HPC the easy way: new technologies for high performance application deployment

Marco Danelutto
Dept. Computer Science
University of Pisa – Italy

Contents

- HPC
  - Pressures & urgencies
  - Existing Programming Models & tools, Desiderata
  - New things around, useful features
  - Personal & group experience

HPC layers

- Hardware
- Middleware
- Programming models & tools
- Applications

SMP, Cluster, GRID

Hw virtualisation, resources

Performance & portability

Application deployment

- Applications
- distribute
- maintain
- doc
- help
- RTS
- Progr. model

Pressures & urgencies

- Architectural advances
  - Single processor, Networking, GRID, cluster
- Software advances
  - OO programming models and technologies
  - Networking facilities
- Standards (de jure or de facto)
  - Languages: C, C++, FORTRAN, Java, C# ...
  - OO interoperability: CORBA, COM, JavaBeans
  - Parallel processing: MPI2, OpenMP
  - WEB: HTML, XML, SOAP, WEB services
  - GRID / distributed processing: Condor, Globus

Pressures & urgencies

- Big challenge/killer applications
  - Climate modeling (CPU intensive, data intensive)
  - Bioinformatics (CPU intensive, data intensive)
  - E-something (highly dynamic & distributed)
- (existing) applications scaled
  - Biochemistry (Water to protein)
  - Climate modeling (5-10Km grid (current) to 1 Km grid)
  - SAR (Real time landslide monitoring)
Which pressures?

**Which pressures? (2)**

- **GRID**
  - Builds on metacomputing
  - Heterogeneous collections of machines
  - Virtualized via middleware (TCP/IP, SETI@home, Condor, Globus)
  - Dynamically handled (brokering)
  - Service based middleware

### Table: NOV 1997

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>Share</th>
<th>R_max</th>
<th>Rpeak</th>
<th>Pros</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPP</td>
<td>328</td>
<td>65.6%</td>
<td>13522</td>
<td>19848</td>
<td>59195</td>
</tr>
<tr>
<td>SMP</td>
<td>161</td>
<td>32.2%</td>
<td>2990</td>
<td>3529</td>
<td>4041</td>
</tr>
<tr>
<td>Constellation</td>
<td>10</td>
<td>2%</td>
<td>391</td>
<td>395</td>
<td>580</td>
</tr>
<tr>
<td>Cluster</td>
<td>1</td>
<td>0.2%</td>
<td>10</td>
<td>33</td>
<td>100</td>
</tr>
</tbody>
</table>

### Table: NOV 2002

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>Share</th>
<th>R_max</th>
<th>Rpeak</th>
<th>Pros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constellation</td>
<td>206</td>
<td>41.2%</td>
<td>49458</td>
<td>71506</td>
<td>36708</td>
</tr>
<tr>
<td>MPP</td>
<td>195</td>
<td>39%</td>
<td>120421</td>
<td>210450</td>
<td>114147</td>
</tr>
<tr>
<td>Cluster</td>
<td>93</td>
<td>18.6%</td>
<td>78052</td>
<td>136048</td>
<td>66614</td>
</tr>
<tr>
<td>SMP</td>
<td>6</td>
<td>1.2%</td>
<td>39126</td>
<td>44352</td>
<td>5544</td>
</tr>
<tr>
<td>Total</td>
<td>500</td>
<td>100%</td>
<td>293059</td>
<td>462357</td>
<td>223053</td>
</tr>
</tbody>
</table>

Which pressures? (3)

- Applications
- Programming models & tools
- Middleware
- Hardware

- User level (0)
- Middleware level
- Resource level

Processes, communications, PBS, DB, OS services...

- Applications
- Programming models & tools
- Middleware
- Hardware

- User level (0)
- Middleware level
- Resource level

Security, discovery, information services, monitoring, resource allocation, scheduling, fault tolerance, storage

- Applications
- Programming models & tools
- Middleware
- Hardware

- User level (0)
- Middleware level
- Resource level

Portals, PSE, GRID API, computational workbenches
Which pressures? (4)

- Earth simulation
- Justifies singular, impressive hardware
- CPU and data intensive application
- Still using traditional programming models

Programming models ...?

