

Local context

The map illustrates the local context of Brazil, showing its geographical location and neighboring countries. The map is color-coded to distinguish between different regions or states. Key features include:

- Neighboring Countries:** Colombia, Venezuela, Ecuador, Peru, Bolivia, Paraguay, Argentina, and Uruguay.
- Major Cities:** Rio de Janeiro, São Paulo, Brasília, Porto Alegre, and others.
- Geographical Features:** The Atlantic Ocean (OCEANO ATLÂNTICO) to the east and south, and the Pacific Ocean (OCEANO PACÍFICO) to the west.
- Scale and Coordinates:** A scale bar indicates distances up to 500 km. Latitude and longitude lines are marked.

Local context

The map displays the geographical context of Brazil, highlighting its state boundaries and major cities. The Amazon region is shaded in green, the central region in yellow, and the southern region in blue. Neighboring countries include Venezuela, Colombia, Peru, and Argentina. The Atlantic Ocean is visible to the east and south. Key cities such as Manaus, Belém, Brasília, and São Paulo are labeled. The map also shows the Equator and the Tropic of Cancer.

The Parallel & Distributed Computing Group (GPPD)

- 4-5 Professors
 - A. Carissimi, C. Geyer, P. Navaux, N. Maillard, T. Divério
 - 1 postdoc, 12 PhD, 15 Master, plus under-grad. students.
- Working at all the stages of High-Performance Computing
 - Architecture (Navaux)
 - Programming (Maillard)
 - Management (Carissimi)
 - Scientific Computing (Divério)
 - Grid and P2P Comp. (Geyer)
 - Etc...



GPPD

Collaborations and Projects

- Academic:
 - INPE CPTEC, LAC (São Paulo);
 - UFRJ/COPPE (Rio de Janeiro);
 - UFSM, PUCRS, UNISINOS, UNISC, UCS, UCPel, etc... (Rio Grande do Sul)
- International:
 - INRIA (France): **Équipe Associée** since 2005/12
 - Pittsburgh (USA); Berlin
 - Portugal (Lisbon)
- Industrial:
 - Microsoft
 - Intel
 - Hewlett-Packard, DELL, etc...



GPPD



Interaction with Grid5000 (France)

- G5K = a grid of 9 sites, network 1 Gbs/s, 4800+ cores today.
 - Plus 2 sites (Luxembourg + Porto Alegre)
- A tool for Computer Scientists.
- Specific middleware:
 - OAR for resource allocation & control,
 - Kadeploy to deploy whatever image you want.
- Contacts with other grids – DAS (netherlands), NAREGI (Japan)
- **The GPPD is about to enter in this Grid.**



GPPD



A few UFRGS clusters

Corisco

Xirú

Labtec

XD1



GPPD



Architectures for HPC

- Network on Chip
 - How do you implement a hierarchy of memories inside a chip?
 - Henrique Freitas
- Cache Hierarchy and Multicore Chips
 - Marco Zanatta
- Clusters of GPUs.
 - Laercio Pilla

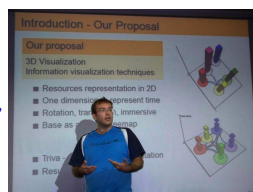


GPPD



Monitoring parallel applications

- Using 3D representations to visualize the behavior of parallel programs
 - Resources (x, y) vs. Time (z)
 - Rotations, zooms...
- Lucas M. Schnorr's PhD
 - Double diploma with G. Huard, Grenoble.



GPPD



A Race for Flops...

Since 2001, Japan's supercomputers have left the U.S. in the dust

SUPERCOMPUTER	PEAK SPEED IN TERAFLOPS**	LOCATION
MOLUCULAR DYNAMICS MACHINE*	79	Institute of Physical & Chemical Research (Riken)
GRAPE-6*	64	University of Tokyo
NEC EARTH SIMULATOR	41	Earth Simulator Center
HP ASCI Q	20.5	Los Alamos National Lab
CALIFORNIA DIGITAL CLUSTER*	20	Livermore/Livermore National Lab
APPLE CLUSTER	17.6	Virginia Tech
DELL CLUSTER	15	National Center for Supercomputing Applications
IBM ASCI WHITE	12	Livermore/Livermore National Lab
HP CLUSTER	11.6	Pacific Northwest National Lab
LINUX NETWORK CLUSTER	11	Los Alamos National Lab

