NASA

ESTO

Computational Technologies (CT) Project

Goal: Demonstrate the power of high-end and scalable cost-effective computing environments to further our understanding and ability to predict the dynamic interaction of physical, chemical, and biological processes affecting the Earth, the solar-terrestrial environment, and the universe.

Round 3 Competitively-Selected Awards

Earth System Modeling Framework ($0.8m over 3 years)

- Kileen NCAR - Part I: Core Earth System Modeling Framework Development
- J. Marshall MIT - Part II: Modeling Applications for the Earth System Modeling Framework
- A. da Silva GSFC - Part III: Data Assimilation Applications for the Earth System Modeling Framework

Earth Science ($6m over 3 years)

- A. Donnellan JPL - Rupture of Simulations for Active Tectonic Processes
- P. Houser GSFC - Land Information System
- C.R. Mechoso UCLA - Atmosphere-Ocean Dynamics and Trace Transport
- J. Scheuer GSFC - Solar Protection: HPCC Infrastructure for Public Health and Em. Forecasting

Space Science ($7m over 3 years)

- Gombosi U Mich - A High-Performance, Adaptive Simulation Framework for Space-Weather Modeling (SWMF)
- P. Sawyer U Illinois - Development of an Interoperable, Extend Em. Framework for Adaptive Meshes (BEAM) with Applications to Radiation-Hydrodynamic Models of Gamma Ray Bursts
- T. Prince Caltech - High-Performance Computational Technologies for the National Virtual Observatory
- P. Colella CHE LBNL - A C++ Framework for Block-Structured Adaptive Mesh Refinement Methods

JPL's CCA Task

- ESTO-CT is a joint JPL-Goddard Space Flight Center (GSFC) Project
- Project includes in-house scientists at JPL and GSFC who:
  - Help the teams meet their milestones
    - Advice Consulting Code Optimization
  - Support the project and the teams by developing tools and libraries
- Some of the teams had mentioned CCA in their proposals as a mechanism useful for developing their applications by:
  - Enabling small groups to write parts of a larger application without understanding the full application's code
  - Allowing the scientists to concentrate on science while still working towards modern, reusable applications
  - Taking advantage of previously developed code
  - Supporting language interoperability
- This led the project to start a task to investigate the CCA
- The investigation task recommended a demonstration task
- The demonstration task runs from Feb. to Sept. 2002
- The task is intended to answer two questions:
  - How usable is the CCA software? What work is involved for a scientist to take previously written software and turn it into components?
  - Once the components exist and are linked together, how does performance of the componentized version of the application compare with that of the original application?
- The task has two deliverables:
  - Report on completed sequential component demonstration of Pyramid AMR library and one application - 5/2002
  - Report on completed parallel component demonstration of Pyramid AMR library and one application - 9/2002

Common Component Architecture (CCA)

- A component model specifically designed for high-performance computing
- Supports both parallel and distributed applications
- Designed to be implementable without sacrificing performance
- Minimalist approach makes it easier to componentize existing software
- Components are peers
  - **No particular component assumes it is "in charge" of the others**
  - **Allows the application developer to decide what is important**
- Components interact through well-defined interfaces, or *ports*
  - **In OO languages, a port is a class**
  - **In Fortran, a port is a bunch of subroutines**
- A given component may *provide* a port – implement the class or subroutines
- The Go port is a special provides port - used to start the app's first component
- Another component may *use* that port – call methods or subroutines in the port
- Links denote a caller/callee relationship, ***not dataflow!***
  - **e.g., linSolve port might contain: solve(*in A, out x, in b*)**
- The framework provides the means to "hold" components and compose them into applications
- The framework is the application's "main" or "program"
- Frameworks allow exchange of ports among components without exposing implementation details
- Frameworks may support sequential, distributed, or parallel execution models, or any combination they choose
- Frameworks provide a small set of standard services to components
- Steps to run an application
  - **Launch framework (use a GUI or a script)**
  - **Instantiate components required for app.**
  - **Connect appropriate provided and used ports**
  - **Start first component**
    - (e.g., in a GUI, right-click the component, select "Go" from a menu)
- CCA Forum is an open community working developing the CCA
  - **Currently, mostly DOE and academic**

Credit: Jim Kohl and the rest of the CCA Forum

Basic Component Examples: Hello World!

