1. Einführung in die parallele Programmierung

- Übersicht
- Parallele Rechnerarchitekturen
- Parallele Programmiermodelle
- Parallele Algorithmen und ihre Analyse
- Paralleles Entwurfsmethodik

Wer bin ich ?

- Nicolas Maillard, nicolas@inf.ufrgs.br
- http://www.inf.ufrgs.br/~nicolas
- Franzose.
- Professor in der UFRGS Porto Alegre, Brasilien, seit 2004.
  - Betriebsystem
  - Compiler

Forschung

- High-Performance Computing (HPC)
  - Paralele Programmierung
  - Clusters & Grids
  - Process scheduling
- Grupo de Processamento Paralelo e Distribuído
  - Profs. Navaux, Geyer, Divério, Carissimi
- Current research projects:
  - National funding foundations (CNPq, CAPES, FINEP)
  - Microsoft (interoperability)
  - Intel
  - International partnerships:
    - » France, Germany, USA, Canada, etc...

Material on-line

- Handouts of the slides, links, etc...
- Web site:

  http://www.inf.ufrgs.br/~nicolas/teaching.en.html

Basic bibiography

- Parallel Algorithms:
  - Henri Casanova and Arnaud Legrand and Yves Robert, Parallel Algorithms, CRC Press, 2008.
  - J. Já, An Introduction to Parallel Algorithms, Addison-Wesley, 1992.
  - Karp e Ramachandran, Parallel Algorithms for Shared-Memory Machines, *Handbook of Theoretical Computer Science*, vol A, J. van Leeuwen Ed., MIT Press, 1990, pp. 869-941.
  - Foster, I. Designing and building parallel programs. Addison-Wesley, 1994, 379p.
- OpenMP:
  - Parallel Programming in OpenMP. R. Chandra *et al., Morgan* Kaufmann, 2001.

Bibliography (seq.)

- MPI:
  - Gropp, William *et al., Using MPI, MIT Press.*
  - Gropp, William *et al., Using MPI-2, MIT Press.*
  - Snir, M. *et al., Dongarra, J., MPI: The Complete Reference.*
  - http://www.mpi-forum.org
- MIT: http://supertech.csail.mit.edu/cilk/
- Berkeley: http://upc.lbl.gov/

Evaluation

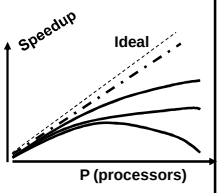
1. First 50% of the final mark: theoretical exam
2. Second 50% of the final mark: practical work.
   1. Pick-up a problem,
   2. Parallelize it!
      1. Report + source code + performance evaluation.
   3. Examples of problems:
      - » Π computation
      - » Sorting
      - » Distributed make
      - » Pattern matching
      - » Data compression (gzip, MP3, MPEG...)
      - » Graph computations
      - » Matrix coputations, etc...

Outline of today´s talk

1. What do you want?
2. What do you have: hardware
   Distributed vs. Shared memory, Clusters and Grids...
3. So, how do you program this?
   - Different Approaches
   - Shared memory
     - » Threads
   - Message Exchange
     - » MPI
4. What do you program?
   - Parallel complexity and algorithms
5. What do you get?
   - Performance evaluation

What do you want?

- A “good” parallel program...
  - Small runtime?
  - Big speed-up?
  - Scalable, i.e. efficient?
- The empirical approach will fail.
- Good programming:
  - Design -> choose the right algorithm
  - Implement it on a given architecture
  - Evaluate the performance you have got.



