Computing with grids

From HPC to Grid Computing

Philippe d'Anfray CEA ANR-CI

Philippe.d-Anfray@cea.fr

Université Paris 13, Institut Galilée, MACS 2, November 2007

Contents

1-Numerical Simulation

Numerical simulation is widely use for research purposes BUT is also necessary for industry:

- to shorten development time for new products -cars, aircrafts, pharmacy, biology...;
- to understand (future) behaviour of existing installations -i.e. nuclear plants-;
- to help design when no experimentation is possible -aerospace industry-, and . . . any way experimentation is always expensive! (e.g. car crash experimentations, wind tunnels for airplanes, etc. . .).

furthermore some research areas also are society stakes:

- climate modelling (global change); medecine;
- studies in pollution; biodiversity.

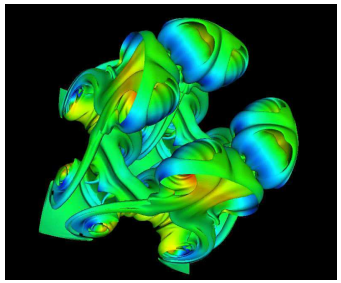
1-Numerical Simulation

Issues in numerical simulation

- modelization of more and more complex phenomena;
- increasing size of problems;
- interactions between mathematical and physical models.

and. . .

- visualization and interpretation of results (graphical applications, virtual reality, . . .).



Multidisciplinarity !

1-Numerical Simulation

What we can actually do in numerical simulation, and the way we do it, is closely related to the developement of both hardware and software technologies.

- in fact . . . mostly hardware;
- and also, as we will see later, something else which is called *middleware*;

We always need more computing power, memory space, storage capacity, etc. . . than what is “locally” available: programming technics are “hardware oriented” to do at best with local resources.

Where are we ?

- **1)** Numerical simulation and scientific computing
- **2)** High Performance Computing
- **3)** Distributed Computing
- **4)** Grid Computing
- **5)** Final remarks

2-High Performance Computing

Hardware technologies:

- PC’s (more and more powerful, sometimes *bi or quadri pro*);
- Workstations (graphic capabilities..);
- Servers (“SMP’s”, shared memory multiprocessors);
- Supercomputers (big “Vector” machines, clusters of SMP’s or massively parallel computers (distributed memory multiprocessors).

"Low cost" supercomputers clusters of PCs -up to several thousands- with high speed interconnection network (Gb ethernet or better).

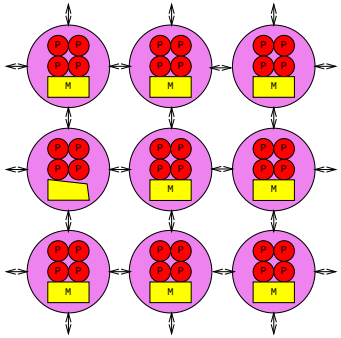
2-High Performance Computing

Widely used because now large numerical simulations can be undertaken on low cost clusters of PC’s.

- distributed memory multiprocessors;
- nodes are often SMP’s, typically two processors;
- run LINUX as OS;
- efficient tools coming from parallel computers (`OpenMP` (directives) or libraries like `PVM`, or -more likely- `MPI`).

2-High Performance Computing

More precisely there is a hierarchical vision of “computing resources”:



where every “level” has its own programming model and we must make the best use of all. Programming models are “hardware or **performance** driven”.

2-High Performance Computing

“node” level

Shared memory MIMD, “threads”;
`Fortran` + directives (`OpenMP`) or library calls.
Library design.

Optimization of nested loops

```
do i=1, M
   do k=1, N
      A(i,k)=
   enddo
enddo
```

external loop: parallel
internal loop: vector (or super scalar)

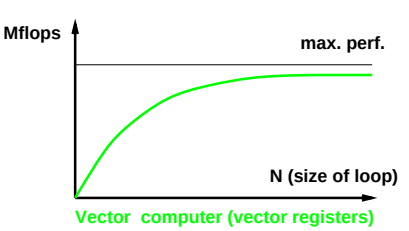
2-High Performance Computing

“processor” level

Vector or superscalar/cache optimization. Efficient at the instruction or loop level.
`Fortran` + directives.
Kernel library design (e.g. `BLAS`).

