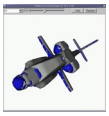
AIAA 2002-2750

Scalable Computational Steering System for Visualization of Large-Scale CFD Simulations

Presented by:
Anirudh Modi
6/24/2002

Co-authored by:
Nilay Sezer-Uzol
Prof. Lyle N. Long
Prof. Paul E. Plassmann



Motivation

- There has been a tremendous growth in speed and memory of parallel computers (+ Advent of cheap Beowulf clusters).
- Computational Fluid Dynamics is getting increasingly important for all kinds of flow computations, from simple to extremely complex.
- Data from large parallel computations will not fit on graphics workstations anymore!
- There is a growing need to be able to steer long running simulations and visualize their results as and when they are being computed.
- Combining remote visualization with complex simulations in real-time gives us a powerful tool to tackle a new world of problems.
- There have been major advances in programming languages and compilers (C++: OOP).

Computational Steering

- Running a complex program on a high-performance computing (HPC) system poses major difficulties in observing output.
- Usually, simulation severely limits interaction with the program during the execution.
- Makes visualization and monitoring very slow and cumbersome (if at all possible).
- It the program is run remotely, additional difficulties arise.
- How do we monitor (observe the output) and steer (change the input) for such a program in real-time?

Computational Steering

- Software tools that support these activities are called computational steering environments.
- They operate in three phases:
  1. Instrumentation: Application code is modified to add monitoring functionality.
  2. Monitoring: Program is run with some initial input data, the output of which is observed by retrieving important data about the program's state change.
  3. Steering: Program's behavior is modified (by modifying the input) based on the knowledge gained during the previous phase by applying steering commands, which are injected on-line.

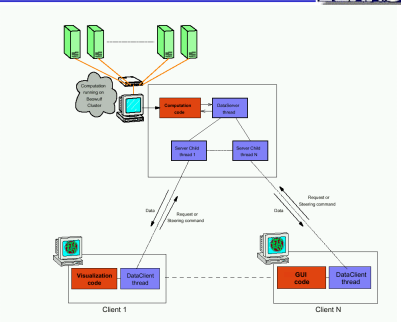
Previous Computational Steering Work

- Powerful, but complex systems with a steep learning curve.
- Bulky. Difficult to setup and use.
- Mainly aimed at computer scientists, not at computational scientists.
- ALICE Memory Snooper from Argonne:
  - Good, lightweight (API similar to MPI).
  - Not object-oriented and therefore not very easy to use.

Computational Steering Library

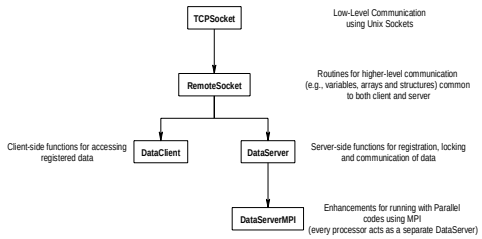
- **POSSE:** *Portable Object-oriented Scientific Steering Environment*
- Written entirely in C++ using advanced features such as classes, templates, polymorphism.
- Uses ***sockets*** for communication and ***pthreads*** for threading
- A simple C++ class interface (DataServer/DataClient).
- **Extremely easy to use!** (Hides most of the complexities involved in the process from the user).
- *Compact* (3500 lines of code), *Fast* (extensive use of templates), *Portable* (has been tested on Linux/HP-UX/SunOS/Windows 2000 as of now), *Multi-threaded* (simultaneous multiple clients supported) and *Lightweight* (very low overhead).

Computational Steering



POSSE

Computational Steering



POSSE design

POSSE: Challenges

- **Challenges:**
  - ***Byte ordering*** (endianness) among various architectures (Intel is little-endian, HP-UX/SunOS are big-endian).
  - ***Byte alignment*** problems. Structs are packed in different ways in different architectures (also by different compilers on the same OS) and therefore may not have the same size. This makes communication of one structure from one platform to the other difficult.
  - Handling user defined structures with ease.

POSSE: Challenges

- ***Data Coherency:***
  - On server-side, use binary semaphores to lock/unlock read-write data (critical portion) when being modified. This has to be done by the user.
  - Modification on server-side is first-received in a buffer and then copied after the data is locked with the semaphore (speed).
  - Client can can either poll or use publish/subscribe methodology (in progress).

