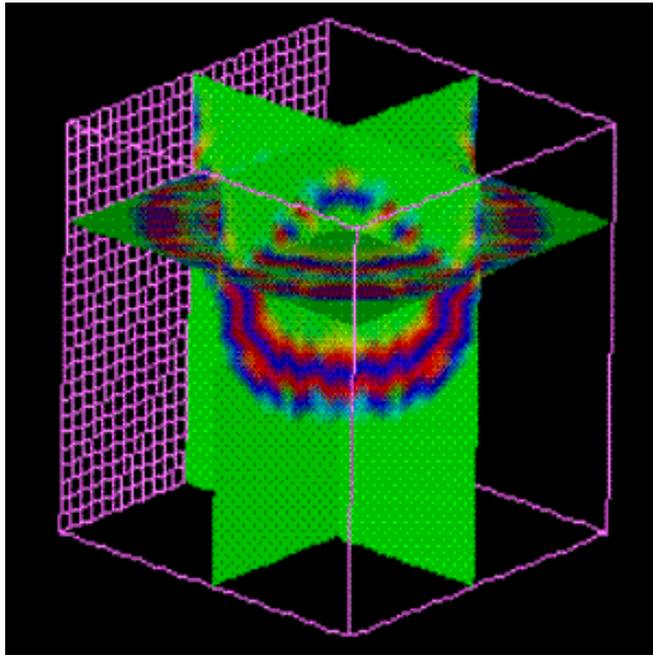
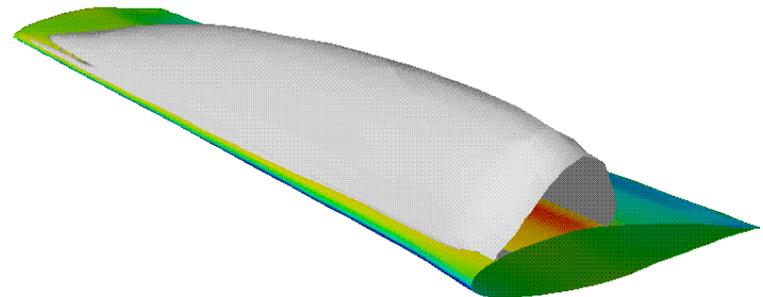


# Interacting with High-Performance Scientific Simulations using CUMULVS: Visualization, Computational Steering, and Fault Tolerance



Dr. James Arthur Kohl  
Oak Ridge National Laboratory  
Oak Ridge, Tennessee



October 7, 1999

# Scientific Simulation

⇒ Explore or Validate Theoretical Models

- \* Very Small / Very Large Physical Scale

- \* Actual Experiments Too Dangerous / Not Isolated

⇒ Alternative to Expensive Physical Prototypes

- \* Explore Variety of Input Parameters & Datasets

- \* Large Startup Overhead ⇔ Fast Evaluation Cycle

⇒ Platform for Remote Collaboration

- \* Not Possible with Traditional Experiments

  - No Ties to Geographical Site or Facility

  - Improved Network Connectivity & Bandwidth...

- \* Scientists Interact to View & Control

# High-Performance Computing

⇒ Solve More Complex Problems

- \* Larger Simulation Domains
- \* Finer Grain / Higher Precision

⇒ Fundamentally Requires Concurrency

- \* Many Computers Cooperate to Solve One Problem
  - Break Problem Into Smaller Sub-Problems
  - Divide Simulation Domain Into Subregions

⇒ Special Hardware

- \* Parallel Computers: SMPs, MPPs
- \* Clusters of Workstations: Beowulf, PCs (Linux/NT)
- \* High-Speed Networking: Fast Ethernet, ATM, Gbit

# Programming Models

⇒ Implicit Parallelism:

- \* Software “Hides” Accesses to Remote Memory
- \* Tuple Spaces, HPF (High Performance Fortran)

⇒ Explicit Parallelism:

- \* Programmer Moves Data “On Purpose”
- \* Message-Passing (PVM, MPI)

⇒ “Locality” ~ Unifying Concept

- \* Reduce Communication Delays, Improve Perf
- \* Direct vs. Automated Approach...



# Parallel Programming Woes...

⇒ Multiple Computational Threads

→ Synchronization, Coordination and *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...*

# Need Software Infrastructure for:

## ⇒ On-The-Fly Visualization

- \* Interactive Access to Intermediate Results
- \* Attach 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, User Migration, User Library for Visualization and Steering)

## ⇒ Collaborative Infrastructure for Interacting with Scientific Simulations:

- \* Run-Time Visualization by Multiple Viewers
  - Dynamic Attachment, Independent Views
- \* Coordinated Computational Steering
  - Model & Algorithm
- \* Heterogeneous Checkpointing / Fault Tolerance
  - Automatic Fault Recovery and Task Migration
- \* Coupled Models...

# CUMULVS Visualization Features

## ⇒ **Interactive Visualization**

- \* Simple API for Scientific Visualization
- \* Use Your Favorite Visualization Tool

## ⇒ **Minimize Overhead when No Viewers**

- \* Application Not Penalized

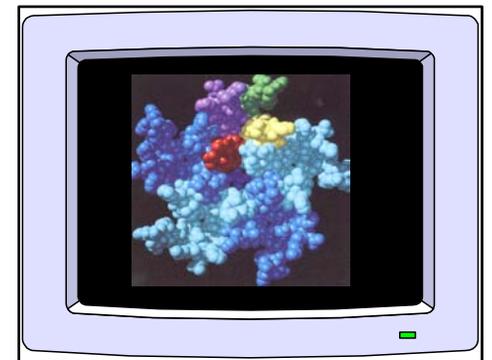
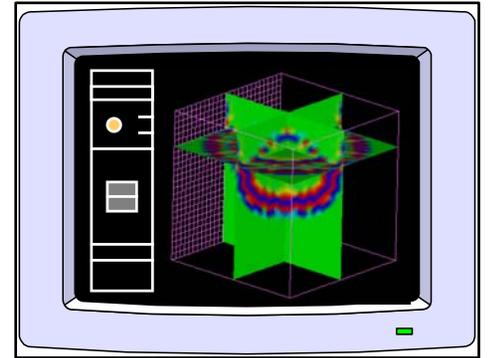
## ⇒ **Send only Viewed Data**

