

Programming High Performances Applications using Components



Thierry Priol
PARIS research group
IRISA/INRIA
<http://www.irisa.fr/paris>

A joint work with A. Denis, C. Perez and A. Ribes

Supported by the French ACI GRID initiative

- High-Performance applications and code coupling
- The CORBA Component Model
- CCM in the context of HPC
- GridCCM: Encapsulation of SPMD parallel codes into components
- Runtime support for GridCCM components
- Conclusions & future works
- Some references

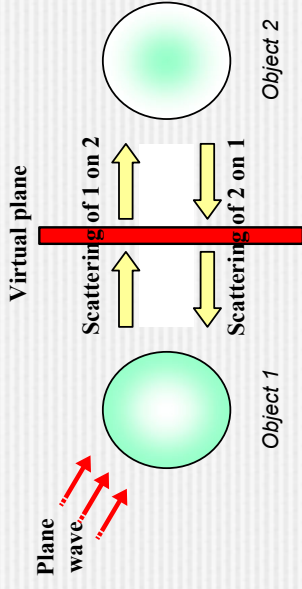
Outline

High Performance applications

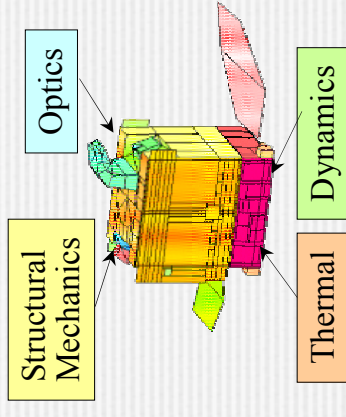
Not anymore a single parallel application but several of them



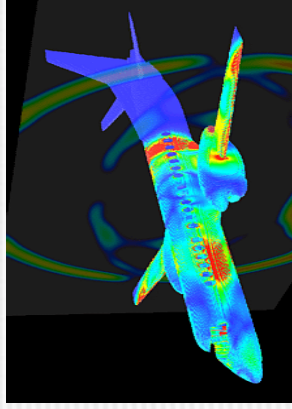
- High performance applications are more and more complex... thanks to the increasing in performance of off-the-shelves hardware
- Several codes coupled together involved in the simulation of complex phenomena
 - Fluid-fluid, fluid-structure, structure-thermo, fluid-acoustic-vibration
- Even more complex if you consider a parameterized study for optimal design
- Some examples
 - e-Science
 - Weather forecast: Sea-Ice-Ocean-Atmosphere-Biosphere
 - Industry
 - Aircraft: CFD-Structural Mechanics, Electromagnetism
 - Satellites: Optics-Thermal-Dynamics-Structural Mechanism



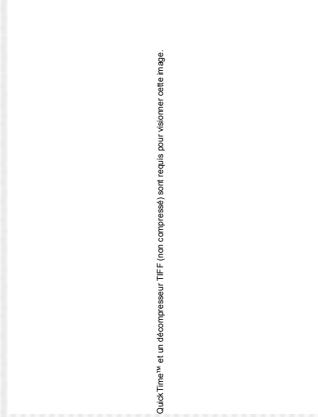
Single-physic multiple-object



Multiple-physic single-object



Electromagnetic coupling



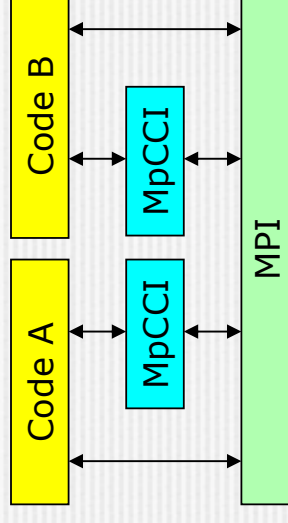
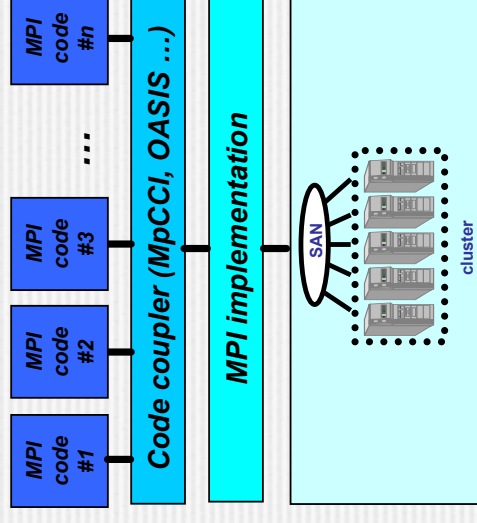
QuickTime™ et un décompresseur TIFF (non compressé) sont requis pour visionner cette image.

Ocean-atmosphere coupling

The current practice...



- Coupling is achieved through the use of specific code coupling tools
 - Not just a matter of communication !
 - Interpolation, time management, ...
 - Examples: MpCCI, OASIS, PAWS, CALCIUM, ISAS, ...
- Limitations of existing code coupling tools
 - Originally targeted to parallel machines with some on-going works to target Grid infrastructure
 - Static coupling (at compile time): not “plug and play”
 - Ad-hoc communication layers (MPI, sockets, shared memory segments, ...)
 - Lack of explicit coupling interfaces
 - Lack of standardization
 - Lack of interoperability

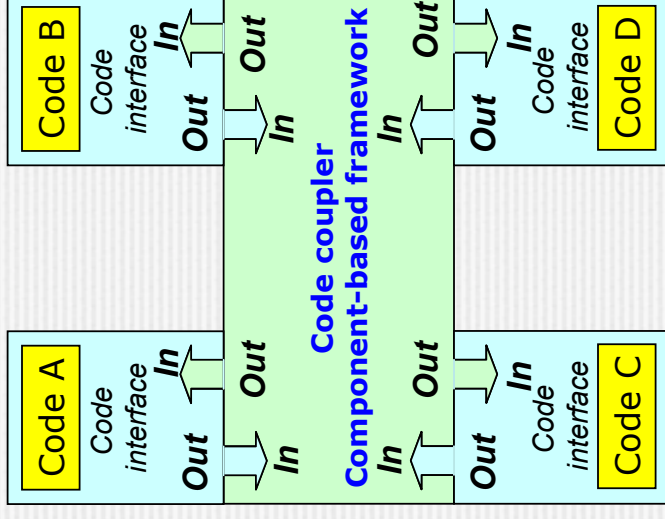
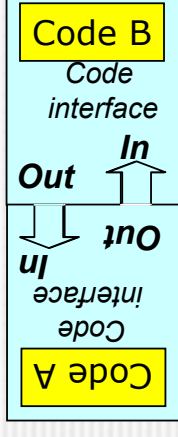


