Advances in Parallel Computing for Computational Mechanics. MPI for Python and the Domain Decomposition Method

by Mario Storti, Norberto Nigro, Lisandro Dalcín and Rodrigo Paz

Centro Internacional de Métodos Numéricos en Ingeniería - CIMEC

INTEC, (CONICET-UNL), Santa Fe, Argentina

`<{mstorti,nnigro,dalcinl, rodrigop} @intec.unl.edu.ar>`

`http://www.cimec.org.ar/mstorti`

`(document-version "jaiio-conf-0.0.2") (document-date "2005/09/02 02:52:42 UTC")`

Outline of presentation

- **During last years, CFD group at CIMEC (Centro Internacional de Métodos Computacionales en Ingeniería, Santa Fe, INTEC-CONICET-UNL), develops a multi-physics, parallel, open-source CFD program called PETSc-FEM (`http://www.cimec.org.ar/petscfem`). We will present here an introduction to PETSc-FEM, its philosophy and recent developments.**
- **Planned future strategy involves inclusion of an scripting extension language as Python or Guile/Scheme. Current work involves extension basic libraries as MPI or PETSc in those scripting languages. We will present here *“MPI four Python”*.**
- **The *“Domain Decomposition Method”* is a scalable algorithm for the parallel solution of large linear system problems. The Interface Strip Preconditioner for DDM is presented. It improves convergence specially for badly conditioned problems.**

Introduction to PETSc-FEM

During 1996 approx. CFD we decided a major rewrite of our FEM code. This rewrite would include the following main features

- **Parallel computing.**
- **Object Oriented Programming.**
- **Multi-physics.**
- **Scripting via extension languages (this feature added later, work is in progress).**

The new was called *"PETScc-FEM"* and has now 7 years of development by a team of 6 people. It has 140K lines of code.

Parallel computing in PETSc-FEM

- **Experiences with parallel computing in CFD in CIMEC started in 1995 with a test code using MPI (Message Passing Interface), with two Digital-Alpha processors and Fast Ethernet (100 Mb/s). The program solved the compressible Euler equations with an explicit scheme.**
- **PETSc-FEM uses PETSc/MPI in C++. PETSc is the Portable, Extensible Toolkit for Scientific Computation (PETSc), a set of libraries and data structures to solve PDE's system of equations on HPC computers. PETSc was developed at the Argonne National Laboratory (ANL) by Satish Balay, William Gropp and others. `http://www.mcs.anl.gov/petsc/`. PETSc, while it is written in C, uses the OOP (Object Oriented Programming paradigm). Uses MPI for the low-level parallel communications.**

Parallel computing in PETSc-FEM (cont.)

- **PETSc-FEM implements the core Finite Element library in charge of assembly element matrices and residuals in parallel. This matrices and residuals are PETSc objects, and systems are solved with the PETSc library.**
- **Also implemented a new Domain Decomposition Method solver and a preconditioner (to be described later in this talk).**
- **PETSc-FEM runs on cluster of PC's using MPI (Beowulf clusters).**

Object Oriented Programming in PETSc-FEM

- **OOP is a programming paradigm that focus on the idea behind object-oriented programming is that a computer program is composed of a collection of individual units, or objects, as opposed to a traditional view in which a program is a list of instructions to the computer. Each object is capable of receiving messages, processing data, and sending messages to other objects. Object-oriented gives more flexibility and increases code reusability.**
- **C++ is a programming language based on C, that supports object-oriented programming while accepting also the procedural programming paradigm.**

Object Oriented Programming in PETSc-FEM (cont.)

The key idea behind OOP is PETSc-FEM is that large sets of data like nodes and elements (atoms) with the same characteristics are grouped in *`Atomsets`* (*vectors* or *arrays* of atoms). Atomsets provide the following items

- **Degrees of freedom (unknown, varying fields).**
- **Spatial data (known, constant fields).**
- **Links to other atoms in this Atomset or another. (e.g. FEM connectivity).**
- **Restrictions (e.g. boundary conditions)**
- **Functions that map the unknowns in a given atom (or its linked atoms) to residual or matrices. (The *“element routine”*).**

Object Oriented Programming in PETSc-FEM (cont.)

Some examples,

- **In finite elements we have nodes and elements as atoms. Nodes have unknown fields and also known fields (constant data). Node coordinates are simply a constant data field.**
- **In cell-centered finite volumes and panel methods (BEM) nodes have the coordinates constant fields, but unknowns reside in the cells. Also FEM has sometimes unknown fields in elements, for instance internal pressure nodes in BB-*a priori* stable interpolation families for incompressible flow.**
- **In applications with moving boundaries (e.g. ALE) coordinates are an unknown field. There are two FEM problems: the physical problem (for which coordinates are a constant field) and the mesh-moving pseudo-elastic problem (for which node coordinates are an unknown field).**

Object Oriented Programming in PETSc-FEM (cont.)

- **Fields can be added at the user-level, mapping functions operate on their specific fields, ignoring additional fields. (e.g. Navier-Stokes for fluid-dynamics with coupled transport on scalars).**
- **Several finite element problems may coexist in the same run, for instance the fluid problem and the mesh-moving one when dealing with moving domains (ALE).**
- **The standard mesh graph is now the connectivity between atoms.**

Object Oriented Programming in PETSc-FEM (cont.)

Elemset class hierarchy

- **Functions that map element states to residuals (and matrices) belong to the `Elemset` class. They represent the physics of each problem.**
- **Specialized mapping functions can share code with more general ones by *“class derivation”*. This originates an Elemset class tree.**
- **In this respect, the most paradigmatic case is the *advective-diffusive module*. A mapping function for generic advective-diffusive problems with the SUPG formulation has been implemented. Physical problems as gas dynamics, shallow water (1D and 2D), linear advection, Burgers eqs., share this. Each physical problem is added by writing a *“flux function class”*. This is an abstraction of the mathematical relation $U \rightarrow \mathcal{F}_c(U), \mathcal{F}_d(U)$, where U is the fluid state and $\mathcal{F}_{c,d}$ are the convective and diffusive fluxes.**
- **Flux function objects are shared between different stabilization schemes (e.g. FEM+SUPG, FVM) or absorbing boundary conditions.**

Physical properties

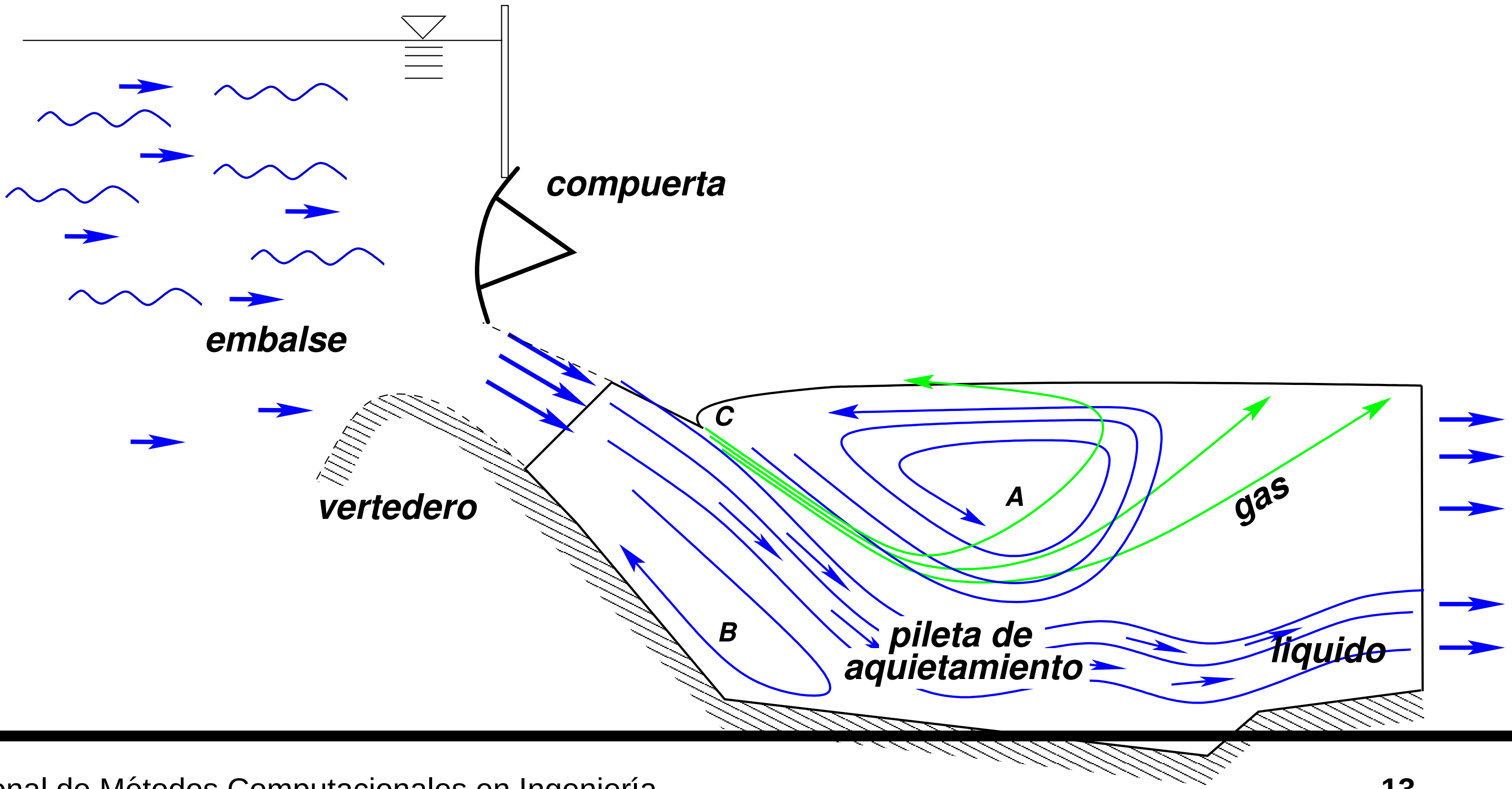
- **Per element physical properties (viscosity, Young modulus,...) can be entered as global, per-elemset or per-element.**
- **Also may be arbitrarily modified at run-time at user level via *"hooks"*. Hooks are a piece of code to be called at well defined points in time. (The concept is borrowed from the Emacs editor). For instance, this allows the user to run some code at the begin/end of a time step, or at the begin/end of the whole run.**
- **Currently hooks must be written in C/C++ (dynamically loaded, with *`dlopen()`* and friends) or shell script language (and run arbitrarily complex problems from there).**

Multi-physics

- **Multi-physics is the ability to manipulate several interacting fields as, for instance, fluid dynamics, elasticity, electromagnetism, transport of scalars, chemical reactions, several phases (multi-phase)...**
- ***Strong interaction* between fields must be implemented monolithically inside an elemset.**
- ***Weak interaction* between fields should be able to be added at the user level, via *hooks*.**

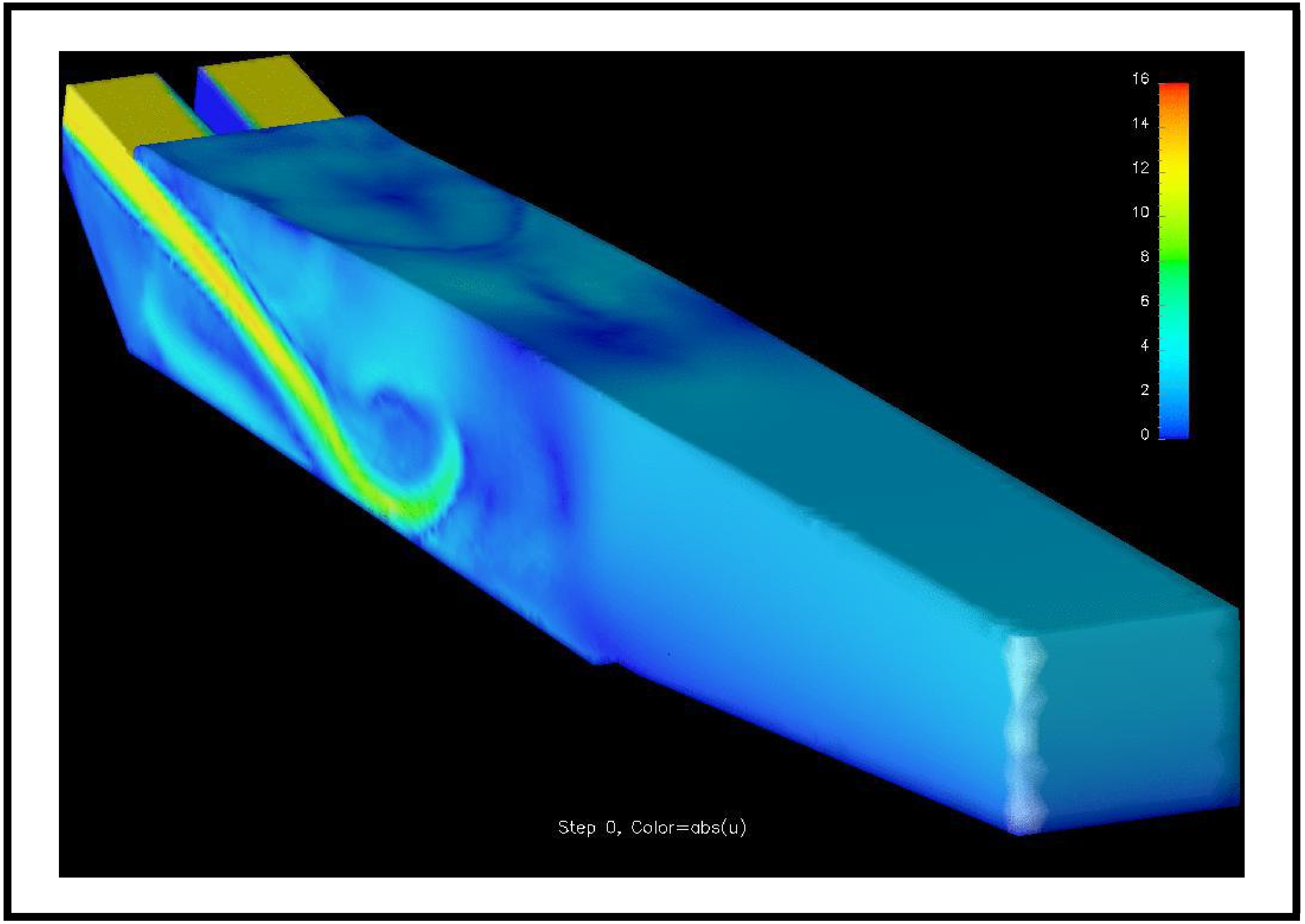
Fluid dynamics with reaction and transport