- Initially, we wrote 2 example applications
  - A single component Hello World! application
    - The hello component has only a Go port
      - The action of the Go port is to print "Hello World!" to standard out, then to exit
  - A two component Hello World! application
    - The hello component has a Go port, and a Uses port
      - The Uses port says that hello component will use a helloServer component
      - The action of the Go port is to instantiate a helloServer component, to call its returnString method, to print the returned string to standard out, then to exit
    - The helloServer component has a Provides port
      - This port provides a returnString method, which returns the string
- Learning the CCA software, then writing and running these examples took about 3 months of part-time effort for two people
  - Most of this effort was learning:
    - What are components?
    - What demonstration code is available?
    - How do we build and run the demos?
    - How do we extract the basics from the complex demos?
  - Very little work in actual writing
    - Create, build, and run our basic examples in C++

Fortran 90 Components?

- We observed that the main items passed across the interface are Fortran 90 pointers
- We had chosen to use the CCAFEINE framework, which requires code to be written in C++
  - We also could have used DCAFE, which allows simple use of BABEL, and thus permits code in C, C++, Fortran 77, Java, and Python
- We decided to write a C++ version of the driver code that could pass Fortran 90 pointers

Componentizing the Parallel Program

- Parallelizing the componentized program was trivial
  - One copy of the framework runs on each parallel processor
  - Each process of each parallel component communicates with the equivalent process of the other other component through the equivalent process of the framework
  - The processes in a parallel component communicate with each other through MPI

Components: Blue, Red

Framework: Gray

PYRAMID: Parallel Unstructured Adaptive Mesh Refinement

Modern... Simple... Efficient... Scalable...

Technology Description

An advanced software library supporting parallel adaptive mesh refinement in large-scale, adaptive scientific & engineering simulations.

State-of-the-Art Design!

- Efficient object-oriented design in Fortran 90 and MPI
- Automatic mesh quality control & dynamic load balancing
- Scalable to hundreds of processors & millions of elements

Application Areas

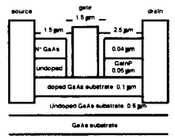
- Computer Modeling & Simulation Applications with complex geometry
- Electromagnetic and semiconductor device modeling
- Structural/Mechanical/Fluid dynamics applications

John Z. Lou, Charles D. Norton, & Thomas A. Cwik
High Performance Computing Systems and Applications Group
http://hpc.jpl.nasa.gov/APPS/AMR

Sample Application: Device Modeling

MICROWAVE ACTIVE DEVICES

- Active devices have very thin layers with extended regions
- Charge resides in thin layers, but is driven by EM fields extending into bulk regions
  - This is a multi-scale problem



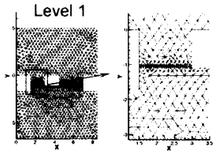
MESFET Model (not to scale)
Lin and Lu, IEEE Elec. Let. Sept 1996

- This problem was examined using Pyramid in: T. Cwik et al., Multi-Scale Meshes for Finite Element and Finite Volume Methods: Device Modeling, AP2000 Millennium Conference on Antennas & Propagation

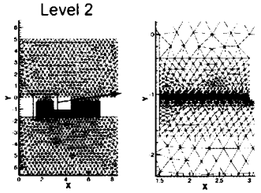
Multi-Scale Mesh: Geometry Driven

- Initial mesh, derived from a commercial mesh generator, contains large elements that just preserve the thin-layered geometry
- Pyramid library performs adaptive refinement of initial mesh in stages
- Problem solved using coupled Hydrodynamic/ Maxwell equations
  - **Irregular FDTD for EM updates**
  - **Box method for transport updates**
- Our sample application is only concerned with building the mesh