... And Tera is just the Beginning

Power needed to complete most complex task in one week

TASK	PEAK SPEED
AUTOMOTIVE DEVELOPMENT	100 TERA FLOPS
HUMAN VISION SIMULATION	100 TERA FLOPS
AERODYNAMIC ANALYSES	1 PETA FLOPS
LASER OPTICS	10 PETA FLOPS
MOLECULAR DYNAMICS IN BIOLOGY	20 PETA FLOPS
AERODYNAMIC DESIGN	1 EXA FLOPS
COMPUTATIONAL COSMOLOGY	10 EXA FLOPS
TURBULENCE IN PHYSICS	100 EXA FLOPS
COMPUTATIONAL CHEMISTRY	1 ZETTA FLOP

China's Tianhe-1a achieves 1.27 PetaFLOPS in LINPACK. Detailed view: 1.27 PetaFLOPS (1.27 million times faster). A petaflop is 1,000,000,000,000,000 operations per second.

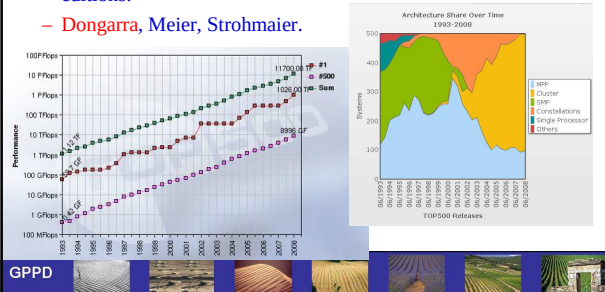
Source: Top500 Supercomputing List, June 2008. **Based on the speed of the fastest node in the system.

GPPD



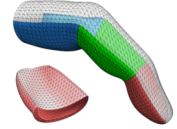
A look at the Top-500

- www.top500.org
 - Ranking of the 500 more powerful computers – 31 editions.
 - Dongarra, Meier, Strohmaier.



So how do you use such machines?

- **Distributed memory programming**
 - Unless you restrict to Multicore architectures.
 - You want to program a cluster of such chips.
- You need support for **heterogeneity**
 - Larger machines tend to lose homogeneity
 - How about CPUs + GPUS?
- You need support for **dynamism**
 - Either because of the platform dynamism
 - Failures, new resources entering...
 - Either because of the application dynamism
 - Local refinement of a mesh
 - Data dependencies
 - Sorting algorithm, for instance.



What do we have?

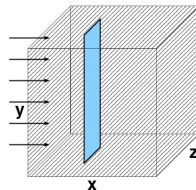
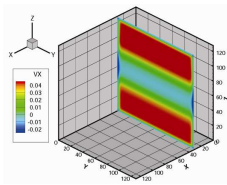
- **MPI, plus:**
 - Posix Threads,
 - OpenMP,
 - CUDA.
- MPI-2 provides some basic mechanisms to launch **new processes at run-time**
 - MPI_Comm_spawn
 - All major distributions have included MPI-2 since 2005.
- In general, the management of the dynamism is left to the programmer.
 - Implementation dependent.

Other solutions

- Coupling MPI + OpenMP/ Pthreads / CUDA
 - It is possible but awkward.
- **Adaptive MPI**
 - MPI interface on top of Charm++
 - A MPI task = 1 charm task = an object.
 - Facility to migrate a task.
 - Nothing for dynamism.
- Using **another Parallel API**
 - Task parallelism: Cilk (MIT), Satin (Amsterdam), Kaapi (Grenoble).
 - “Loop parallelism”: OpenMP, UPC, Fortress (Berkeley)
 - Great... But who is using them?

Typical case of “adaptive MPI program”

- A **fluid flows** in a square channel,
- There is an **obstacle** in the middle.
- How does it impact on the velocity/pression of the fluid?