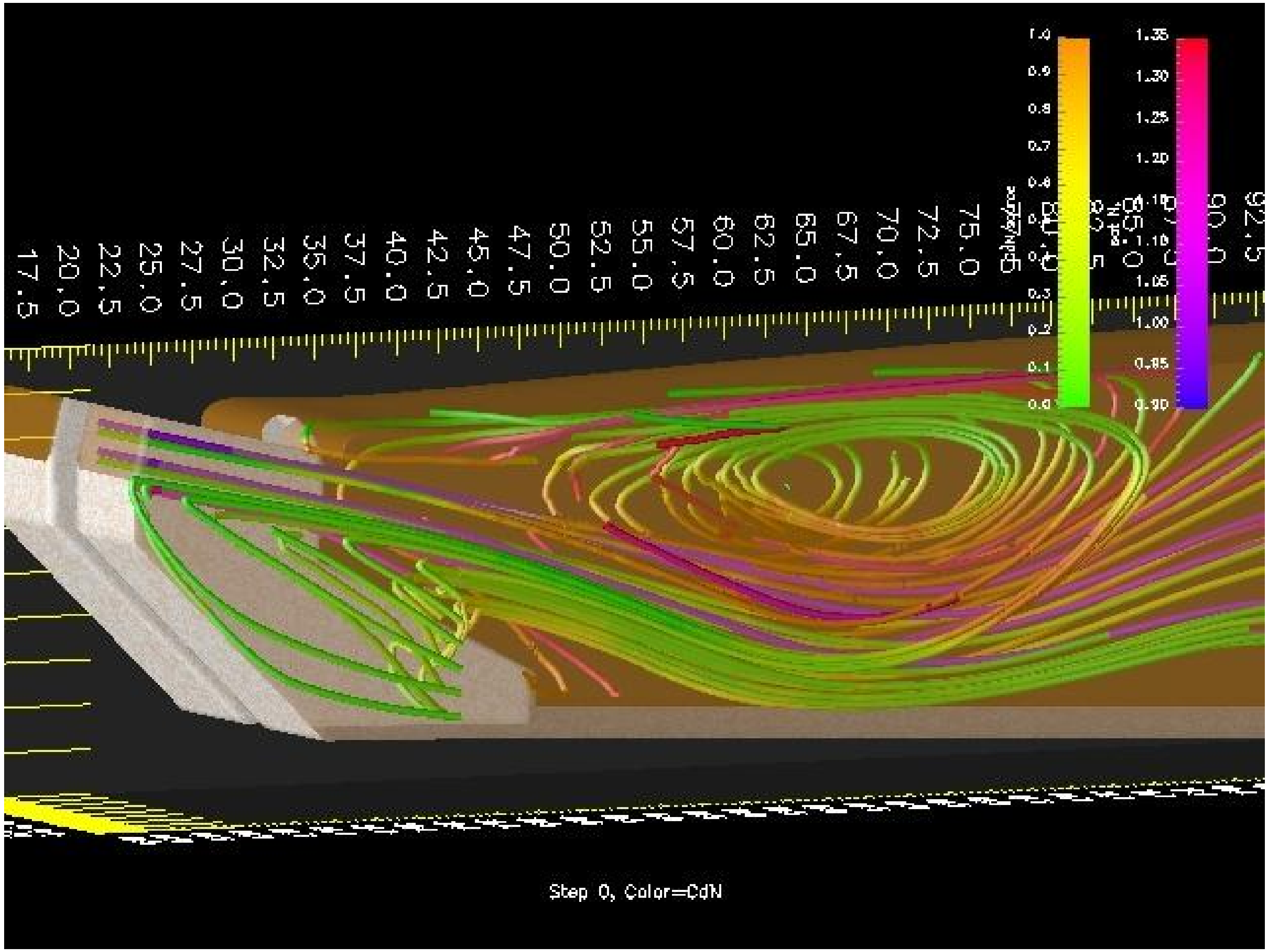
- **Jet impinging on pool produces high bubble content (foam). High energy jet drags bubbles to the bottom of the pool. Dilution of O2/N2 from gas phase to liquid phase is promoted by high pressures. Bubbles quickly return to air by buoyancy but high concentration of O2/N2 in the liquid remains for long distances downstream.**



Fluid dynamics with reaction and transport (cont.)

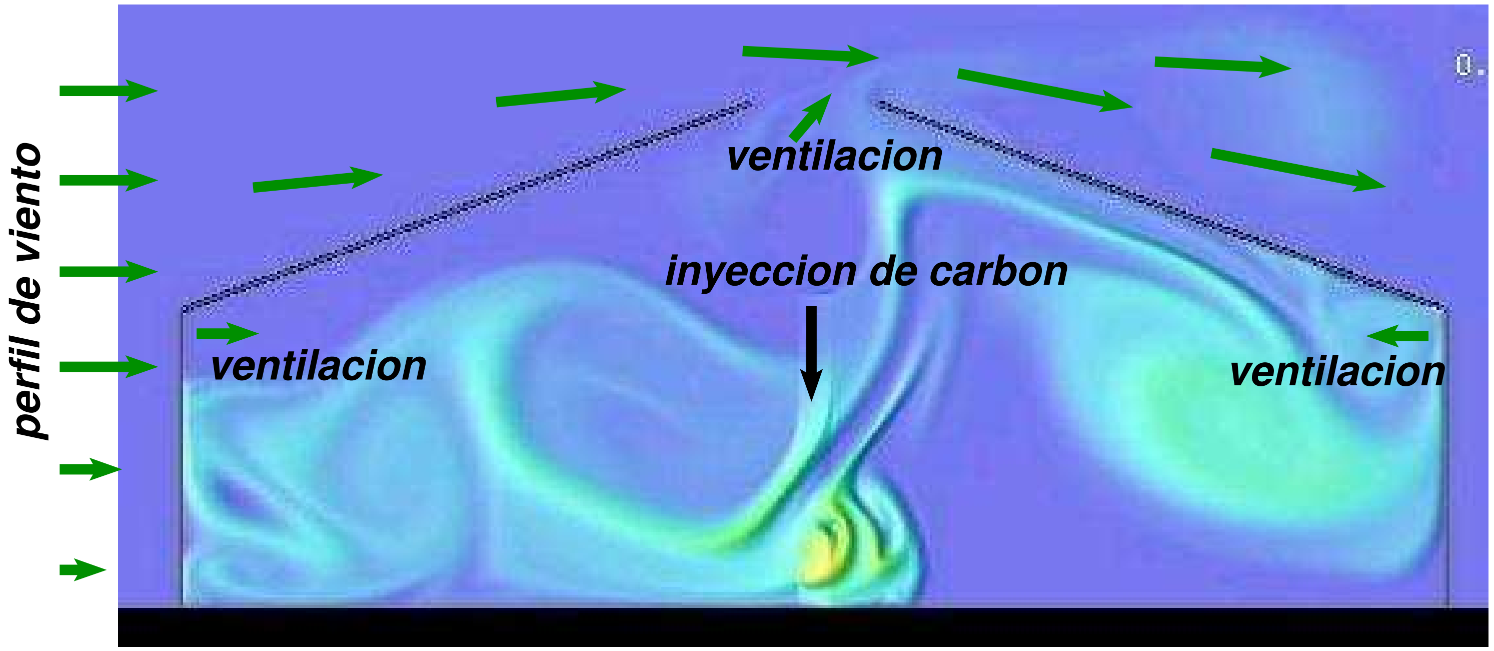
- **Model predicts concentration of O2/N2.**
- **The model includes solving the momentum equation and continuity for the (continuous) liquid phase. Driving force for the movement of water in the pool is the impinging jet and bubble buoyancy.**
- **Momentum eq. for gas phase may be treated as a restriction (algebraic model) or as another evolution equation.**
- **Solve transport eq. for bubble number density.**
- **Transport of O2/N2 in the liquid is solved, with reaction terms involving the difference in O2/N2 concentration at the gas phase and liquid phase, bubble density, etc...**
- **Has $2 \cdot n_d + 2$ unknown fields and equations.**
- **May be solved monolithic or staggered in time: advance fluid dynamics, then dilute gas concentration.**


16
14
12
10
8
6
4
2
0
Step 0, Color=abs(u)


Step 0, Color=CdN

Multi-phase flow gas/solid

***Multi-phase gas/solid flow*: Coke processing facility want to assess total amount of particle emission under several conditions (forced and natural convection).**



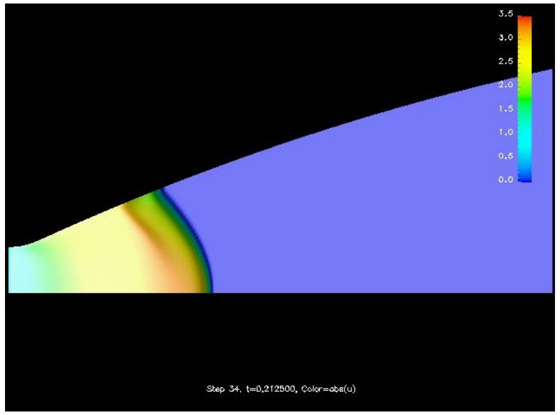
Multi-phase flow gas/solid (cont.)

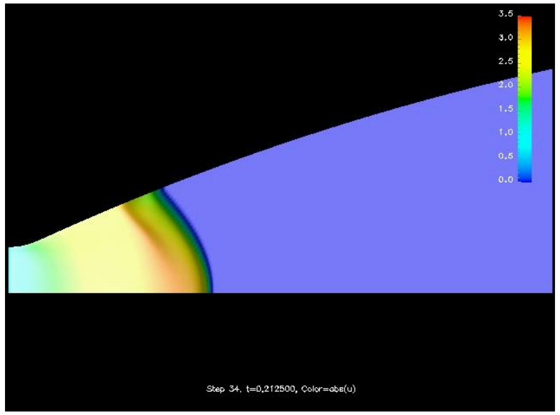
- **Solve momentum equations and continuity for air velocity field.**
- **Solve momentum equations for solid particle velocity field or assume algebraic slip model.**
- **Solve particle number transport equation.**
- **May use several particle sizes (one scalar unknown field for each particle size).**

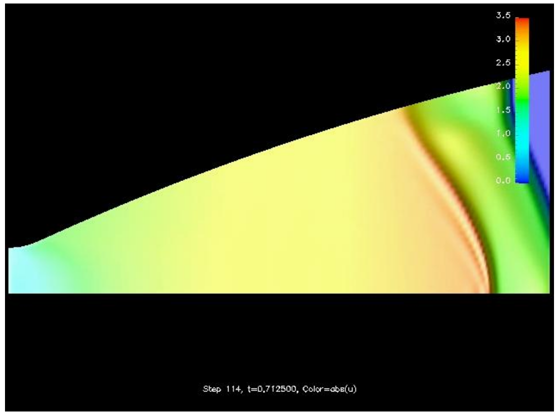
Multi-phase flow gas/solid (cont.)

Nozzle chamber fill

The case is the ignition of a rocket launcher nozzle in a low pressure atmosphere. The fluid is initially at rest (143 Pa, 262 $^\circ\mathrm{K}$). At time $t = 0$ a membrane at the throat is broken. Behind the membrane there is a reservoir at $6\times10^5\ \mathrm{Pa}$, $4170^\circ\mathrm{K}$. A strong shock (intensity $p_1/p_2 > 1000$) propagates from the throat to the outlet. The gas is assumed as ideal ($\gamma = 1.17$). In the steady state a supersonic flow with a max. Mach of 4 at the outlet is found. The objective of the simulation is to determine the time needed to fill the chamber ($< 1\mathrm{msec}$) and the final steady flow.







Multi-phase flow gas/solid (cont.)

We impose density, pressure and tangential velocity at inlet (assuming subsonic inlet), slip condition at the nozzle wall. The problem is with the outlet boundary. Initially the flow is subsonic (fluid at rest) there, and switches to supersonic. The rule dictaminates to impose 1 condition, as a subsonic outlet (may be pressure, which is known) and no conditions after (supersonic outlet). If pressure is imposed during the wall computation, then a spurious shock is formed at the outlet.

This test case has been contrasted with experimental data obtained at ESTEC/ESA (European Space Research and Technology Centre-European Space Agency, Noordwijk, Holanda). The predicted mean velocity was 2621 m/s to be compared with the experimental value of 2650+/-50 m/sec.

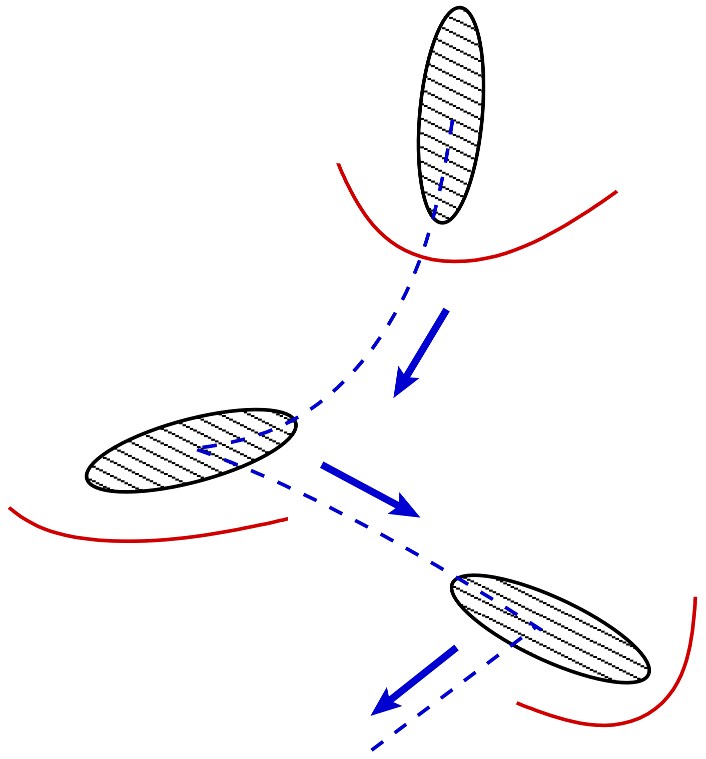
Successful modeling requires imposing boundary conditions *dynamically*. From one condition to rest, to subsonic outlet to supersonic outlet *during the computation*.