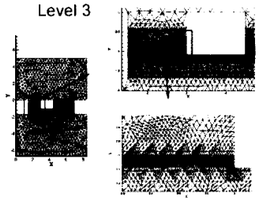
Level 1



Level 2



Level 3



Credit: Tom Cwik

A Sample Pyramid Program

- A sample Pyramid program is Fortran90, and looks object-oriented
  1. Instantiate a mesh object
  2. Work with the mesh object, by calling method functions
- Calls to Pyramid are made with a first argument that is the mesh object to be worked on
  - `call PAMR_METHOD (input_mesh, …)`
- In an object-oriented programming langauge, these calls would look like
  - `input_mesh.method (…)`

Componentizing the Software

- Fairly short effort
  - About 3 weeks of part-time effort for two people
- We basically took what we learned from the basic examples (written in C++) and applied it to Pyramid
- However, Pyramid driver and library are Fortran 90
  - Understanding how to build components out of Fortran 90 code was our biggest challenge
  - Fortran 90 integration issues took a couple of weeks to work out
- First, a test program was written that used a Fortran 90 pointer
  - This was compiled into object code, to understand the routine names that the compiler was generating, so that these routines could later be called from C
  - Additionally, the code was compiled to assembler, which was studied to understand how a Fortran 90 pointer was stored and passed
- Once these two issues were clear, it was a simple matter to write a C main program, and to wrap the Pyramid library with a C wrapper
  - Neither of the main nor the wrapper are portable to other machines, OSes, or compilers, but the non-portable code is limited to two specific files, and can be rewritten for other environments
- Next, a C++ main program was written, and a C++ wrapper was written around the C-wrapped Fortran 90 library
- Once this was working, it was a simple matter to use the knowledge gained in the two-component Hello World example to turn the main and the wrapped library into components, and run them in the CCAFEINE framework

Sequential Timing Results

- Overall result - the overhead of componentization is negligible

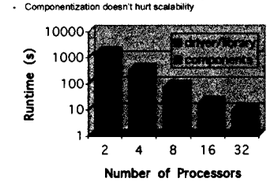
	Original F90 application	Componentized application
User time, as returned by the Unix time call	19.51 s	19.49 s
Wall clock time, as measured from the first Pyramid call to the last Pyramid call	20.37 s	20.43 s
	- 0.87 s system time	- 0.91 s system time
	= 19.50 s	= 19.52 s
One call to Pyramid made one million times	98.83 s	102.24 s

Lessons Learned

- There is currently a fair amount of learning associated with use the CCA Forum's technology, including the CCAFEINE framework
  - It may take 1-3 months for a computational scientist to be able to componentize an initial application
  - A second should be able to be componentized fairly quickly
- The lack of a means to write Fortran 90 components is a serious shortcoming for many science applications
  - It is possible to get around this shortcoming
  - This introduces additional work for the componentizer
  - This adds the chance for additional errors to come into the application
- Once an application is componentized, if the amount of work done in each component call is large when compared with the time needed to make a function call, it is likely that the componentized version of the application will perform well

Parallel Timing Results

- Overall results
  - The overhead of componentization is negligible
  - Componentization doesn't hurt scalability



Conclusions and Future Work

- Knowledge of ongoing work within the CCA Forum (including our own) leads us to believe that the problems with learning the CCA methodology and using Fortran 90 will be resolved in time, most likely in less than a year
- Once this is done, the CCA model will be a promising method for building large single-processor and parallel applications
- Next Steps:
  - Try using CCA technology for one of the ESTO-CT applications
  - Work with the CCA Forum on the learning and Fortran 90 issues
  - We hope to do these during the next fiscal year (Oct. '02 - Sep. '03)