- \* Partial Array / Lower Resolution

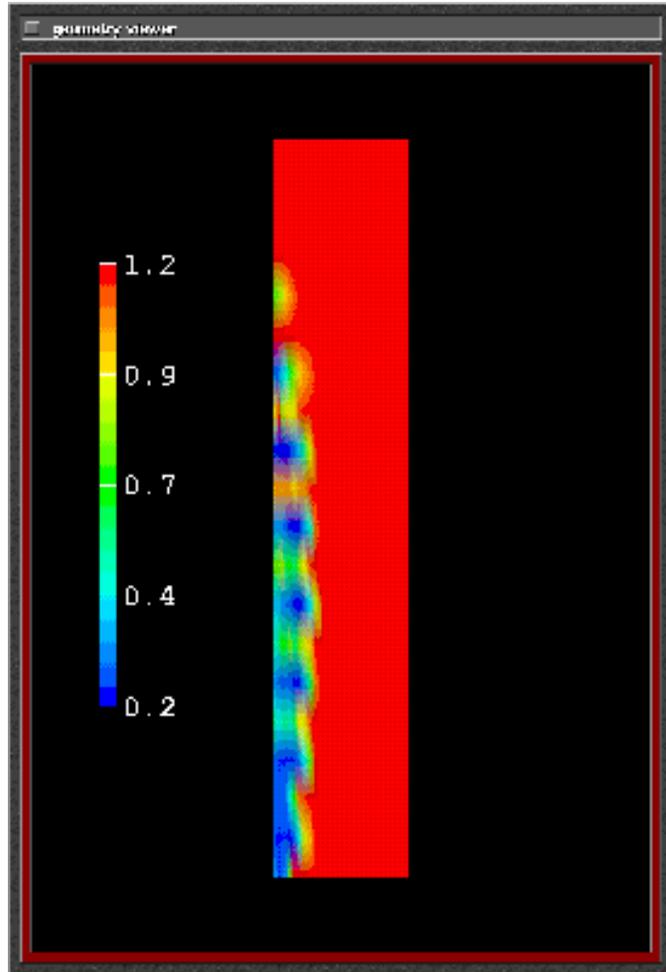
## ⇒ **Both Field and Particle Data**

## ⇒ **HPF Data Distributions**

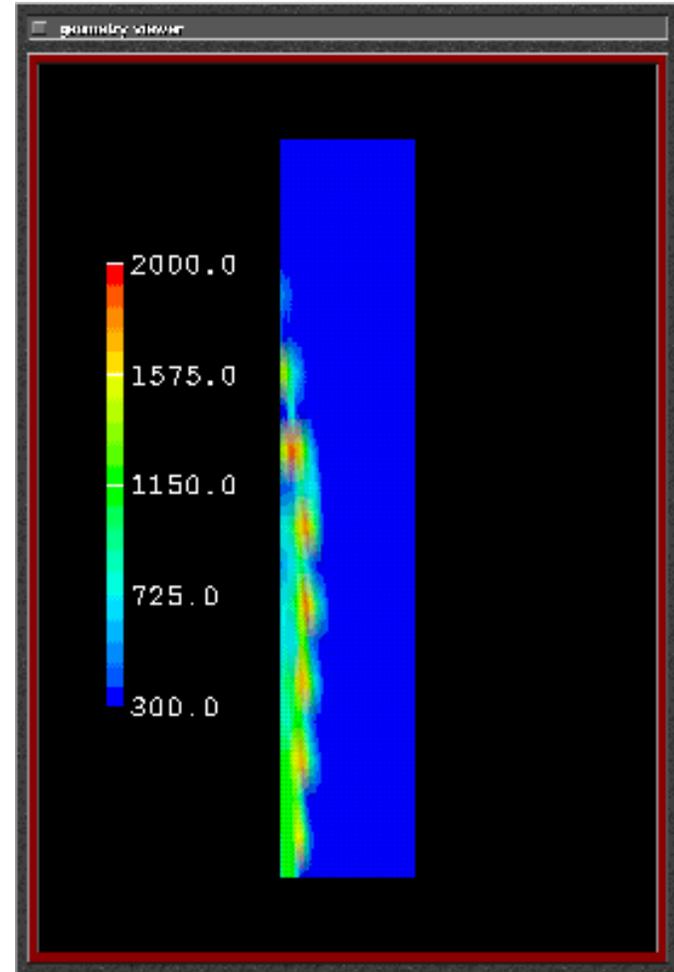
- \* BLOCK, CYCLIC, EXPLICIT, COLLAPSE



# Multiple Simultaneous Views

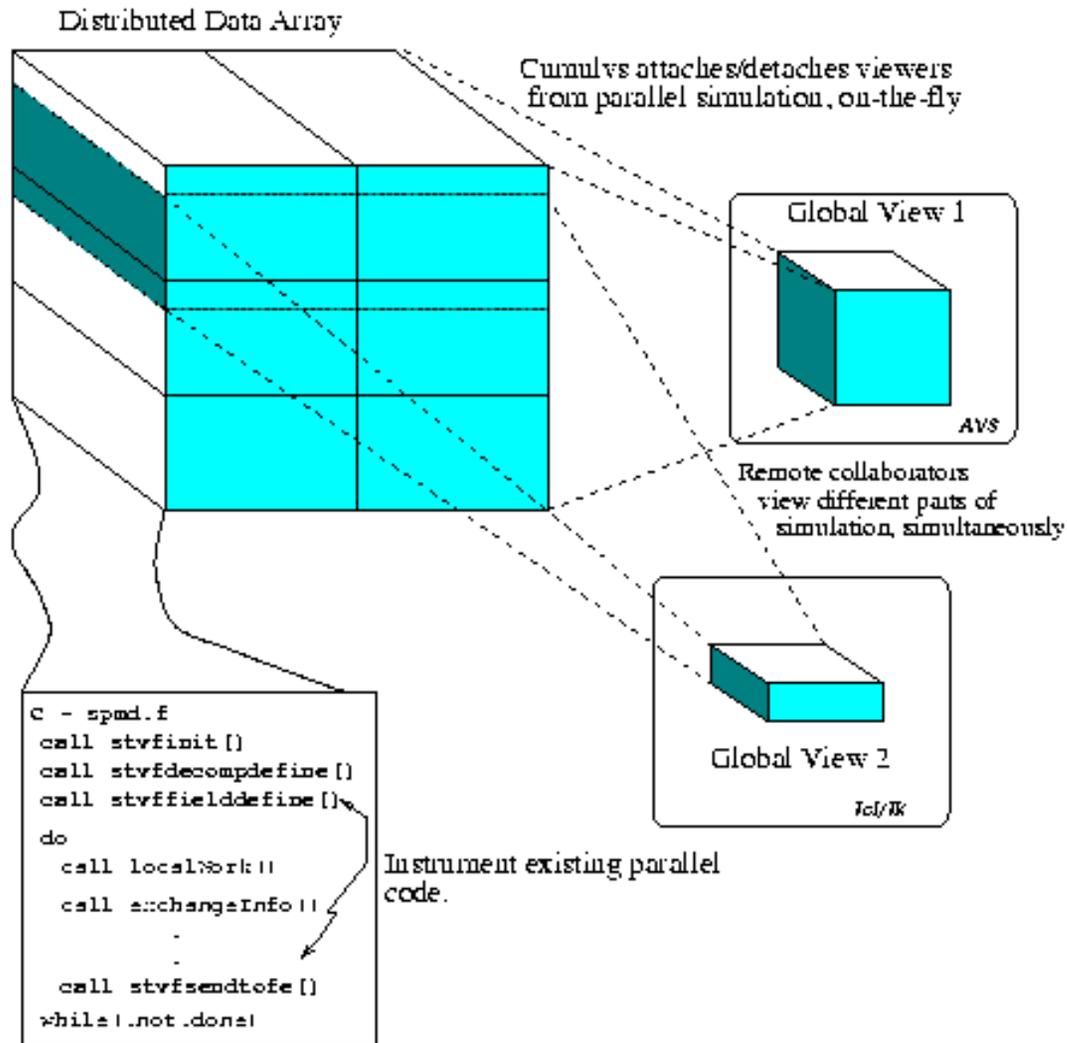


Density



Temperature

# Multiple Distinct Views



# Instrumenting Simulations

⇒ Initialization ~ `stv_init( ) / stvfinal( )`

\* Each Task: Logical Name, Number of Tasks

⇒ Data Fields (Visualization & Checkpointing)

\* Data Distribution: Dim, Decomp, PE Topology

\* Local Allocation: Name, Type, Offsets

⇒ Steering Parameters

\* Name, Type, Reference

⇒ Typically 10s of Lines of Code...