Object falling at supersonic regime

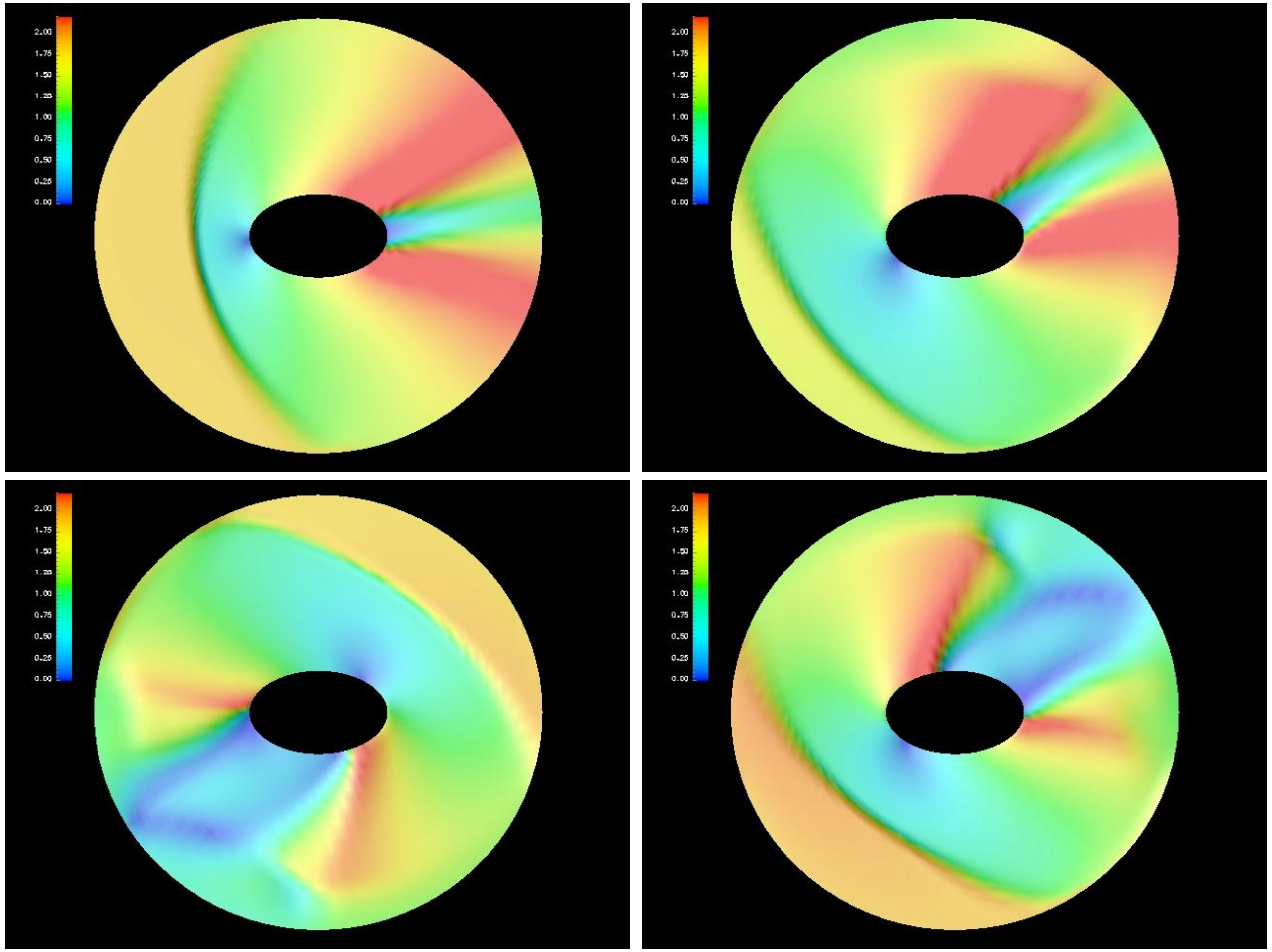
We have an object in free fall, under certain conditions in size and density relation to the surrounding atmosphere it reaches supersonic speeds. At supersonic speeds the principal source of drag is the shock wave, we use slip boundary condition at the body in order to simplify the problem. This kind of falling mechanism is typical of slender bodies with relatively small moment of inertia like a sheet of paper and is called *"flutter"*. However, depending of several parameters, but mainly depending of the moment of inertia of the body, if it has a large angular moment at (B) then it may happen that it rolls on itself, keeping always the same sense of rotation. This kind of falling mechanism is called tumbling and is characteristic of less slender and more massive objects. For massive objects (like a ballistic projectile, for instance) tumbling may convert a large amount of potential energy in the form of rotation, causing the object to rotate at very large speeds.

Object falling at supersonic regime (cont.)

We also do the computation in a non-inertial system following the body, so that non-inertial terms (Coriolis, centrifugal, etc...) are added. In this frame some portions of the boundary are alternatively in all the conditions (subsonic incoming, subsonic outgoing, supersonic incoming, supersonic outgoing).
Again, the ideal would be to switch* dynamically *from one condition to the other during the computation.

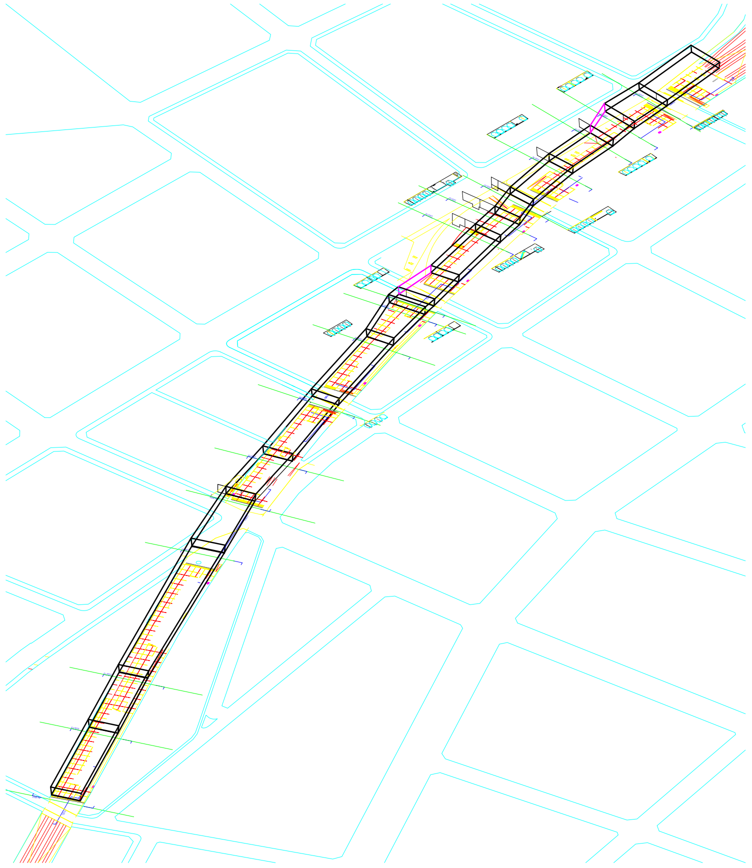


Object falling at supersonic regime (cont.)



Fire protection

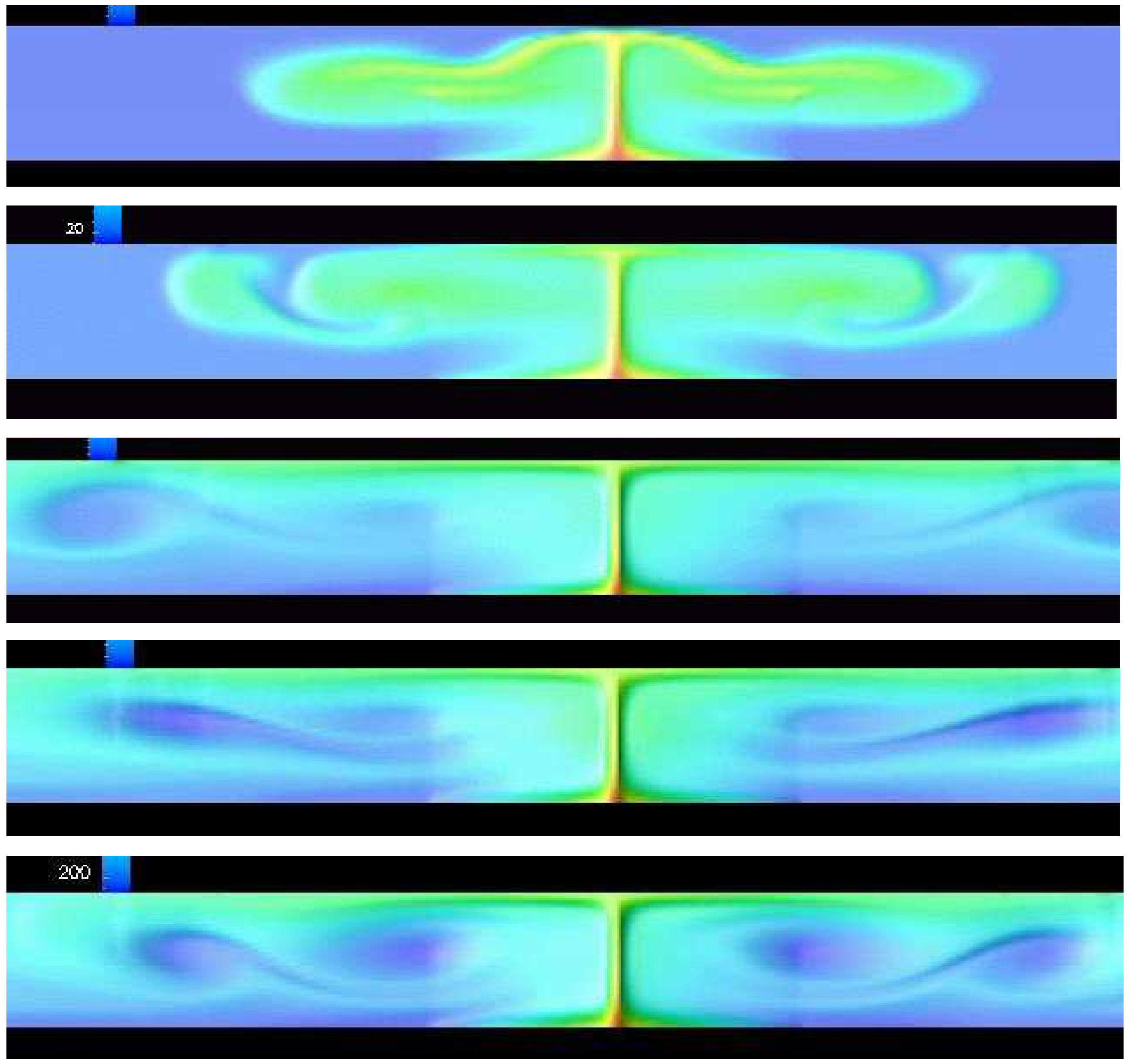
- **City of BA plans to cover the Sarmiento railway corridor for 800 m starting at Once Station.**
- **Assessment of TAE (for *"Time Available for Escape"*) and TNE (for *"Time Needed for Escape"*) under accidental fire development is demanded by constructors (Vialmani S.A., HP-IC S.A. and KB Eng. S.A.).**
- **Standard fire on a wagon is simulated (1.7 MW, 1500 C, 6 % CO, 1e9 solid part/m3, 10 micra).**
- **Work currently in progress.**



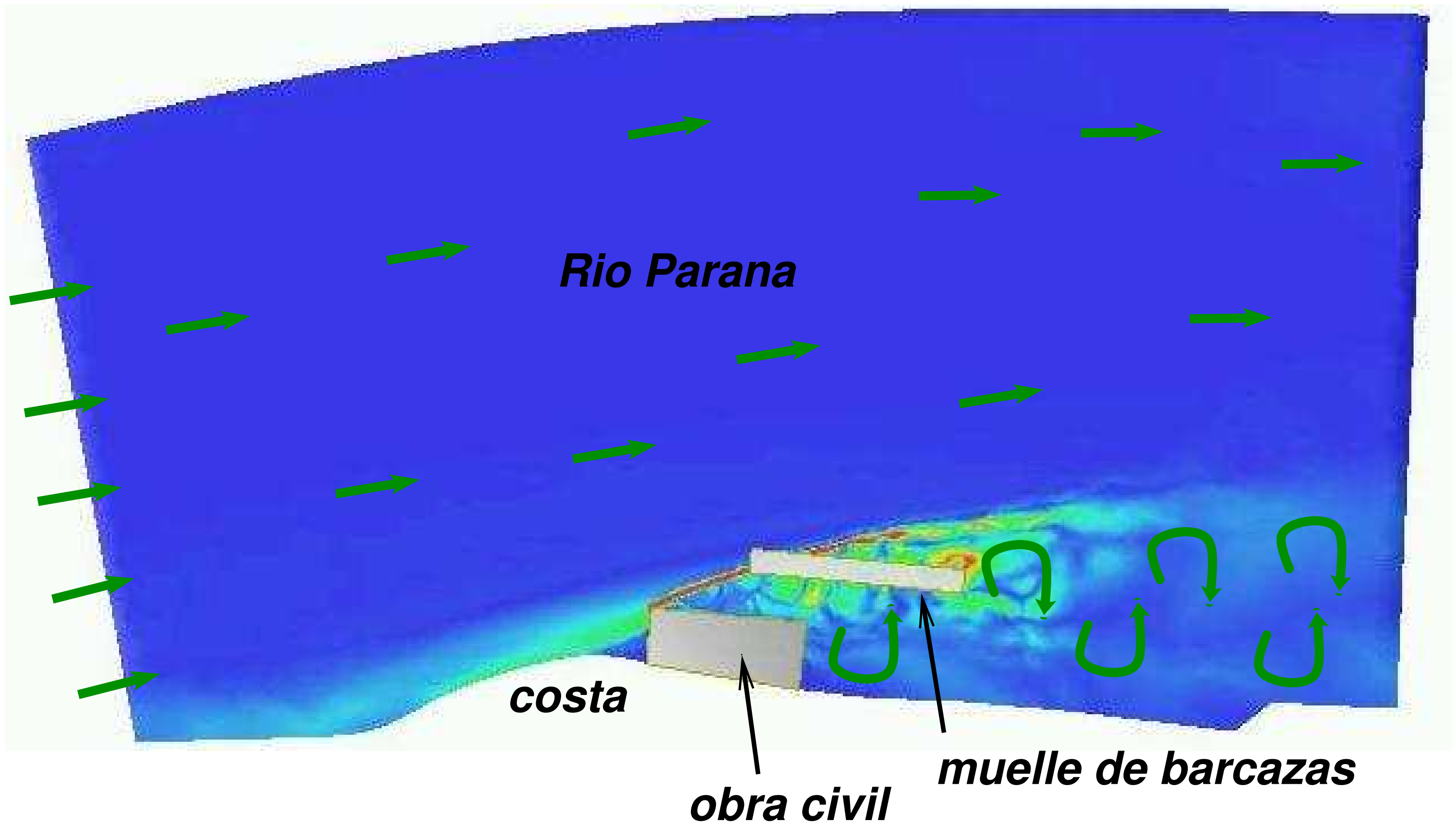
Fire protection (cont.)

Map
Satellite
Hybrid
200 ft
100 m
©2005 Google - Imagery ©2005 DigitalGlobe - Terms of Use
Map
Satellite
Hybrid
200 ft
100 m
©2005 Google - Imagery ©2005 DigitalGlobe - Terms of Use

Fire protection (cont.)



