, ports to new machines including BlueGene/L, performance prediction capabilities via BigSim, and extensions to *Projections*, the *Charm++/AMPI* performance analysis tool.

The year before, we had demonstrated basic capability for automatic coordinated checkpointing. This has now been added to the production version of AMPI distributed on the web. We started on a project to support fault tolerance capabilities this year. We extended the checkpointing system with the ability to automatically detect failure of individual processors, terminate existing computation and restart it from the last known checkpoint. Further, we added a new type of checkpointing protocol: in-memory double checkpointing. Here, the checkpoint is stored in the memory of a remote processor; a second checkpoint is stored in the memory of local processor itself. On failure, all processors except the failed one recover the checkpoints from their own memory and resume, while the objects on the failed processor are resurrected on other (possibly existing) processor/s from their remote checkpoints. This works well for applications which have a small memory footprint at checkpoint time. For memory intensive applications, one can replace memory with local disk. In either case, we demonstrated that we get significantly better performance over checkpointing to NFS. A long-term ambitious project that aims at fault recovery without sending all processors back to their checkpoint

has been started (with new funding outside of CSAR) and has demonstrated some preliminary success based on sender-side logging.

Projections, the performance analysis and visualization tool that comes with *Charm++*, was extended to provide support for AMPI with function-based views that MPI programmers expect. With its existing support for detailed timelines, performance counters and communication analysis, projections is now a close to complete tool for performance analysis.

Leveraging support from an NSF grant, we have built an emulator (to ease porting) and simulator (for performance prediction) that allow us to emulate/simulate applications on future machines with large numbers of processors using currently available parallel machines. We were able to port *Charm++* and AMPI to BlueGene/L at IBM with one day's effort, because of the prior emulation. Performance prediction capabilities via parallel discrete event simulation are allowing us identify bottlenecks in scaling to large number of processors before BG/L at LLNL becomes operational, for example.

Mesh Coarsening and Refinement (Wilmarth, Chakravorty, and Kale)

In an attempt to gain an understanding of the mesh-refinement and in general mesh modification issues, we have embarked on a small effort to create parallel implementations of refinement-coarsening algorithms. Leveraging support from other NSF grants, Terry Wilmarth and Sayantan Chakravorty have implemented 2-D and 3-D refinement and coarsening algorithms that run in parallel, and the 2-D version of the refinement has been integrated into the *Charm++* unstructured mesh framework. This code has been tested by Philippe Geubelle and students on a simple 2-D shock-wave propagation problem.