Beschleunigung durch parallele Programmierung

Das Gesetz von Amdahl

- Häufiges Ziel:
  - *Verbessern* einer existierenden Implementierung
- Maß: *Beschleunigung / Speedup*

$$Speedup = \frac{Ausführungszeit ohne Verbesserung}{Ausführungszeit mit Verbesserung}$$

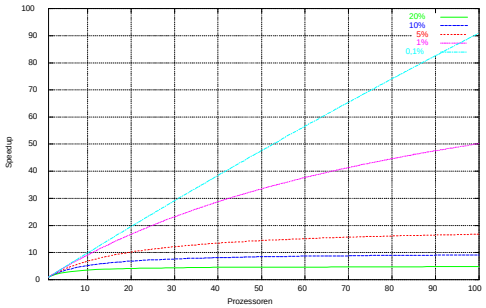
- bzw.: $$Speedup = \frac{Leistung mit Verbesserung}{Leistung ohne Verbesserung}$$

Entscheidend: Welcher Anteil der Implementierung ist von Beschleunigung betroffen?!

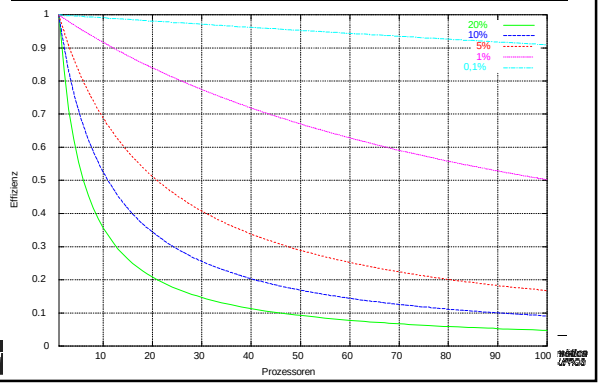
Amdahls Gesetz

- Gehe davon aus, Laufzeit des nicht-parallelen Programms ist 1.
- Beschleunigung des parallelen Programms ist durch den sequentiellen Anteil s = 1-p beschränkt
  - B(n) = 1/(p/n+s)
  - Im Grenzwert vieler Prozessoren: B → 1/s
- Konsequenz
  - Nur 1% Ein-/Ausgabe (s=0.01) beschränkt die Beschleunigung auf 100!
  - Oder: Effizientes Programm auf ES (5120 CPUs) muss sequentiellen Anteil unter 0,019% haben!

Amdahls Gesetz: Speedup

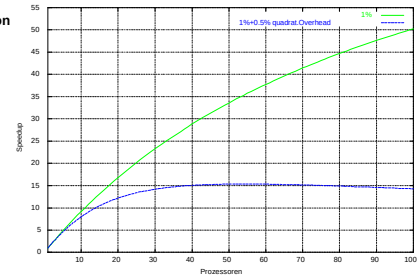


Amdahls Gesetz: Effizienz



Amdahls Gesetz in der Praxis

- Berücksichtigt keine Kosten für
  - Ein-/Ausgabe
  - Kommunikation
  - Balancierung
  - Scheduling

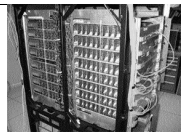


Outline of today´s talk

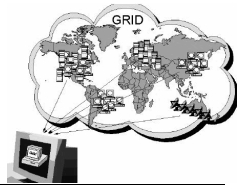
1. What do you want?
2. What do you have: hardware
   Distributed vs. Shared memory, Clusters and Grids...
3. So, how do you program this?
   - Different Approaches
   - Shared memory
     - » Threads
   - Message Exchange
     - » MPI
4. What do you program?
   - Parallel complexity and algorithms
5. What do you get?
   - Performance evaluation

What do you have? Parallel hardware

- Context: High-Performance Computing

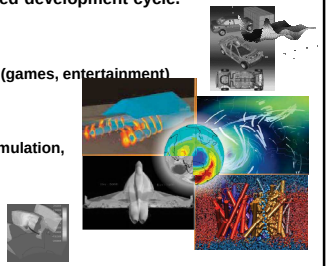


- From Clusters...
  - What are they
  - Management & Configuration
  - Programmation
- ... To Grids
  - What are they
  - Management & Configuration
  - Use



High-Performance Computing

- Which Applications? “Hard” problems require HPC (sciences, technoloy, business)
- Why? If you need an accelerated development cycle.
  - Aircraft Design,
  - Improvement in car industry,
  - Digital Imaging and animation (games, entertainment)
  - Proteine engineering,
  - Drug engineering,
  - Nano-technolgy design and simulation,
  - Stock market prevision,
  - Climate prevision.
  - Etc...