Incidence of flow on ship maneuverability

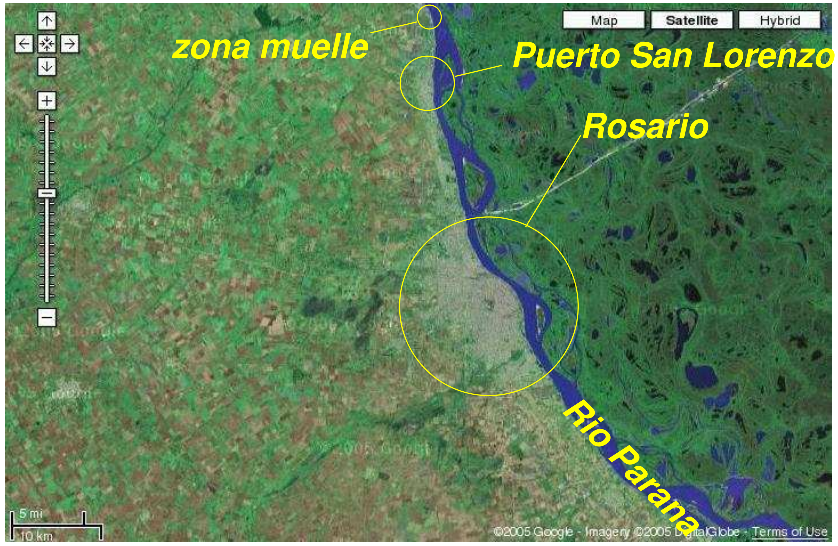


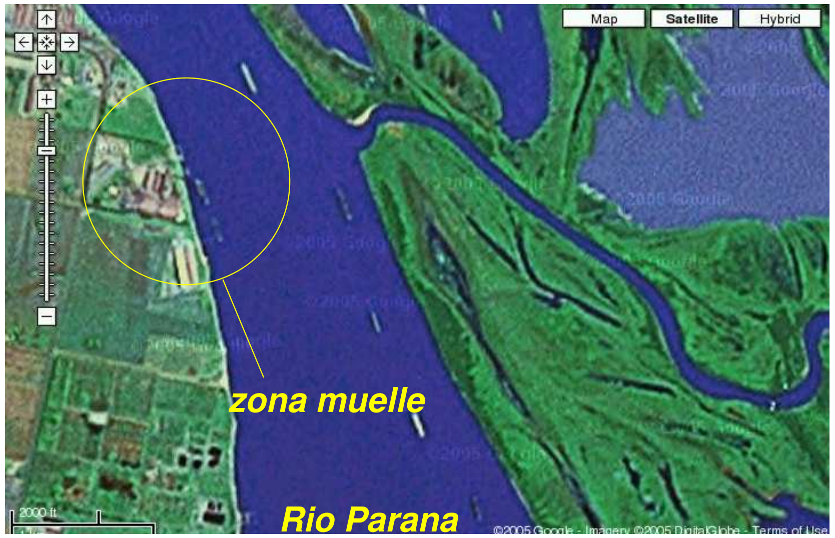
Which is the competitivity niche for PETSc-FEM?

- **Commercial codes give multi-physics support via add-hoc combination of several physical models (e.g. NS+transport, fluid+structure,...)**
- **Users can't use combinations of physical modules that have not been anticipated by the developer.**
- **PETSc-FEM allows weak coupling of physical modules at the user level (requires good knowledge of the library and programming though!! ☹)**

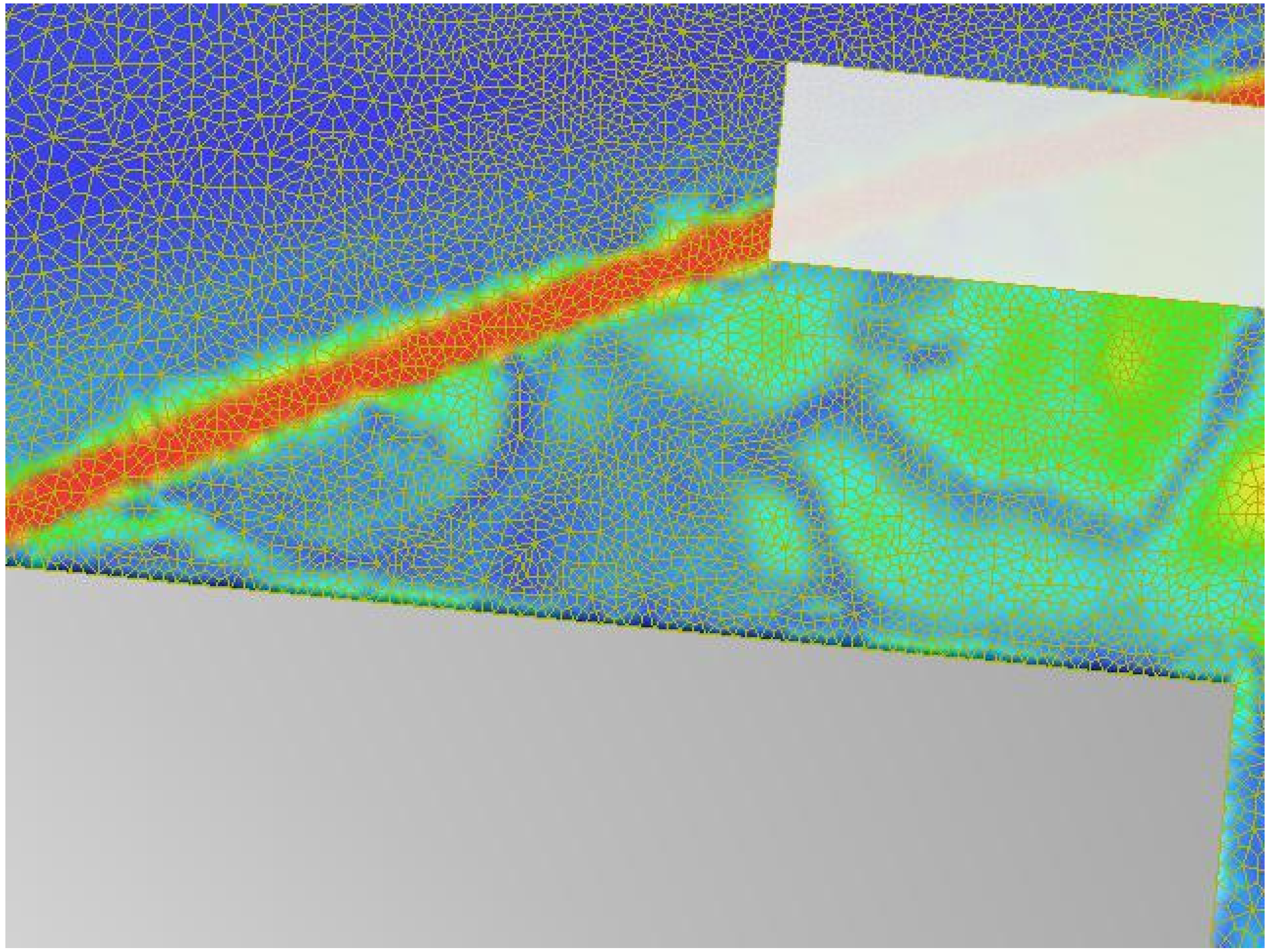
Which is the competitivity niche for PETSc-FEM? (cont.)

- ***Objective:*** **Compute turbulence intensity produced by a civil construction near the ship maneuvering zone.**
- ***Equations to solve:*** **Navier-Stokes 3D.**

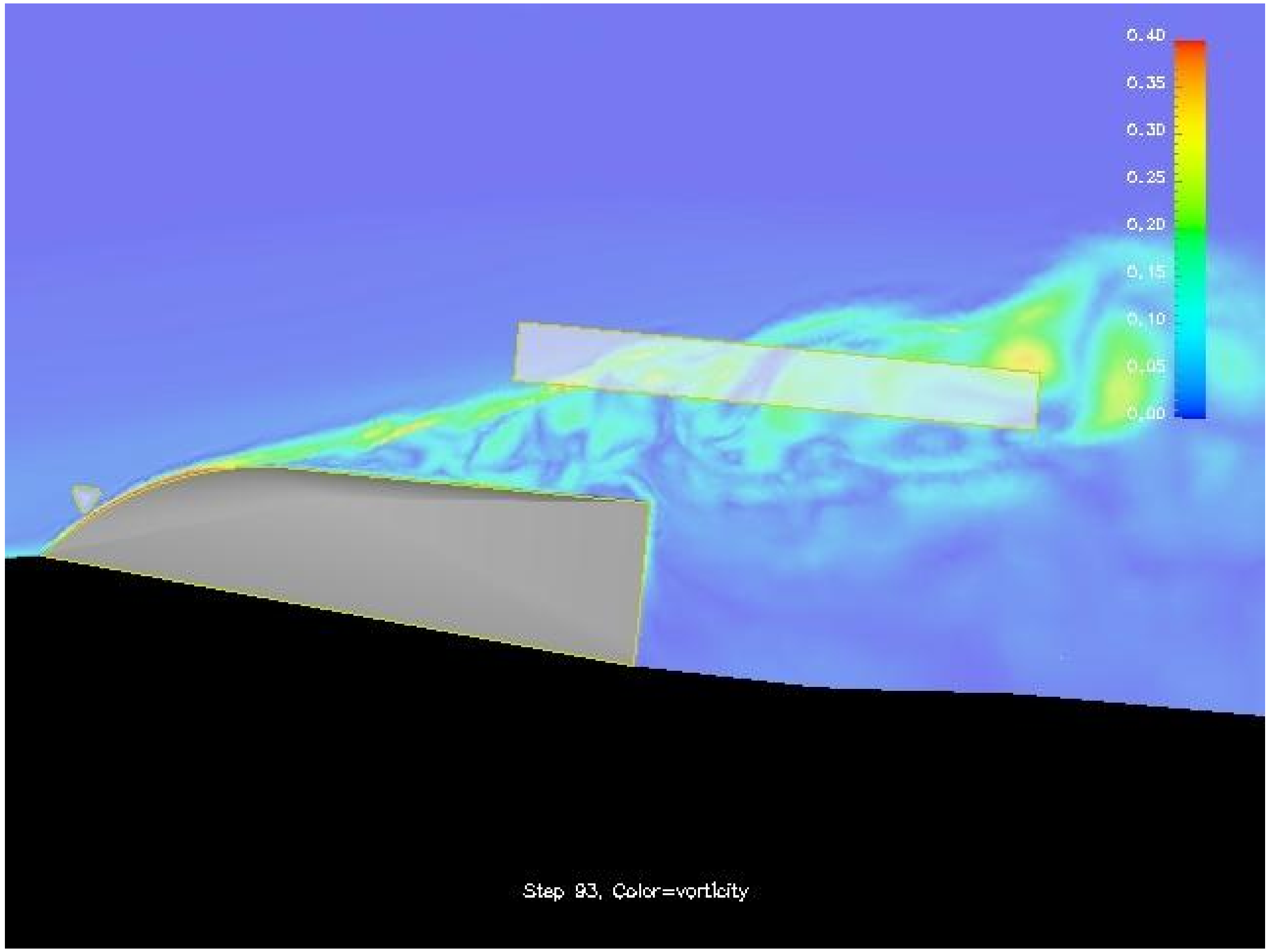




Which is the competitivity niche for PETSc-FEM? (cont.)



Which is the competitivity niche for PETSc-FEM? (cont.)



Scripting via extension languages

- **Currently, user data file has a tree-like data structure with nodes meaning of the form (key,value) assignments.**
- **As the syntax of the user data files grew in complexity and more preprocessing is needed (calling functions, system calls...) the syntax *evolves eventually in a language*.**

```
global_options

alpha 1. # Integration rule param.

#if !$restart # Conditional processing...
nstep 100000
#else
nstep 1
#endif

tol_newton 1e-10 # Iterative parameters
atol 0
rtol 1e-7
viscosity <:=$U*$L/$Re:> # handle math exps
...
```

Scripting via extension languages (cont.)

Adding features to the user data file syntax and making it a language has many disadvantages.

- **Developer has to define a correct language (syntax, ...) and implement it.**
- **Users have to learn a new language/syntax.**

This happened with a series of Open Source projects (Apache, PHP, Perl, ...). Developers started with a small language definition and ended with a full featured language. A better solution is to use an already existing high level scripting language and extend it with the functions of the new library. In this way the user needs not to deal with coding in a low level language like C/C++ directly. Candidates for extension language are Perl, Python, Lisp/Scheme, Octave...

Scripting via extension languages (cont.)

The subject was at the heart of a war over the net (the *“Tcl war”*, see `http://www.vanderburg.org/Tcl/war/`) and the Free Software Foundation proposed to design a language specific to be an extension language for Free Software projects. This is Guile, a complete implementation of the Scheme language, a dialect of Lisp. Advantages of Guile are

- **It implements Scheme a full featured language. It implements dynamic typing, supports functional programming, has first class procedures, hygienic macros, first class continuations. There are many implementations (interpreters) of Scheme, Guile is just one of them.**
- **It is the extension language proposed by the FSF.**
- **Full support for loading C functions in libraries.**
- **Can call C functions from Scheme and vice-versa.**

Scripting via extension languages (cont.)

User data file will be a Guile/Scheme script

```
(use-modules (petsc-fem base))
(use-modules (petsc-fem ns))
(use-modules (petsc-fem elasticity))

(define global-options
  ‘(alpha . 1.0) ;;; Integration rule param.
  (nstep . ,(if restart 1000 1)) ;;; Cond. proc. . .
  (tol-newton . 1e-10) ;;; Iterative parameters
  (atol . 0)
  (rtol . 1e-7)
  ;;; Handles arbitrarily complex math exps
  (viscosity . ,(/ (* rho U L) Re)))
(define coords (elemset-load "./coords.dat"))
(define elemset (elemset-load "./connect.dat"))
(define ns-problem
  (ns-init coords elemset global-options))
. . .
```

Scripting via extension languages (cont.)

- **Currently best candidates are Scheme and Python.**
- **Written wrappers for MPI and PETSc libraries in Python (Lisandro Dalcín). [Dalcín, L., Paz, R., Storti, M. “MPI for Python”, Journal of Parallel and Distributed Computing, 65/9, pp. 1108-1115 (2005)]. Can exchange arbitrary Python objects between processors (via `cPickle` module).**
- **Basic MPI wrappers in Scheme have been written.**
- **Sends/receives special vector objects (PETSc-FEM `dvector<T>` class).**
- **Sends complex Scheme objects with serialization via write/read (or *hash-comma* syntax for extended objects (SRFI-10)).**

```
(use-modules (mpi))
(use-modules (dvector))
(mpi-initialize)

;;; Get rank and size
(define my-rank (mpi-rank))
(define size (mpi-size))

;;; Standard parallel hello...
(format t "Hello world I am ~A of ~A\n" my-rank size)

;;; Define vectors v,w, fill v with random
(define N 1000)
(define v (make-dvdbl N))
(define w (make-dvdbl ))
(dv-rand! v)

;;; Rotates data (sends to myrank+1 and
;;; receives from myrank-1, cyclically)
(cond ((even? my-rank)
        (mpi-send v (modulo (+ my-rank 1) size))
        (mpi-recv w (modulo (- my-rank 1) size)))
       (else
        (mpi-recv w (modulo (- my-rank 1) size)))
        (mpi-send v (modulo (+ my-rank 1) size)))

(mpi-finalize)
```

Functional programming style

Functional programming promotes intermix of code and data (CODE=DATA slogan).

```
;;; Saves state file to disk
;;; each 10 time steps
(define nsave 10)

;;; Save each 10 or if large variations in
;;; state vector are detected
(define nsave
  (lambda (step)
    (if (or (= (modulo step 10) )
            (check-state step)))))
```

Functional programming style (cont.)

```
(define visco 1e-3) ;;; Fixed value for viscosity

(define visco ;;; Use Sutherland's law
  (sutherland-law 1e-3 300 110 120))

;;; This takes the physical parameters
;;; and returns a function of temperature.
(define (sutherland-law muinf Tinf T1 T2 expo)
  (let ((T1 T1) (T2 T2) (expo expo))
    (lambda (T)
         (if (< T T2)
             (* (muinf (/ T Tinf)))
             (* muinf
                (/ T2 Tinf)
                (expt (/ T T2) expo)
                (/ (+ Tinf T1) (T T1))))))))
```

MPI for Python

- **Another possibility for scripting language is Python.**
- **Python is a powerful programming language, easy to learn, with a large library set and a simple and strongly integrated Object Oriented Programming system.**
- **It is an ideal candidate for writing higher-level parts of large scale scientific applications and driving simulations in parallel architectures.**
- **Can call C functions from Python and vice-versa.**
- **Previous to extend PETSc-FEM with Python we worked on adding some parallel functionality to Python. This evolved in the package *“Mpi4py”*.**
- **Related work is found in the OOMPI, Pypar, pyMPI, Scientific Python, Numeric, Numarray, PyFort, SciPy and SWIG projects.**

MPI for Python (cont.)