# CUMULVS Particle Handling

⇒ Particle Data Fundamentally Different

- \* Nested Data Fields, Explicit Coordinates

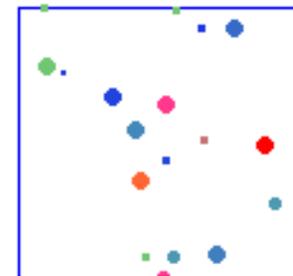
⇒ Particle-Based Decomposition API

- \* User-Defined, Vectored Accessor Routines

⇒ Viewing Particle Data

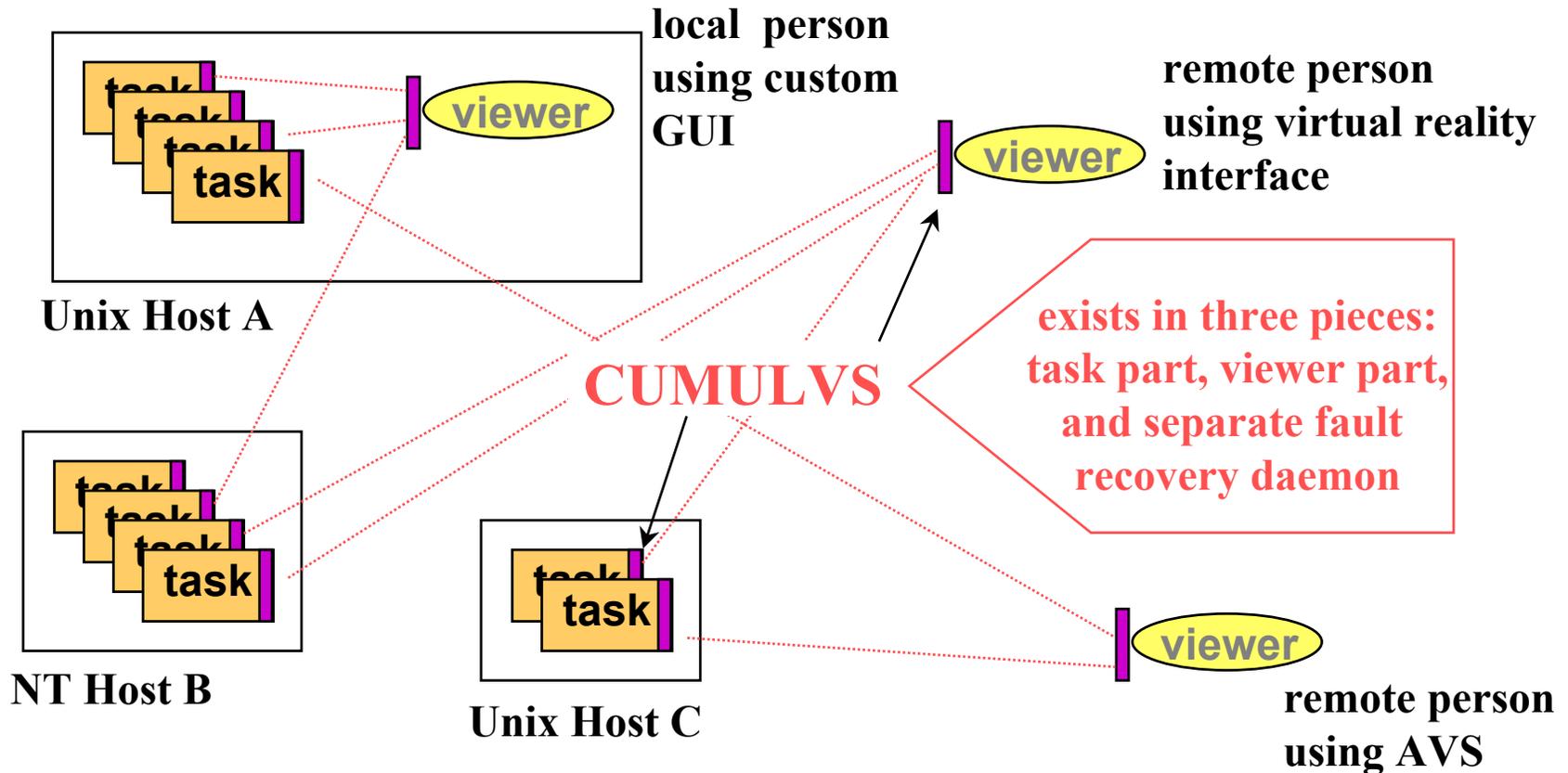
- \* AVS Module Extensions

- \* Tcl/Tk Slicer Particle Mode



# CUMULVS

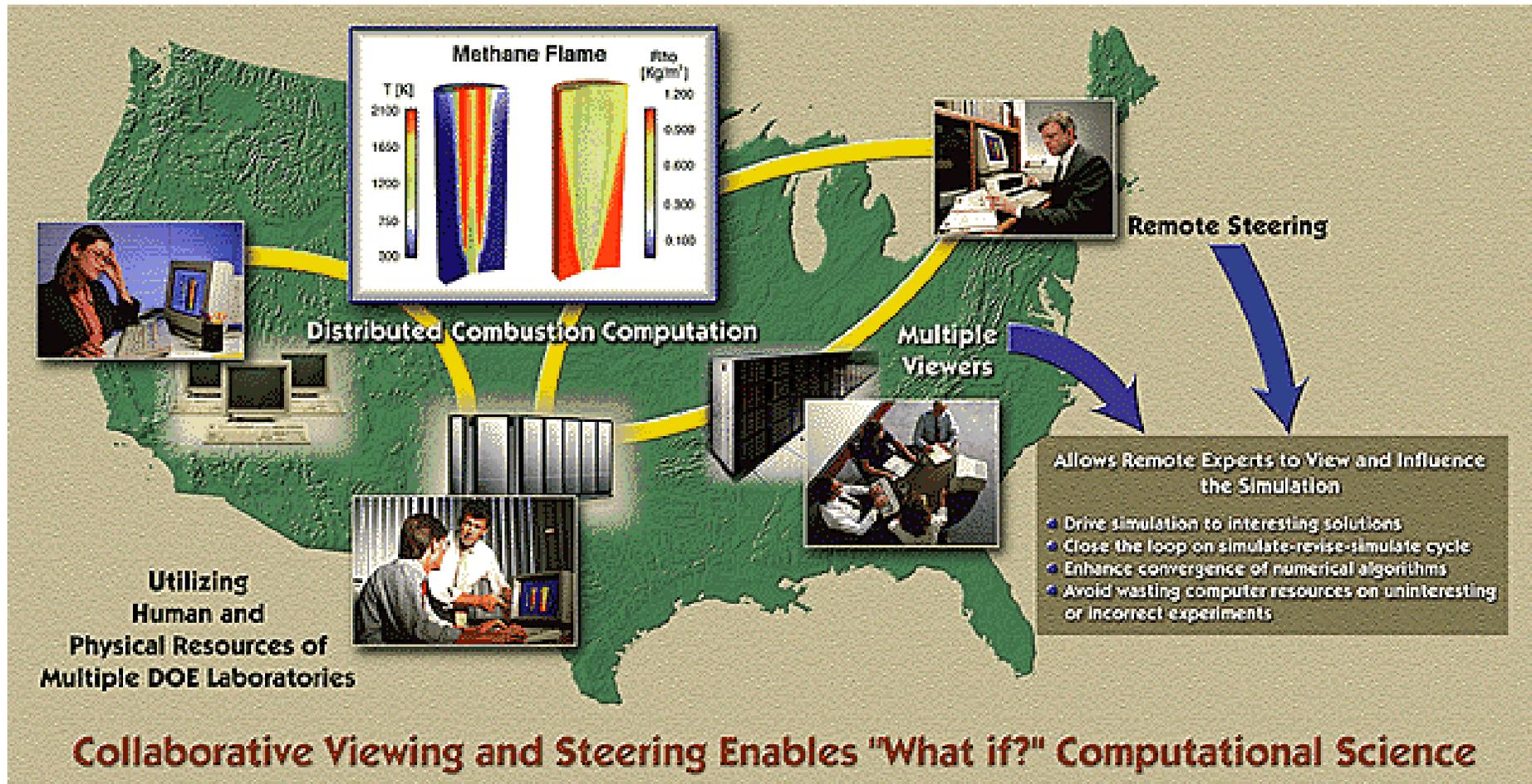
**coordinates the consistent collection and dissemination of information to/from parallel tasks to multiple viewers**



distributed parallel application or simulation

supports most target platforms (PVM/MPI, Unix/NT, etc.)

# Collaborative Combustion Simulation



# CUMULVS

## Steering Features

### ⇒ Computational Steering

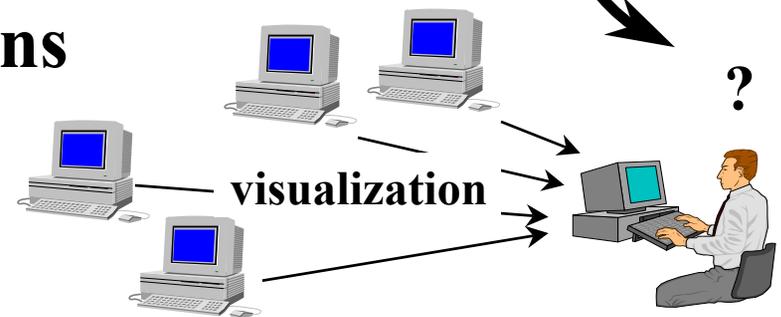
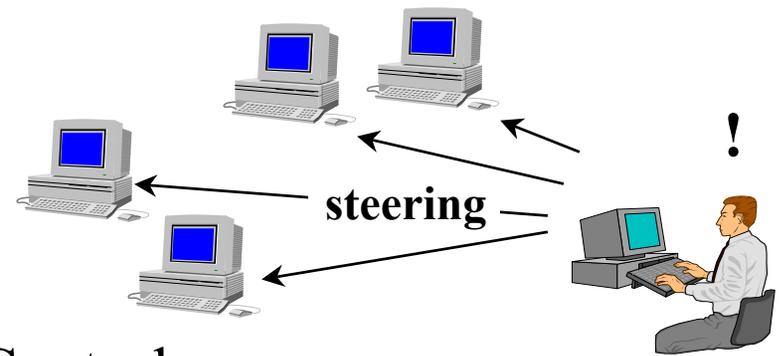
- \* API for Interactive Application Control

### ⇒ Modify Parameters While Running

- \* Eliminate wasteful cycles of ill-posed simulation
- \* Drive simulation to more interesting solutions
- \* Enhance convergence of numerical algorithms

### ⇒ Allows “What If” Explorations

- \* Closes the loop of standard simulation cycle