Computational Steering Library

- *Server Side:*
  - Functions for registration of data.
  - Initializing DataServer class.
  - Functions for locking data (if necessary)
- *Client Side:*
  - Initializing DataClient class.
  - Calling send/recv functions.
- Has support to pass user-defined structures very easily by defining functions packStruct() and unpackStruct().

Computational Steering Library

```
#include "dataserver.h"
int dummyInt = 0, n1, n2;
double **dyn2D;
REGISTER_DATA_BLOCK() { // Register global data
  REGISTER_VARIABLE("testvar", "rw", dummyInt);
  REGISTER_DYNAMIC_2D_ARRAY("dyn2D", "ro", dyn2D, n1, n2);
}
int main(int argc, char *argv[]) {
  DataServer server(4096);
  n1 = 30; n2 = 40;
  ALLOC2D(&dyn2D, n1, n2);
  for (int iter = 0; iter < MAX_ITER; iter++) {
      server.Wait("dyn2D"); // Lock DataServer access for dyn2D
        Compute(dyn2D); // Update dyn2D with new values
      server.Post("dyn2D"); // Unlock DataServer access for dyn2D
    }
  FREE2D(&dyn2D, n1, n2);
}
```

Example POSSE server code

Computational Steering Library

```
#include "dataclient.h"
int main(int argc, char *argv[])
{
 DataClient client("cocoa.ihpca.psu.edu", 4096);
 double **dyn2D;
 client.SendVariable("testvar", 100); // Send new value for "testvar"
 int n1 = client.getArrayDim("dyn2D", 1);
 int n2 = client.getArrayDim("dyn2D", 2);
 ALLOC2D(&dyn2D, n1, n2);
 client.RecvArray2D("dyn2D", dyn2D);
 Use(dyn2D); // Utilize dyn2D
 FREE2D(&dyn2D, n1, n2);
}
```

Example POSSE client code

Beowulf Clusters

- Multi-computer architecture which can be used for parallel computations.
- Uses commodity personal computers, standard network adaptors and switches.
- Does not contain any custom hardware components and is trivially reproducible. **Extemely cost-effective!**
- Using commodity (and usually public domain) software like the Linux OS, MPI, and other widely available open-source software.
- First Beowulf cluster was built by NASA in 1994 (consisted of 16 486DX4-100 MHz machines).

Beowulf Clusters

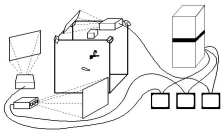
- IHPCA has its own clusters: COCOA (COst effective COmputing Array) and COCOA-2.
- COCOA: 50-proc PII-400 cluster with 12.5 GB RAM, fast ethernet
- COCOA-2: 40-proc PIII-800 cluster with 20 GB RAM, dual fast ethernet

COCOA ($100K in 1998)

COCOA-2 ($48K in 2000)

Visualization Hardware

- **RAVE**: Reconfigurable Automatic Virtual Environment
  - A popular and sophisticated VR installation similar to the CAVE (with just 1 screen).
  - It is projection-based system that surrounds the viewer with 1 or more screens and uses supports stereo display using shutter glasses, and electro-magnetic tracking equipment with a wand for interaction.



CAVE/RAVE schematic

Snapshot of the IHPCA RAVE

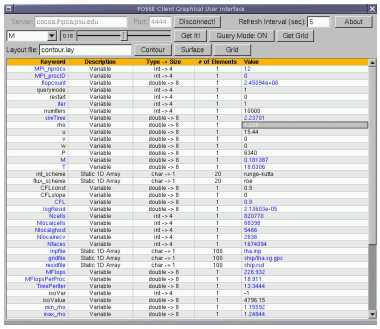
Flow solver: PUMA2

- Parallel Unstructured Maritime Aerodynamics, ***PUMA***, originally written by Dr. Christopher W.S. Bruner.
- Modified extensively and re-organized for efficient performance on Beowulf Clusters [**PUMA2**].
- ***3-D Euler/Navier-Stokes solver.***
- Written entirely in ***ANSI C*** using ***MPI*** library.
- Based on ***Finite Volume method.***
- Supports mixed topology ***unstructured grids*** composed of tetrahedral, wedges, pyramids and hexahedral (bricks)
- ***Preserve time accuracy*** or ***pseudo-unsteady formulation.***
- Uses ***dynamic memory allocation.***
- ***Runge-Kutta***, ***Jacobi*** and various ***Successive Over-relaxation Schemes (SOR)***, as well as both ***Roe*** and ***Van Leer*** numerical flux schemes