- **First, a special Python interpreter was created. The Python interpreter that comes with the `python` package would be OK for non-interactive use. In interactive use the interpreter must read a line on the master node and broadcast to the slaves. Then the line is evaluated, as is in he usual REPL (Read-Eval-Print loop).**
- **Any Python object can be transmitted using the standard `cPickle` module. The object to be transmitted is first serialized. After that, string data is communicated (using MPI CHAR datatype). Finally, received strings are unpacked and the original object is restored. Serialization process introduces some overheads: dynamic memory allocations, heavier messages and extra processing. However, this methodology is easily implemented and quite general. Direct communication, i.e., without serialization, of consecutive numeric arrays is feasible but not currently supported. This issue will be addressed in the near future.**

MPI for Python (cont.)

- `Comm` class implemented with methods: `Get_size()`, `Get_rank()`, `Clone()`, `Dup()`, `Split()`.
- Set operations with Group objects like `Union()`, `Intersect()` and `Difference()` are fully supported, as well as the creation of new communicators from groups. Virtual topologies (`Cartcomm` and `Graphcomm` classes, both being a specialization of `Intracomm` class) are fully supported.

MPI for Python (cont.)

- **Methods *`Send()`*, *`Recv()`* and *`Sendrecv()`* of communicator objects provide support for blocking point-to-point communications within *`Intracomm`* and *`Intercomm`* instances. Non-blocking communications are not currently supported.**
- **Methods *`Bcast()`*, *`Scatter()`*, *`Gather()`*, *`Allgather()`* and *`Alltoall()`* of *`Intracomm`* instances provide support for collective communications. Global reduction operations *`Reduce()`*, *`Allreduce()`* and *`Scan()`* are supported but naively implemented.**

MPI for Python (cont.)

- **Error handling functionality is almost completely supported. Errors originated in native MPI calls will throw an instance of the exception class *`Exception`*, which derives from standard exception *`RuntimeError`*.**

MPI for Python (cont.)

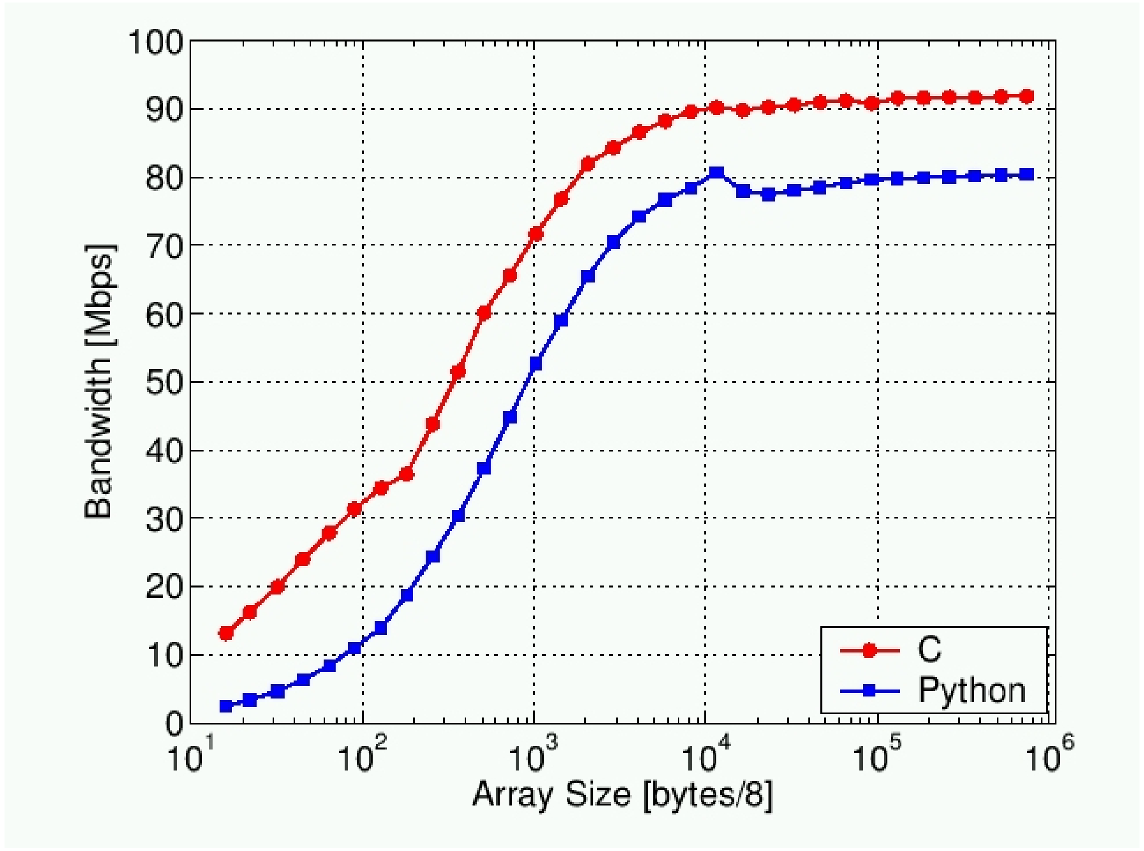
- **Efficiency: Some efficiency tests were run on the Beowulf class cluster Geronimo at CIMEC. Hardware consisted of ten computing nodes with Intel P4 2.4Ghz processors, 512KB cache size, 1024MB RAM DDR 333MHz and 3COM 3c509 (Vortex) NIC cards interconnected with an Encore ENH924-AUT+ 100Mbps Fast Ethernet switch. MPI for Python was compiled with MPICH 1.2.6 and Python 2.3, Numarray 1.1 was also used.**
- **The first test was a bi-directional blocking send and receive between pairs of processors. Messages were numeric arrays (NumArray ob jects) of double precision (64 bits) floating-point values. A basic implementation of this test using MPI for Python (translation to C or C++ is straightforward) is shown below.**

MPI for Python (cont.)

```
from mpi4py import mpi
import numarray as na
sbuff = na.array(shape=2**20,
                 type= na.Float64)
wt = mpi.Wtime()
if mpi.even:
   mpi.WORLD.Send(buffer,mpi.rank+1)
   rbuff = mpi.WORLD.Recv(mpi.rank+1)
else:
   rbuff = mpi.WORLD.Recv(mpi.rank-1)
   mpi.WORLD.Send(buffer,mpi.rank-1)
wt = mpi.Wtime() - wt
tp = mpi.WORLD.Gather(wt, root=0)
if mpi.zero: print tp
```

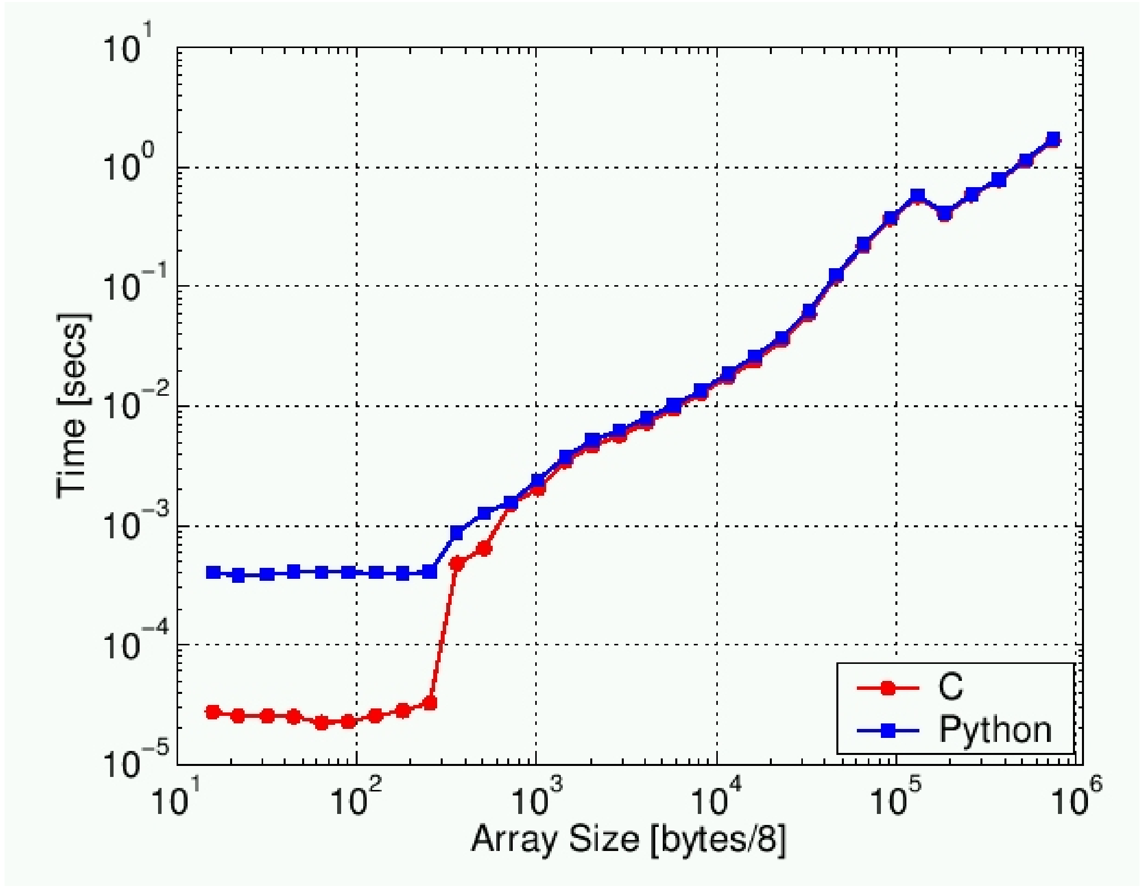
MPI for Python (cont.)

Maximum bandwidth in Python is about 85 % of maximum bandwidth in C. Clearly, the overhead introduced by object serialization degrades overall efficiency.

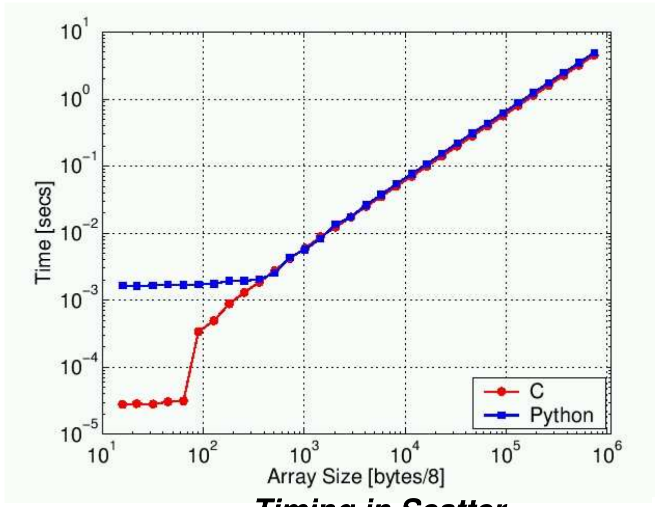


MPI for Python (cont.)

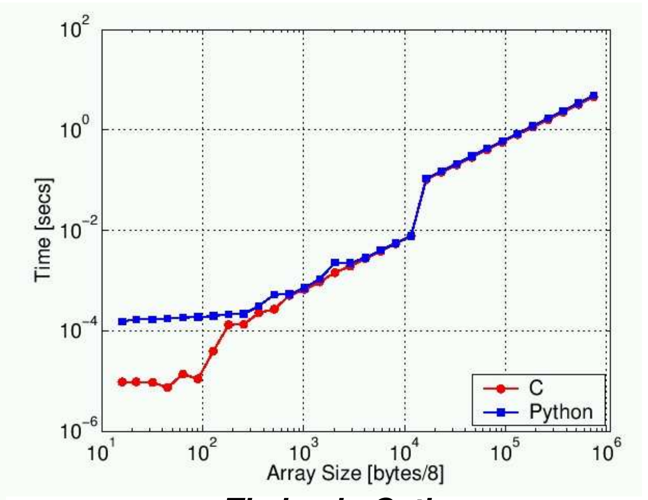
The second test consisted in wall-clock time measurements of some collective operations on ten uniprocessor nodes. Messages were again numeric arrays of double precision floating-point values. For array sizes greater than 103 (8KB), timings in Python are between 5 % (for Bcast) to 20 % (for Alltoall) greater than timings in C. (*Timing in Broadcast*)



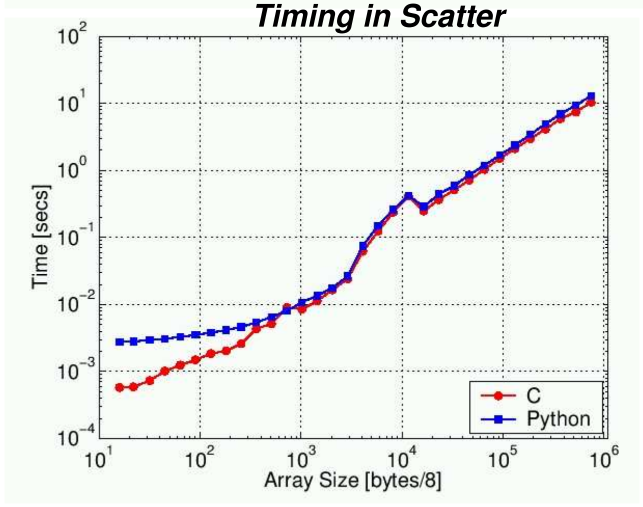
MPI for Python (cont.)



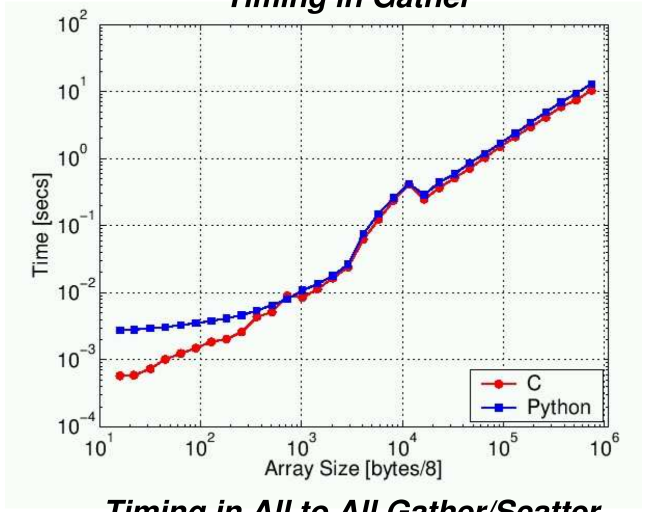
Timing in Scatter



Timing in Gather



Timing in Gather to All



Timing in All to All Gather/Scatter

MPI for Python (cont.)