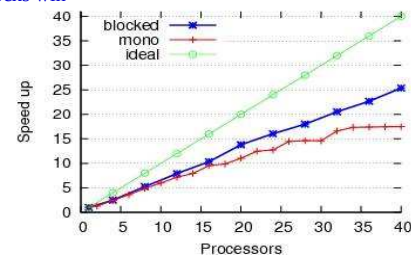


The adaptativity is provided by the **block size** of the mesh.

(All the study in SCHEPKE et al.,
Performance Improvement of the Parallel Lattice Boltzmann Method. SBAC '07)

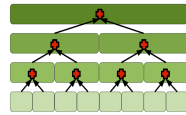
Performance vs. Block size [Schepke'07]

- The geometry and size of the block impacts on the performance.
 - The 3D blocks win



Using D&C with dynamic creation of processes

- You need to program in order to “enable” the dynamic creation of tasks.
 - Either you enable a dynamic mapping of the iterations (tasks) to the processes/threads
 - See OpenMP
 - Trouble with recursive parallelism
 - either loop parallelism is excluded
 - Use a more “functional” approach, or recursivity.
- Our solution (not original): use **Divide & Conquer**.
 - Think of Cilk, for instance.
 - Each “divide” phase enables either the **creation of parallel tasks**, either the sequential continuation of the computation.



GPPD



D&C and MPI-2

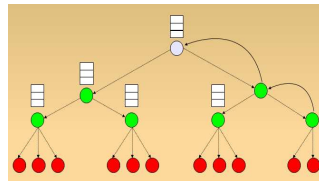
- Use **MPI_Comm_spawn** to (recursively) create new tasks.
 - 1 task == 1 (MPI) process.
- Two problems remain:
 - How do you **communicate** with these new processes?
 - The basic mechanism is MPI's inter-communicator.
 - It only works between parent and children processes.
 - Where do you **schedule** these new processes?
 - The basic mechanism is to run the new processes on the same CPU as the parent one.
 - This quickly overloads the original CPU.

GPPD



D&C with MPI - communication

- First, the communication are limited to parent/child relationship.
- Second, you need a hierarchy of inter-communicators to route the communication.
- Routing the communication this way also enables a distributed scheduling by **Work Stealing**.



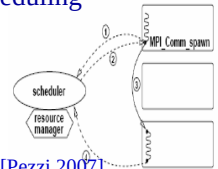
GPPD



D&C with MPI - scheduling

We have devised two solutions:

- Centralized algorithm** [Cera 2006]
 - A process that spawns children first interacts with a daemon.
- Distributed algorithm (Workstealing)** [Pezzi 2007]
 - Decouple the notion of task from the notion of process
 - The MPI process becomes a task.
 - Each processor owns a deque of tasks, and a worker process runs an active task.
 - When a processor turns idle the worker process **tries to steal tasks** from other deques.
 - The routing of the request is done by the tree of manager processes.

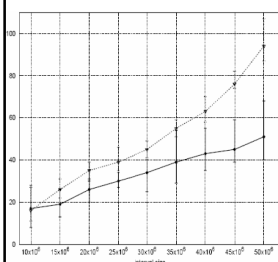


GPPD

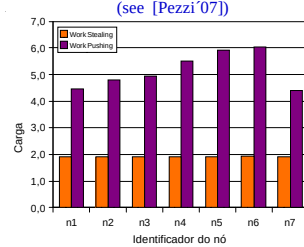


A few experimental results

- Centralized algorithm (see [Cera'06])



- Load balancing by Work Stealing (see [Pezzi'07])



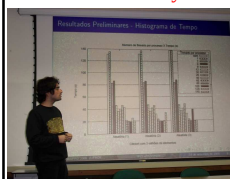
GPPD



Dynamic control of the granularity in MPI

- MPI_Comm_spawn starts tasks as new **heavy** processes.
- Various MPI implementations enable the mapping of tasks to threads, when you start the initial program.
 - You can even compile a MPI program to get a threaded program.

- What about **dynamically creating MPI tasks**, whether threads or processes, depending on the architecture and of the load of the nodes?



João Lima (MoS)

GPPD



Dynamic processes + threads

- Mergesort, $3 \cdot 10^9$ elements – 3 different input sets