- \* Non-physical effects...



# Coordinated Steering

⇒ Multiple, Remote Collaborators

⇒ Simultaneously Steer Different Parameters

- \* Physical Parameters of Simulation

- \* Algorithmic Parameters ~ e.g. Convergence Rate

⇒ Cooperate with Collaborators

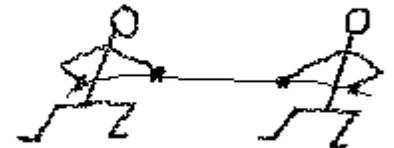
- \* Parameter Locking ~ Prevent Conflicts

- \* Vectored Parameters...

⇒ Parallel / Distributed Simulations

- \* Synchronize with Parallel Tasks

- \* All Tasks Update Parameter in Unison



# CUMULVS Fault Tolerance Features

## ⇒ **Application Fault Tolerance**

- \* Automatic detection and recovery from failures

## ⇒ **User Directed Checkpointing**

- \* User decides what and where to checkpoint
- \* Minimizes amount of stored data

## ⇒ **Heterogeneous Task Migration**

- \* Tasks can be restarted on heterogeneous hosts
- \* Restart file is automatically repartitioned if host pool is of a different size (yikes!)

## ⇒ **Avoids Synchronizing Distributed Tasks**

- \* Asynchronous checkpointing and detection
- \* Minimize intrusion of distributed checkpoint/restart



