CUMULVS Tutorial

ACTS Collection Workshop

Dr. James Arthur Kohl
Computer Science and Mathematics Division
Oak Ridge National Laboratory

August 7, 2003

Research sponsored by the Applied Mathematical Sciences Research Program, Office of Mathematical, Information, and Computer Sciences, U.S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
Scientific Simulation Issues…

- Fundamental Parallel Programming
  → Synchronization, Coordination & Control
- Distributed Data Organization
  → Locality, Latency Hiding, Data Movement
- Long-Running Simulation Experiments
  → Monitoring, Fault Recovery
- Massive Amounts of Data / Information
  → Archival Storage, Visualization
- Too Much Computer, Not Enough Science!
  → Need Some Help…
Potential Benefits from Computer Science Infrastructure:

- **On-The-Fly Visualization**
  - Interactive Access to Intermediate Results
  - Attached as Needed, Minimize Overhead

- **Computational Steering**
  - Apply Visual Feedback to Alter Course / Restart
  - “Close Loop” on Experimentation Cycle

- **Fault Tolerance**
  - Automatic Fault Recovery / Load Balancing
  - Keep Long-Running Simulations Running Long
Collaborative Infrastructure for Interacting with Scientific Simulations:

⇒ Run-Time Visualization by Multiple Viewers
    → Dynamic Attachment, Independent Views
⇒ Coordinated Computational Steering
    → Model & Algorithm Parameters
⇒ Heterogeneous Checkpointing / Fault Tolerance
    → Automatic Fault Recovery and Task Migration
⇒ Coupled Models…
Collaborative Combustion Simulation

Collaborative Viewing and Steering Enables "What if?" Computational Science
CUMULVS Visualization Features

⇒ Interactive Visualization
  * Simple API for Scientific Visualization
  * Use Your Favorite Visualization Tool

⇒ Minimize Overhead When No Viewers
  * One Message Probe, No Application Penalty

⇒ Send Only Viewed Data
  * Partial Array / Lower Resolution

⇒ Rect Mesh & Particle Data

⇒ Common (HPF) Data Distributions
  * BLOCK, CYCLIC, EXPLICIT, COLLAPSE

⇒ Soon Unstructured, Sparse & Adaptive Meshes…
Multiple Simultaneous Views

Density

Temperature
Multiple Distinct Views

```
C = spmd.f
  call stvfinite()
  call stvdecompdefine()
  call stvtfiledefine()
  do
    call localwork()
    call exchangeinfo()
  end do
  call stvsendtofe()
  whilst not done!
```
CUMULVS Architecture
coordinate the consistent collection and dissemination of information to / from parallel tasks to multiple viewers

- local person using custom GUI
- remote person using virtual reality interface
- remote person using AVS

CUMULVS exists in 3 pieces: application library, viewer library, and separate fault recovery daemon

interact with distributed / parallel application or simulation supports most target platforms (PVM / MPI, Unix / NT, etc.)
Instrumenting Programs for CUMULVS

• CUMULVS Initialization ~ One Call (Each Task)
  ⇒ Logical Application Name, # of Tasks
• Data Fields (Visualization & Checkpointing)
  ⇒ Local Allocation: Name, Type, Size, Offsets
  ⇒ Data Distribution: Dim, Decomp, PE Topology
• Steering Parameters
  ⇒ Logical Name, Data Type, Data Pointer
• Periodic CUMULVS Handler
  ⇒ Pass Control for Transparent Access / Processing
• Typically 10s of Lines of Code…
Local Allocation Organization

```
dataValues

dataDim=2

dataOffset[0]

Valid Data

dataSize[0]

dataSize[1]

dataAllocSize[1]

dataAllocSize[0]
```
CUMULVS Particle Handling

• Particle Data Fundamentally Different
  ⇒ Data Fields Encapsulated in a Particle Container
  ⇒ Explicit Coordinates Per Particle

• Particle-Based Decomposition API
  ⇒ User-Defined, Vectored Accessor Routines

• Viewing Particle Data
  ⇒ AVS Module Extensions
  ⇒ Tcl/Tk Slicer Particle Mode
CUMULVS
Steering Features

