Building Coupled Parallel and Distributed Scientific Simulations

Alan Sussman

Department of Computer Science & Institute for Advanced Computer Studies
A Simple Example (MxN coupling)

Visualization station

Parallel Application LEFTSIDE (Fortran90, MPI-based)

Parallel Application RIGHTSIDE (C++, PVM-based)

InterComm: Data exchange at the borders (transfer and control)

M=4 processors

N=2 processors

2-D Wave Equation
Coupling Parallel Programs via InterComm

- **Introduction**
  - Problem Definition (the MxN problem)
  - InterComm in a nutshell

- **Design Goals**
  - Data Transfer Infrastructure
  - Control Infrastructure
  - Deploying on available computational resources

- **The Landscape**
  - What else is out there?
The Problem

- Coupling *codes*, not models
- Codes written in different languages
  - Fortran (77, 95), C, C++/P++, …
- Both parallel (shared or distributed memory) and sequential
- Codes may be run on same, or different resources
  - One or more parallel machines or clusters (the Grid)
Our driving application:

- Production of an ever-improving series of comprehensive scientific models of the Solar Terrestrial environment
- Codes model both large scale and microscale structures and dynamics of the Sun-Earth system
What is InterComm?

- A programming environment and runtime library
- For performing efficient, direct data transfers between data structures (multi-dimensional arrays) in different programs
- For controlling *when* data transfers occur
- For deploying multiple coupled programs in a Grid environment
Data Transfers in InterComm

- **Interact** with data parallel (SPMD) code used in separate programs (including MPI)
- **Exchange** data between separate (sequential or parallel) programs, running on different resources (parallel machines or clusters)
- **Manage** data transfers between different data structures in the same application
- CCA Forum refers to this as the **MxN** problem
InterComm Goals

- One main goal is **minimal modification** to existing programs
  - In scientific computing: plenty of legacy code
  - Computational scientists want to solve their problem, not worry about plumbing
- Other main goal is **low overhead and efficient data transfers**
  - Low overhead in *planning* the data transfers
  - Efficient data transfers via customized all-to-all message passing between source and destination processes
Issues in Coupling Codes

To enable a program to be coupled to others, we need to:

- Describe data distribution across processes in each parallel program
  - Build a data descriptor
- Describe data to be moved (imported or exported)
  - Build set of regions
- Build a communication schedule
  - What data needs to go where
- Move the data
  - Transmit the data to their proper locations
Data Transfer

- It all starts with the **Data Descriptor**
  - Information about how the data in each program is distributed across the processes
  - Usually supplied by the program developer
- **Compact or Non-Compact descriptors**
  - Regular Blocks: collection of offsets and sizes (one per block)
  - Irregular Distributions: enumeration of elements (one per element)
- Performance issue is that different algorithms perform best for different combinations of source/destination descriptors and local vs. wide area network connections
Coupling OUTSIDE components

- Separate coupling information from the participating components
  - Maintainability – Components can be developed/upgraded individually
  - Flexibility – Change participants/components easily
  - Functionality – Support variable-sized time interval numerical algorithms or visualizations
- Matching information is specified separately by application integrator
- Runtime match via simulation time stamps
Controlling Data Transfers

- A flexible method for specifying when data should be moved
  - Based on matching export and import calls in different programs via timestamps
  - Transfer decisions take place based on a separate coordination specification
    - Coordination specification can also be used to deploy model codes and grid translation/interpolation routines (how many and where to run codes)
Simulation exports every time step, visualization imports every 2\textsuperscript{nd} time step.
Separate codes from matching

Exporter Ap0

```
define region Sr12
define region Sr4
define region Sr5
...
Do t = 1, N, Step0
    ... // computation jobs
    export(Sr12,t)
    export(Sr4,t)
    export(Sr5,t)
EndDo
```

Importer Ap1

```
define region Sr0
...
Do t = 1, M, Step1
    import(Sr0,t)
    ... // computation jobs
EndDo
```

Configuration file

```
#
Ap0 cluster0 /bin/Ap0 2 ...
Ap1 cluster1 /bin/Ap1 4 ...
Ap2 cluster2 /bin/Ap2 16 ...
Ap4 cluster4 /bin/Ap4 4
#
Ap0.Sr12  Ap1.Sr0  REGL  0.05
Ap0.Sr12  Ap2.Sr0  REGU  0.1
Ap0.Sr4   Ap4.Sr0  REG   1.0
#
```

Diagram:

- Ap0.Sr12 -> Ap1.Sr0
- Ap0.Sr4 -> Ap2.Sr0
- Ap0.Sr5 -> Ap4.Sr0
Exporter Ap0 produces a sequence of data object $A$ at simulation times 1.1, 1.2, 1.5, and 1.9
- $A@1.1$, $A@1.2$, $A@1.5$, $A@1.9$

