Out-of-order Execution Parallel Virtual Machine

George Bosilca, Franck Cappello

Laboratoire de Recherche en Informatique
Universite d’Orsay, France
Global view

High performance computing required by a large spectre of users. (companies, research labs)

Weather/Climate
Physics (high energy particle)
Pharmaceutics
Bioinformatic
Medical Images
Cinema
Microprocessors Simulation
Etc.

Regular Applications:
Without load balancing

Non-Uniform Applications:
With load balancing
Branch and bound
Dynamic generation of additional computations

Evolution of parallel architectures toward a multi-level NUMA approach

Clusters of SMP

PC Clusters

GRID, P2P
A central place for the execution model

Applications
Programming Models
Languages
Ex: data parallel, concurrent model, master/slave
Languages: Fortran, C, C++, Java, Matlab, Scilab

Execution Model
API
Execution environment
Virtual Machine
Message Passing: MPI, OVM (RPC),
Virtual Shared Memory: Scach, TreadMarks,
Global Computing Infrastructure: Netsolve, Ninf,
Computational Grids: Globus, Legion, XtremWeb
GRID enabled message passing: MPICH-G,
Virtual shared memory for GRID: OmniOpenMP

Operating System
GRID, P2P

Architectures
Clusters
Clusters of SMP

Hardware
Proposer aux programmeurs des environnements portables permettant de pérenniser sinon leurs applications au moins leurs outils/environnements de programmation.
Programming Models.

- **RPC**
- **OpenMP**
- **MPI**
- **Dataflow**

- Initially not really developed for applications
- Concurrent RPC using several threads
- Data always cross the network with the RPC request
- “global communication” like operations should be done locally

+ A whole range of applications could be easily expressed using RPC
  - Master/slave
  - Recursive
  - Some kind of branch-and-bound
Programming Models ..

- RPC
- OpenMP
- MPI
- Dataflow

Suitable only on shared memory architectures
- Complex to handle irregular applications
- Static environment (MPI2)

Unpredictable results on complex // architectures

+ Perfect for regular applications is a homogeneous environment
+ Have global communications
Programming Models …

• RPC
• OpenMP
• MPI
• Dataflow

• Most of the time the dataflow graph is statically created during the compilation step
  • Internal load-balancing
  • Still no “global communications”

SMART: horizontal and vertical parallelism only for shared memory computers
Programming Model

• PVM
  – The user should do the load-balancing depending on the application
  – Blocking global communication
  – MP layer too complex and quite slow

+++ Dynamic : pvm_spawn

• Harness
  – Support multiple instance of PVM and MPI applications
  – More related to a execution environment
Load Balancing environments

Load-balancing in multi-flow environments:
Athapascan (static dataflow graph + distributed resolution), lab ID, ENSIMAG
PM2 (threads migration), LIFL, ENS Lyon
Cilk (work stealing with a recursion heap), MIT

OO with load-balancing capabilities (95, 96):
Dome (c++ et PVM), SPMD, check point and migration
Charm, Charm++ (thread based) Urbana Champaign

Load-balancing for PVM/MPI (93-94-95; 01):
MPVM (process migration) Oregon Graduate Institute of S & T.
UPVM (threads migration) Oregon Graduate Institute of S & T.
Dynamic PVM (condor checkpoint), University of Amsterdam
MPI TM (task migration – core dump) Mississippi State University
AMPI (threads migration using Charm++) Urbana Champaign
Dynamite PVM (task migration)
Choosing a programming environment

• Multiple parameters equation
  – Computer architecture
  – Application type
  – Level of programming paradigm (performances)

The ideal parallel environment?
• Dynamic: resources can join/leave the environment
• Automatic load balancing (user hints)
• Suitable for HPC
• Suitable for different kind of applications
• Same performances on all architectures
OVM general principles

Research goal:
A virtual machine based on a dynamic macro-dataflow graph and RPC
→ Continuum between SPMD and master/slave approaches

• Why Out-of-Order?