• Computational Steering
  ⇒ API for Interactive Application Control

• Modify Parameters While (Long) Running
  * Eliminate Wasteful Cycles of Ill-Posed Simulation
  * Drive Simulation to More Interesting Solutions
  * Enhance Convergence of Numerical Algorithms

• Allows “What If” Explorations
  * Closes Loop of Standard Simulation Cycle
  * Explore Non-Physical Effects…

ORNL Kohl/2003-13
Coordinated Steering

• Multiple, Remote Collaborators

• Simultaneously Steer Different Parameters
  ⇒ Physical Parameters of Simulation
  ⇒ Algorithmic Parameters ~ e.g. Convergence Rate

• Cooperate with Collaborators
  ⇒ Parameter Locking Prevent Conflicts
  ⇒ Vectored Parameters…

• Parallel / Distributed Simulations
  ⇒ Synchronize with Parallel Tasks
  ⇒ All Tasks Update Parameter in Unison
Parallel Model Coupling in CUMULVS

• Natural Extension to CUMULVS Viewer Scenario
  ⇒ Promote “Many-to-1” → “Many-to-Many”

  ![Graph Diagram](image)

• Translate Disparate Data Decompositions
  ⇒ Parallel Data Redistribution Among Shared Data Fields
    → stv_couple_fields( fieldID, appname, fieldname, ... );
    ⇒ Fundamental Model Coupling Capability
      → Next Step ~ Interpolation in Space & Time, Units Conversion…

E.g. Regional Climate Assessment

![Graph Diagram](image)
CUMULVS Fault Tolerance Features

- **Application Fault Tolerance**
  - ⇒ Automatic Detection and Recovery from Failures

- **User Directed Checkpointing**
  - ⇒ User Decides What / Where to Checkpoint
  - ⇒ Minimizes Amount of Stored Data

- **Heterogeneous Task Migration**
  - ⇒ Restart Tasks on Heterogeneous Hosts
  - ⇒ Restart is Automatically *Repartitioned* if Host Pool is of Different Size or Topology (Yikes!)

- **Avoids Synchronizing Distributed Tasks**
  - ⇒ Asynchronous Checkpoint Collection and Fault Detection
  - ⇒ Minimize Intrusion of Checkpoint / Restart
Run-Time Fault Monitor

- One Checkpointing Daemon (CPD) Per Host
  ⇒ Ckpt Collector / Provider
  ⇒ Run-Time Monitor
  ⇒ Console for Restart / Migrate
- CPDs Comprise Fault-Tolerant Application…
  ⇒ Handle Failure of Host / CPD
  ⇒ Coordinate Redundancy
  ⇒ Ring Topology
Rollback Versus Restart…

• Rollback Recovery:
  ⇒ Only Replace Failed Tasks, “Roll Back” the Rest
  ⇒ Elegant & Cool, But You Must…
    → Monitor ALL Communication for Restart Notification
    → Unroll Program Stack, Reset Comm & File Pointers…
  ⇒ Necessary for High Overhead Restart Cases

• Restart Recovery:
  ⇒ “Genocide” ~ Kill Everything & Restart All Tasks
  ⇒ Simple Approach, No Additional Instrumentation
  ⇒ Not as Efficient a Recovery in All Cases…
Checkpoint Data Collection

- Data from Each Local Task Collected/Committed
  \[ \text{\texttt{stv_checkpoint();}} \]
- Invoke When Parallel Data / State “Consistent”…
  \[ \Rightarrow \text{Highly Non-Trivial in General! (Chandy/Lamport)} \]
  \[ \Rightarrow \text{Straightforward for Most Iterative Applications} \]
  \[ \Rightarrow \text{Save Checkpoint at Beginning or End of Main Loop} \]
- No Automatic Capturing of Other Internal State:
  \[ \Rightarrow \text{Open Files, I/O, Messages-in-Transit…} \]
  \[ \Rightarrow \text{CUMULVS Assumes User Handles This Recovery} \]
  \[ \Rightarrow \text{Can Be Done Manually Using Saved Checkpoint State} \]
  \[ \Rightarrow \text{Future Extensions to Assist…} \]
Manual Software Instrumentation