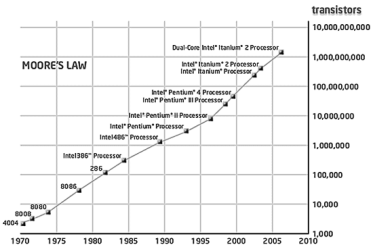


Personal Supercomputers?

	1991	1998	2005
System	Cray Y-MP C916	Sun HPC10000	Shuttle @ NewEgg.com
			
Architecture	16 x Vector 4GB, Bus	24 x 333MHz Ultra-SPARCII, 24GB, SBus	4 x 2.2GHz x64 4GB, GigE
OS	UNICOS	Solaris 2.5.1	Windows Server 2003 SP1
Performance	**~10 GFlops**	**~10 GFlops**	**~10 GFlops**
Top500 #	1	500	N/A
Price	**$40,000,000**	**$1,000,000 (40x drop)**	**< $4,000 (250x drop)**
Customers	Lab. Governo	Grandes Empresas	Engenheiros e Cientistas
Applications	Secretas, Clima, Pesquisa em Física	Manufatura, Energia, Finanças, Telecom	Bioinformática, Ciências, Mídia Digital

Why do you need parallel programming?

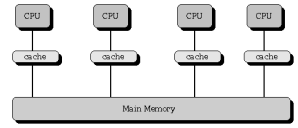
You need more power? Just wait a while... It will come for free!



But you can not always wait...

Shared Memory

- **All the CPUs access to a same memory**
  - **Solution that dates back to the 90´s, and much actual due to Multicore chips.**
- **Expensive solution, but easy to use!**
- **The shared bus supports up to a few tens of CPUs.**
  - **It is also a limiting factor**



Clusters – basic idea

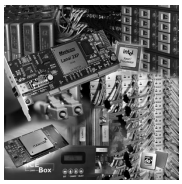
off-the-shelf CPU (?)