Performance of sample loop

```
do i=1, N
   x(i)=y(i−+z(i)
enddo
```



2-High Performance Computing

“machine” level

Distributed memory MIMD. Message passing or Distributed Shared Memory (`DSM`);
`Fortran` + library calls (`MPI`, . . .).
Application design.

Programming model: communicating tasks with message passing (for exchanges of information and synchronizations) or emulation of a shared memory space.

For each task, node and processor optimizations.

2-High Performance Computing

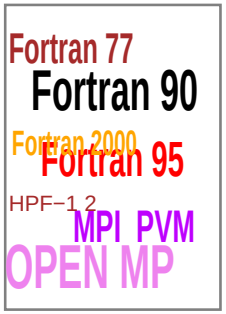
Solution, design with the most general (abstract) model then perform more and more local optimizations:

- task model with **message passing**;
- **SPMD** programs (same code on each node, processing different data);
- communications between nodes are expensive: **locality**;
- communications. . . OK if enought processing: **granularity**;
- a “good” application may be launched efficiently on a large number of nodes: **scalability**.

2-High Performance Computing

Software technologies: programming tools and models are often far behind hardware innovations, and . . .
everyday practise evolves slowly.

- programming language (`Fortran` 77, 90 or whatever);
- programming model most often “procedural”;
- parallel paradigms coded with low level tools (directive, libraries);
- “legacy” numerical codes developped as “stand alone” applications.



2-High Performance Computing

Yes but... for good software techniques ... what about **object oriented techniques**:

- Object Oriented approach is NOT something new !!! (ADT in the 70’s) but does not care *a priori* about performances.

Much progress has been made in the use of “object technologies” (e.g. `C++`). They are found throughout new tools and standards (`Java`, `Corba`, `...`).

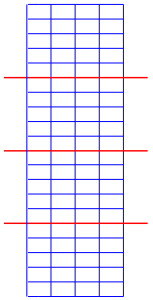
- 00 allows us to describe, browse, access, the large data structures used in numerical simulation codes. (sounds good)...

2-High Performance Computing

HPC (High Performance Computing) applications deal with huge data structures (meshes, matrices,. . .):
Most often, parallelization is driven by decomposition of the data:
Good news!

- the object oriented approach will help to model those decompositions;
- the associated methods will perform **local** computations on the encapsulated data.

This seems to be fine but WaRnInG: from a performance point of view, the key question remains: “where is the data ?” (does not sound good from the OO point of view)...

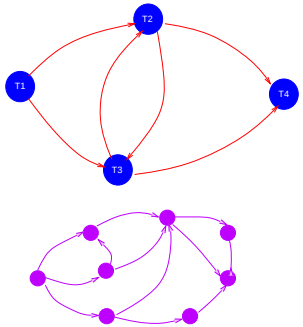


2-High Performance Computing

Which run time model for OO programming in this context ?

Our “usual model”, communicating tasks is the key to “scalability” ?

Data decomposition often imply a SPMD approach and we want to keep this model *versus* a more general “set of interacting objects” model. Why ??



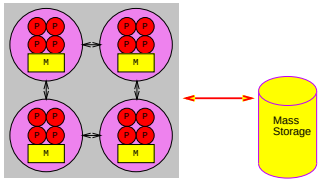
- to control the “granularity”;
- (non) availability of a sophisticated run time environment.

2-High Performance Computing

Solution: use OO but up to a given level, keep it “procedural” (This is fine with C++ or Java).
As a consequence: at the “node” level, user’s objets are local.

data movements between nodes are “above” user’s objects.

- **cloning** and **migration** (rather than message passing or distributed shared memory);
- include notions of IO and persistant objects.



2-High Performance Computing

C++ is the most common object language used so far in numerical simulation.
Many associated tools:

- “basic” libraries, e.g. MPI are ported;
- “stardard” libraries e.g. STL are available;
- “High level” tools e.g. distributed arrays of objects, etc. . . are developped

C++

But:

- can be really intricate (standard document $>$ 500 pages), semantic of some constructs are not clear!
- low performances. . . if one uses the object features (caricatural but somewhat true).

