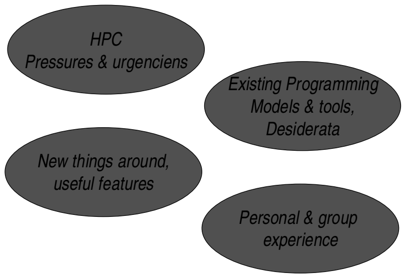
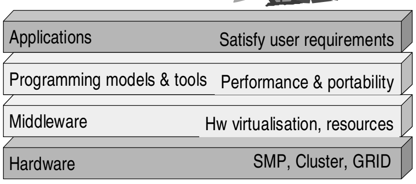
HPC the easy way: new technologies for high performance application deployment

Marco Danelutto
Dept. Computer Science
University of Pisa – Italy

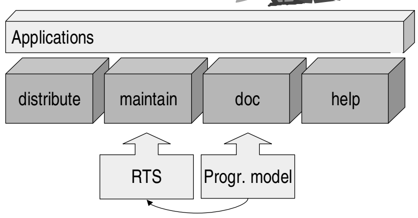
Contents



HPC layers



Application deployment



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 ?

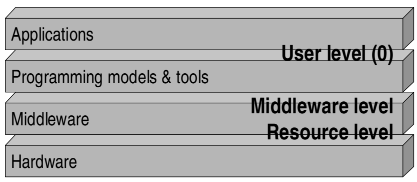
NOV 1997	Count	Share	Rmax	Rpeak	Procs
MPP	328	65.6 %	13522	19848	59195
SMP	161	32.2 %	2990	3529	4041
Constellations	10	2 %	391	595	580
Cluster	1	0.2 %	10	33	100

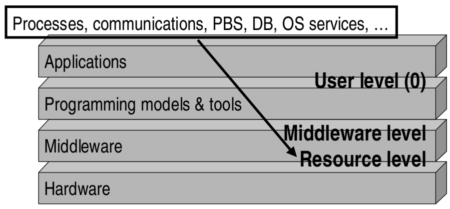
NOV 2002	Count	Share	Rmax	Rpeak	Procs
Constellations	206	41.2 %	49458	71506	36708
MPP	195	39 %	126421	210450	114187
Cluster	93	18.6 %	78052	136048	66614
SMP	6	1.2 %	39126	44352	5544
Total	500	100 %	293058	462357	223053

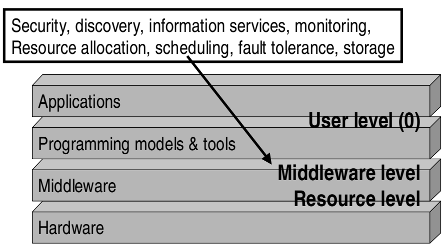
Which pressures ? (2)

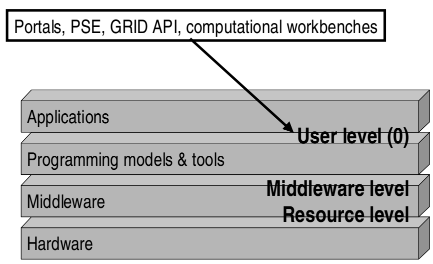
- GRID
  - Builds on metacomputing
  - Heterogeneous collections of machines
  - Virtualized via middleware (TCP/IP, SETI@home, Condor, Globus)
  - Dynamicity handled (brookering)
  - Service based middleware

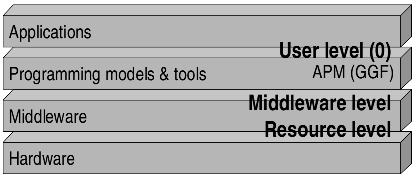
Which pressures ? (3)

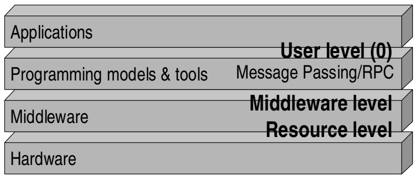






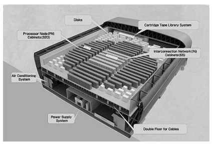






Which pressures ? (4)

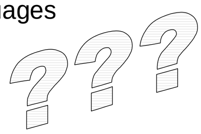
- Earth simulation
- Justifies singular, impressive hardware
- CPU *and* data intensive application
- Still using *traditional* programing models



	Hybrid model	Flat model
Inter-PN	HPF/MPI	HPF/MPI
Intra-PN	Microtasking/ openMP	
AP	Automatic vectorization	

Programming models …

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



Programming models ???