+ standard network (?) Gbit Ethernet, SCI, Myrinet, infiniband…

+ a few Operating System tricks

= cluster

- **Thousands of CPUs**
- **"cheap"**
- **"home-made"**



Some BIG machines

Deep Blue

Earth Simulator

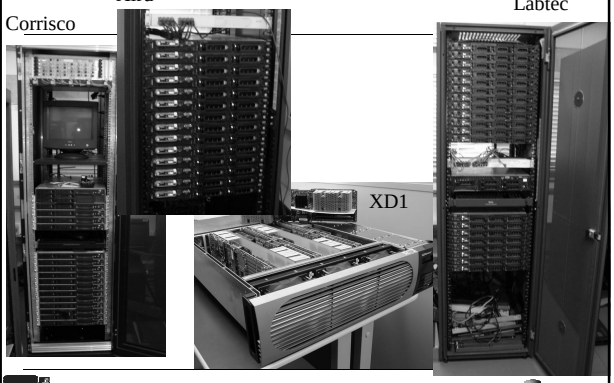
A few UFRGS clusters

Corrisco

Xirú

Labtec

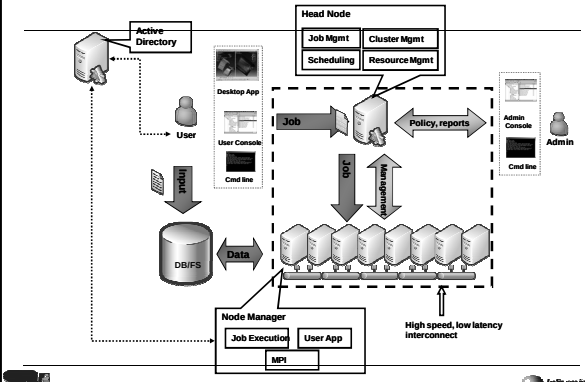
XD1



Architecture of a Cluster

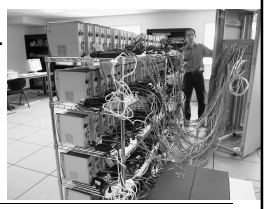
- **A cluster = a set of nodes**
  - **PC, dedicated CPUs...**
- **A special node serves as front-end**
  - **Interface with the users,**
  - **Disc server,**
  - **Controls the acess to other (compute) nodes,**
  - **HTTP server,**
  - **Hosts the compilers...**
- **The other (many) nodes are used for computation**
  - **Reduced version of the O. S.,**
  - **The users can not access them directly,**
  - **May use specific network,**
  - **Exclusive use.**

Example: Microsoft Compute Cluster Solution



Installing a cluster

1. Install (linux, for instance) in the front-end.
   1. Create a dir. /home and the user logins,
   2. Install the compilers, sshd, nfsd, and software to control the access to the compute nodes.
      - » BRAMS, INTEL´s MKL, LAPACK, etc...
      - » OAR
2. Configure a compute node.
3. Clone this image to the other nodes.
   - Kadeploy
4. Set up the network.
   1. Only the front-end needs an access to the external network.
5. Export /home to the nodes..



The Grid

In 1999 appears a new idea:

"A Computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities." [Foster:99]

- **CPU power = Electrical power**
- **"Internet Computing"**

From The Grid to the grids

- **The Grid has soon turned into many grids:**
  - **Computing Grids**
    - » Cluster of clusters
    - » Concerns: Performance
  - **Data Grids**
    - » The point is the amazing quantity of data that is generated/analized.
    - » Concerns: storage, network, data formats.
  - **Service Grids**
    - » Utility Servers: a provider supplies IT services.
    - » Concerns: service spec, quality of service, Grid economics
  - **Besides, there are different interests depending on the scientific community**
    - » Grid as large-scale production tool, vs.
    - » Grid as a large-scale experiment tool.

Middleware for the Grid

- **Globus: a set of standards, and implementations:**
  - **Open Grid Services Architecture (OGSA)**
  - **Web Services Resource Framework (WSRF)**
  - **Job Submission Description Language (JSDL)**
  - **Etc...**
  - **Rather heavy to setup and use...**
- **gLite**
  - **Description of the Virtual Orgs, the sites, the users, the jobs, etc...**
- **BOINC (Berkeley)**
  - **Base of SETI@home**
  - **Client/server for volunteer computing**
- **OurGrid,**
  - **Grid & P2P computing – current integration with gLite**
- **OAR,**
- **And many other... (ARC, Advanced Resource Connector)**

Enabling Grids for E-SciencE (EGEE)

- **European project, started in 2004.**
- **260 sites, 50+ countries,**
- **79,000+ CPU cores.**
- **Initially started to treat the data from the Large Hadron Collider (CERN) – Particles with High Energy.**

- **Based on gLite.**

A Race for **Flops**...

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

SUPERCOMPUTER	PEAK SPEED IN TERAFLOPS**	LOCATION
MOLECULAR DYNAMICS MACHINE*	78	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	Lawrence Livermore National Lab
APPLE CLUSTER	17.6	Virginia Tech
DELL CLUSTER	15	National Center for Supercomputing Applications
IBM ASCI WHITE	12	Lawrence Livermore National Lab
HP CLUSTER	11.6	Pacific Northwest National Lab
LINUX NETWORX CLUSTER	11	Los Alamos National Lab

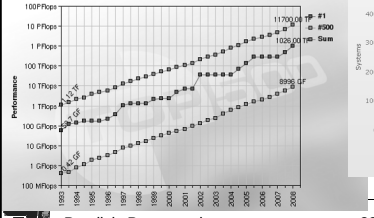
...And **Tera** is Just the Beginning

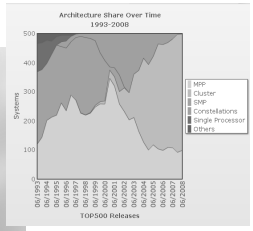
Power needed to complete most complex task in one week