2-High Performance Computing

Java: more than a “cleaned up” C++ (no pointers, operator overloading, multiple inheritance, etc. . .)

- “total” portability (JVM) (*“write once, run everywhere”*); many libraries;
- “web” aspects (applets);
- parallel features (threads defined as classes), Remote Method invocation RMI;

Java™

But:

- slow ! (interpreted language, but better with JIT);
- problems with floating point operations; primitive types *vs* objects; etc. . .

see Jave Grande forum and especially JGNWG Numerics Working Group (somewhat outdated).

2-High Performance Computing

Conclusion, state of the art: Object Oriented Cluster Computing: Clusters are widely available for numerical computing:

- hierarchical (hardware) architectures ;
- hierarchical design (classes, libraries, application)
- object oriented approach but . . . **keep control over data localization**.

`C++` and `Java` are good candidates, they are not 100% OO languages:

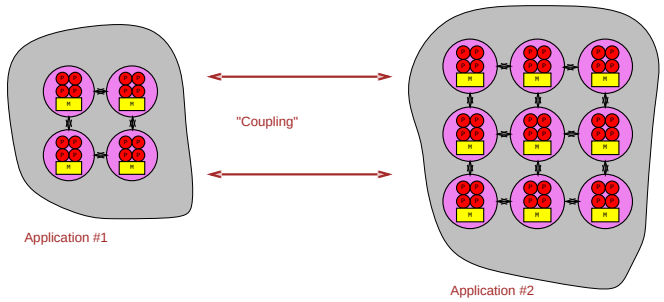
- Object design can be helped with analysis tools (`UML`, . . .);
- “procedural” at the higher level;
- run time model, communicating tasks (most often `SPMD`).

Where are we ?

- **1)** Numerical simulation and scientific computing
- **2)** High Performance Computing
- **3)** Distributed Computing
- **4)** Grid Computing
- **5)** Final remarks

3-Distributed Computing

What if we have several applications running on different platforms ?? (e.g. numerical computations and real time visualization)



“State of the art solution”: coupling features are incorporated inside the applications. and we use an extension of the message passing model *“`meta-MPI`”*. but control is only data driven

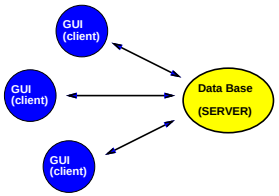
3-Distributed Computing

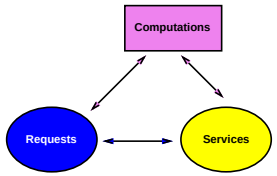
Client/server: a general framework for distributed applications:

- **clients** are programming units which make requests upon providers;
- **servers** are programming units which provide services and resources.

A functionality <=> a specialized piece of software.

Objects are “servers” They are able to perform a given task, i.e. *provide a service* upon reception of a message, i.e. *upon request from a “client”*.

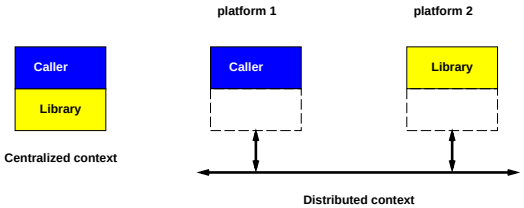




3-Distributed Computing

It is easy to design a client-server architecture from existing software.

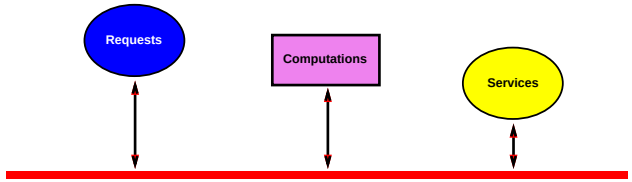
- the “main program” of your application is the client code;
- the routines called in this main program belong to the server side (cf. library).



In a monolithic application everything is on the same platform. In a distributed context, the caller (client side) and the library (server side) may reside on separate platforms.

3-Distributed Computing

Just like hardware components are connected by a bus in a computer, a software bus is a model where applications are seen as components that can be “plugged” on the bus.
All components may interact without modifications of the existing ones:



Of course one can have a “software bus” vision of a “Client-Server” application.