- **SPDT 98 Case Study ~ SW Instrumentation Cost**

<table>
<thead>
<tr>
<th>Instrumentation:</th>
<th>Seismic:</th>
<th>Wing Flow:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Lines of Code</td>
<td>20,632</td>
<td>2,250</td>
</tr>
<tr>
<td>Vis / Steer System Init</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Vis / Steer Variable_DECLS</td>
<td>48</td>
<td>73</td>
</tr>
<tr>
<td>CP Restart Initialization</td>
<td>21</td>
<td>12</td>
</tr>
<tr>
<td>CP Rollback Handling</td>
<td>41</td>
<td>34</td>
</tr>
<tr>
<td><strong>Total Instrumentation</strong></td>
<td><strong>204 ~ 1.0 %</strong></td>
<td><strong>188 ~ 7.7 %</strong></td>
</tr>
</tbody>
</table>
# Checkpointing Efficiency

- **SPDT 98 Case Study ~ Execution Overhead**

<table>
<thead>
<tr>
<th>Experiment</th>
<th>SGI:</th>
<th>Cluster:</th>
<th>Hetero:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seismic - No Checkpointing</td>
<td>2.83</td>
<td>6.23</td>
<td>9.46</td>
</tr>
<tr>
<td>Seismic - Checkpoint for Restart</td>
<td>2.99</td>
<td>6.50</td>
<td>10.76</td>
</tr>
<tr>
<td>Seismic - Checkpoint for Rollback</td>
<td>3.03</td>
<td>6.66</td>
<td>10.90</td>
</tr>
<tr>
<td>Wing - No Checkpointing</td>
<td>0.69</td>
<td>1.58</td>
<td>6.14</td>
</tr>
<tr>
<td>Wing - Checkpoint for Restart</td>
<td>0.77</td>
<td>1.71</td>
<td>7.10</td>
</tr>
<tr>
<td>Wing - Checkpoint for Rollback</td>
<td>0.79</td>
<td>1.71</td>
<td>7.30</td>
</tr>
</tbody>
</table>

(Checkpointing Every 20 Iterations ~ every 15 sec to 4 mins…!)  

**Seismic Overhead:** 4-14% Restart, +1-3% Rollback.  
**Wing Overhead:** 8-15% Restart, +0-2.5% Rollback.
An Aside: PVM vs. MPI

- Comparison of Features and Philosophy
- Which One to Choose?
  ⇒ Both Useful for Given Application Needs…
- CUMULVS Issues
  ⇒ Internals and General Usage
PVM (Parallel Virtual Machine)

- Use Arbitrary Collection of Networked Computers as a Single, Large Parallel Computer
  ⇒ Workstations, PCs (Unix or NT) ~ Clusters
  ⇒ SMPs, MPPs

- Programming Model & Runtime System
  ⇒ Message-Passing ~ Point-to-Point, Context, Some Collective Operations, Message Handlers
  ⇒ Resource & Process Control, Message Mailbox, Dynamic Groups, Application Discovery
  ⇒ Fault Notification
**Message Passing Interface Standard**

- **Library Specification for Message-Passing**
  - Designed By Broad Committee of Vendors, Implementors and Users
  - High Performance on Massively Parallel Machines and Workstation Clusters

- **Comprehensive Message-Passing System**
  - Point-to-Point, Collective, One-Sided
  - Groups/Communicators, Topology, Profiling, I/O
  - Some Process Control (MPI-2 ~ MPI_SPAWN)
PVM vs. MPI: Different Goals

- MPI
  ⇒ Stable Standard, Portable Code
  ⇒ High Performance on Homogeneous Systems

- PVM
  ⇒ Research Tool, Robust, Interoperable
  ⇒ Good Performance on Heterogeneous Systems
PVM vs. MPI: Different Philosophies

• MPI
   ⇒ Static Model (~ MPI_SPAWN in MPI-2…)
   ⇒ “Rich” API (MPI-1 / 128, MPI-2 / 288)
   ⇒ Performance, Performance, Performance,…

• PVM
   ⇒ Dynamic Model
   ⇒ “Simple” API (PVM 3.4 / 75)
   ⇒ Flexibility (& Performance)
Portability vs. Interoperability

• Portable:
  ⇒ Re-compile Source Without Modification on a Different System, with C, C++, Fortran Support
  ⇒ True of Both MPI and PVM.