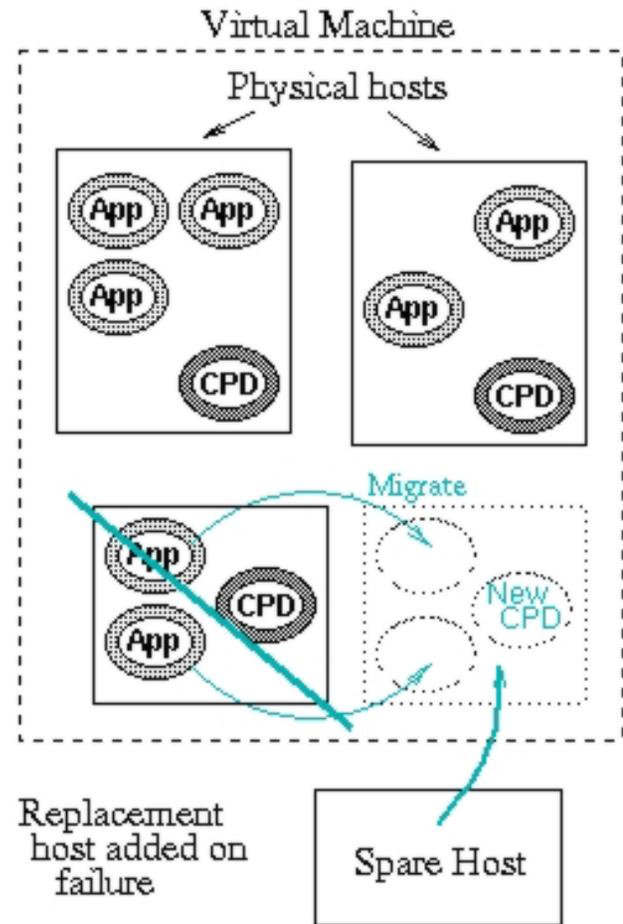
# Run-Time System Architecture

⇒ 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



# Manual Software Instrumentation

⇒ SPDT 98 Case Study ~ SW Instrumentation Cost

Instrumentation:	Seismic:	Wing Flow:
Original Lines of Code	20,632	2,250
Vis / Steer System Init	3	3
Vis / Steer Variable Decls	48	73
CP Restart Initialization	21	12
CP Rollback Handling	41	34
Total Instrumentation	204 ~ 1.0 %	188 ~ 7.7 %

# Checkpointing Efficiency

⇒ SPDT 98 Case Study ~ Execution Overhead

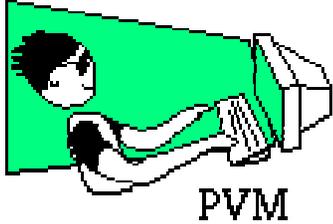
Seconds per Iteration

Experiment:	SGI:	Cluster:	Hetero:
Seismic - No Checkpointing	2.83	6.23	9.46
Seismic - Checkpoint for Restart	2.99	6.50	10.76
Seismic - Checkpoint for Rollback	3.03	6.66	10.90
Wing - No Checkpointing	0.69	1.58	6.14
Wing - Checkpoint for Restart	0.77	1.71	7.10
Wing - Checkpoint for Rollback	0.79	1.71	7.30

(Checkpointing Every 20 Iters.)

**Seismic Overhead: 4-14% Restart, +1-3% Rollback.**

**Wing Overhead: 8-15% Restart, +0-2.5% Rollback.**



# PVM (Parallel Virtual Machine)

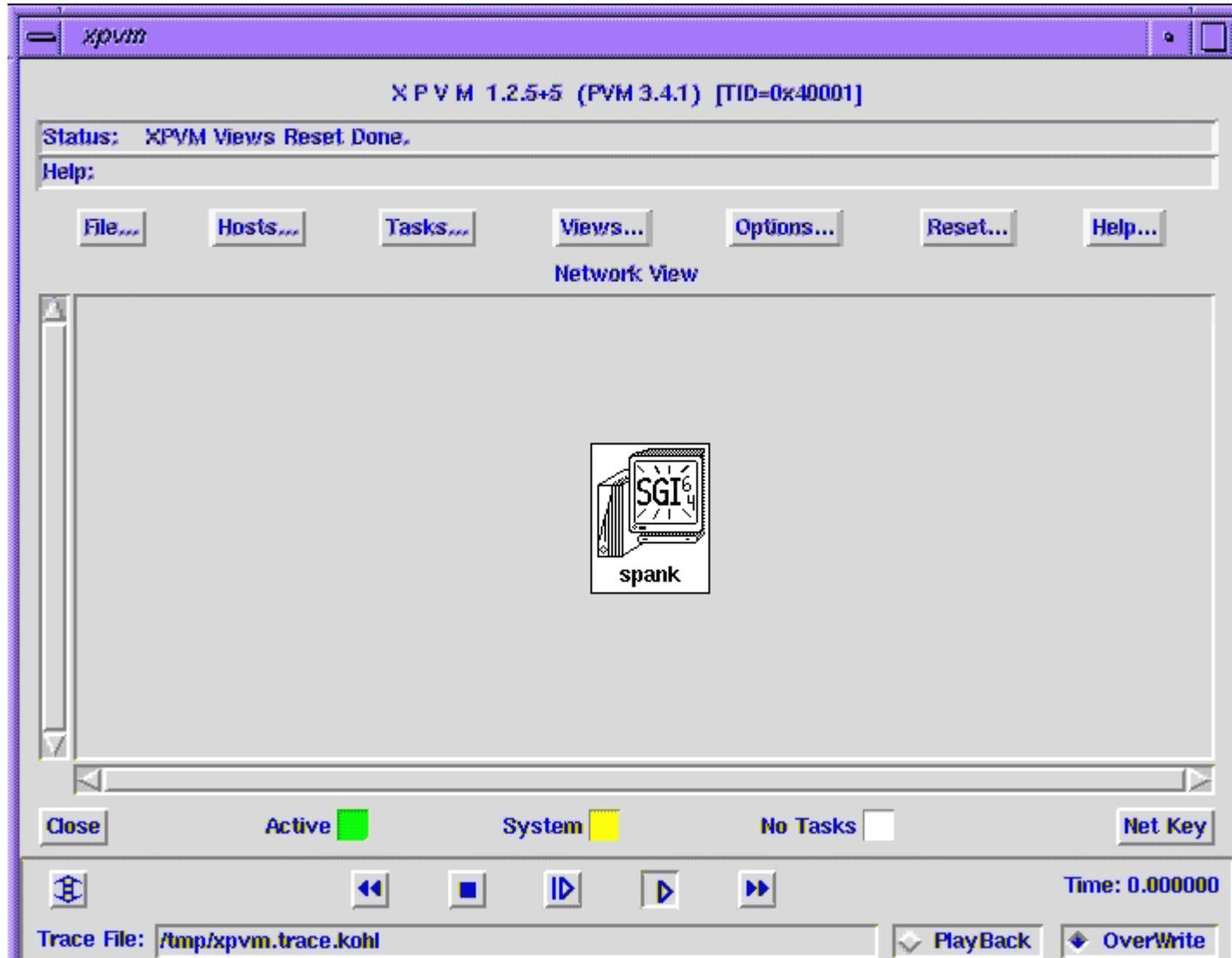
⇒ Use Arbitrary Collection of Computers as a Single, Large, Uniform Parallel Computer

- \* Workstations, PCs (Unix or NT) ~ Clusters
- \* SMPs, MPPs
- \* Connected by a Network

