High-Performance Scientific Simulation and Distributed Computing at ORNL: Harness, CUMULVS and the Common Component Architecture (CCA)

Dr. James Arthur Kohl
Oak Ridge National Laboratory

Wednesday, September 13, 2000

Scientific Simulation is HOT!

• Flexible, Powerful, Inexpensive (?)
• Remote Collaboration Possible
• Infrastructure Issues:
  ⇒ Interaction / Control of Parallel Simulations
  ⇒ Fault Tolerance ~ Clusters, Networks, etc.
  ⇒ Adaptable to Changing Resources / Technology

• ORNL Projects:
  ⇒ Harness, CUMULVS, CCA
Distributed Computing History at ORNL

PVM was developed as a tool to help us explore HDC issues. Software has been redesigned from scratch every 2-3 yrs to study emerging architecture, network, and user needs.

1990 **Heterogeneity** - architecture, data, power, network... explore the problems and research issues in heterogeneous computing

1991 **Robust, Portable Programming Environment** study how to create a robust, portable programming environment for HDC

1993 **Transparent Multiprocessor integration** efficient multi-protocol handling of distributed and shared memory computers

1995 **Fault Tolerance and Extensibility** study VM and application fault tolerance. Design first VM plug-in interfaces

1998 **Windows and Unix interoperability** enable the millions of NT and Win2000 hosts to exploit cluster computing

Next Step is Harness
Building on our experience and success with PVM we will create a fundamentally new heterogeneous virtual machine environment based on three key research concepts:

- **Parallel Plug-in Environment**
  Extend the concept of a plug-in to the parallel computing world. Dynamic with no restrictions on functions.

- **Distributed Peer-to-Peer Control**
  No single point of failure unlike typical client/server models.

- **Multiple Distributed Virtual Machines Merge/Split**
  Provide a means for short-term sharing of resources and collaboration between teams.
HARNESS Motivation

Needs of Simulation Science and Cluster Computing

• develop applications by plugging together component models.

• customize/tune virtual environment for application’s needs and for performance on existing resources.

• support long-running simulations despite maintenance, faults, and migration (dynamically evolving VM).

• adapt virtual machine to faults and dynamic scheduling in large clusters (C-Plant).

• Provide framework for collaborative simulations (in spirit of CUMULVS or a collaborative PSE).
HARNESS Team
Collaborative Effort by the Developers of PVM

Al Geist,
Jim Kohl, Stephen Scott, Conrad Albrecht-Buehler, Wael Elwasif
Oak Ridge National Laboratory

Jack Dongarra,
Graham Fagg, Martin Swany, Nathan Garner
University of Tennessee

Vaidy Sunderam,
Paul Gray, Mauro Migliardi, Gopi Sankar
Emory University

www.csm.ornl.gov/harness
HARNESS Virtual Machine
Scalable Distributed Control and CCA-Based Daemon

Operation within VM uses Distributed Control

Merge/split with other VMs

Component based daemon

Customization and extension by dynamically adding plug-ins

process control
user features

HARNESS daemon
HARNESS Architecture
Hlib and Harness daemon within host

Example: PVM Application on Harness

Host A

Application task

Hlib

PVM plug-in

Language independent interface

TCP/IP

Harness daemon

PVMD plug-in

Host B

Application task

Hlib

PVM plug-in

TCP/IP

Harness daemon

PVMD plug-in

Communication direct task-task

Task to pvmd API

Distributed control
HARNESS Development Plan

Plan is to produce the core framework and a couple key plug-ins that will provide a practical environment and illustrate the capabilities of HARNESS:

- **Harness Core**
  Task library and Harness Daemon software
  Provides API to load, unload plug-ins and distributed control.

- **PVM Plug-in**
  Provides PVM API veneer to support exiting PVM applications.

- **Fault tolerant MPI plug-in**
  Provides MPI API for 30 most used functions. Semantics adjusted to allow recovery from corrupted communicator.