• Interoperable:
  ⇒ Executables on Different Systems Communicate
  ⇒ PVM ~ Yes, MPI ~ Sometimes (Not Required)
  ⇒ Different MPI Implementations? IMPI ~ Soon…
  ⇒ Language Interoperability?
    → PVM ~ Yes, IMPI ~ Soon…
Performance vs. Flexibility

• To Be Flexible, You Must Pay the Price…

• Heterogeneity Overheads:
  ⇒ Data Conversion, Network Protocol Selection,
     Extra Message Headers (on top of Native Comm)…

• Choose the Lowest Common Denominator?
  ⇒ Not the Best on Any System.

• Performance Dictates Locally Optimal Solution.
  ⇒ Lose Interoperability…
Interesting Result

• You can build an MPI implementation that supports interoperability and system dynamics across different systems / languages (some already do ~ Mpich, LAM, IMPI…).

• But, given all these conditions:
  ⇒ It Would Perform About the Same as PVM!!
Supporting MPI Applications in CUMULVS

- CUMULVS Works with MPI Applications! 😊
- But MPI Doesn’t Have Everything We Need (Internally)
  ⇒ Static Model, Minimal Operating Environment
  ⇒ No Name Service / Database, Fault Recovery / Notification?
  ⇒ MPI_SPAWN( )…? Proxy Server for Viewer Attachment?
- Existing CUMULVS Solution:
  ⇒ Applications Communicate Using MPI or PVM or ???
  ⇒ CUMULVS Viewers / CPDs Still Attach Using PVM
- Possible “Reduced-Functionality” MPI Version…?
  ⇒ Currently Under Development…
CCA “MxN” Parallel Data Redistribution

⇒ Builds on CUMULVS Viz & Coupling Protocols
⇒ CUMULVS & PAWS (LANL) Being Integrated
→ “MxN” Generalizes Capabilities of Both Systems
  * Point-to-Point versus Persistent Connections (a la Viz)
→ CUMULVS Complements PAWS Coupling Work
Future CUMULVS Plans (2 of 3)

CUMULVS as a Foundry to Forge New Technology

High-Performance Visualization

- Full Parallel Integration of Pipeline
- CCA “MxN” → “M x N x P x Q x R”!

Proposed “Fully Connected” User-Centric Simulation Cycle

ORNL

Scalable Visualization Cache Architecture

Terabyte+ → 100s GBs → 10s GBs → GBs → 100s MBs

Data Reorg For Viz

Reduction & Filtering

Parallel Render

Kohl/2003-32
Future CUMULVS Plans (3 of 3)

Feature Extensions...

• Application Interface:
  ⇒ Assist Manual Instrumentation of Applications
    → GUI, Pre-Compiler...

• Checkpointing Efficiency:
  ⇒ Tasks Write Data in Parallel / Parallel File System?
  ⇒ Redundancy Levels, Improve Scalability…

• Portability:
  ⇒ Other Messaging Substrates
    → Reduced Functionality / Direct Connect for CCA & MPI
CUMULVS Summary

• Interact with Scientific Simulations
  ⇒ Dynamically Attach Multiple Visualization Front-Ends
  ⇒ Steer Model & Algorithm Parameters On-The-Fly
  ⇒ Automatic Heterogeneous Fault Recovery & Migration

• Future Opportunities
  ⇒ Couple Disparate Simulation Models
  ⇒ Integrate as “MxN” Component in CCA
  ⇒ Application Instrumentation GUI / Pre-Compiler

http://www.csm.ornl.gov/cs/cumulvs.html
Seismic Example ~ 2D (Tcl/Tk)

Seismic Example ~ 3D (AVS)

Air Flow Over Wing Example ~ 3D (AVS)