- Message passing/shared memory
- RPC/RMI/…
- ➢ these are *mechanisms !!!*
- OO
- Structured (parallel) programming
- HPF
- ➢ these are *models !!!*

… & 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

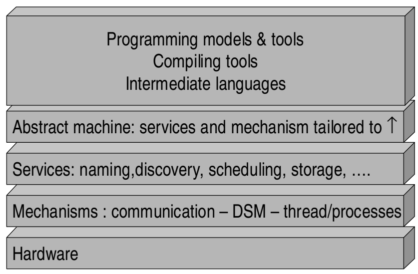
if available …

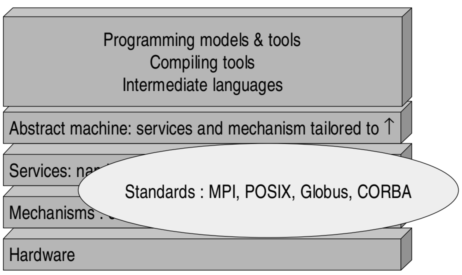
- Shorter design to deploy time
- “*write once, run everywhere*”
- Less debugging/tuning required
- Interoperate with other HPC sw
- HPC programming *in-the-large*

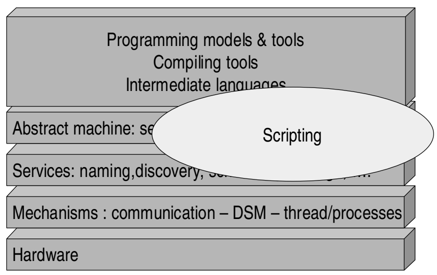
Desiderata (Runtime System)

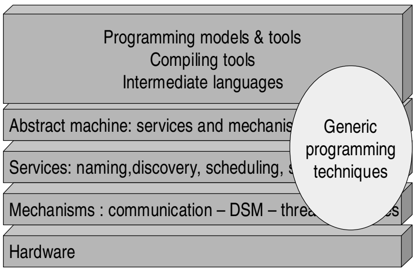
- Integral part of HPC models & tools
- Exploit known techniques in HPC frameworks
- Layered implementation
- Incapsulate standard tools
- Support dynamicity, heterogeneity

RTS









The success stories …

MPI

✓portable

✓clear semantics

✓P2p & collective operations
one-way & parallel I/O

- completeness (non SPMD?)
- performance portability (?)
- software reuse (?)
- interoperability (?)

The success stories … (2)

HPF

✓portable

✓clear semantics
(owner computes rule)

✓performance portability (…)

✓software reuse (Fxx!!!)

- completeness (task parallelism ?)
- software reuse (C++ and the rest ?)
- interoperability (?)

"New" models

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

"New" models (2)

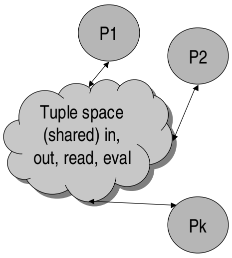
- Overcome traditional problems
  - Higher level programming model
    - Automatic handling of cumbersome features
  - Allow extensive software reuse
    - Sequential portions of code
  - Adhere to interoperability standards
    - Provide & use "standard" services
  - Guarantee performance, portability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

Coordination

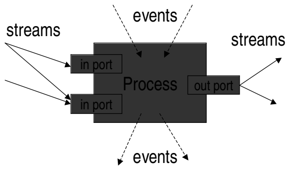
- Data (Linda) or control (Manifold) oriented
- Separate computation from coordination
  - endogenous
    - primitives within the coordinated code
  - or exogenous
    - primitives outside the coordinated code