Importer Ap1 requests the same data object $A$ at time 1.3
- $A@1.3$

Is there a match for $A@1.3$? If Yes, which one and why?
Controlling Data Transfers

- **Import** and **Export** operations are time-stamped ($T_i$ and $T_e$)
- Issues in designing *Decision Functions*
  - **Matching Policy**
    - Does the import timestamp *match* any of the exported timestamps, subject to a particular policy?
  - **Precision**
    - Which of the exported data most closely matches what is requested to be imported?
- Decision functions directly affect InterComm buffering decisions!
Plumbing

- Bindings for C, C++/P++, Fortran77, Fortran95
- Currently *external* message passing and program interconnection via PVM
- Each model/program can do whatever it wants internally (MPI, pthreads, sockets, …) – and start up by whatever mechanism it wants
Deploying Components

Infrastructure for deploying programs and managing interactions between them

1. **Starting** each of the models on the desired Grid resources
2. **Connecting** the models together via the InterComm framework
3. Models communicate via the **import** and **export** calls
Motivation

- Developer has to deal with …
  - Multiple logons
  - Manual resource discovery and allocation
  - Application run-time requirements

- Process for launching complex applications with multiple components is
  - Repetitive
  - Time-consuming
  - Error-prone
Deploying Components

- A single environment for running coupled applications in the high performance, distributed, heterogeneous Grid environment
- We must provide:
  - **Resource discovery**: Find resources that can run the job, and automate how model code finds the other model codes that it should be coupled to
  - **Resource Allocation**: Schedule the jobs to run on the resources – without you dealing with each one directly
  - **Application Execution**: Start every component appropriately and monitor their execution
- Built on top of basic Web and Grid services (XML, SOAP, Globus, PBS, Loadleveler, LSF, etc.)
1. Assign Job ID
2. Create scratch directories
3. Create/ manage app specific runtime info
4. Allocate requests (with retry)
   - Send script to local resource manager
   - Lock file creation
1. Transfer input files to resources (put)
2. Deal with runtime environment (e.g., PVM)
3. Split single resource across components
4. Launch using appropriate launching method
5. Transfer created files to user’s machine (get)
Current implementation

- **Discovery:**
  - Simple resource discovery service implemented
  - Resources register themselves with the service, including what codes they can run

- **Resource Allocation**
  - Scripts implemented to parse XJD files and start applications on desired resources
  - Works with local schedulers – e.g., PBS, LoadLeveler

- **Job Execution:**
  - Startup service implemented – for machines without schedulers, and with schedulers, for PBS (Linux clusters) and LoadLeveler (IBM SP, p Series)
  - Works via ssh and remote execution – can startup from anywhere, like your home workstation
  - Need better authentication, looking at GSI certificates
  - Building connections automated through startup service passing configuration parameters interpreted through InterComm initialization code
Current status

- **First InterComm release (data movement) April 2005**
  - source code and documentation included
  - Release includes test suite and sample applications for multiple platforms (Linux, AIX, Solaris, …)
  - Support for F77, F95, C, C++/P++
- **Next release, InterComm 1.5, imminent**
  - adds support for automated launching and resource allocation, tool for building XJD files for application runs
  - new version of library with export/import support, broadcast of scalars/arrays between programs
- Control mechanisms in advanced stage of development
  - will be in next major release (2.0)
- **New project to integrate with ESMF**
  - wrap ESMF objects for communication via InterComm
What else is out there?

- **CCA MxN Working Group**
- **Parallel Application Work Space (PAWS)** [Beckman et al., 1998]
- **Collaborative User Migration, User Library for Visualization and Steering (CUMULVS)** [Geist et al., 1997]
- **Model Coupling Toolkit (MCT)** [Larson et al., 2001]
- **Earth System Modeling Framework (ESMF)**
- **Space Weather Modeling Framework (SWMF)**
- **Roccom** [Jiao et al., 2003]
- **Overture** [Brown et al., 1997]
- **Cactus** [Allen et al., 1999]
The team at Maryland

- **Norman Lo (staff programmer)**
  - InterComm testing, debugging, packaging
  - Looking at augmenting PVM with MPICH-G2
- **Jae-Yong Lee (graduate student)**
  - InterComm data transfer design and implementation
- **Shang-Chieh Wu (graduate student)**
  - InterComm control design and implementation
- **Il-Chul Yoon**
  - Deployment and startup of multiple programs on 1 or more resources
Summary and Ongoing Work

- **InterComm**: a comprehensive high-performance framework for coupling parallel scientific codes
- Plumbing for high performance data transfers is fully functional and released, deployment services about to be released, control functions prototyped and will be released when thoroughly tested
- Working with our customer base to modify their codes and couple their *models*