Migration of Applications Across Different Systems: A Case Study

Marisa Gil, Edgar Juanpere, Xavier Martorell and Nacho Navarro (BSC, UPC)
Riccardo Rossi and Pooyan Dadvand (CIMNE, UPC)

Outline

- Context
- Overview
- Goals
- Methodology phases
- Migration development
- Evaluation
- Conclusions
Wide spectrum of platforms: variety of components, heterogeneity at different levels (sometimes enforcing changes on code and in algorithm)

- More complex applications, based on different Programming Models and execution paradigms.
- Need of benchmarks to be able to take decisions, compare platforms, improve performance.
Kratos Multi-Physics Framework

- Open source framework for the development of multi-disciplinary solvers
  - Modular composition based on support libraries and solvers.
  - Written in C++ and organized according to object-oriented paradigms.

- Free, open source packages
  - Trilinos (http://trilinos.sandia.gov)
    - supporting algorithms for the solution of large-scale multiphysics engineering.
  - ParMETIS (http://glaros.dtc.umn.edu)
    - implementing problem partitioning
  - Boost (http://www.boost.org/)
    - with various services offered as C++ objects
  - SuperLU_Dist (http://acts.nersc.gov/superlu/)
    - dealing with distributed solving systems of linear equations
  - Blas/Lapack (http://www.netlib.org)
Overview

- Migration process from an Intel cluster to an IBM supercomputer

Goals

- Find the main steps needed in a migration process
- Achieve correct execution in the target environment
  - Varying the number of resources (processors, memory...)
  - Maintaining scalability
- Productivity evaluation
  - Obtain results on installation and execution times
- Improve performance
  - Parameters and flag tuning
  - Adapt programming models or function modules
Methodology phases

Learning → Application Structure → Target platform Structure → Test installation

Application requirements → Migrating → Improving

Library dependencies Graph → Configure environment

Build executable → Run

Analyze executing flow

Test different configurations

Collect and compare data

Target (MareNostrum)

<table>
<thead>
<tr>
<th>Token</th>
<th>Mare Nostrum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microchip</td>
<td>IBM PowerPC 970MP (64 bits, dual core 2’3 GHz)</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB per node</td>
</tr>
<tr>
<td>Hard disk</td>
<td>36 GB per node</td>
</tr>
<tr>
<td>Queuing system</td>
<td>Moab client 5.2.3 and SLURM 2.1.9</td>
</tr>
<tr>
<td>Interconnection network</td>
<td>Myrinet express 1.2.7</td>
</tr>
<tr>
<td>Cores per node</td>
<td>4 (2 xDual core)</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>2560 (BladeCenter JS21)</td>
</tr>
<tr>
<td>Number of processors</td>
<td>10240</td>
</tr>
<tr>
<td>Total memory</td>
<td>20 TB</td>
</tr>
<tr>
<td>Total Hard disk</td>
<td>280 TB (GPFS)</td>
</tr>
<tr>
<td>Operating system</td>
<td>Linux Suse</td>
</tr>
</tbody>
</table>

TRONDHEIM, Dec. 2011
**Kratos structure**

**Mesh**
- Nodes
- Elements
- Properties
- Conditions

**Communicator**
- Local nodes
- Ghost nodes

**ProcessInfo**

**Modellpart**

---

**Kratos execution**

**Pre-processing (serial):**

**Kratos initiates its kernel and imports applications (shared libraries)**

Data input and associated Modellpart is created

**METIS decides node-partitioning and divides input data into as many files as needed to compute in parallel**

---

**Executing (parallel):**

**METIS assigns files partitions to processors**

400 steps

- `Solver.Solve()` Result is stored and printed
- `Solver.Solve()` Result is stored and printed
- `Solver.Solve()` Result is stored and printed
- `Solver.Solve()` Result is stored and printed

As many as processors required

---

**UPC**

TRONDHEIM, Dec. 2011
Migration stages

- Need to know application dependencies to install

Configure environment
Build executable
Run

**Tools and libraries:**
- Python, swig, cmake
- Compilers
- MPI, BLAS/LAPACK, METIS/ParMETIS
- SuperLU_Dist, BOOST/bjam
- NumPy, Trilinos

**Compilation flags**
Installation paths (64 bits needed)
Jamroot file

**Compile and link with libraries (bjam utility)**
Ensure proper include file location
Ensure proper linking of libraries

**Ensure proper loading of tools and libraries:**
Set PATH, LD_LIBRARY_PATH and PYTHONPATH

- Run Application (Python interpreter on script defining problem to solve)
- Interactive runs on small environments
- Batch runs on large machines

**View results using GiD**

---

Results and comparative - Ahmed 25

- Wind tunnel containing a car-shaped object
- The Kratos solver used computes the airflow across the wind tunnel, and around the object
- The experiment has over 1,600,000 elements
Results and comparative - Ahmed 25

- With 512 cores, only 3000 elements per core

Results and comparative - TELESCOPE

- Canarias telescope model
- The Kratos solver used computes the airflow across telescope surroundings
- The experiment has over 24,000,000 elements
Results and comparative - TELESCOPE

> 90% efficiency at 1280 cores

Workload concerns

- Work performed at each step
- Amount of computational work decreases

TRONDHEIM, Dec. 2011
Comparing I/O vs Computation

- Reading the input data under SF threshold
- Computation time decreases

![Graph showing execution time in seconds vs number of cores]

Productivity Evaluation

- Smaller platform after the first Marenostrom Supercomputer: Picasso
  - 256 nodes (2 x single core): 512 processors PowerPC
  - 4Gbytes per node.

- Equivalent hardware components and software tools
- From 9 months to 7 days!!
Improving Performance

- Compilation based
- Auto-installable through a script

```
START
i=1
env54.x1.sh is created

i>1?
NO
i:=i+1
YES

package[i]=Python?
NO
install package[i]
YES

env54.x1.sh is created
```

<table>
<thead>
<tr>
<th>Installation</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation 1: GCC</td>
<td>1 (baseline)</td>
</tr>
<tr>
<td>Installation 2: GCC with native libraries</td>
<td>8-10% (libs 30%)</td>
</tr>
<tr>
<td>Installation 3: GXLC</td>
<td>1</td>
</tr>
<tr>
<td>Installation 4: XLC</td>
<td>2-3%</td>
</tr>
<tr>
<td>Installation 5: Optimized XLC</td>
<td>4-7%</td>
</tr>
</tbody>
</table>

TRONDHEIM, Dec. 2011
Conclusions

- Systematization of migration process following the proposed steps
- Scalability is an application characteristic and must be checked after each change in the environment
  - If application has a good data decomposition and parallelization, it can obtain high efficiency in any architecture
- “Automating” the migration process eases the administration of different installations
  - Several installations can be kept easily at the same time
- ...and also future improvements on software.

Migration of Applications Across Different Systems: A Case Study

Marisa Gil, Edgar Juanpere, Xavier Martorell and Nacho Navarro (BSC, UPC)
Riccardo Rossi and Pooyan Dadvand (CIMNE, UPC)