Coordination : Linda

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



Coordination : Manifold




```
example()
port in input.
port out output.
{
process A is A_type.
process B is B_type.
process C is C_type.
start: (activate A, activate B, activate C): do begin.
begin: (A → B, output → C, input → output).
e1: (B → input, C →A, A → B, B → C, input → output).
e2: C → B.
}
```



```
example()
port in input.
port out output.
{
process A is A_type.
process B is B_type.
process C is C_type.
start: (activate A, activate B, activate C): do begin.
begin: (A → B, output → C, input → output).
e1: (B → input, C →A, A → B, B → C, input → output).
e2: C → B.
}
```

I/O ports

Internal process/components (defined elsewhere)

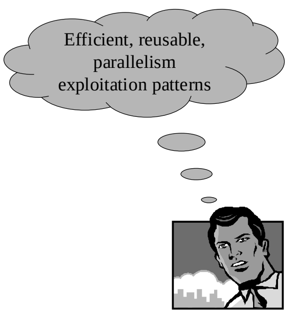
Stream handling

React to events

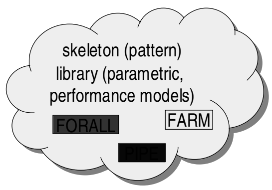
Algorithmical skeletons

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

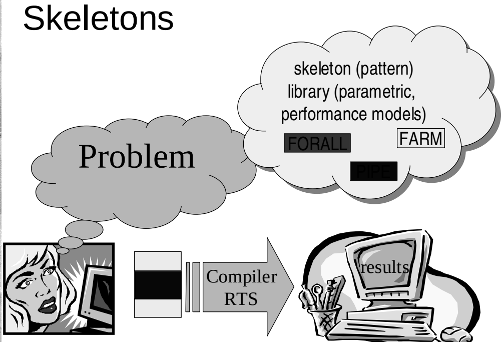
Skeletons



Skeletons

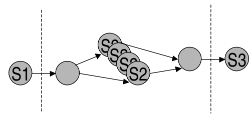


Skeletons



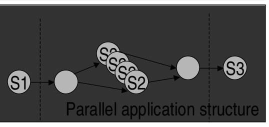
Skeletons : P3L/SkIE

```
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)
  s2(a,b)
end farm
seq S3 in(t_b b) out() $c++{ … } end seq
pipe main in() out()
  S1 in() out(t_a x)
  aFarm in(x) out(t_b y)
  S3 in(y) out()
end pipe
```




```
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)
  s2(a,b)
end farm
seq S3 in(t_b b) out() $c++{ … } end seq
pipe main in() out()
  S1 in() out(t_a x)
  aFarm in(x) out(t_b y)
  S3 in(y) out()
end pipe
```

seq code reuse



Skeletons : Lithium

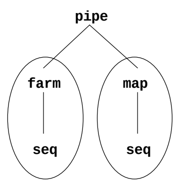
- **Control & data parallel skeletons**
- Macro data flow execution model
- Optimization rules
- Full Java (RMI)

Seq
Pipe
Farm
Map
Reduce
Div&Con
Loop

Skeletons : Lithium

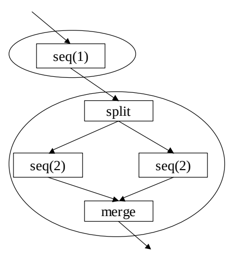
- Control & data parallel skeletons
- **Macro data flow execution model**
- Optimization rules
- Full Java (RMI)

pipe(farm(seq1),map(seq2))



Skeletons : Lithium

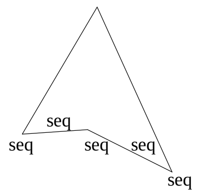
- Control & data parallel skeletons
- **Macro data flow execution model**
- Optimization rules
- Full Java (RMI)



Skeletons : Lithium

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

Stream parallel skeleton tree (farm, pipelines & seqs)