The MpCCI coupling library

Another approach for code coupling



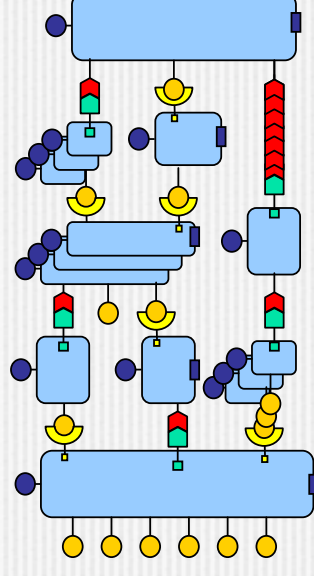
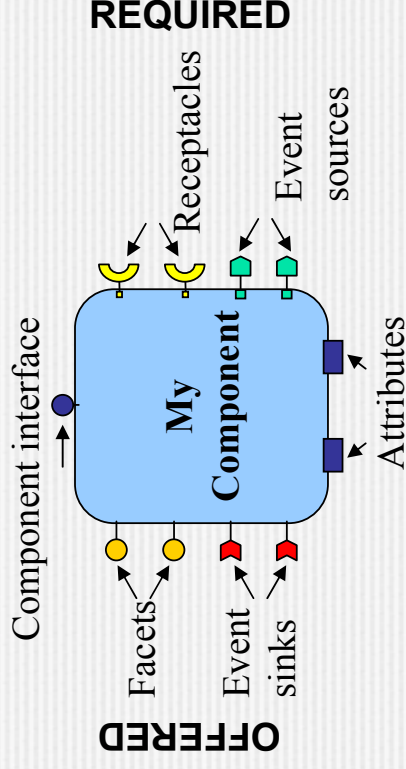
- Component definition by C. Szyperski
 - “A component is a unit of independent deployment”
 - “Well separated from its environment and from other components”
- Component programming is well suited for code coupling
 - Codes are encapsulated into components
 - Components have public interfaces
 - Components can be coupled together or through a framework (code coupler)
 - Components are reusable (with other frameworks)
 - Application design is simpler through composition (but component models are often complex to master !)
- Some examples of component models
 - HPC component models
 - CCA, ICENI
 - Standard component models
 - EJB, DCOM/.NET, OMG CCM



Distributed components: OMG CCM



- A distributed component-oriented model
 - An architecture for defining components and their interaction
 - Interaction implemented through input/output interfaces
 - Synchronous and asynchronous communication
 - A packaging technology for deploying binary multi-lingual executables
 - A runtime environment based on the notion of containers (lifecycle, security, transaction, persistent, events)
 - Multi-languages, multi-OS, multi-ORBs, multi-vendors, ...
- Include a deployment model
 - Could be deployed and run on several distributed nodes simultaneously



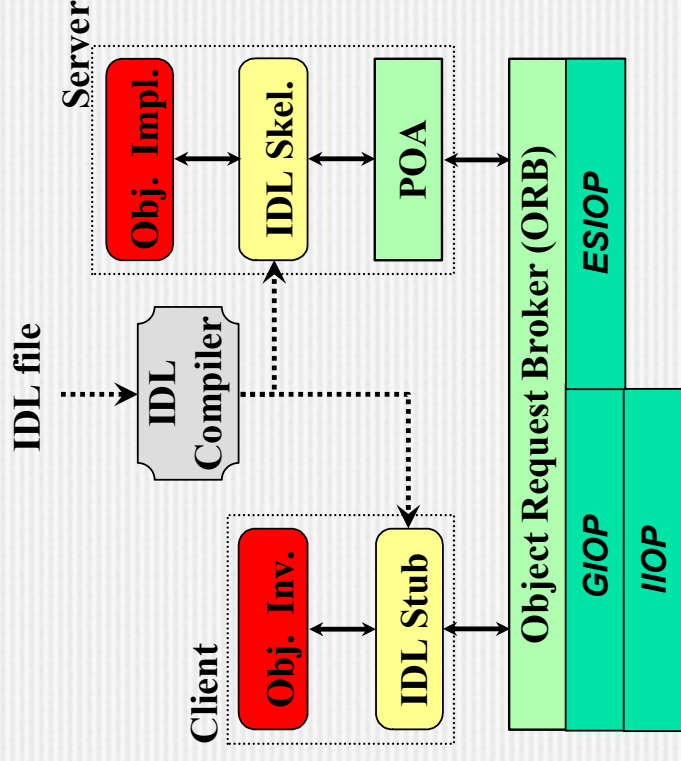
Component-based application

From CORBA 2.x to ...



- Open standard for distributed object computing by the OMG
- Software bus, object oriented
- Remote invocation mechanism
- Hardware, operating system and programming language independence
- Vendor independence (interoperability)

```
interface MatrixOperations {  
    const long SIZE = 100;  
    typedef double Vector[ SIZE ];  
    typedef double Matrix[ SIZE ][ SIZE ];  
    void multiply( in Matrix A, in Vector B,  
                 out Vector C );  
};
```



... CORBA 3.0 and CCM



- Component interface specified by the OMG IDL 3.0

```

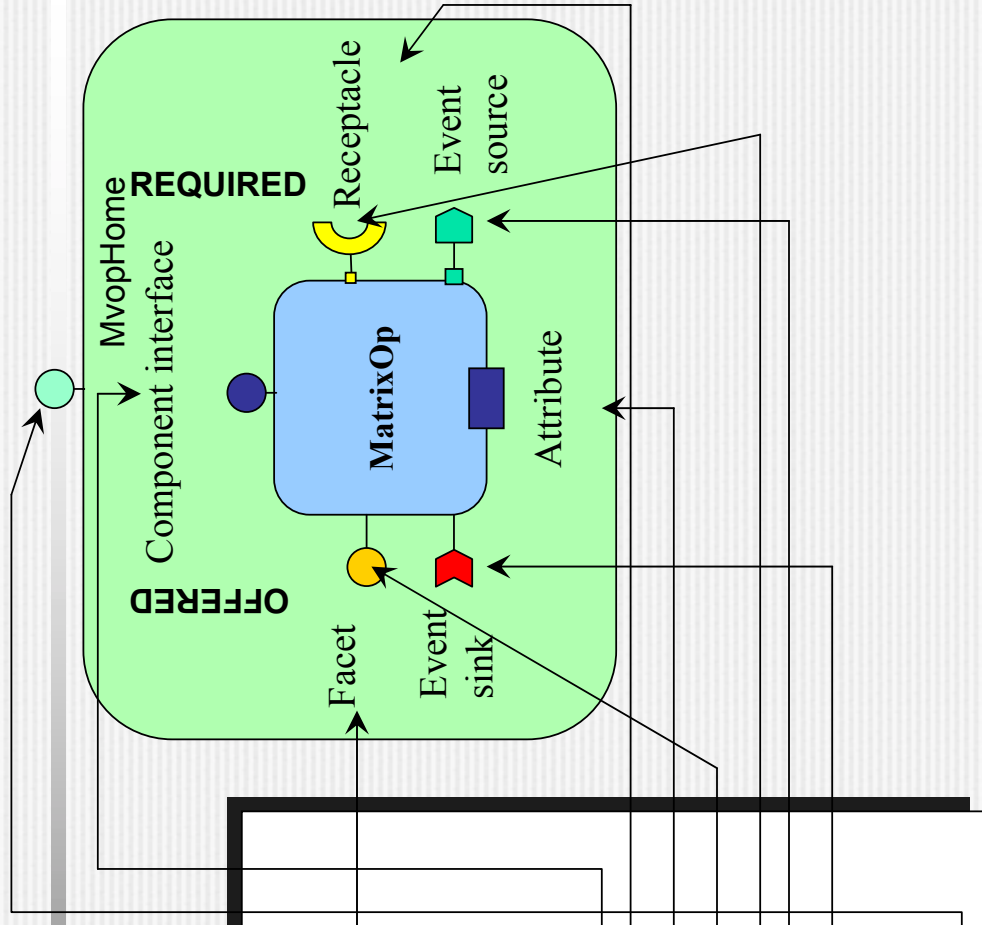
interface Mvop
{
    void mvmult(some parameters);
};
interface Stat
{
    Void sendt(some parameters);
}
component MatrixOp
{
    attribute long size;
    provides Mvop the_mvop;
    uses Stat the_stat;
    publishes StatInfo avail;
    consume MustIgo go;
}