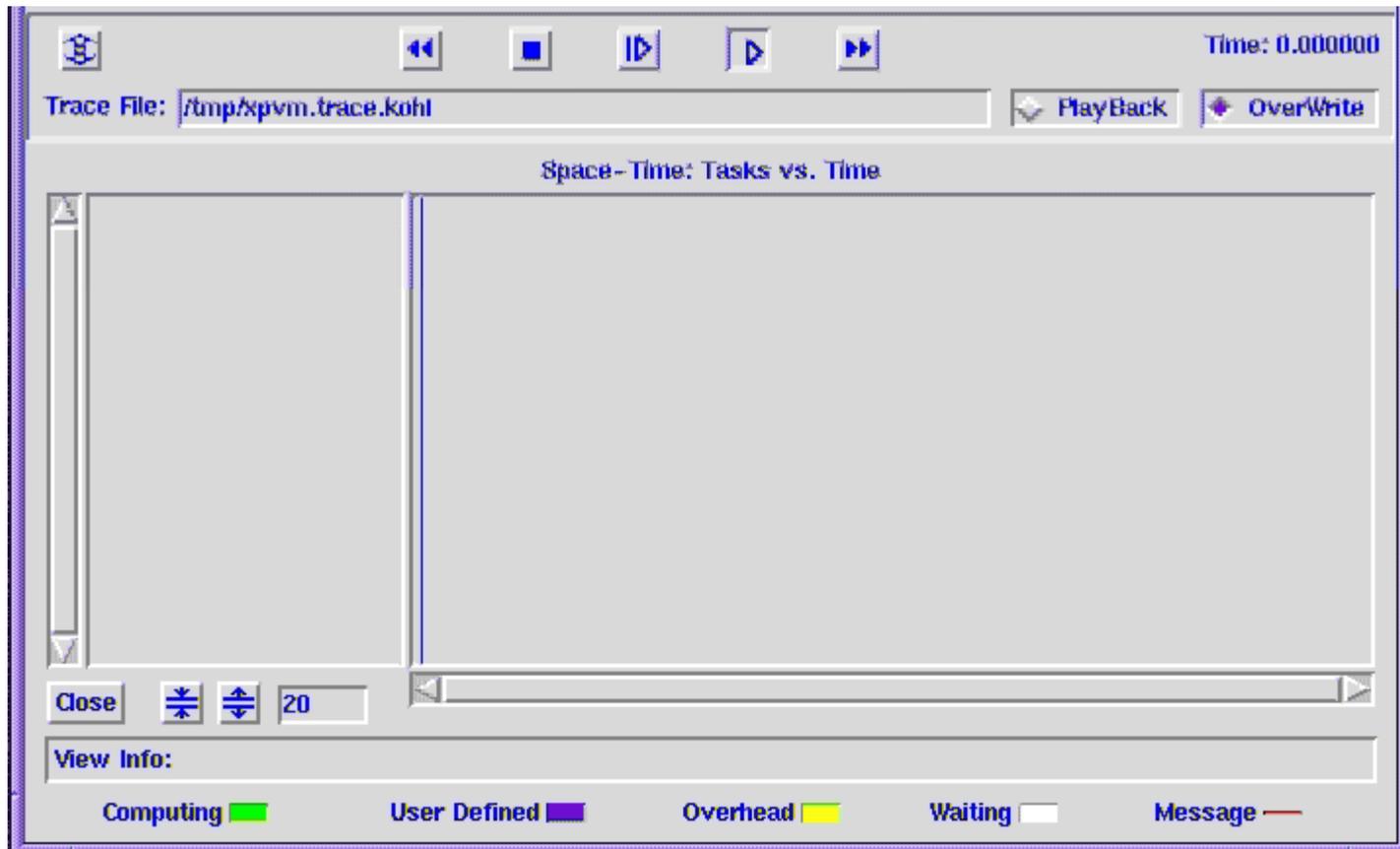
⇒ Programming Model & Runtime System

- \* Message-Passing ~ “Point-to-Point”
- \* Process Control, Key-Value Database
- \* Fault Notification