TASK	PEAK SPEED
AUTOMOTIVE DEVELOPMENT	100 TERAFLOPS
HUMAN VISION SIMULATION	100 TERAFLOPS
AERODYNAMIC ANALYSES	1 PETAFLOPS
LASER OPTICS	10 PETAFLOPS
MOLECULAR DYNAMICS IN BIOLOGY	20 PETAFLOPS
AERODYNAMIC DESIGN	1 EXAFLOPS
COMPUTATIONAL COSMOLOGY	10 EXAFLOPS
TURBULENCE IN PHYSICS	100 EXAFLOPS
COMPUTATIONAL CHEMISTRY	1 ZETTAFLOP

A look at the Top-500

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





What are the problems?

- **Everything is difficult with such machines**
  - **For instance, think about the time it takes to boot 100.000 PCs...**
  - **Access, configuration, manutention of the software, security policy...**
    - » **Heat, weight, space, cables...**
  - **Monitoring & debugging**
  - **Porting the programs**
  - **Life cycle of the programs...**
- **Programming such platforms is hard.**

Outline of today´s talk

1. **What do you want?**
2. **What do you have: hardware**

   Distributed vs. Shared memory, Clusters and Grids...
3. **So, how do you program this?**
   - **Different Approaches**
   - **Shared memory**
     - » **Threads**
   - **Message Exchange**
     - » **MPI**
4. **What do you program?**
   - **Parallel complexity and algorithms**
5. **What do you get?**
   - **Performance evaluation**

How do you program THAT?

A few approaches (more to come in the next lectures):

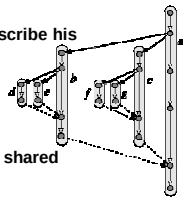
1. **Provide a high-level abstraction – e.g. the Object (Java/Corba)**
   - **High-level, nice, robust...;**
   - **How about performance?**
2. **Let the compiler do the work...**
   - **Automatic parallism**
1. **Have the programmer define n executing flows of instructions:**
   - **With (possibly virtually) shared memory – Threads**
   - **With message passing (distributed memory)**

First approach – Distributed Objects

- **Virtualize the HW and OS with a Virtual Machine;**
- **All data and processing is hidden into the Object;**
- **Publish / Discover the (remote) methods provided by a clas**
- **Remote Method Invocation**
  - **The data exchange relies on serialization!**
- **Very easy to program, to port to heterogeneous environments... But with high overhead!**
  - **Nice for Ubiquous Computing, Pervasive Computing, ...**
  - **Examples: Ibis (H. Bal/Amsterdã); ProActive (INRIA/France)**

2nd approach: let the compiler make magic.

- Automatic parallelization: very hard.
  - The loop dependencies are hard to compute...
  - ... And most of the time they can only be resolved at runtime.
- Current nice solution: let the programmer describe his (sequential) program with parallel directives:
  - OpenMP (static)
  - Cilk (dynamic) – MIT project.
- Sadly, it only provides good performance for shared memory machines...

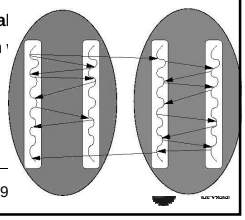


Some OpenMP code

Sequential code	Parallel Open-MP code
double res[10000];	double res[10000];
	#pragma omp parallel for
for (i=0 ; i<10000; i++) compute(res[i]);	for (i=0 ; i<10000; i++) compute(res[i]);

Third approach – Communicating processes

- Extensions of classical sequantial languages
  - Thread libraries (e.g. Posix Threads)
    - » Define, create and synchronize threads.
  - Message exchange libraries
    - » PVM, MPI.
    - » A library to extend C/Fortran programs.
    - » Many open-source and commercial
- Write one unique program, which will run
  - Each process runs the same program,
  - Each process has its own rank (id),
  - Each process may send/receive data to/from the others.