3-Distributed Computing

When Object Oriented Programming is used to design a client-Server architecture, we will use the term

Distributed Objects

In this framework, the run time model is made of objects interacting by exchanging messages.

Those objects may live (i.e. also be born or die) anywhere in a distributed system. This is transparent to the user.

=> The Remote Method Invocation (`RMI`) mechanism introduced in the `Java` programming language allows one to design distributed objects applications.

3-Distributed Computing

A well known tool in this area is `Corba` *“Common Object Request Broker Architecture”*.

It is an attempt to define a standard for programming distributed applications with the three models discussed above.

- Object Oriented
- Client-Server
- Software Bus

`Corba` is defined by the `OMG`, a consortium of users and hardware or software vendors.

3-Distributed Computing

Many applications in numerical simulation fit into this distributed or meta-computing framework:

- coupling of applications (ocean and atmospheric modeling, electro-mechanic, vibro-acoustic, fluid-structure, . . .), multi-scale problems (atomic, particules, continuum);
- cooperative computing;
- interactive visualization;
- and also... videoconferencing, e-learning, video on demand, ...

Where applications are "software Components"

3-Distributed Computing

Warning (as usual):

- performances needed!: keep usual "hardware" models (i.e. keep control over data layout...) and don't forget the hierarchical aspect of distributed architectures;

Very important:

- . . . do not throw away "legacy software";

AND. . .

- a new problem: interoperability of softwares (different implementations of `MPI`, `MPI` and `Corba`, . . .);

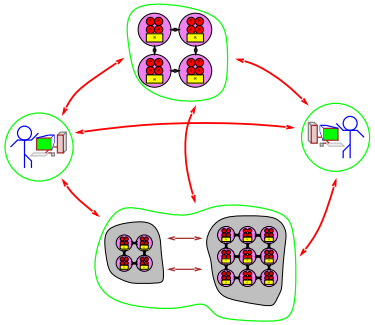
Where are we ?

- **1)** Numerical simulation and scientific computing
- **2)**High Performance Computing
- **3)** Distributed Computing
- **4)** Grid Computing
- **5)** Final remarks

4-Grid Computing

New applications go far beyond the parallel or meta computing approaches and we want to deal with:

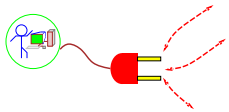
- several users who communicate through the "web";
- multiple "sites" connected through high bandwidth networks;
- a set of applications running simultanously and interacting.
- huge set of data from scientific instruments or results of simulation.



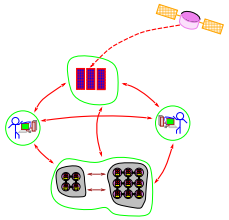
4-Grid Computing

many definitions of the Grid. . .

Analogy with *power grid* (electricity): producers, network, consumers, a “systemic approach” (globalization). The user (consumer) does not see the complexity of the system and potentially accesses the whole available power. Such design issue is the basis of some projects cf. `Globus`.



The **grid** is a “working space” where communities of users (virtual organizations) will achieve common goals by sharing (geographically) distributed resources.



4-Grid Computing

Shared resources might be:

- computing power;
- memory space;
- storage devices;
- networks;
- applications (software);
- information (typically: data bases);
- data sources (sensors, large scientific instruments);
- people!

4-Using Grids

Grids are used to share:

- hardware (computing power)
- software resources (simulation codes)

e.g. in Nuclear energy (national projects), climate modelling, production grid EGEE (EU projects), worldwide projects: HEP, Virtual observatory, etc. ..

and Grids also:

- are used for production and/or acquisition and storage of large amounts of data;
- support (distributed) data retrival and mining tools.

“compute grids”, “data grids”, “community grids” ... same tools !!

4-More on Grids/models

Computer science models:

The **“high performance”** Grid: aggregate computer power, storage elements.

Typically the idea is to build a virtual computing center based on :

- (real) computing centers, départemental resources (clusters);
- typically <100 sites, where:
  -stablility, authentification, security are required

`Globus`, `Unicore`, . . . and many related projects: EU `EGEE`, `Deisa`...