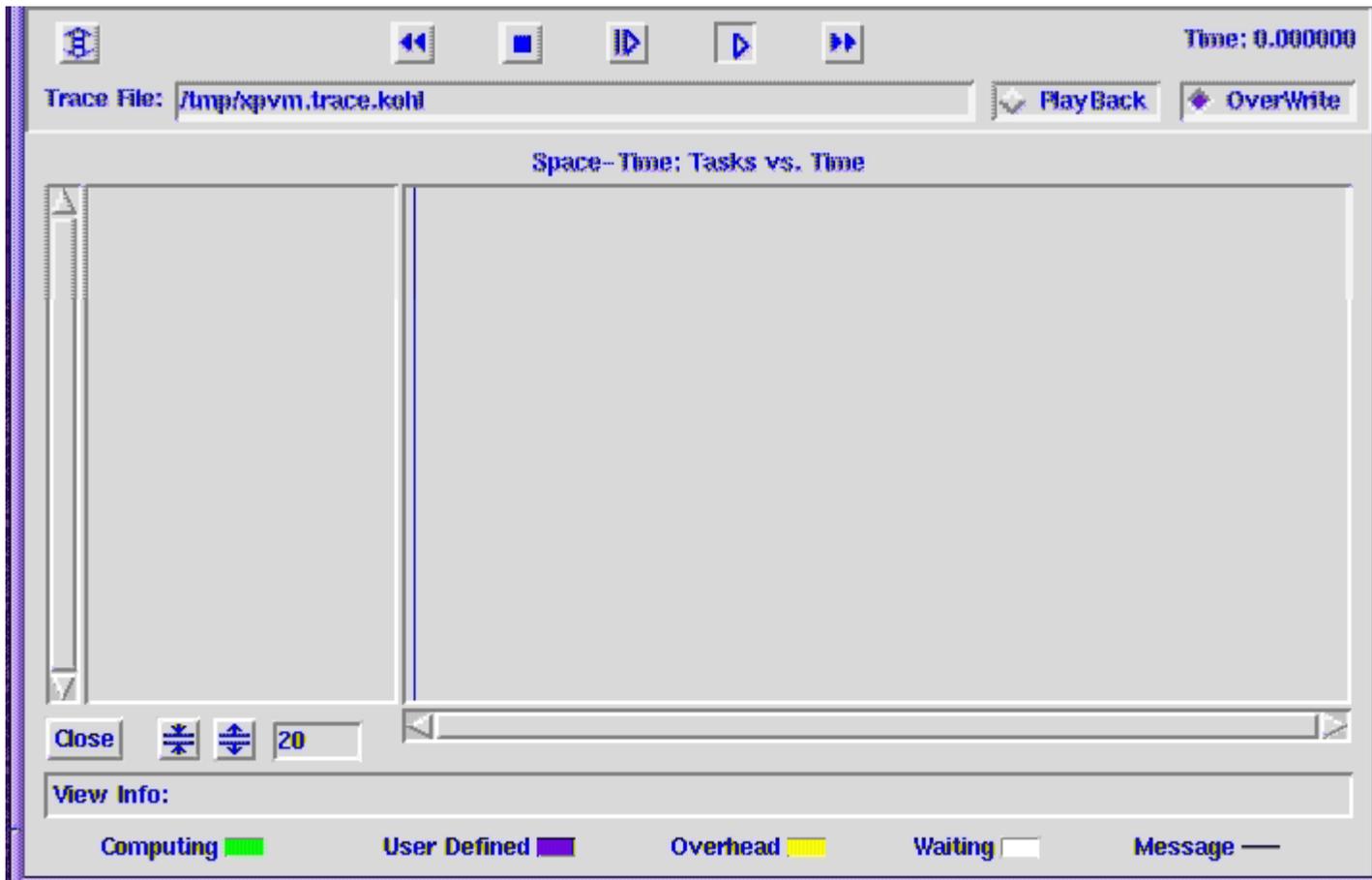
# Virtual Machine Hosts & Spawning



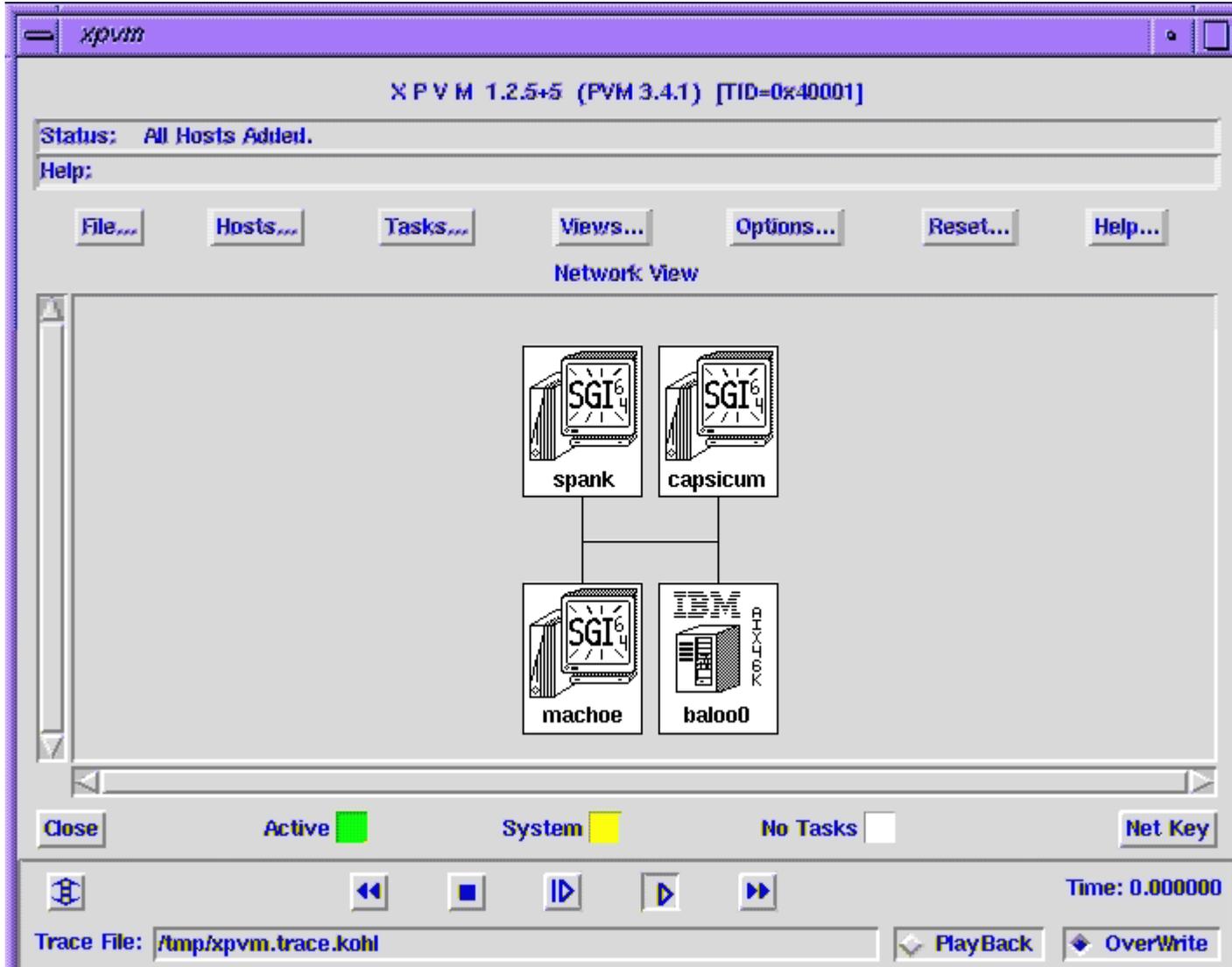
# PVM Tasks ~ Space-Time View



# CUMULVS ~ Fault Tolerance Demo



# Hosts View ~ Fault Tolerance Demo



# 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...)
- \* “Rich” API (MPI-1 / 128, MPI-2 / 288)
- \* Performance

## ⇒ PVM

- \* Dynamic Model
- \* “Simple” API (PVM 3.4 / 75)
- \* Flexibility

# Portability vs. Interoperability

⇒ Portable:

- \* Re-compile Source Without Modification on a Different System.
- \* Both MPI and PVM.

⇒ Interoperable:

- \* Executables on Different Systems Communicate
- \* PVM, Sometimes MPI (Standard Doesn't Require)
- \* Different MPI Implementations? No Way!

# Language Support

⇒ Write Programs in C, Fortran, C++, F90?

\* Yes, in Both MPI and PVM.

⇒ Communicate Among Such Programs?

\* Yes, in PVM. No Problem.

\* MPI? Maybe... Not Required by Standard.

⇒ Getting the Idea? ☺

# Scenario 1: Homogeneous Systems

⇒ Interoperability is Irrelevant

⇒ MPI

\* Best Performance using Optimized Native Comm.

⇒ PVM

\* Trades Performance for Flexibility, Unnecessary.

⇒ MPI “Wins”!

# Scenario 2: Heterogeneous Systems

⇒ Typical Situation

- \* Front-End/Back-End Model.

⇒ PVM

- \* Transparent Handling by Default, “It Works.”

⇒ MPI

- \* No Guarantees... Additional Standard Required.

- \* No Vendor Cooperation? (Performance Loss...)

⇒ PVM “Wins”!

# Performance vs. Flexibility

⇒ To Be Flexible, You Must Pay the Price.

⇒ Overheads:

- \* Data Conversion, Network Protocol Selection, Extra Message Headers (on top of Native Comm)...

⇒ Choose the Common Denominator.

- \* Not the Best on Any System.

⇒ Performance Dictates Locally Optimal Solution.

- \* Lose Interoperability.

# Interesting Result

⇒ Someone could build an MPI implementation that supports interoperability across different systems / languages.

⇒ But:

\* It Would Perform About the Same as PVM!!

# Supporting MPI Applications in CUMULVS

- ⇒ Vendor-Supported Standard, Growing Application Base
  - \* Selected for ASCI Applications
- ⇒ MPI Built for High Performance
  - \* Static Model, Minimal Operating Environment
  - \* No Name Service / Database, Fault Recovery / Notification?
  - \* MPI\_SPAWN( )...?
- ⇒ Existing CUMULVS Solution:
  - \* Applications Communicate Using MPI
  - \* CUMULVS Viewers/CPDs Attach Using PVM
- ⇒ Possible “Reduced-Functionality” MPI Version...?