- Message passing
  - PVM, MPI, Nexus, MPI-G
- Shared memory
  - OpenMP
- (data) Parallel languages
  - HPF

... & tools

- Toolset inherited from the seq world
  - gcc, gdb, gprof, ...
- Specific tools and toolsets
  - MPICH tools (MPE & UpShot)
  - HPF tools
- then?
Desiderata

- New programming models
  - Clear semantics, expressive power, completeness, software reuse, interoperability, portability, performance, performance portability, nice user interface, open source
- New tools
  - Development, deployment, documentation, maintainance

if available ...

- Shorter design to deploy time
- "write once, run everywhere"
- Less debugging/tuning required
- Interoperate with other HPC sw
- HPC programming in-the-large

Desiderata (Runtime System)

- Integral part of HPC models & tools
- Exploit known techniques in HPC frameworks
- Layered implementation
- Incapsulate standard tools
- Support dynamicity, heterogeneity

 RTS

Programming models & tools
Compiling tools
Intermediate languages
Abstract machine: services and mechanism tailored to
Services: naming, discovery, scheduling, storage, ...
Mechanisms: communication – DSM – thread/processes
Hardware

Services: naming, discovery, scheduling, storage, ...
Mechanisms: communication – DSM – thread/processes
Hardware

Programming models & tools
Compiling tools
Intermediate languages
Abstract machine: services and mechanism tailored to
Services: naming, discovery, ...
Mechanisms: communication
Hardware

Scripting
The success stories ...

- portable
- clear semantics
- P2p & collective operations
  one-way & parallel I/O
  completeness (not SPMD?)
  performance portability (?)
  software reuse (?)
  interoperability (?)

"New" models

- Already on the scene
  - Coordination languages
  - Algorithical skeletons
  - Design patterns
  - Components
  - PSE (frameworks)
- Currently not yet exploited
  - but in (sometimes limited) research frameworks

"New" models (2)

- Overcome traditional problems
  - Higher level programming model
    - Automatic handling of cumbersome features
  - Allow extensive software reuse
    - Sequential portions of code
  - Adhere to interoperability standards
    - Provide & use "standard" services
  - Guarantee performance, portability & performance portability
    - Complex (effective) compiler/RTS

Coordination

- Data (Linda) or control (Manifold) oriented
- Separate computation from coordination
  - endogenous
    - primitives within the coordinated code
  - or exogenous
    - primitives outside the coordinated code
Coordination: Linda

- Shared tuple space
- Pattern matching operations on tuples
- API (endogenous)
- Parallel/concurrent aspects → OS, ...
- Recently revisited in standard Java: JavaSpaces

Coordination: Manifold

Algorithmtical skeletons

- Structured parallelism exploitation
- Small number of parallelism exploitation constructs/patterns/library entries
- Sequential computation with standard languages/tools
- Data + control parallelism coexist
- Three tier structure: control par → data par → sequential

Skelettons

- Efficient, reusable, parallelism exploitation patterns
Skeletons

Problem

Compiler

RTS

Skeletons

Skeletons : P3L/SkIE

seq S1 in( ) out(t_a a) $C( ) end seq

seq S2 in(t_a a) out(t_b b) $F77( ) end seq

farm aFarm in(t_a a) out(t_b b)

a2(a,b)

end farm

seq S3 in(t_b b) out() $C++( ) end seq

pipe main in( ) out( )

S1 in( ) out(t_a a)

aFarm in(x) out(t_b y)

S3 in(y) out( )

end pipe

Skeletons : Lithium

- Control & data parallel skeletons
- Macro data flow execution model
- Optimization rules
- Full Java (RMI)

Skeletons : Lithium

- Control & data parallel skeletons
- Macro data flow execution model
- Optimization rules
- Full Java (RMI)
Skeletons : Lithium

- Control & data parallel skeletons
- Macro data flow execution model
- Optimization rules
- Full Java (RMI)

Stream parallel skeleton tree
(farm, pipelines & seqs)

Skeletons : Lithium

- Control & data parallel skeletons
- Macro data flow execution model
- Optimization rules
- Full Java (RMI)

Normal form skeleton tree

\[ T_s(\Delta_{id}) \leq T_s(\Delta) \]