4-More on Grids/models

Computer science models:

“large scale systems”, where a very large number of “standard” and (widely ?) available resources are involved.

Another family of distributed systems in the sense that they imply a large and -not so well defined- community, there might not be a common projects or goals.

- use standard `PC's` (`Linux` ou `Windows`);
- up to 100.000's of nodes, where:
  -volatility, no authentification, no trust. are common “properties”.

P2P computing and/or storage projects (`desktop Grid`, `internet computing` (*seti@home*), `Kaaza`, `Gnutella`, `napster`, `OceanStore`)

4-More on Grids/models

Users' models, the **“community grid”**

Sharing knowledge and know-how. The community has common (long term) goals; may not have common projects -may even compete- (fundamental physics, biomedical and geosciences applications).

- data banks (coming from scientific instruments or résultats de simulations);
- related application and software;
- data retriving and data mining tools

All projects with a grid dimension cf.: Gripps (biologie, fr), HealthGrid (Europe), Virtual Observatory, etc...

4-More on Grids/models

Users' models, the**“entreprise grid”**

- **“rationalization”** (equipments, costs); ease cooperations and communications.

Focus on **services** rather than servers.

Users' models, the **“business grids”**,

access to products and services.

e-business, ASP, outsourcing.

4-More on Grids/models

Common characteristics:

- heterogeneous (sites, hardwares, OS's, programming languages and models, run time model . . . , applications, users) interoperability !!!;
- grids are built piece by piece not just deployed from scratch (**grids grow**).

A new generation of (computing) environments called *“middleware”* takes place between *“hardware”* and *“software”* resources . . . and . . . also *“humanware”*).

A grid platform is first of all a “middleware”. New participants must adopt the underlying model and aggregate to the grid ?

4-More on Grids/applications

Caracterization of applications, **typology + metrics**:

- *“data intensive”*, *“cpu intensive”*, *“IO intensive”*, ***“network intensive”***;
- distributed applications (number of sites);
- dynamicity (load balancing, adaptative application, fault tolerancy);
- synchronicity (coupling or simply MPI);
- **does it scale** ? (bounded by the numerical method, the language, etc...)
- performances on the grid (*benchmark ????*).

4-More on Grids/applications

Evaluate the impact of “gridification” on the applications:

- do programming and run time models fit to the Grid context (sequential, parallel, objet oriented programming, client/server, components, etc. . .);
  - compatibility with other platforms (`Corba`-based, `Java`-based, . . .).
- need to reuse existing (legacy) software but how ??? encapsulation, restructuration, rewriting. . . ;
  - straightforward portability is a myth (grid=heterogeneity).
  - what about libraries;
  - what about licences for proprietary software.

4-More on Grids/usages

Environments (middlewares) should be:

- **powerful** (robust and stable) to ensure management and coordination among a large number of resources (dynamicity, fault tolerancy, etc . . .);
- **simple to use** allowing **“seamless”** and **“transparent”** access to resources;
- **secured** (authentification, protection, confidentiality,. . .).

Caracterization of usages, **typology + metrics ????**;

- identify communities of users . . . and practices inside those communities: deployment, production,
- “social” organization (users, administrators, providers, ...);
- security requirement (grid security *versus* site security), data protection . . . .

4-More on Grids

Some basic users’ needs:

1. information system: available resources;
2. language/tool to describe the distributed application;
3. automatic data transfers, monitoring (how jobs are progressing);
4. integrated environment *versus* toolbox;
5. sécurity, security, security (. . .).

But users need to “adapt” to the grid: change of culture

- develop new applications with new paradigms (components, web services ?) ;
- do not build on top of legacy software (integrate it);
- standardization to share “knowledge” (format, metadata)
- semantic associated with resources.

4-More on Grids

main issues... (as seen from the users' side):

- availability, reliability and performances (up to the investement..);
- trust among "actors";
- deployment and maintenance of the middleware on a large number of nodes;
- synchronicity (*workflow*, etc. . . , coupling);
- fault tolerancy (application, library, language, run time, OS, middleware level(s));
- OS "grid-aware"
- accounting (and billing).