Conclusions:

- **Like any scripting language, Python is not as efficient as compiled languages. However, it was conceived and carefully developed to be extensible in C (and consequently in C++).**
- **Python can be used as a glue language capable of connecting existing software components in a high-level, interactive, and productive environment.**
- **Running Python on parallel computers is a good starting point for decreasing the large software costs of using HPC systems.**
- **MPI for Python can be used for learning and teaching message-passing parallel programming taking advantage of Python's interactive nature.**

MPI for Python (cont.)

- **Efficiency tests have shown that performance degradation is not prohibitive, even for moderately sized objects. In fact, the overhead introduced by MPI for Python is far smaller than the normal one associated to the use of interpreted versus compiled languages.**
- **Examples of Mpi4py use in the JPDC paper or in the package (`http://www.cimec.org.ar/python`, maintainer `mailto:dalcinl@intec.unl.edu.ar`)**

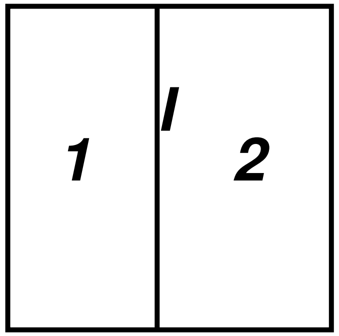
- ***Future work:* Add some currently unsupported functionalities like non blocking communications and direct communication of numeric arrays.**
- **Develop Python packages providing access to very well known and widely used MPI based parallel libraries like PETSc and ParMETIS (almost done)**
- **Furthermore, the higher-level portions of the parallel multi-physics finite elements code PETSc-FEM [32,33] developed at CIMEC are planned to be implemented in Python in the next major rewrite.**

Parallel solution of large linear systems

- **Direct solvers are highly coupled and don't parallelize well (high communication times). Also they require too much memory, and they asymptotically demand more CPU time than iterative methods even in sequential mode. But they have the advantage that the computational cost do not depend on condition number.**
- **Iteration on the global system of eqs. is highly uncoupled (low communication times) but has low convergence rates, specially for bad conditioned systems.**
- ***"Substructuring"* or *"Domain Decomposition"* methods are somewhat a mixture of both: the problem is solved on each subdomain with a direct method and we iterate on the interface values in order to enforce the equilibrium equations there.**

Global iteration methods

$$\begin{bmatrix} A_{11} & 0 & A_{1I} \\ 0 & A_{22} & A_{2I} \\ A_{I1} & A_{I2} & A_{II} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_I \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_I \end{bmatrix}$$



- **Computing matrix vector operations involve to compute diagonal terms $A_{11}x_1$ and $A_{22}x_2$ in each processor and,**
- **Communicate part of the non-diagonal contributions.**

Global iteration methods (cont.)

Eliminate x_1 and x_2 to arrive to the condensed eq.

$$\begin{aligned}(A_{II} - A_{I1}A_{11}^{-1}A_{1I} - A_{I2}A_{22}^{-1}A_{2I})\,x_I \\ = (b_I - A_{I1}A_{11}^{-1}\,b_1 - A_{I2}A_{22}^{-1}\,b_2) \\ \tilde{A}\,x_I = \tilde{b}\end{aligned}$$

Evaluation of $y_I = \tilde{A}\,x_I$ implies

- **Solving the local equilibrium equations in each processor for x_j:** $A_{jj}\,x_j = -A_{jI}\,x_I$
- **Summation of the interface and local contributions:** $y_I = A_{II}\,x_I + A_{I1}\,x_1 + A_{I2}\,x_2$
- **This method will be referred later as DDM/SCMI (*“Domain Decomposition Method/Schur complement matrix iteration”*).**

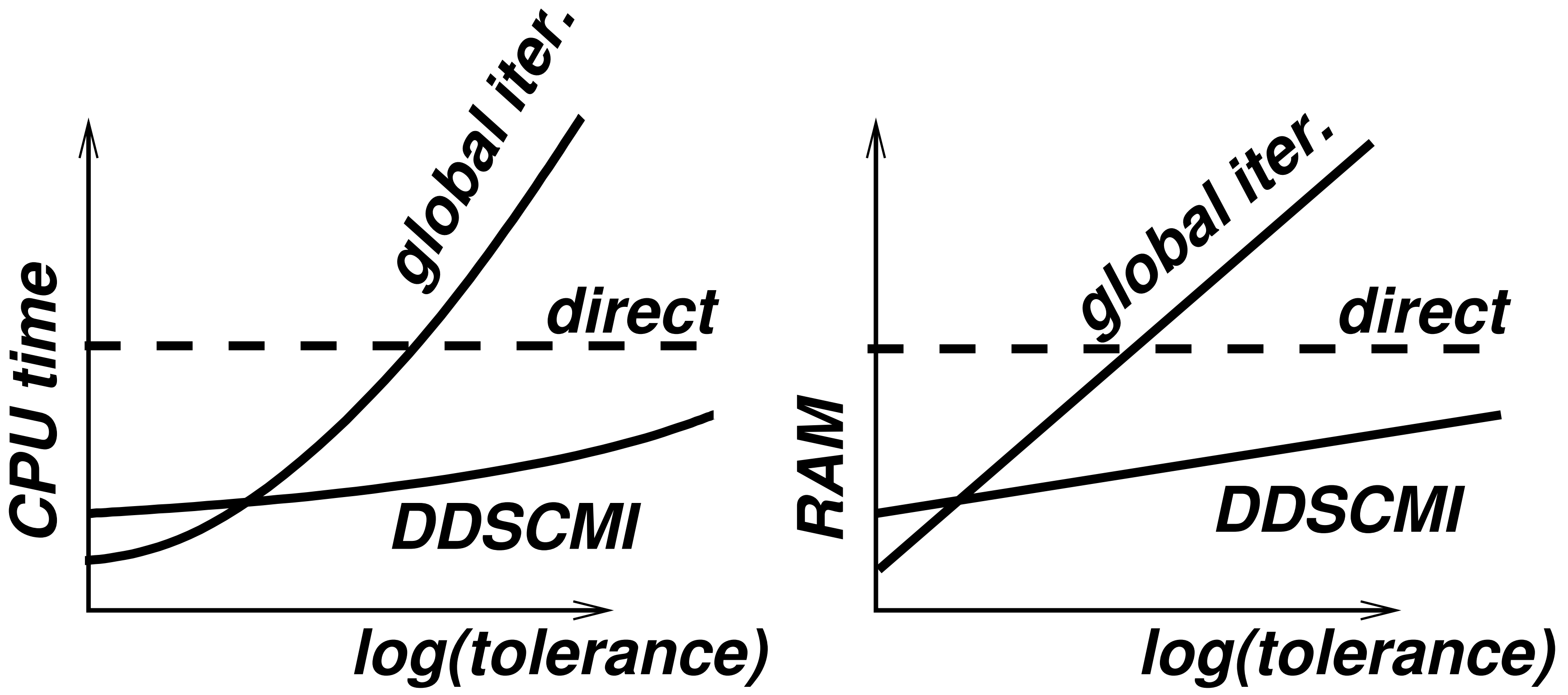
DDM/SCMI vs. Global iter.

- **Iteration on the Schur complement matrix (condensed system) is equivalent to iterate on the subspace where the local nodes (internal to each subdomain) are in equilibrium.**
- **The rate of convergence (per iteration) is improved ☺ due to: a) the condition number of the Schur complement matrix is lower, b) the dimension of the iteration space is lower (non-stationary methods like CG/GMRES tend to accelerate as iteration proceeds). However this is somewhat compensated by factorization time and backsubst. time ☹.**
- **As the iteration count is lower and the iteration space is significantly smaller, RAM requirement for the Krylov space is significantly lower ☺, but this is somewhat compensated by the RAM needed by the factorized internal matrices $LU(A_{jj})$ ☹.**

DDM/SCMI vs. Global iter. (cont.)

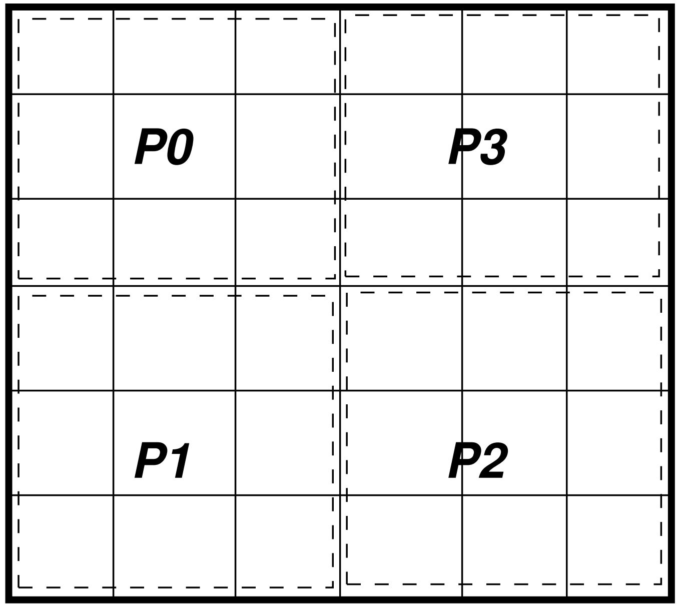
- **Better conditioning of the Schur comp. matrix prevents CG/GMRES convergence break-down ☺**
- **As GMRES CPU time is quadratic in iteration count (orthogonalization stage) and global iteration requires usually more iterations, Schur comp. matrix iteration is comparatively better for lower tolerances (☺/☹).**
- **Global iteration is usually easier to load balance since it is easier to predict computation time accordingly to the number of d.o.f.'s in the subdomain ☹**

DDM/SCMI vs. Global iter. (cont.)



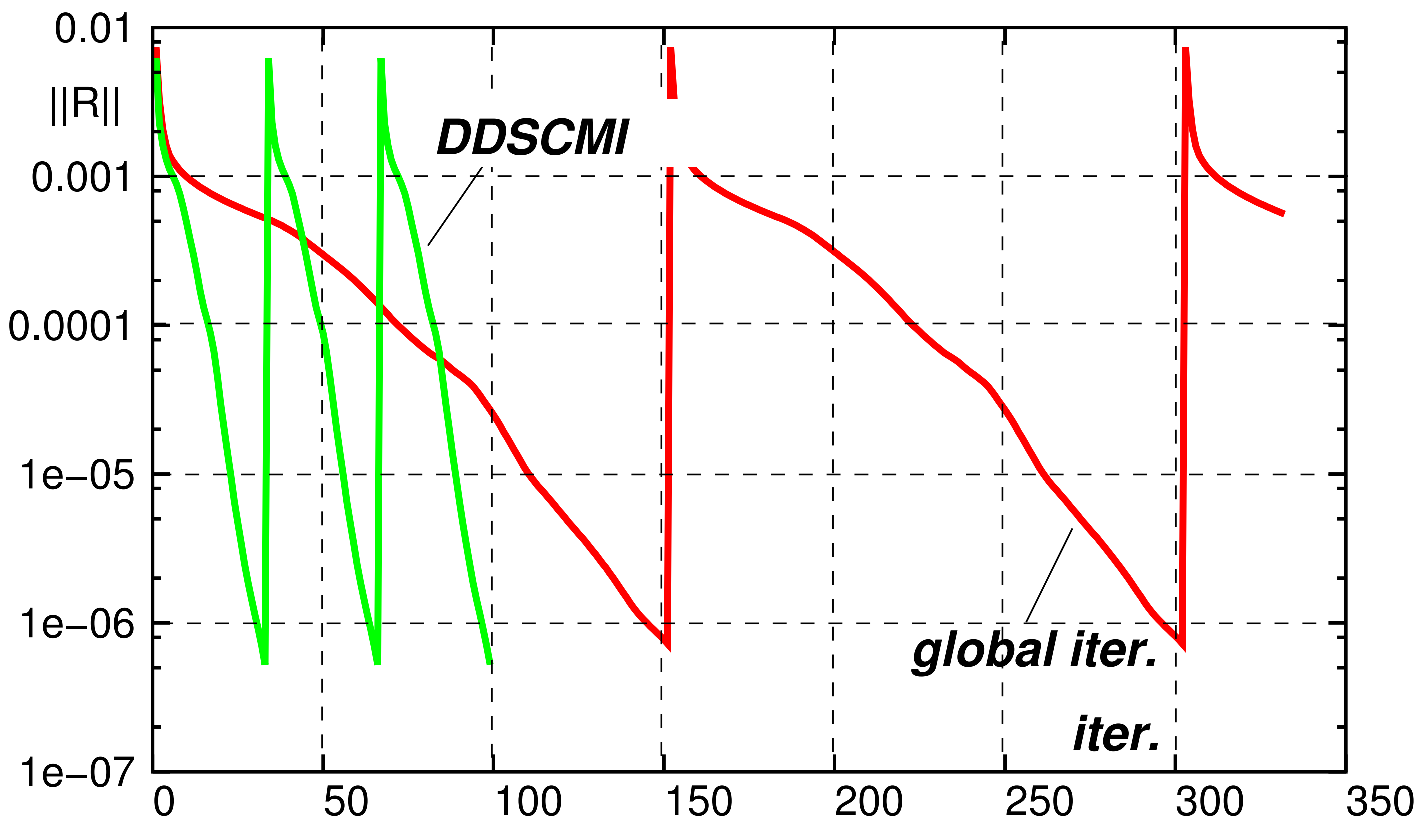
Subpartitioning

- **For large problems, the factorized part of the A_{jj} matrix may exceed the RAM in the processor. So we can further subpartition the domain in the processor in smaller sub-subdomains.**
- **In fact, best efficiency is achieved with relatively small subdomains of 2,000-4,000 d.o.f.'s per subdomain.**



Example. Navier Stokes cubic cavity Re=1000

625,000 tetras mesh, rtol=10^{-4}, NS monolithic, [Tezduyar et.al. TET (SUPG+PSPG) algorithm, CMAME, vol. 95, pp. 221-242, (1992)]



Example. Navier Stokes cubic cavity Re=1000 (cont.)