Skeletons : Lithium

- Control & data parallel skeletons
- Macro data flow execution model
- Optimization rules
- Full Java (RMI)

M. Danilo Euromicro PDPR 2003

Skeletons : Kuchen’s Skelib

- Define application structure
- Setup/terminate library
- Execute parallel code (MPI)

Design patterns

- From OO software engineering
- Patterns of computation
  (intent, motivation, applicability, structure, ..., consequences, example code, implementation)
- Sequential → parallel
- OO techniques vs. languages (debate)
Design patterns: CO₂P₂S
- Correct OO Pattern based Parallel Programming System
- Generate code for Java/SMP
- Layered framework (different levels of intervention)
- Extensible (restricted access)
- Fully exploits design patterns

Components
- Stateless components
- Ports (interfaces to services)
- Building blocks for more complex applications
- LEGO model

Components (CCA)
- Ports
  - Interfaces between components
  - Uses/provides model
- Framework
  - Allows assembly of components into applications
- Direct Connection
  - Maintain performance of local inter-component calls
- Parallelism
  - Framework stays out of the way of parallel components
- Language Interoperability
  - Babel, Scientific Interface Definition Language (SIDL)

Components: Ccaffeine
- GUI & Scripting facility (create, operate on components)
- Use/provides ports
- BABEL guarantees interop.

Components: Java beans
- JavaBean is a reusable software component that is written in the Java™
  - Introspection, Properties, Customization, Events, Persistence, Methods
- Live in standard Java environments
- JEE provides tools for Beans
- High performance?

PSE (frameworks)
- User friendly collections of tools
- Dedicated to a single application field
- GUI or command line
- Allow to solve a set of problems
- Sometimes allow to insert new components
NetSolve
- Numerical algorithms
- Different bindings (FORTRAN, Mathematica)
- GRID enabled services
- Agents mediate services

The good news
- Semantics
  - Clear, formal, parallel semantics (skeletons)
  - Simple compositional semantics for skeletons, components, design patterns and coordination (control oriented)
- Expressive power
  - Orders of magnitude far away MP/RPC/ShMem

The bad news
- Completeness
  - Fixed construct/skeleton/pattern set (not for components and (some) PSEs)
  - No escape if needed
- Interoperability
  - Need to cope with standards (CORBA, …)
  - What if StageA and StageB are implemented with different tools
- User interface
  - Often either cmd line or GUI

Worth inheriting …
- Clear interfaces
  - accessible from different environments
  - providing limited but functional set of abstractions (IDL, ports, channels)
  - HPC → performance !!!
- component call vs. F90 call is 200 μsec vs. 80 μsec
- JIT + compile time techniques

Worth inheriting …
- Compositional semantics
  - Develop components (LEGO ones)
  - separate debugging
  - composition mechanisms
    - data flow (stream), RPC, events
- HPC → Lessons learned since CSP
  - Design, development and debugging made simpler
  - Need mechanisms to implement compositionality (the lowest the level, the higher the performance)
Worth inheriting ...
✓ Structured parallelism exploitation
  ✓ provide common tasks as primitives
  ✓ macro expansion (comm, synch, sched,...)
  ✓ nestable patterns
  ✓ performance models
  ✓ Portability
  ✓ compiler-RTS design
  ● HPC \rightarrow Exploit brick structure to achieve performance:
    ● At compile time : templates, compile policies
    ● At run time : optimization of comms/copies

Worth inheriting ...
✓ Interoperability
  ✓ use services
  ✓ provide services
  ✓ according to common standards
  ● HPC \rightarrow problem is latency!
    ● CORBA latency 10-100 times that of plain TCP/IP
    ● WEB Services/soap \rightarrow RPC over XML over TCP/IP ...