MPI in 6 words

1) MPI_Init(&argc, &argv) // No comment.
2) MPI_Comm_rank(&r, communicator)
   // returns the rank in the var. int ´r´
3) MPI_Comm_size(&p, communicator)
   // returns the number of processes in the var. int ´p´
4) MPI_Send(/* a bunch of args */)
5) MPI_Recv(/* almost the same bunch of args */)
6) MPI_Finalize() // No comment

The basic communicator is MPI_COMM_WORLD.

Programming with MPI

p processes interact through messages.

Programming with MPI

p processes interact through messages.

```
void main() {
  int r, tag = 103;
  MPI_Init(...) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &r);
  if (r==0) {
     val = 3.14 ;
     MPI_Send(&val,..., 2, tag, ...);  }
  if (r==2)
     MPI_Recv(&val, ..., 0, tag, ...);
```

Process 0

```
void main() {
  int r, tag = 103;
  MPI_Init(...) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &r);
  if (r==0) {
     val = 3.14 ;
     MPI_Send(&val,..., 2, tag, ...);  }
  if (r==2)
     MPI_Recv(&val, ..., 0, tag, ...);
```

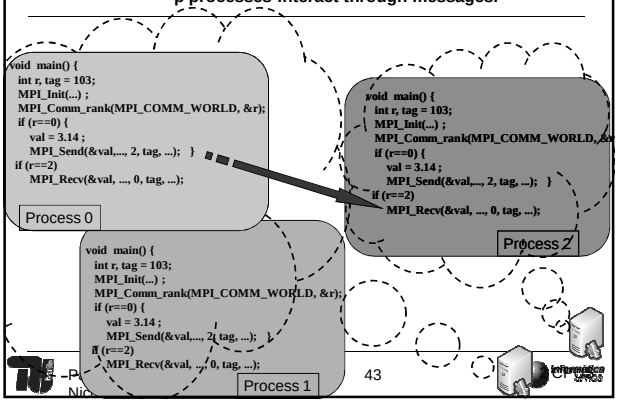
Process 1

```
void main() {
  int r, tag = 103;
  MPI_Init(...) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &r);
  if (r==0) {
     val = 3.14 ;
     MPI_Send(&val,..., 2, tag, ...);  }
  if (r==2)
     MPI_Recv(&val, ..., 0, tag, ...);
```

Process 2

Programming with MPI

p processes interact through messages.



Conclusion about Parallel Programming

- There is no unique API for parallel programming.
- There are many solutions:
  - MPI / Open-MP
  - These two are usable for production code.
- What exists is hard to use when you need:
  - dynamicity
  - heterogeneity
- The research has focused on these two points.
  - MPI-2, Cilk, Charm++, Satin/Ibis, Nesl, UPC, etc...

Outline of today´s talk

1. What do you want?
2. What do you have: hardware
   Distributed vs. Shared memory, Clusters and Grids...
3. So, how do you program this?
   - Different Approaches
   - Shared memory
     - » Threads
   - Message Exchange
     - » MPI
4. What do you program?
   - Parallel complexity and algorithms
5. What do you get?
   - Performance evaluation

What do you program?

- "Complexity" = metrics to evaluate the quality of your program.
- It depends on:
  - The model of the algorithm and of the program:
    - » Data, computation, memory usage, for instance.
  - The model of the host machine:
    - » Processor(s), memory hierarchy, network...?
- In the sequential case, everything´s fine!
  - Von Neumann model – fetch & run cycles.
  - Turing machine...
  - A very SIMPLE model enables the correct prediction and categorization of any algorithm.

What do you expect from a model?