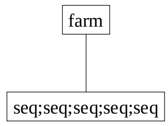


Skeletons : Lithium

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

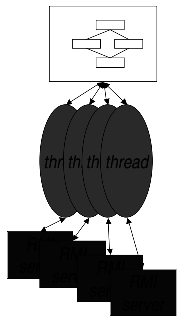
Normal form skeleton tree

$$Ts(\Delta_{nf}) \leq Ts(\Delta)$$



Skeletons : Lithium

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



Skeletons : Kuchen's Skelib

```
template <class I, class O>
inline Process* NestedFarm(Process& worker, int length){
  int nw = (int) (sqrt(length)+0.1);
  Farm<I,O>*  p1 = new Farm<I,O>(worker,nw);
  Farm<I,O>*  p2 = new Farm<I,O>(*p1,nw);
  return p2;}
int main(int argc, char **argv){
  try{
    InitSkeletons(argc,argv);
    Initial<int>     p1(init);
    Atomic<int,int>  p2(square,1);
    Process*         p3 = NestedFarm<int,int>(p2,4);
    Final<int>       p4(fin);
    Pipe             p5(p1,*p3,p4);
    p5.start();
    TerminateSkeletons();}
  catch(Exception&){…}
}
```



```
template <class I, class O>
inline Process* NestedFarm(Process& worker, int length){
  int nw = (int) (sqrt(length)+0.1);
  Farm<I,O>*  p1 = new Farm<I,O>(worker,nw);
  Farm<I,O>*  p2 = new Farm<I,O>(*p1,nw);
  return p2;}
int main(int argc, char **argv){
  try{
    InitSkeletons(argc,argv);
    Initial<int>     p1(init);
    Atomic<int,int>  p2(square,1);
    Process*         p3 = NestedFarm<int,int>(p2,4);
    Final<int>       p4(fin);
    Pipe             p5(p1,*p3,p4);
    p5.start();
    TerminateSkeletons();}
  catch(Exception&){…}
}
```

define a farm of farm of seq

define appl. par. structure

execute parallel code (MPI)

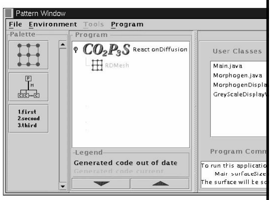
Setup/terminate lib

Design patterns

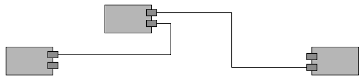
- From OO software engineering
- Patterns of computation (intent, motivation, applicability, structure, …, consequences, example code, implementation)
- Sequential → parallel
- OO techniques vs. languages (debate)

Design patterns : CO_3P_2S

- Correct OO Pattern based Parallel Programming System
- Generate code for Java / SMP
- Layered framework (different levels of intervention)
- Extensibile (restricted access)
- Fully exploits design patterns



Components



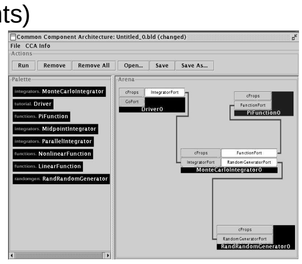
- Stateless components
- Ports (interfaces to services)
- Building blocks for more complex applications
- LEGO model

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

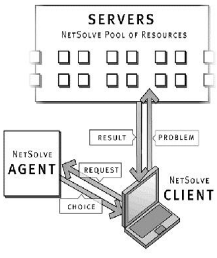


Figure 1: NetSolve's organization

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 good news

- Software reuse
  - C, C++, FORTRAN, Java, …
  - (coordination, skeleton)
- Portability
  - Layered implementation
    (source, intermediate, hw specific RTS)
    (skeleton, design patterns, coordination)
- Performance
  - Clusters, SMP → close to MPI, HPF
    (modulo expressive power)
    (skeleton, coordination, PSE)