- **Of course, each iteration of DDM/SCMI takes more time, but finally, in average we have**

$$\begin{aligned}\textbf{CPU TIME(DDM/SCMI)} &= \textbf{17.7 secs},\\ \textbf{CPU TIME(Global iteration)} &= \textbf{63.8 secs}\end{aligned}$$

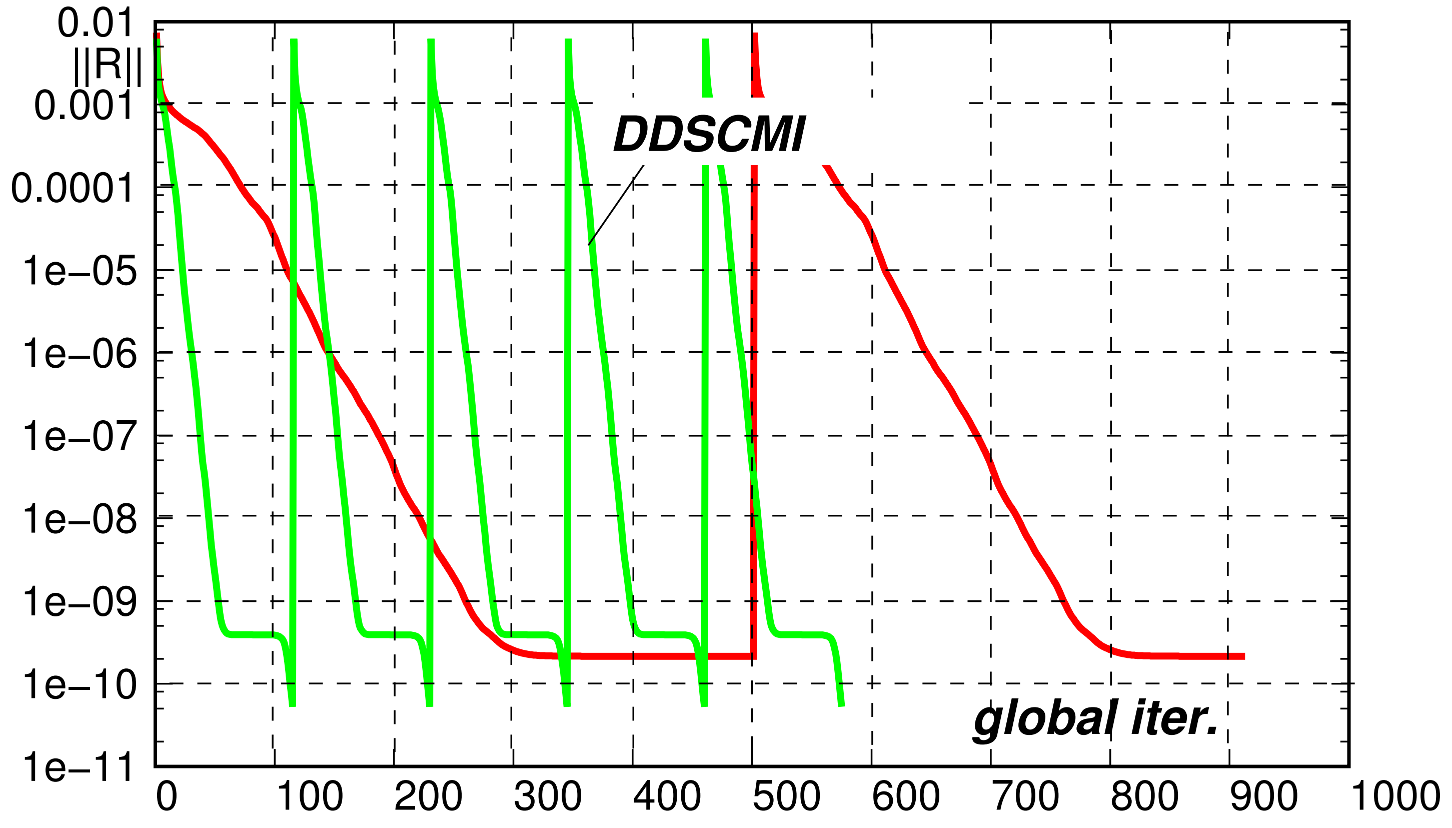
- **Residuals are on the interface for Schur compl, global for Global iter. But vector iteration for the SCM is equivalent to a global vector with null residual on the internal nodes. (So that they are equivalent).**
- **SCMI requires much less communication** ☺
- **Complete scalability study in**

Example. Navier Stokes cubic cavity Re=1000 (cont.)

- **V. Sonzogni, A. Yommi, N. Nigro, M. Storti, *A Parallel Finite Element Program on a Beowulf Cluster*; Advances in Engineering Software, vol 33, pp. 427-443 (2002)**
- **R.R. Paz, M.A. Storti, *An Interface Strip Preconditioner for Domain Decomposition Methods: Application to Hydrology*, International Journal for Numerical Methods in Engineering, 62(13), pp. 1873-1894 (2005)**

Example. Navier Stokes cubic cavity Re=1000 (cont.)

For a lower tolerance (rtol=10^{-8}), Global iteration breaks down, while DDM/SCMI continues to converge.



Example. Navier Stokes cubic cavity Re=1000 (cont.)

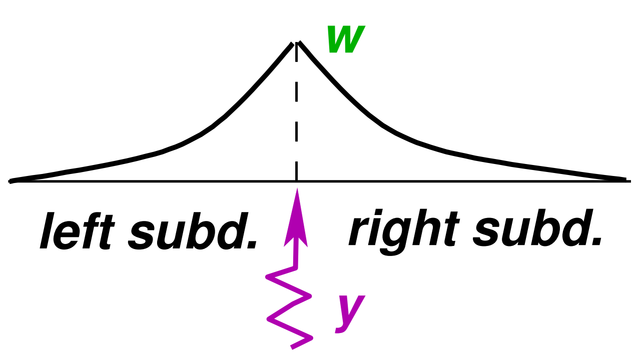
- **Following results obtained with Beowulf cluster Geronimo at CIMEC (see http://www.cimec.org.ar/geronimo). 8 x P4 1.7 Ghz (512MB RIMM (Rambus)).**
- **With DDM/SCMI we can run 2.2 Mtetras with NS monolithic (cubic cavity test) 252secs/iter (linear solution only, to rtol=10^{-4}).**
- **Global iteration crashes (out of memory!!) at iteration 122 having converged a factor 5×10^{-3}.**
- **DDM/SCMI is specially suited to stabilized methods, since they are in general worse conditioned. Also when Lagrange multipliers, penalization, strong Darcy terms are used.**

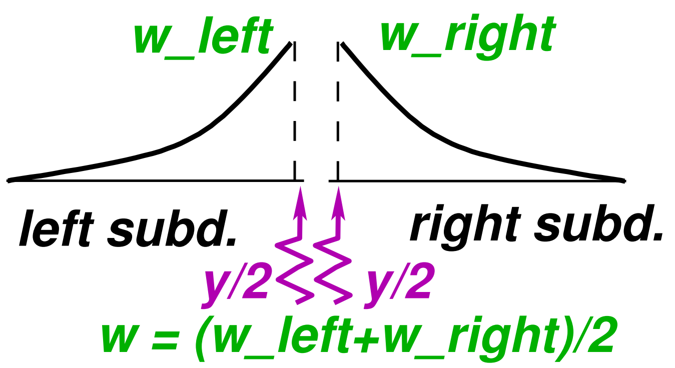
Fractional step solver

- **Predictor and projection steps have $\kappa = O(1)$ are better solved by global iteration (even Richardson).**
- **Poisson step can be solved with DDM/SCMI.** *[Some of the following observations apply in general for symmetric, positive definite operators.]*
- **Better conditioning (than monolithic) makes global iteration more appropriate ☹. As Conjugate Gradient can be used in place of GMRES, CPU time vs. iteration is no more quadratic ☹**
- **Factorized internal subdomain matrices can be stored and no re-factorized, so we can use larger internal subdomains (but requires more RAM) ☺**
- **We can solve 4.2 Mtetra mesh in 58sec (Poisson solution time only...) (rtol=10^{-4}, 130 iters.)**

Schur CM preconditioning - NN

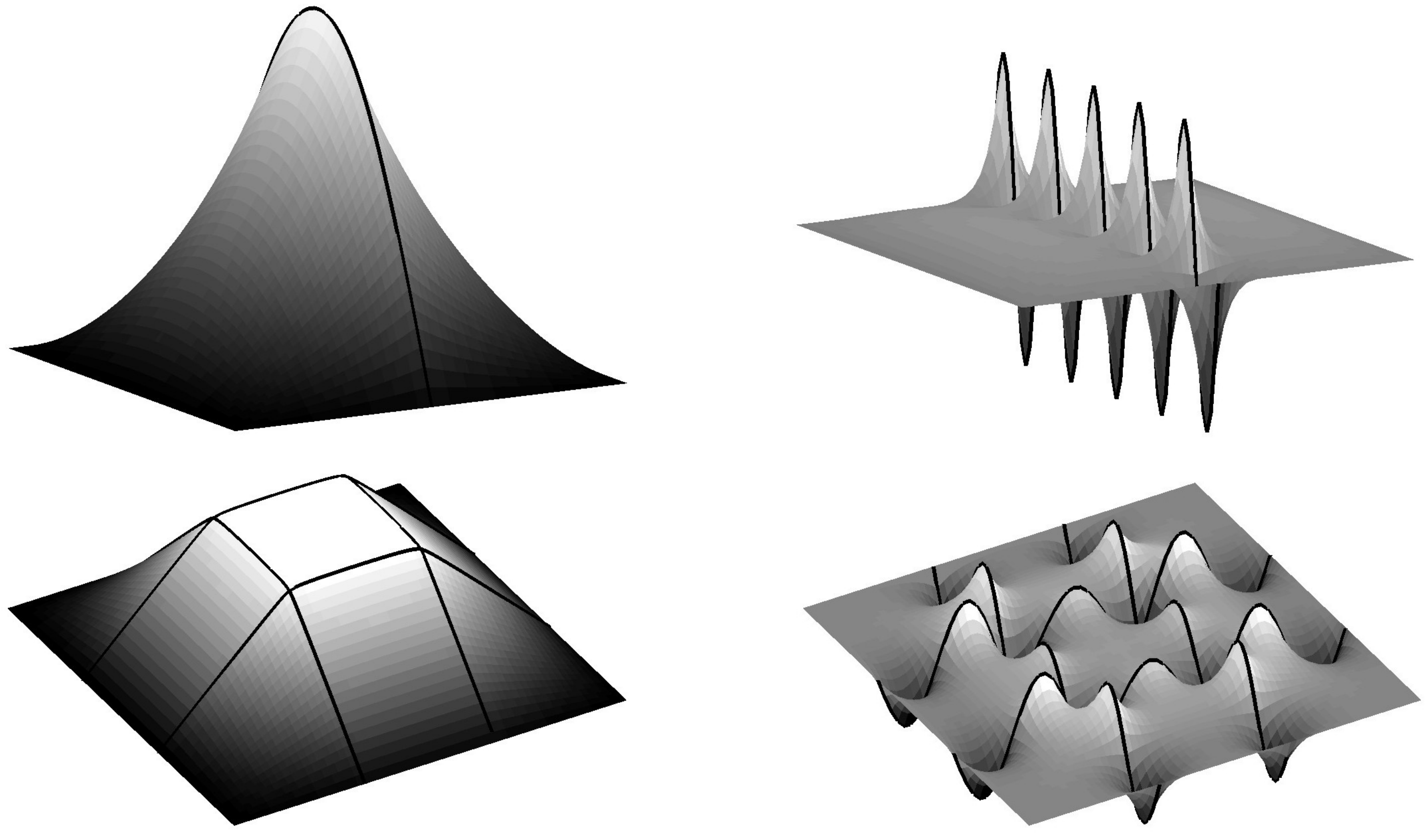
- **In order to further improve SCMI several preconditionings have been developed over years. When solving $\tilde{A}x = \tilde{b}$ a preconditioner should solve $\tilde{A}w = y$ for w in terms of y approximately.**
- **For the Laplace eq. this problem is equivalent to apply a *“concentrated heat flux”* (like a Dirac’s δ) y at the interface and solving for the corresponding temperature field. Its trace on the interface is w.**
- **Neumann-Neumann preconditioning amounts to split the heat flux one-half to each side of the interface (${}^1\!/_2 y$ for each subdomain).**





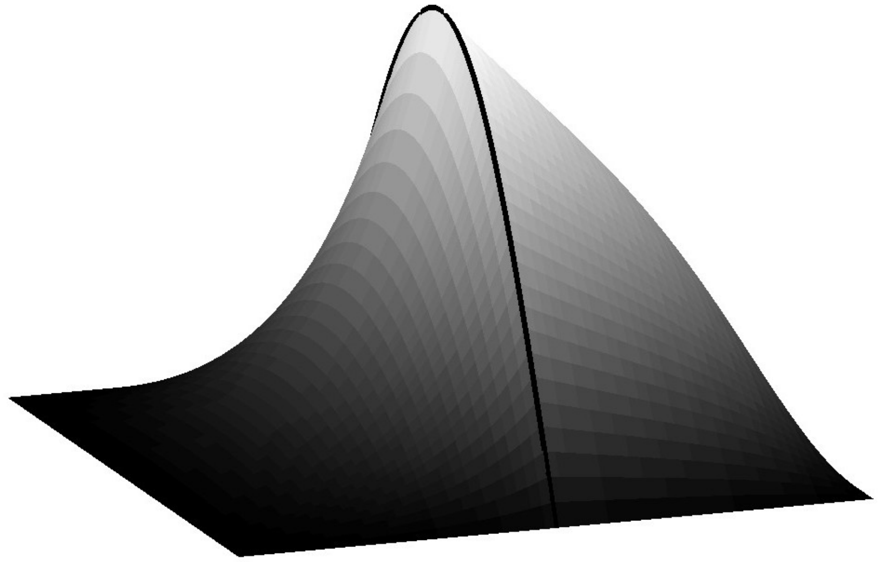
Flux splitting

Neumann-Neumann prec. works well whenever *“equal splitting”* is right: right subdomain equal to left subdomain, symmetric operator... Eigenfunctions of the Stekhlov operator (Schur comp. matrix) are symmetric.



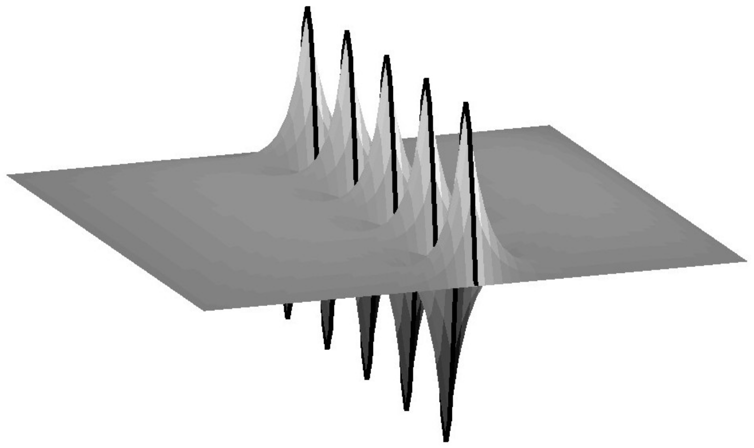
Flux splitting (advective case)

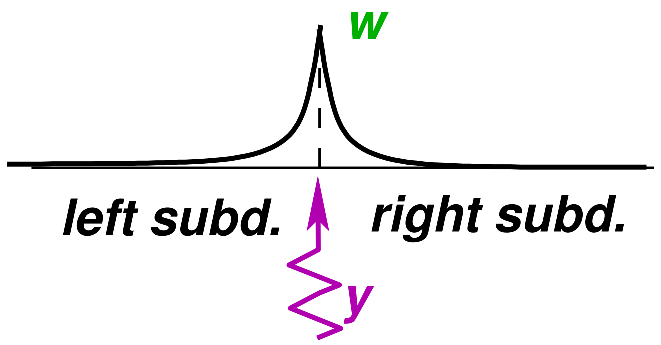
In the presence of advection splitting is biased towards the down-wind sub-domain. Eigenfunctions are no more symmetric.