```

```

Home MvopHome manages MatrixOp {};

```

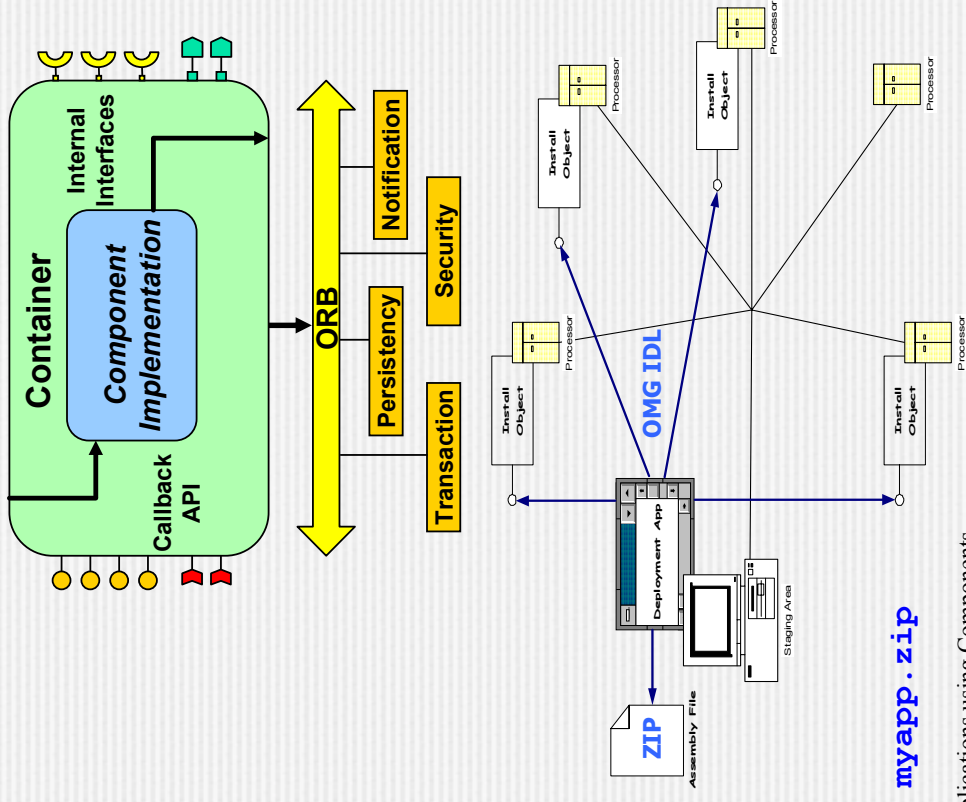


CCM Runtime Environment & Deployment



- CCM includes a runtime environment called Container
 - A container is a component's only outside contact
- CCM defines a Packaging and Deployment model
 - Components are packaged into a self-descriptive package (zip)
 - One or more implementations (multi-binary) + IDL of the component + XML descriptors
 - Packages can be assembled (zip)
 - A set of components + XML descriptor
 - Assemblies can be deployed
 - Through a deployment tool to be defined... such as a Grid middleware...

[\[my_favorite_grid_middleware\]run myapp.zip](#)



CCM in the context of HPC

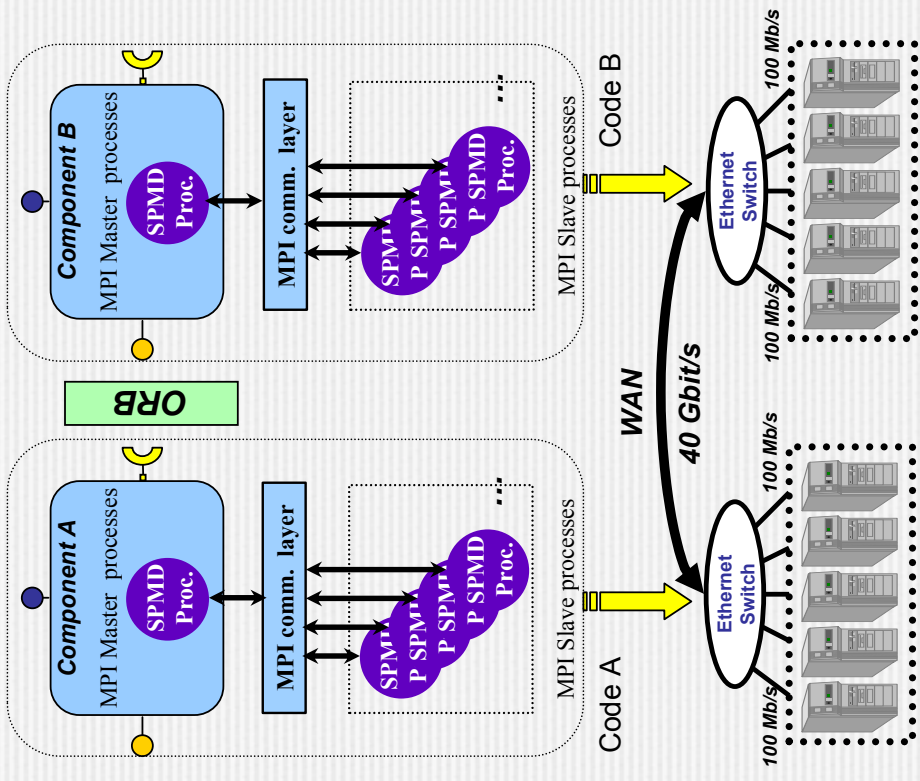


- Encapsulation of parallel codes into software components
 - Parallelism should be hidden from the application designers as much as possible when assembling components
 - Issues:
 - How much my parallel code has to be modified to fit the CCM model
 - What has to be exposed outside the component ?
- Communication between components
 - Components should use the available networking technologies to let component to communicate efficiently
 - Issues:
 - How to combine multiple communication middleware/runtime and to make them to run seamlessly and efficiently ?
 - How to manage different networking technologies transparently ?
 - Can my two components communicate using Myrinet for one run and using Ethernet for another run without any modification, recompilation,..?