- **VIA/FM communication plug-in**
  To illustrate how different low level communication plug-ins can be used within Harness, and to provide high performance.
Harness Core Implementations

Two Different Schemes are Being Explored

- **C Scheme (ORNL)**
  - Based on dynamically linked / shared libraries
  - Advantage ~ DLL / lib can be written in the language of the app
  - Potential for higher performance plug-ins (compiled binary)

- **Java (2) Scheme (Emory)**
  - Based on JVM dynamic class loader
  - Advantage ~ fast prototyping
  - Leverages wealth of existing Java specs, JINI, JavaBeans, RMI, Java Spaces, etc.
  - Good integration with emerging Java SC apps and interfaces
  - Beta Release!
Symmetric Peer-to-Peer Distributed Control
Requirements

- No single point of failure for Harness.
  ⇒ It survives as long as one member still lives.

- All members know the state of the virtual machine
  ⇒ Knowledge is kept consistent w.r.t. the order of changes of state
  (Important parallel programming requirement!)

- No member is more important than any other
  ⇒ At any instant
  ⇒ There’s no “control token” being passed around
**Distributed Control**

Harness Overlapping Two Phase Arbitration

Harness kernels on each host have arbitrary priority assigned to them (new kernels are always given the lowest priority)

1. A task on this host requests a new host be added
2. Send host/T#/data to neighbor in ring
3. Each adds request to a list of pending changes

Virtual machine

VM state held by each kernel

Each adds request to a list of pending changes
Harness Distributed Control

Control is Asynchronous and Parallel

- Supports multiple simultaneous updates
- Fast host adding or recovery from fault
- Parallel recovery from multiple host failures
HARNESS Scalability
Variable Distributed Control Loop Size

Virtual machine

Size of the Control Loop
1 <= S <= (size of VM)

For small VM and ultimate fault tolerance S = (size of VM)

For large VM a random selection of a few hosts (f.e. S = 10) gives a balance of multi-point failure and performance.

For S = 1, distributed control becomes simple client/server model.
Fault Tolerant MPI

Motivation

Two major drawbacks to MPI are:
• lack of interoperability - being addressed by IMPI and MPI-Connect
• lack of fault tolerance - any failure is catastrophic

As application and machine sizes grow the MTBF is less than the application run time.

MPI standard is based on a static model any decrease in tasks leads to corrupted communicator (MPI_COMM_WORLD).

Develop MPI plugin that takes advantage of Harness robustness to allow a range of recovery alternatives to an MPI application. Not just another MPI implementation.

FT-MPI follows the syntax of MPI standard.

Being used on Blue at LLNL
HARNESS Research Status

Java-based HARNESS prototype created at Emory
  • used to test parallel plug-in concepts
  • integration with JINI underway

IceT package developed by Paul Gray (Iowa St.)
  • demonstrates merging and splitting of virtual machines
  • dynamically switching communication (MPI to CCTL)

C-based HARNESS kernel and distributed control
  • feasibility demonstrated at ORNL
  • production release in progress

C-based FT-MPI plug-in prototype developed at UTK
  • built on top of PVM 3.4 API
Harness Enables New Kinds of Applications

Thinking “Outside the Box” -- It’s not just for Scientific Computing

Harness is still just a research project but its potential is great

Applications follow user roaming wearable computers

On-the-fly simulation tuning plug-in different methods if simulation evolves to need them

Teams of tasks patrol and monitor local network for performance or security
(Collaborative, User Migration, User Library for Visualization and Steering)

- Collaborative Infrastructure for Interacting with Scientific Simulations:
  ⇒ Run-Time Visualization by Multiple Viewers
    → Dynamic Attachment
  ⇒ Coordinated Computational Steering
    → Model & Algorithm
  ⇒ Heterogeneous Fault Tolerance
    → Automatic Fault Recovery and Task Migration
  ⇒ Coupled Models…
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.)

remote person using AVS

remote person using virtual reality interface

exists in three pieces: task part, viewer part, and separate fault recovery daemon

local person using custom GUI

Unix Host A

Unix Host B

Unix Host C

Kohl/2000-19
Collaborative Combustion Simulation

Collaborative Viewing and Steering Enables "What if?" Computational Science
Multiple Simultaneous Views

Density

Temperature
Multiple Distinct Views

Distributed Data Array

Cumulus attaches/detaches viewers from parallel simulation, on-the-fly

Global View 1

AVS

Remote collaborators view different parts of simulation, simultaneously

Global View 2

$ic/i_k$

Instrument existing parallel code.