- A model has to be:
  - extensive:
    - » Many parameters – in general, it ends up in a complx system.
    - » These parameters reflect the program/machine.
  - Abstract
    - » i.e. generic
    - » You do not want to change your model each 18 months (see Morore´s law)
    - » You want the general trend, not the details.
  - Predictive
    - » So it must lead to something you can calculate on.
    - » (It does not mean that it must be analytical)

In the parallel world...

- There is no universal model. ⊗
- There are many models
  - For each type of machine, and many types of programs.
- You have no simple "reduction" from a model for distributed memory machine to a model for shared memory machine.
  - Because of the network model...
- Most theoretical models have been obtained with shared memory models.
  - Much simples, less parameters.
  - Scalability limited!

Basic ideas for the machine model

- Disconsider the communication (PRAM)
  - Adapted for shared memory machines, multicore chips...
- Consider a machine as being homegeneous, static, perfectly interconnected, with zero latency, and a fixed time for message passing (delay model).
- Consider a homogeneous, static machine, with a network that has latency and/or a given bandwith (LogP)
  - Okay for a cluster
- Considerar a dynamic, heterogeneous machine (Grid)...
  - No one knows hot to model this.

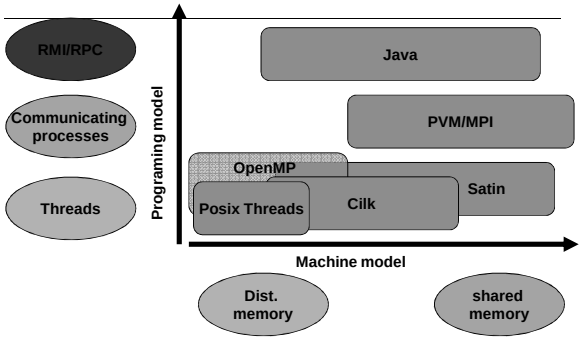
Parallel program model

- How do you describe a parallel program?
- Task parallelism:
  - The program is made of tasks (sequential unit);
  - The tasks are (partially) ordered by dependencies
  - A correct execution if a (possibly parallel) schedule which respects the dependencies.
  - Very close to functionnal programming.
- The dependencies can be explicit (e.g.: depend on messages) or implicit (e.g.: arguments).
- Trivial case: no dependencies ("embarassingly parallel" (EP), "parameter sweeping", "task farm", "master/slave", "client/server", "bag of tasks",...
- More complex case: Divide & Conquer.
  - The dependencies order the tasks in a tree-like way.

Parallel program model

- Data Parallelism
  - Distribute the data, and each process/thread computes on its local data.
    - » Owner Compute Rule
  - *Single Program Multiple Data*
- Loop parallelism
  - Comes from the Compiler world
  - Just tell which iterations of a loop can be performed in parallel.
- *Templates* para programação paralela
  - Provides skeletons (frameworks).

Programming Model vs. Machine Model



Outline of today´s talk

1. What do you want?
2. What do you have: hardware
   Distributed vs. Shared memory, Clusters and Grids...
3. So, how do you program this?
   - Different Approaches
   - Shared memory
     - » Threads
   - Message Exchange
     - » MPI
4. What do you program?
   - Parallel complexity and algorithms
5. What do you get?
   - Performance evaluation

Performance evaluation

- A parallel program is intrinsically non-deterministic
  - The order of execution of the tass may change from execution to execution
  - The network (if any) adds its part of random.
- You are interested in runtime.
  - The usual argument "I compiled it, therefore the program is okay" does not serve at all!
- It is mandatory to use statistical measurements:
  - At least: x runs (x=10,20, 30...), and mean, min. and max. Runtime (or speedup, or efficiency) indicated.
  - Better: x runs, mean and standard deviation
    - » If the standard dev. is high, run it more – or ask yourself if there is something wrong...
  - Event better: x runs, confidence interval about the mean.

That´s all, folk!

- **Tomorrow: Basic parallel algorithms, PRAM complexity, an introduction.**
- **The slides of today´s lecture are on:**

http://www.inf.ufrgs.br/~nicolas/teaching.en.html