Encapsulating SPMD codes into CORBA Components



- The obvious way : adopt a master/Slave approach
 - One SPMD process acts as the master whereas the others act as slaves
 - The master drives the execution of the slaves through message passing
 - Communication between the two SPMD codes go through the master process
- Advantage
 - Could be used with any* CCM implementation
- Drawbacks
 - Lack of scalability when communicating through the ORB
 - Need modifications to the original MPI code



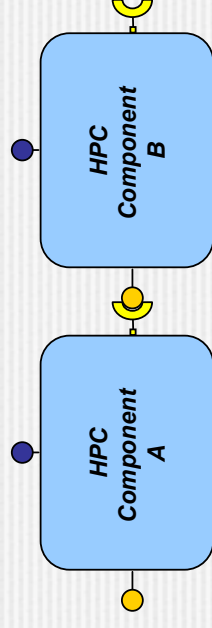
* In theory ... but practically ...

Making CORBA Components parallel-aware

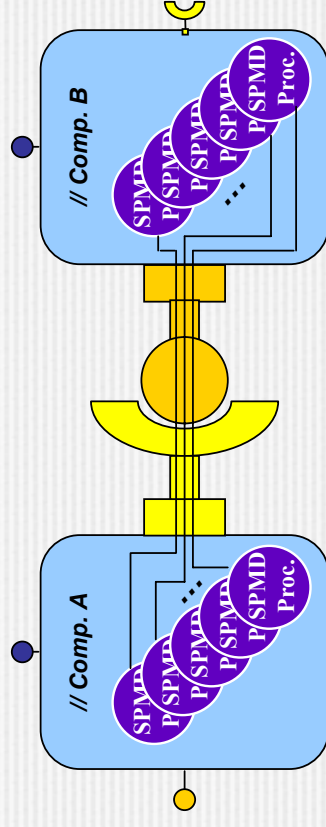


- Just to remind you :
 - Communication between components is implemented through a remote method invocation (to a CORBA object)
- Constraints
 - A parallel component with one SPMD process = a standard component
 - Communication between components should use the ORB to ensure interoperability
 - Little but preferably no modification to the CCM specification
 - Scalable communication between components by having several data flows between components

What the application designer should see...



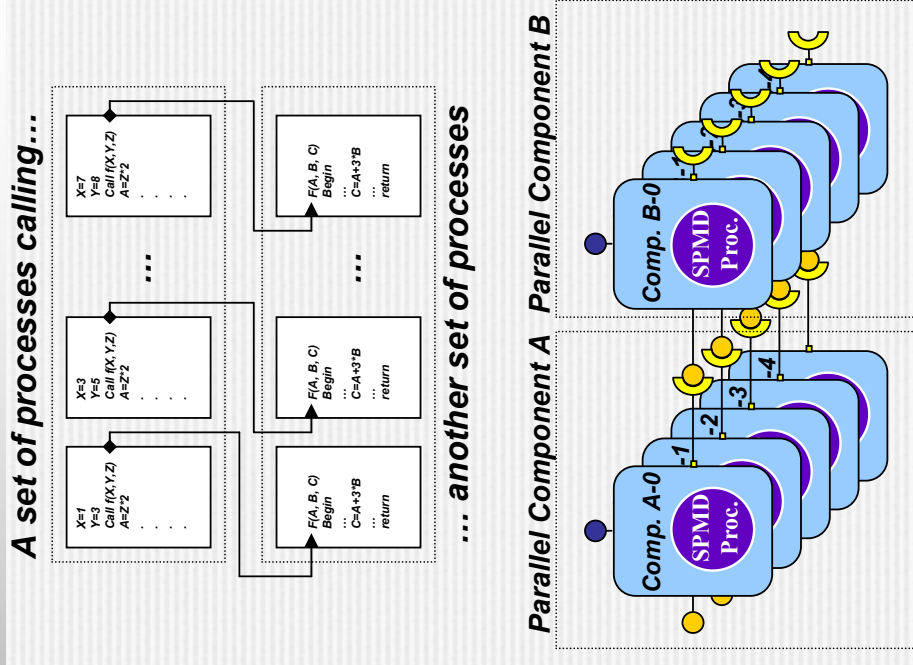
... and how it must be implemented !



Parallel Remote Method Invocation



- Based on the concept of “multi-function” introduced by J-P. Banâtre et al. (1986)
 - A set of processes collectively calls another set of processes through a multi-function (multi-RPC)
- Parallel remote Method invocation
 - Notion of collection of objects (parallel objects)
 - Multi-RMI
- Parallel Component
 - A collection of identical components
 - A collection of parallel interfaces (provide/use)