C - spad.f
  call stvfieldinit()
  call stvfielddefine()
  do
    call localwork!!
    call exchangeinfo!!
    call stvfielddefine()
  while: not: done!!
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
Coordinated Steering

- Multiple, Remote Collaborators
- Simultaneously Steer Different Parameters
  ⇒ Physical Parameters of Simulation
  ⇒ Algorithmic Parameters ~ e.g. Convergence Rate
- “What If?” Analyses
  ⇒ Explore Non-Physical Effects
- Efficient Experimentation Cycle
  ⇒ Keep Simulation On Track
  ⇒ Crop Off Experiments Gone Awry…
Heterogeneous Checkpointing

• Application Defines Its Own State
  ⇒ Provide CUMULVS with Semantic Information
  ⇒ CUMULVS Handles Checkpoint Collection
  ⇒ Automatic Fault Recovery System

• Efficient & Flexible Checkpoints
  ⇒ Not Full Core Image
  ⇒ Semantics → Heterogeneous Restart & Migration
  ⇒ On-The-Fly Reconfiguration Also Possible…
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**

<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>Total Instrumentation</td>
<td>204 ~ 1.0 %</td>
<td>188 ~ 7.7 %</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 Iters.)*

Seismic Overhead: 4-14% Restart, +1-3% Rollback.  
Wing Overhead: 8-15% Restart, +0-2.5% Rollback.
Seismic Example ~ 3D (AVS)
Air Flow Over Wing Example ~ 3D (AVS)
Coupling Data Fields in Simulation Models Using CUMULVS

• 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

ORNL Kohl/2000-31
Future CUMULVS Plans

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

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

• Portability:
  ⇒ Other Messaging Substrates
  → Reduced Functionality for MPI / MPI-2…

http://www.csm.ornl.gov/cs/cumulvs.html
Common Component Architecture (CCA)

- Component Architecture for Scientific Simulation
  ⇒ Special Emphasis on High-Performance / Parallelism
- Reusable “Components” Connect Via “Ports”
  ⇒ Forum Creating Specification and Reference Framework

Direct Connect Ports
(Local Components Share Memory)

Collective / MxN
(Parallel Data Exchange)
Common Component Architecture (CCA) / ACTS Toolkit
Oak Ridge National Laboratory

CCA

MxN (Kohl)

Mesh & Field (SNL/ANL)

MxN (Kohl)

MxN (LLNL/Utah)

MxN Framework (LLNL/Utah)

Visualization (Wilde)

Comp. Chemistry (Bernholdt)

Comp. Steering

Global Arrays (PNNL)

CCA Demo SC99

ESI (SNL)

GIST (LoCascio)

CUMULVS (Kohl)

Fault Tolerance

Components

Framework

ORNL

Kohl/2000-34
MxN Parallel Data Mapping
Collective MxN Example

Ocean

getDataField( space, time )

Collective Port

Atmosphere

Kohl/2000-36
High-Performance Parallel Collective Port

getDataField( space, time )
getDataField( space, time )
getDataField( space, time )
getDataField( space, time )
getDataField( space, time )
getDataField( space, time )
getDataField( space, time )
getDataField( space, time )
getDataField( space, time )
Summary of ORNL Scientific Simulation

- Harness ~ Next Generation HDC Environment
  ⇒ Pluggable Virtual Machine, Distributed Control
- CUMULVS ~ Interacting with Simulations On-The-Fly
  ⇒ Visualization, Steering, Fault Tolerance
- Common Component Architecture (CCA)
  ⇒ Harness Pluggability Builds on CCA Foundation
  ⇒ CUMULVS Technology Used for MxN / Coupling