# But... What About Mpich-G?

⇒ Globus:

- \* Provides Rich Operating Environment
- \* Nexus Communication Substrate
- \* Duroc Bootstrap & Runtime
- \* MDS Database Server

⇒ Implement CUMULVS using Globus

- \* Short-Circuit to Mpich-G Applications!
- \* Enhance Globus Toolkit, Other Apps...

# CUMULVS Needs

- ⇒ Application Discovery
- ⇒ Dynamic Attachment
- ⇒ Heterogeneous Messaging
- ⇒ Fault Notification / Recovery
- ⇒ I/O ~ Checkpoint Archiving

# Application Discovery

- ⇒ Independently Started Tasks Find Each Other
- ⇒ Simulation Tasks Register in Global Server
  - \* Master Task Serves as Proxy for Attachment
  - \* CUMULVS Viewers Look Up Master
  - \* Master Forwards Attach Requests to Rest of App
- ⇒ Alternatives:
  - \* Nexus “Allow Attach” ~ URL-Based...
  - \* MDS ~ “Real” Global Meta-Data Server (LDAP)

# Dynamic Attachment

⇒ Multiple Viewers Attach to Same Simulation

\* Need Dynamic Communication Setup

→ Nexus Startpoint / Endpoint Trading

⇒ Two-Phase Attachment

1. Request List of Simulation Tasks & Info

→ Send Viewer Startpoint, Get Back Startpoints from App

→ Data Fields & Decompositions, Steering Parameters

2. Request Specific Data Field(s)

→ Each Viewer Talks with Proper Subset of Tasks

# Heterogeneous Messaging

⇒ Simulation Tasks Pack & Send Data to Viewer

- \* “Push” Model (Simulation → Viewer(s))

- \* Viewers Request Data at Desired Frequency

  - Subregion in Global Coordinates, Level of Detail

- \* Viewers Send Steering Updates Asynchronously

- \* Normal Nexus “Send RSR” Communication

⇒ Flow Control Protocol for Data Frames

- \* Loose Synchronization

  - App Only Blocks at Next Iteration if No “XON”...

- \* Nexus Message Handlers Update Protocol State

# Fault Notification

## ⇒ All Protocols Fault-Tolerant

- \* Survive Failures in Simulation Tasks or Viewers
- \* Indirect Failures from Host / Network Crashes
- \* Viewers Come and Go Without Interfering
  - With Each Other or With Simulation Tasks
- \* Use `globus_nexus_enable_fault_tolerance()`

## ⇒ Fault Notification Messages

- \* Allow Bail Out from Protocols – But “Who”?
- \* Don’t Kill Off Remaining Tasks
  - Let CUMULVS Handle, No “MPI Genocide” ☺

# Fault Recovery...

- ⇒ Restart Failed Simulation Tasks
  - \* User Can Restart Any Viewers...
  - \* Need Mechanism to Spawn Replacement Tasks
- ⇒ Either Restart or Roll Back Remaining Tasks
  - \* Roll Back is Elegant, Efficient, and NASTY!
  - \* Restart is Simple but EASY...
  - \* Application Chooses
- ⇒ Checkpoint Data Sent in Regular Messages
  - \* Program State Automatically Reconciled...

# I/O ~ Checkpoint Archiving

## ⇒ Checkpoints Saved to Local Disk

- \* CPD Per Host Collects From Local Tasks

- \* Local Checkpoints in Binary Format

  - CPDs Translate Using Messaging Substrate (RSR)

## ⇒ Parallel File System / Disk Arrays

- \* Handle Redundancy at Hardware Level?

- \* Assistance in Committing Checkpoints

  - Collect & Assemble Individual Task Checkpoints...

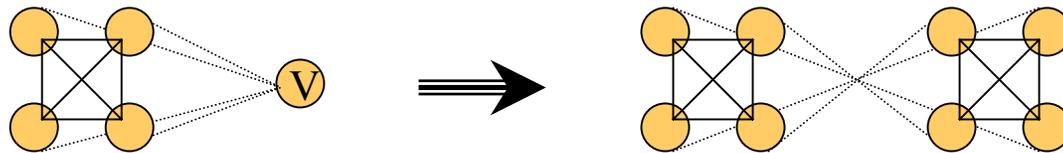
- \* Disk Access ~ Performance Bottleneck

# Future CUMULVS Plans (1 of 3)

## Coupling Data Fields in Simulation Models

⇒ Natural Extension to Viewer Scenario

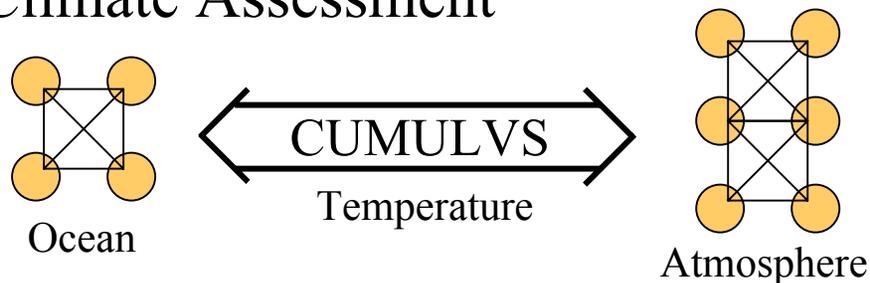
- \* Promote “Many-to-1” → “Many-to-Many”



⇒ Translate Disparate Data Decompositions

- \* Complements PAWS Coupling Work
- \* Builds on CCA (Common Component Architecture) Forum

E.g. Regional Climate Assessment



# Future CUMULVS Plans (2 of 3)

## Building on Harness

⇒ “Harness” ~ Next Generation HDC Environment

- \* Pluggable Virtual Machine, Distributed Control

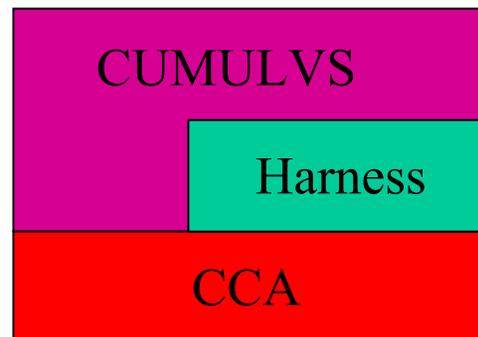
  - Follow-on to PVM...

- \* Inspired by CUMULVS Coupling Needs

- \* ORNL, UT and Emory (Basic Research)

⇒ Reinforces Need for CCA

- \* Harness Pluggability Builds on CCA Foundation



## Future CUMULVS Plans (3 of 3)

### ⇒ 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
  - Nexus, Reduced Functionality for 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 with Other Frameworks via CCA
- \* Application Instrumentation GUI / Pre-Compiler

<http://www.csm.ornl.gov/cs/cumulvs.html>

# Seismic Example ~ 2D (Tc1/Tk)

# Seismic Example ~ 3D (AVS)

# Air Flow Over Wing Example ~ 3D (AVS)