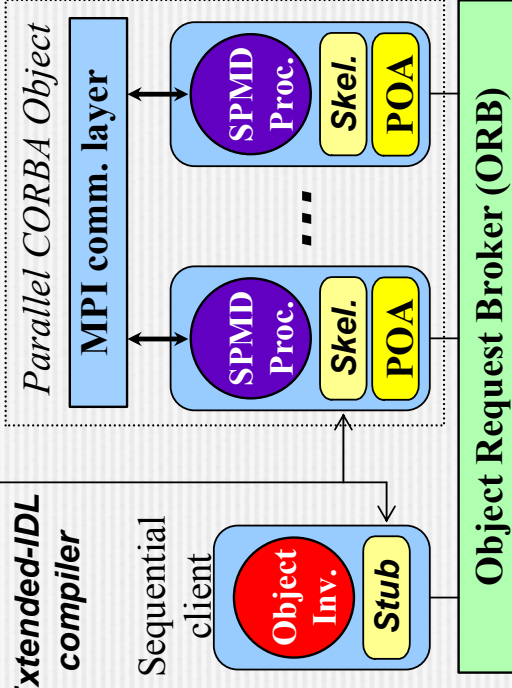
Parallel interface



- A parallel interface is mapped on a collection of identical CORBA objects
 - Invocation to a collection of objects is transparent to the client (either sequential or parallel)
- Collection specification through IDL extensions
 - Size and topology of the collection
 - Data distribution
 - Reduction operations
- Modification to the IDL compiler
 - Stub/skeleton code generation to handle parallel objects
 - New data type for distributed array parameter
 - Extension to CORBA sequence
 - Data distribution information
- Impact on standard:
 - Does not require the modification of the ORB
 - Require extensions to IDL and naming service
 - Is not portable across CORBA implementations

```

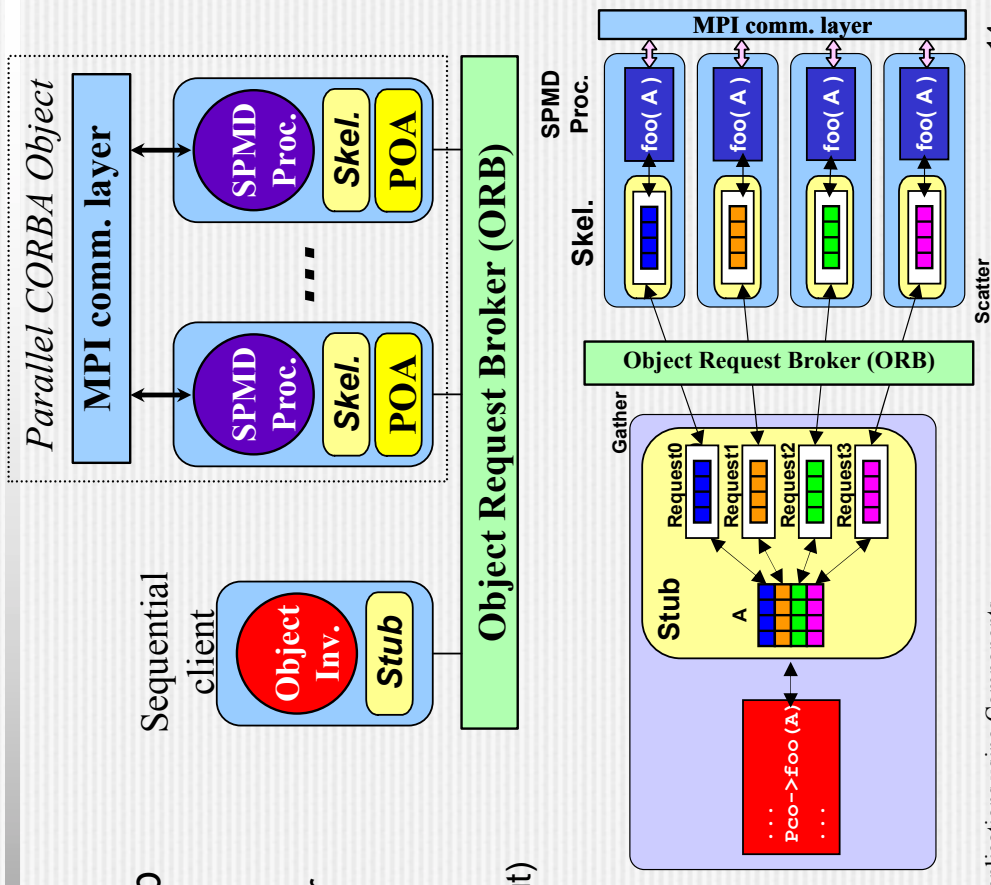
interface [*:2*n] MatrixOperations {
  typedef double Vector[SIZE];
  typedef double Matrix[SIZE][SIZE];
  void multiply(in dist[BLOCK] [*] Matrix A,
               in Vector B,
               out dist[BLOCK] Vector C);
  void skal(in dist[BLOCK] Vector C,
            out csum double skal);
};
    
```



Parallel invocation and data distribution



- Data management is carried out by the stub and the skeleton
- What the stub has to do:
 - generates requests according to the number of objects belonging to the collection
 - According to the data distribution
 - Scatter the data for the input parameters (in)
 - Gather the data for the output parameters (out)

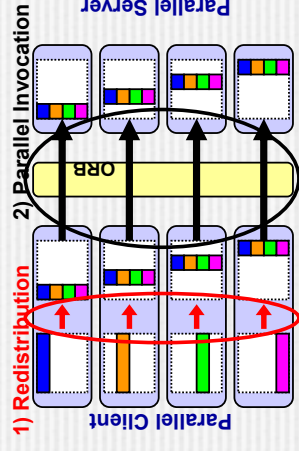
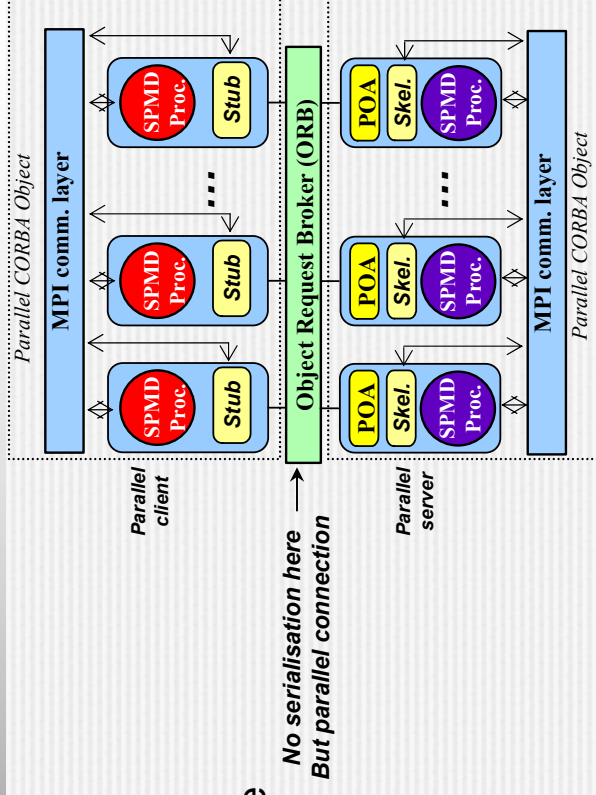


```
interface [*] Operations {
    typedef double Matrix[100][100];
    void foo(inout dist[BLOCK][*] Matrix A);
};
```

Parallel invocation and data distribution



- Different scenarios
 - $M \times 1$: the server is sequential
 - $M \times N$: the server is parallel
- Stubs synchronized themselves to elect those that call the server objects
 - Synchronisation through MPI
- Data redistribution is performed by the stub before calling the parallel CORBA object
 - Data redistribution library using MPI



Lessons learnt...