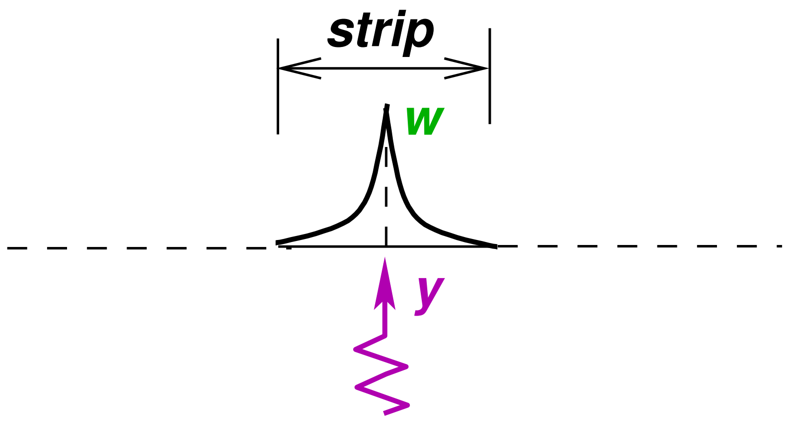


Interface strip preconditioner

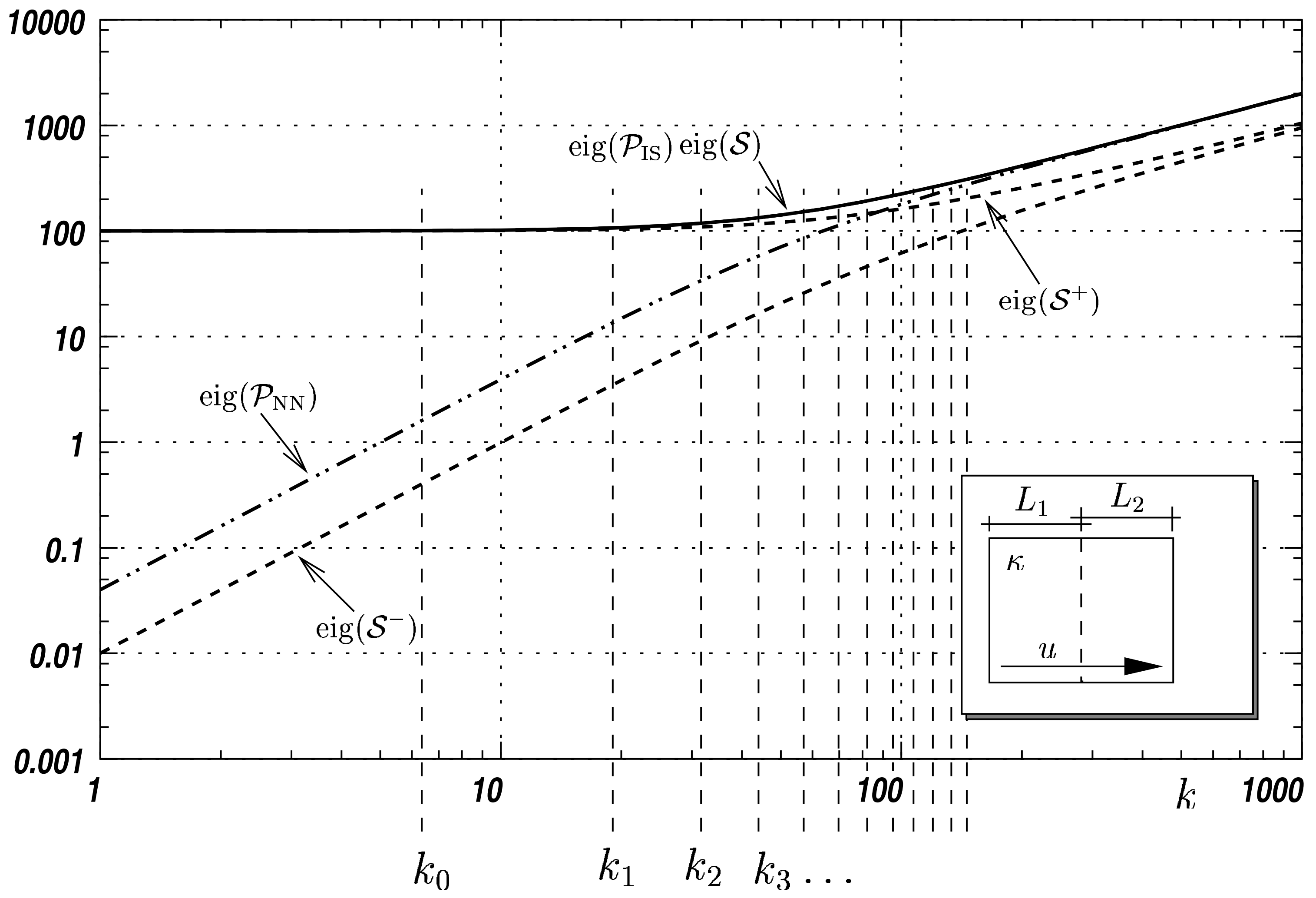
- **For high-frequency modes (high eigenvalues) the eigenfunctions concentrate near the interface.**
- **ISP: Solve the problem in a narrow strip around the interface**







Strongly advective case (Pe=50



Condition number, 50x50 mesh

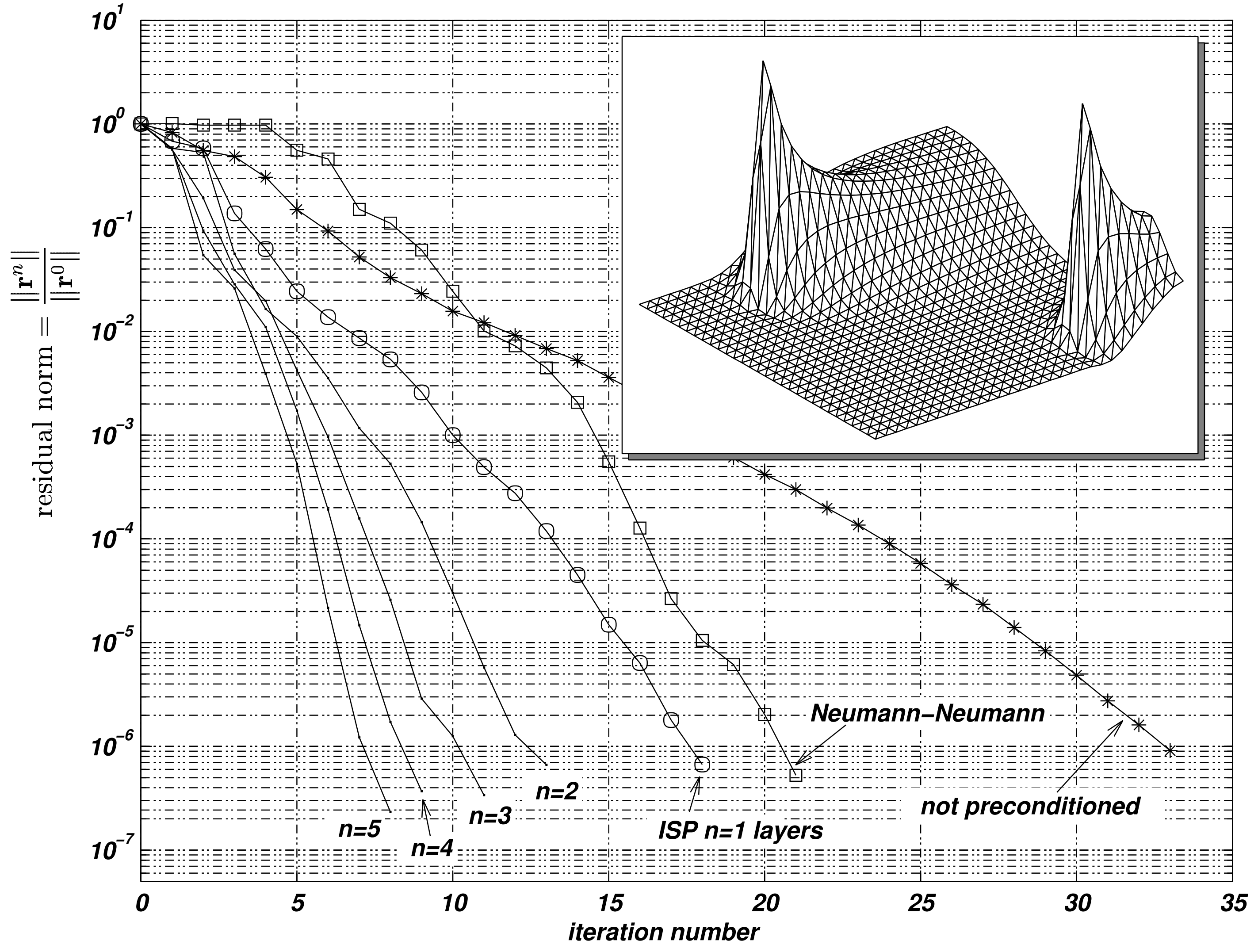
u	$\mathrm{cond}(\mathcal{S})$	$\mathrm{cond}(\mathcal{P}_{\mathrm{NN}}^{-1}\mathcal{S})$	$\mathrm{cond}(\mathcal{P}_{\mathrm{IS}}^{-1}\mathcal{S})$
0	**41.00**	**1.00**	**4.92**
1	**40.86**	**1.02**	**4.88**
10	**23.81**	**3.44**	**2.92**
50	**5.62**	**64.20**	**1.08**

Cuadro 1: Condition number for the Stekhlov operator and several preconditioners for a mesh of 50×50 elements.

Condition number, 100x100 mesh

u	$\mathrm{cond}(\mathcal{S})$	$\mathrm{cond}(\mathcal{P}_{\mathrm{NN}}^{-1}\mathcal{S})$	$\mathrm{cond}(\mathcal{P}_{\mathrm{IS}}^{-1}\mathcal{S})$
0	**88.50**	**1.00**	**4.92**
1	**81.80**	**1.02**	**4.88**
10	**47.63**	**3.44**	**2.92**
50	**11.23**	**64.20**	**1.08**

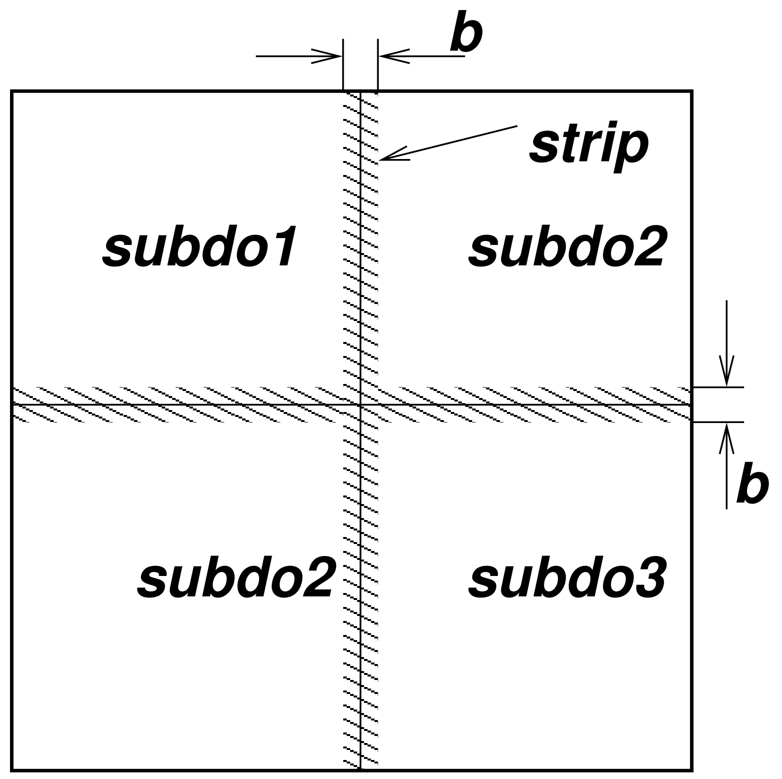
Cuadro 2: Condition number for the Stekhlov operator and several preconditioners for a mesh of 100×100 elements.



10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
residual norm $= \frac{\|\mathbf{r}^n\|}{\|\mathbf{r}^0\|}$
0
5
10
15
20
25
30
35
iteration number
Neumann−Neumann
not preconditioned
ISP n=1 layers
n=2
n=3
n=4
n=5

ISP features

- Better than NN for strong advective case ☺ (NN can improve splitting, but here splitting should depend on wave-length)
- No floating subdomains.
- Cost depends on strip width. Good conditioning can be obtained with thin strips ($\ll$NN) ☺
- Solution of strip problem should be done iteratively. Two possibilities: communicates (weakly coupled) or doesn't communicate (diagonal dominant).



ISP Implementation

- **ISP matrix is *“global”* (has elements in all processors): can’t be inverted with a direct solver.**
- **Iterative solution can’t use a CG/GMRES solver (can’t nest CG/GMRES inside CG/GMRES).**
- **FGMRES (Flexible GMRES) can be used.**
- **Use either preconditioned Richardson iteration or disconnect (i.e. modify) ISP matrix in some way in order to get a more disconnected matrix and use a direct method.**
- **Only preconditioned Richardson iteration is currently implemented in PETSc-FEM.**

Cubic cavity

- Use N^3 regular hexahedral mesh or $5N^3$ tetra mesh.
- Very dense connectivity (high LU band-width, strong 3D).
- Covers many real flow features (separation, boundary layers...)
- For N=30 (135Ktet), using 128 subdomains, 50 % savings in CPU time, 25 % savings in RAM.

N=30 cubic cavity stat.

nlay	iters
0	340
1	148
2	38

Conclusions

- **Domain Decomposition + iteration on the Schur complement matrix is an efficient algorithm for solving large linear systems in parallel or sequentially.**
- **Specially suited for ill-conditioned problems (stabilized methods, Lagrange multipliers, penalization, strong Darcy terms...)**
- **Interface strip preconditioner improves convergence, specially for advection dominated problems or floating subdomains.**

Acknowledgment

This work has received financial support from Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, Argentina, grants PIP 0198/98, PIP 02552/00, PIP 5271/05), Universidad Nacional del Litoral (UNL, Argentina, grants CAI+D 2000/43) and Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT, Argentina, grants PICT 6973/99, PID-74/99, PICT Lambda 12-14573/2003, PME 209/2003).

We made extensive use of *Free Software* (`http://www.gnu.org`) as GNU/Linux OS, MPI, GNU-Guile, Python, PETSc, GCC compilers, Octave, Open-DX among many others.