4-More on Grids

optimistic conclusions:

- better services for the users; scientists will do more science;
- rationalization of costs;
- encourage pluridisciplinarity; tighter cooperations among scientific and industrial projects.

There is a need for open operational platforms! (*"download the middleware AND use it to connect to an existing platform"*)

- experimentations cannot be local;
- investigate new usages, organisations, work practices, *business models*.

Contents

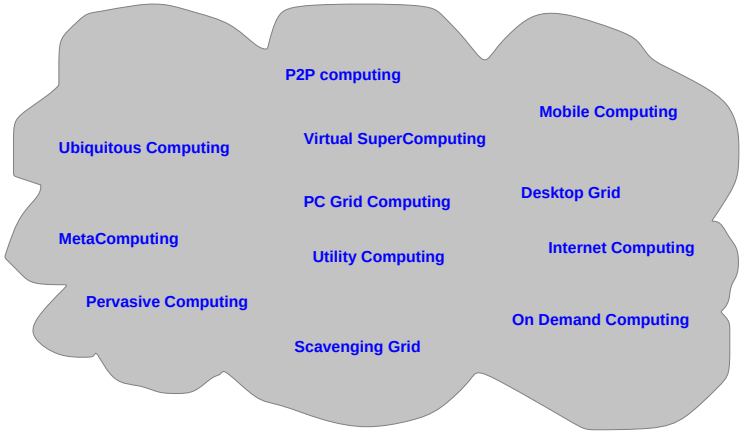
5-Final remarks

Recall: (distributed) modelization is a pluridisciplinary activity:

- physics; coupling of models (fluid-structure interactions, ocean and atmospheric models for climate modelling); multi-scale problems (material sciences); . . . ;
- numerical analysis: new methods (domain decomposition. . .);
- computer science: new tools and standards for distributed modelisation: languages (`Java`), standards (`Corba`), *"middlewares"* and Problem solving environnements (*"PSE's"*).

5-Final remarks

All scientific projects now have some “grid dimension” mainly because they involve communities. Technology trend=global interconnexion of resources, **grid(s) is (are) everywhere**.



(picture taken from Thierry Priol, IRISA, presentation at ORAP Forum Juin 2004)

5-Final remarks

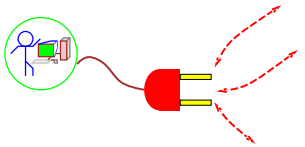
- software tools:
  - **MPI**: `http://www-unix.mcs.anl.gov/mpi`
  - **PVM**: `http://www.csm.ornl.gov/pvm`
- langages and standards:
  - **Java**: `http://www.javagrande.org/`
  - **Corba**: `http://www.corba.org`
- *“batch”* schedulers:
  - **OpenPBS**: `http://www.openpbs.org`
- middlewares:
  - **SGE**: `http://www.gridengine.sunsource.net`
  - **Globus,OGSA**: `http://www.globus.org`
  - **Unicore**: `http://unicore.eu/`
  - **gLite**: `http://glite.web.cern.ch/glite/`

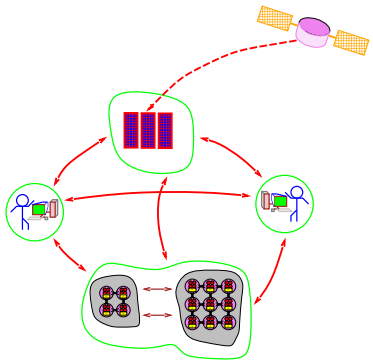
5-Final remarks

- *“Peer to Peer”* or *“Desktop Computing”*:
  - **XtremWeb**: `http://www.xtremweb.net`
- european projects:
  - Enabling Grids for E-Science in Europe **Egee**: `http://www.eu-egee.org`
  - Distributed European Infrastructure for Supercomputing Applications **Deisa**: `http://www.deisa.org`
- who’s doing what ???, the *“Open Grid Forum”*:
  - **OGF**: `http://www.ogf.org`
- everything you need: *“Grid Computing Info Centre”*:
  - **GRID Infoware**: `http://www.gridcomputing.com`

The end

Thank you for your attention!!!





Any question(s) ????