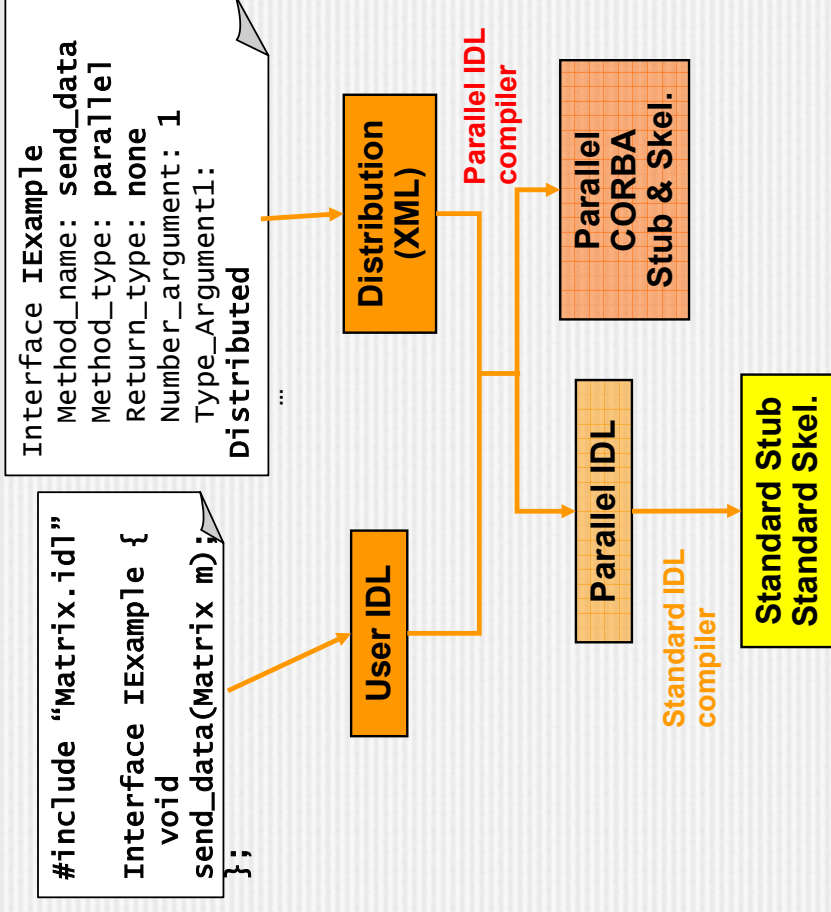


- Implementation dependant
 - Parallel CORBA object only available for MICO
- Modification of the CORBA standard (Extended-IDL)
 - The OMG does not like such approach
 - The end-users too...
- Data distribution specified in the IDL
 - Difficulties to add new data distribution schemes
- Other approaches
 - PARDIS (Indiana University, Prof. D. Gannon)
 - Modification of the IDL language (distributed sequence, futures)
 - Data Parallel CORBA (OMG)
 - Require modifications to the ORB to support data parallel object
 - Data distribution specification included in the client/server code and given to the ORB

Portable parallel CORBA object



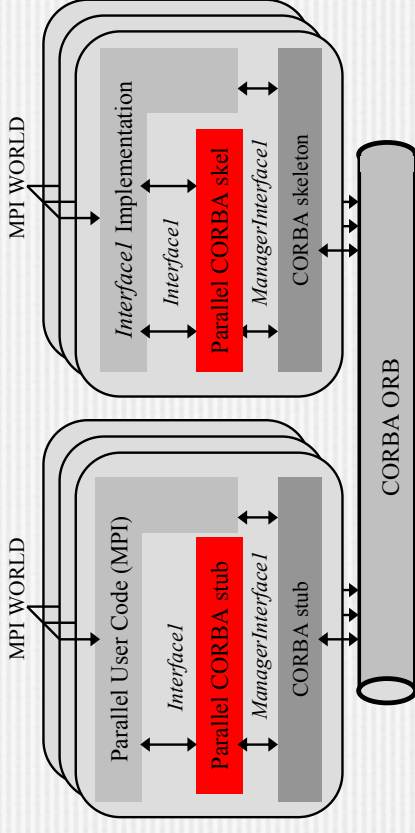
- No modification to the OMG standard
 - Standard IDL
 - Standard ORB
 - Standard naming service
- Parallelism specification through XML
 - Operation: sequential, parallel
 - Argument: replicated, distributed
 - Return argument: reduction or not
- Automatic generation of a new CORBA object (*Parallel Stub and Skeleton*)
 - Renaming
 - Add new IDL interfaces



Portable parallel CORBA object



- Behave like standard CORBA objects
 - `I_var o = I::_narrow(...);`
`o->fun();`
- A parallel CORBA object is:
 - A collection of identical CORBA object
 - A Manager object
- Data distribution managed by the Parallel CORBA stub and skeleton
 - At the client or the server side
 - Through the ORB
- Interoperability with various CORBA implementations



Back to CCM: GridCCM



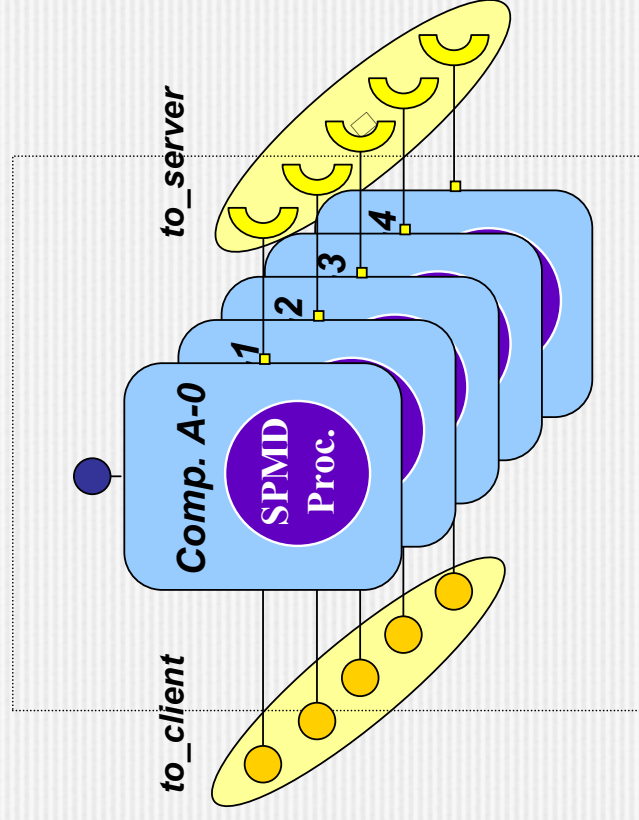
```
interface Interfaces1
{
    void example(in Matrix mat);
};
...
component A
{
    provides Interfaces1 to_client;
    uses Interfaces2 to_server;
};
```

IDL

```
Component: A
Port: to_client
Name: Interfaces1.example
Type: Parallel
Argument1: *, bloc
...
```

XML

Parallel Component Of type A



Runtime support for a grid-aware component model



- Main goal for such a runtime
 - Should support several communication runtime/middleware at the same time
 - Parallel runtime (MPI) & Distributed Middleware (CORBA) such as for GridCCM
- Underlying networking technologies not exposed to the applications
 - Should be independent from the networking interfaces
- Let's take a simple example
 - MPI and CORBA using the same protocol/network (TCP/IP, Ethernet)
 - MPI within a GridCCM component, CORBA between GridCCM components
 - The two libraries are linked together with the component code, does it work ?
 - Extracted from a mailing list:

Message 8368 of 11692 | [Previous](#) | [Next](#) [Up Thread] [Message Index](#)

From: ----- < .-----@n... >

Date: Mon May 27, 2002 10:04 pm

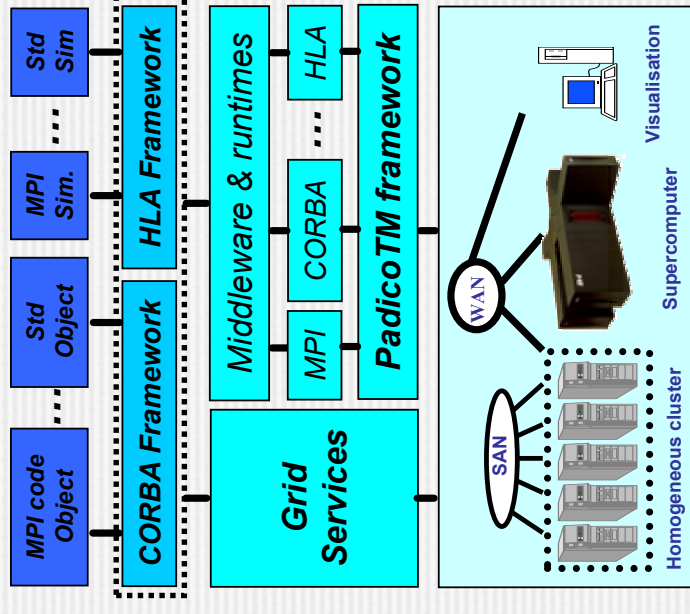
Subject: [mico-devel] mico with mpich

I am trying to run a Mico program in parallel using mpich. When calling CORBA::ORB_init(argc, argv) it seems to caredu.mp. Does anyone have experience in running mico and mpich at the same time?