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 → 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 → problem is latency !
  - CORBA latency 10-100 times that of plain TCP/IP
  - WEB Services/SOAP → 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 → Costly integration (depends on the RTS provided)
  - meta-link format (DLL)
    vs.
    wrappers
  - Data structures !
    - FORTRAN/C matrixes (column/row major)
    - Pointers !
    - Object (Serialization)

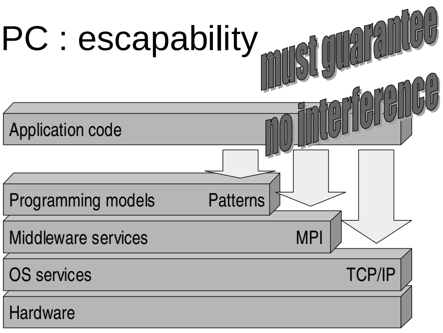
Worth inheriting …

- ✓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



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)

My personal experience

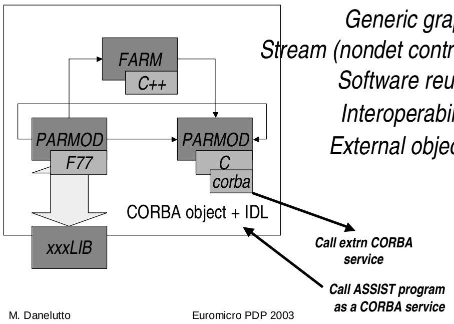
- P3L design & implementation (FGCS 91, …)
- Skeleton library & embedding
  - OcamlP3l (ML embedding, ACM ML WS 1998)
  - Skelib (C library, Europar 2000)
  - MPISke (MPI library, PDCS 2002)
- Macro data flow implementation model (Parco 1999, PPL 2001)
- Design pattern and skeletons (PARCO 2001)
- Pure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

Current experience

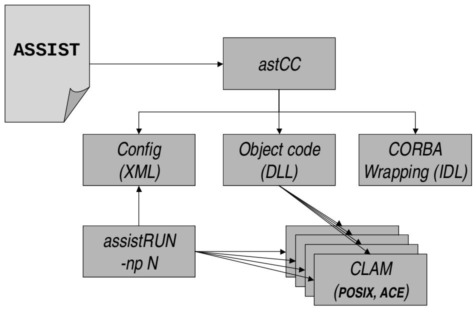
- ASSIST (A Software development System based on Integrated Skeleton Technology)
  - Overcome problems evidenciated by P3L/SkIE
  - Introduce more flexibility in the programming model
  - Guarantee interoperability
  - Portable RTS
  - GRID version
  - ***[M. Vanneschi Tech Rep Ago2002 www.di.unipi.it]***

ASSIST

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



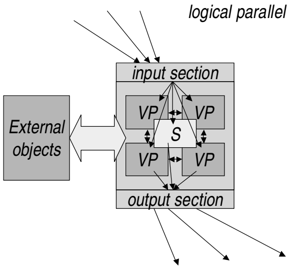
ASSIST compiler



ASSIST parmod

*(VPs may execute all the same code (**SPMD**) or subsets may exec 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)



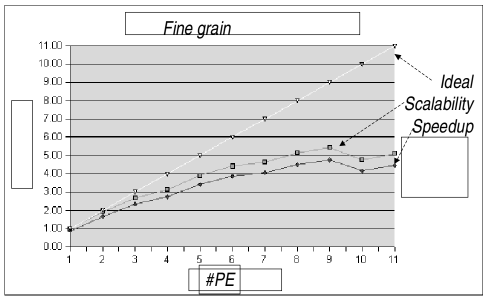
… and therefore

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

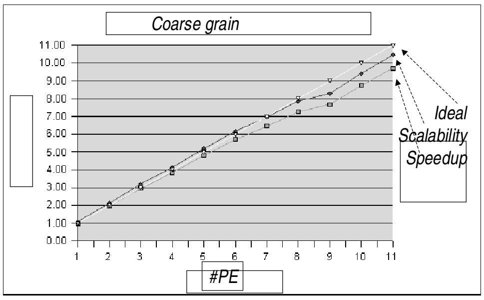
ASSIST status

- 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)