Worth inheriting ...
✓ Expandability
  ✓ Pattern repository
  ✓ MetaInfo
    ✓ XML
    ✓ reflection/introspection
  ● HPC
    ● Compile time techniques (JIT)
    ● Templates/pre-compilation
    ● User classes (allow/deny new patterns)

Worth inheriting ...
✓ Layered implementation
  ✓ Compiler (perform static optimisations and prepare suitable (instances of) object code)
  ✓ abstract machine (runs high level intermediate code)
  ✓ middleware (supports high level mechanisms (services))
  ✓ OS/hw (supports low level mechanism (resources))
  ● Fundamental to guarantee performance, efficiency & protection

Worth inheriting ...
● Upper levels (near source code)
  ● Compile as far as possible
  ● Static optimizations
  ● Libraries/macro expansion
  ● Lower levels (near middleware/hw)
    ● JIT, dynamic linking
    ● Dynamic, discovery
    ● Specialized code to cope with heterogeneous machines

Worth inheriting ...
✓ Software reuse
  ✓ Lots of existing HPC code (libraries)
  ● HPC \rightarrow Costly integration (depends on the RTS provided)
    ● meta-link format (DLL)
      VS.
      wrappers
    ● Data structures !
      ● FORTRAN/C matrices (column/row major)
      ● Pointers !
      ● Object (Serialization)
Worth inheriting …

- Frameworks
  - Provide abstract environments for program development, deployment and production usage
  - Provide suitable user interfaces
  - Support expandability

- HPC →
  - Low level, low latency mechanisms required
  - Pre-compilation (JIT) vs. wrapping

HPC: “escapability”

- I know there is X downside, I want to use it for performance!
- Layered structure of language and RTS
  - Source → Intermediate → Object (Abstract machine)
  - Abstract machine → object code → OS calls
  - Different classes of users

HPC: escapability

My personal experience

- P3L design & implementation (FGCS 91, …)
- Skeleton library & embedding
  - OcamlP3l (ML embedding, ACM ML WS 1998)
  - Skelib (C library, Europar 2000)
  - MPI3ke (MPI library, PDCC 2002)
- Macro data flow implementation model
  (Parco 1999, PPL 2001)
- Design pattern and skeletons (PARCO 2001)
- Pure Java skeleton framework: Lithium
  ([ICCS 2002, FGCS 2003])
- ASSIST design/implementation (ongoing)

Group experience

- Skeleton activity started in 1990: P3L
- (HP Pisa Science Center joint project)
- Industrial version with QSW in 1997: SkIE (PQE2000 project)
- Moving to coordination frameworks: ASSIST (ASI – PQE project 2001-2002)
- Components (FIRB project 2002-2005)

Current experience

- ASSIST (A Software development System based on Integrated Skeleton Technology)
  - Overcome problems evidenced by P3L/SKIE
  - Introduce more flexibility in the programming model
  - Guarantee interoperability
  - Portable RTS
  - GRID version
  - M. Vanneschi Tech Rep Apr 2002 www.df.unipi.it
ASSIST

Based on skeletons
Traditional and innovative
Generic graph
Stream (nondet control)
Software reuse
Interoperability
External objects

PARMOD
C++

xml2IDL

CORBA object + IDL

Call ASSIST program as a CORBA service

ASSIST compiler

assCC

Config

Object code

XML

CORBA Wrapping (IDL)

CLAM

PARMOD

FARM

Call xml2IDL CORBA service

FARM

External objects

A S S I S T  c o m p i l e r

A S S I S T  p a r m o d

... and therefore

• standard patterns as subcases
• stateless skeleton/pattern technology overtaken
• escapes to existing (possibly parallel) libraries
• provides users with handy abstraction of parallel activities

ASSIST parmod

(VPs) may execute all the same code (SPMD) or subsets
may execute different code (MPMD)

logical parallel activities (virtual processors)

may have a topology ...

and share a state

parmod has non det control on input stream,

may deliver results on

multiple output streams...

and may access external objects (services)

ASSIST status

• Designed and developed (version 1.0) in ASI-PQE
• Currently begin extended (5% & FIRB)
  • moving to GRID
  • exploit component technology
• Runs on POSIX clusters/networks
  (using ACE)
  • homogeneous, moving to heterogeneous, now

ASSIST performance
ASSIST performance (2)

Conclusions

- Existing models developed (almost) independently with different goals
- Common features useful for
  - design, implementation & deployment
- The ASSIST proposal

... does it stimulate discussion ???

Thank you for the attention

http://www.di.unipi.it/~marcod
(these slides will be there)