Padico™: an open integration framework



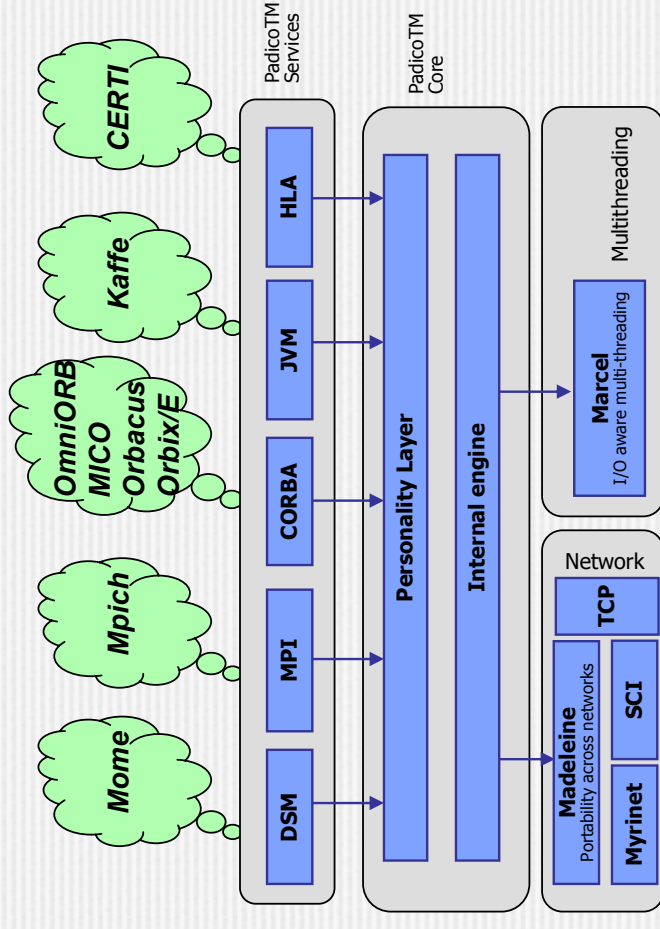
- Provide a framework to combine various communication middleware and runtimes
 - For parallel programming:
 - Message based runtimes (MPI, PVM, ...)
 - DSM-based runtimes (TreadMarks, ...)
 - For distributed programming
 - RPC/RMI based middleware (DCE, CORBA, Java)
 - Middleware for discrete-event based simulation (HLA)
- To handle a large number of networking interfaces
 - Avoid the NxM syndrome !
- Transparency *vis à vis* the applications
- Get the maximum performance from the network!
 - Offer zero-copy mechanism to middleware/runtime
- What are the difficulties ?
 - Provide a generic framework for parallel-oriented runtime (SPMD-based + fixed topology) and distributed-oriented middleware (MPMD-based + dynamic topology)
 - Multiplexing the networking interface
 - A single runtime/middleware use several networking interfaces at the same time
 - Multiple runtime/middleware use simultaneously a single networking interface



Padico™ architecture overview



- Provide a set of personalities to make easy the porting of existing middleware
 - BSD Socket, AIO, Fast Message, Madeleine, ...
- The Internal engine controls the access to the network and scheduling of threads
 - Arbitration, multiplexing, selection, ...
- Low level communication layer based on Madeleine
 - Available on a large number of networking technologies
 - Associated with Marcel (multithreading)