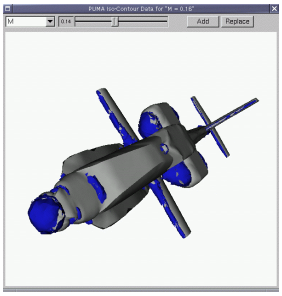
Modifications to PUMA2

- The POSSE server component, *DataServerMPI*, was added to the *main()* function of PUMA2.
- This was done by registering the cell-centered flow vector *[ρ , u, v, w, p]* and various important flow parameters in the code (Grid and flow properties, CFL number, flux scheme, integration scheme, etc).
- Several new global variables were added and registered to receive iso-surface requests and store resulting iso-surfaces.
- An iso-surface extraction routine (based on the marching-tetrahedra algorithm) was added to PUMA2.
- Since this implementation expects the flow data at the nodes of every tetrahedron, a subroutine to interpolate the flow data from cell-centers to the nodes also had to be added to PUMA2.

Graphical User Interface

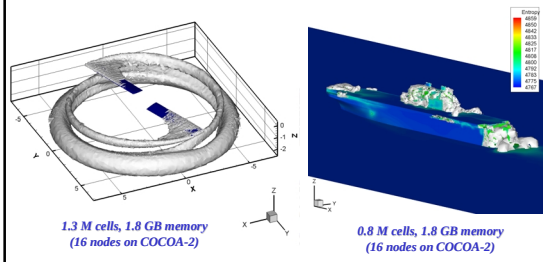


Graphical User Interface



Screenshot of client application showing VTK window with an Apache helicopter with Mach number iso-surfaces

Visualization Snapshots



1.3 M cells, 1.8 GB memory (16 nodes on COCOA-2)

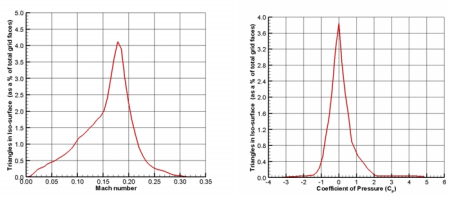
0.8 M cells, 1.8 GB memory (16 nodes on COCOA-2)

Visualization using the Tecplot integration feature of the POSSE GUI

Scalability and Dimensional Reduction

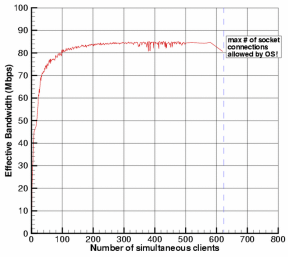
- For an evenly distributed grid, the number of grid points on each processor of a parallel computation is *N/P* where *N* is the total number of grid faces and *P* is the total number of processors.
- ***Scalability:*** Computational time for extraction of an iso-surface is *O(N/P)* as compared to the sequential algorithm which takes *O(N)* for the same procedure.
- ***Dimensional Reduction:*** Data required for the CFD simulation lives in higher dimensional space (3-D) than the data that is required for visualization (which are in 2-D and 1-D space for iso-surfaces and chord plots, respectively) => $O(N^{2/3})$
- ***Combined effect:*** $O(N^{2/3}/P)$
  - Computation is perfectly scalable *O(N/P)*. Less network bandwidth is required due to only $O(N^{2/3})$ data traveling between the client and the server.

Dimensional Reduction



Percentage of Mach and Cp iso-surface triangles for the Apache Helicopter case

POSSE Performance



Multiple client performance

Conclusions

- POSSE has proven to be a very powerful, yet easy to use software with a high rate of acceptance and approval in our research Group. If scientists are given an easy to use software system with a mild learning curve,they will use it.
- It enables us to carry out and debug complex simulations in real-time with considerable ease using a client-server architecture.
- The coupling of computational steering to our parallel simulation makes the real-time visualization of the CFD simulations possible. Scalability and dimensional reduction arising from this approach make the implementation efficient.
- At a more basic level, this ability to interact and visualize a complex solution as it unfolds and the real-time nature of the computational steering system opens a whole new dimension to the scientists for interacting with their simulations.

Webpage:

http://www.personal.psu.edu/lnl/
http://www.anirudh.net/phd/

Questions?