Based on decoupled RPC (data and requests are separate entities)
Client

Simplified view of the system: Unique architecture

- Create and initialize sets of data
- Define affinities between data
- Define the relationship between the data: 
  \[(A,B) = \text{operation}(A,B,C,D)\]
- Collect the final results

- Provide services using plug-ins (shared libraries) on the server side
- Provide a long-time scheduling plug-in for the broker
Servers

- Multi-threaded approach
  - Only one server by node
- Supporting dynamic multi-protocol communication devices
- Provide generic functions through the default plug-ins
- And/Or additional services through user plug-ins
Broker

- Correct execution
- Resolve dependencies
- Resource Management
  - Available nodes
  - Available services
Different RPC modes

1) Classic RPC

2) Decoupled RPC
Requirement: starting a function on the servers require that all the data used by the function should be available on that server.

Interest: only the real data dependencies limit the parallelism.
SPMD Applications

Purpose: execution flow as close as possible to MPI
Cyclic placement of the persistent data
Problem: Hide the cost of the macro-dataflow management

- persistent data
- static scheduling
- global operations
Irregular Applications

Purpose: limit the migration, increase the number of concurrent tasks
Scheduling: First Come First Serve

- Volatile data
- Dynamic scheduling
- No global operations

Servers

Shared Libraries

Volatiles data

Data flow

Scheduling function

Client + broker
Optimizations

• Classic multi-protocole RPC:
  - embedded data if the size < 4 Ko
  - or asynchronous data retrieval from the client ( >= 4 Ko )
    - blocking or non-blocking mode for the client
• Non-blocking global communications
  - each server start a new execution after his contribution to a global communication
• Asynchronous global communications
  - the completion order of the global communications could not be the same as the starting order (independent global communications could be executed in parallel)
• Task prefetch when persistent data are located on the same server – hierarchic dataflow
• Task prefetch on mode RPC with volatile data, without checking for ressource dependencies to hide data transfert cost for the next task
• Data cache on the servers side (for volatile data)
• Both kind of RPC can be used in same time ( (a) = operation(2, b, c) )
Regular applications

OVM and MPI for 3 of the kernels from the NAS NPB benchmark 2.3: CG, FT, EP

CG: frequent reduction with short messages  
FT: frequent all-to-all with long data  
EP: pure parallel kernel with no communication

Cluster of PentiumPro 200 + Myrinet, Score environnement (RWCP)  
Optimized MPI global operations
Non-uniform Load Application

Parallel version of Aires Auger Project

Simulation of very high energy particle entering the Earth atmosphere

Rely on the load-balancing mechanism
Iso-energy blocs

Speedup

With initialization
Without initialization

<table>
<thead>
<tr>
<th>Iterations</th>
<th>With Initialization</th>
<th>Without Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3.72</td>
<td>3.96</td>
</tr>
<tr>
<td>8</td>
<td>7.2</td>
<td>7.66</td>
</tr>
<tr>
<td>16</td>
<td>15.4</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td>26.9</td>
<td>20.8</td>
</tr>
<tr>
<td>64</td>
<td>56.1</td>
<td>36.8</td>
</tr>
</tbody>
</table>
Another Irregular Application

Real-time ray tracing with OVM and Povray
Parallel algorithm

- Global distribution of the scene
- Dynamic work distribution
- Compute each part of the final image and put them together to obtain the final image
Radiosity (any light)

Master/Slave approach with the master involved in the computation

Number of Procs

1 2 4 8 16 32 64 128

Speedup

1 10 100

Bohn/Garmann
Singh
Carter
Benavides
OVM
Garmann 2000

2000
2001
1994

Master/Slave approach with the master involved in the computation
Conclusion

- Fit well for large classes of applications
- We can reach the same performances as MPI for some regular applications (even with different ratio communication/computation)
- OVM allow linear speedup with irregular applications (AIRES, raytracing, radiosity) using a Master/Slave approach
- Multi protocols (PM, TCP/IP, BIP), multi architectures: OVM can be used on multiples parallel computers
Fault Tolerance

• How to became a fault-tolerant environment?
• The system keep trace of all requested services for the clients.
• Data replication, not necessary the last version
• Multiple parallel execution for the same service
• Single point of failure: **the broker**

But the most important thing is that the fault tolerance could be achieved at 100% automatically.