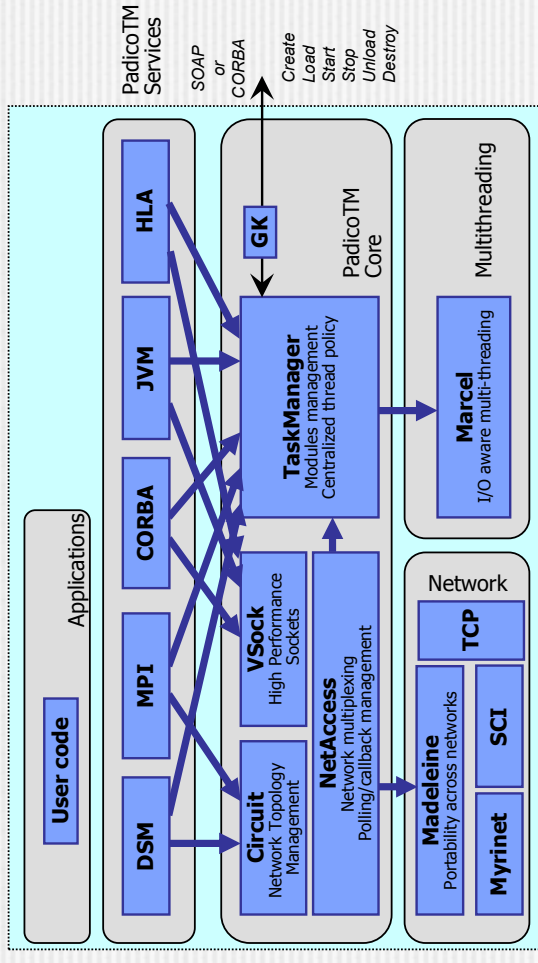


Padico™ implementation



SPMD process on each participating node

- Implemented as a single process
 - To gain access to networks that cannot be shared
 - Launched simultaneously on each participating node
- Middleware and user's applications are loaded dynamically at runtime
 - Appear as modules
 - Modules can be binary shared objects (" .so " libraries on Unix system) or Java classes



```

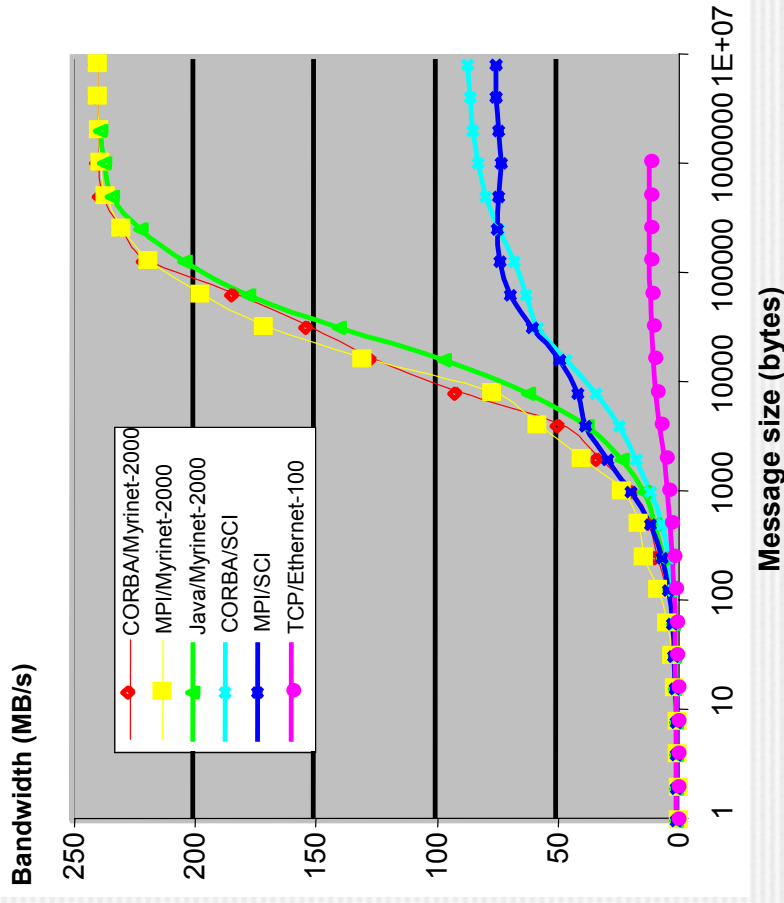
<defmod name="CORBA-omniORB-4.0" driver="binary">
  <attr label="NameService"> corbaloc::paraski.irisa.fr:8088/NameService </attr>
  <requires>SysW</requires>
  <requires>VIO</requires>
  <requires>LibStdC++</requires>
  <unit>CORBA-omniORB-4.0.so</unit>
  <file>lib/libomniorb-4.A.so</file>
  <file>lib/libomniThread4.A.so</file>
</defm=od>
    
```

**XML
Module
Descriptor**

Performances



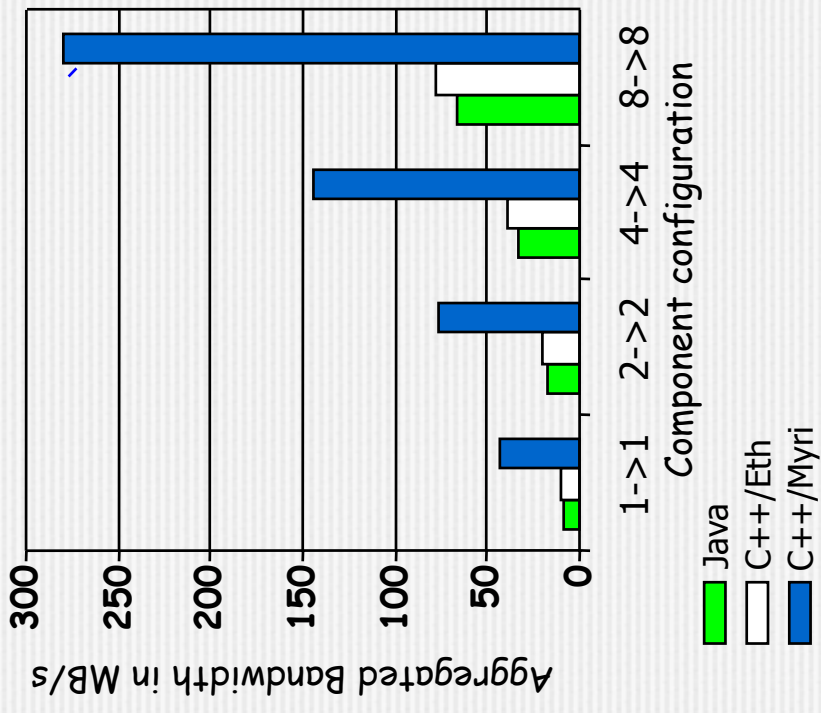
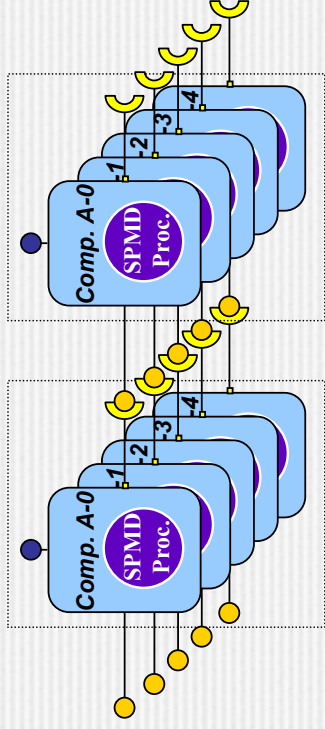
- Experimental protocols
 - Runtimes and middleware:
 - MPICH, CORBA OmniORB3, Kaffe JVM
 - Hardware: Dual-Pentium III 1 Ghz with Myrinet 2000, Dolphin SCI
- Performance results
 - Myrinet-2000
 - MPI: 240 MB/s, 11 μ s
 - CORBA: 240 MB/s, ~~20~~ 18 μ s
 - 96 % of the maximum achievable bandwidth of Madeleine
 - SCI
 - MPI: 75 MB/s, 23 μ s
 - CORBA: 89 MB/s, 55 μ s



Preliminary GridCCM performance



- Composition scalability
 - GridCCM code hand written
 - 1D block distribution
- Experimental Protocol
 - Platform
 - 16 PIII 1 Ghz, Linux 2.2
 - Fast-Ethernet network + Myrinet-2000 network
 - GridCCM implementation
 - JAVA: OpenCCM
 - C++: MicoCCM
- C++/Myri based on MicoCCM/PadicoTM



Conclusion & Future works



- **Conclusions**
 - Code coupling tools can benefit from component models
 - Provide a standard to let HPC codes to be reused and shared in various contexts
 - An existing component model such as CCM can be extended to comply with HPC requirements without modifying the standard
 - We do not need to reinvent the wheel...
 - CORBA is a mature technology with several open source implementations
 - Should be careful when saying that CORBA is not suitable for HPC
 - It is a matter of implementation ! (zero-copy ORB exists such as OmniORB)
 - It is a matter of a suitable runtime to support HPC networks (PadicoTM)
 - Yes, CORBA is a bit complex but do not expect to have a simple solution to program distributed systems...
- **Future works**
 - Deployment of components using Grid middleware & dynamic composition

Some references



- The Padico project
 - <http://www.irisa.fr/paris/Padico>
- One of the best tutorial on CCM
 - <http://www.omg.org/cgi-bin/doc?ccm/2002-04-01>
- A web site with a lot of useful information about CCM
 - <http://ditec.um.es/~dsevilla/ccm/>