Wer bin ich?

- Nicolas Maillard, nicolas@inf.ufrgs.br
- http://www.inf.ufrgs.br/~nicolas
- Franzose.
- Professor in der UFRGS
  - Betriebssystem
  - Compiler

Material on-line

- Handouts of the slides, links, etc...
- Web site:
  http://www.inf.ufrgs.br/~nicolas/teaching.en.html

Basic bibliography

- Parallel Algorithms:

- 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.
  - 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.
3. Pick-up a problem,
4. Parallelize it!
5. Examples of problems:
   - Pi computation
   - Sorting
   - Distributed make
   - Pattern matching
   - Data compression (gzip, MP3, MPEG...)
   - Graph computations
   - Matrix computations, 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 \rightarrow 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
  - Ausführungszeit
    - Verbesserung
    - Leistung
  - Leistung
  - Leistung
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
  - \( B(n) = 1/(p+ns) \)
  - Im Grenzwert vieler Prozessoren: \( B \rightarrow 1/s \)
- 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: Effizienz

Amdahls Gesetz in der Praxis

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
  - Programming

• ... To Grids...
  - What are they
  - Management & Configuration
  - Use

High-Performance Computing

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

Personal Supercomputers?

<table>
<thead>
<tr>
<th>1991</th>
<th>1998</th>
<th>2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Cray Y-MP C916</td>
<td>Sun HPC10000 Shuttle</td>
</tr>
<tr>
<td>Architecture</td>
<td>In a Box/ knoll</td>
<td>Sun 20K 10/800</td>
</tr>
<tr>
<td>DX</td>
<td>4-12,570 MIPS</td>
<td>24 L 12,000 Hertz</td>
</tr>
<tr>
<td>Performance</td>
<td>10 1.0 Ghz</td>
<td>10 1.0 Ghz</td>
</tr>
<tr>
<td>Price</td>
<td>140,000.000</td>
<td>11,100,000 (26% drop)</td>
</tr>
<tr>
<td>Customers</td>
<td>IBM, Boeing</td>
<td>Lockheed, Boeing</td>
</tr>
<tr>
<td>Applications</td>
<td>Aerospace, Computers, Financial, Telecommunications, Medicine, Engineering, Agriculture</td>
<td>Financial, Telecommunications, Medicine, Engineering, Agriculture</td>
</tr>
</tbody>
</table>

Parallel Programmierung
Nicolas Maillard, Marcus Ritt

13/10/2008
Some BIG machines

- Earth Simulator
- Deep Blue

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”

Architecture of a Cluster

- A cluster = a set of nodes
  - PC, dedicated CPUs...
- A special node serves as front-end
  - Interfaced with the users,
  - Disc server,
  - Controls the access 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.
Parallele Programmierung
Nicolas Maillard, Marcus Bii

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
  – Concern: Performance
  – Data Grids
  – The point is the amazing quantity of data that is generated/unalyzed.
  – Concern: storage, network, data formats.
  – Service Grids
  – Utility Services: a provider supplies IT services.
  – Concern: service spec, quality of service, Grid economics

– Besides, there are different interests depending on the scientific community
  – Grid as a 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 Orga, 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)

Installing a cluster
1. Install (linux, for instance) in the front-end.
2. Create a dir. .home and the user login.
3. Install the compilers, gsh, rsh, and software to control the access to the compute node.
  – BRAMS, INTEL MKL, LAPACK, etc...
  – OAR
4. Configure a compute node.
5. Clone this image to the other nodes.
6. Kadeploy
7. Set up the network.
8. Only the front-end needs an access to the external network.
9. Export .home to the nodes...

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.
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
   - Threads
   - Message Exchange
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 parallelism

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 class
- 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...

Third approach – Communicating processes

- Extensions of classical sequential languages
  - Thread libraries (e.g. Posix Threads)
  - Define, create and synchronize threads.
  - Message exchange libraries
    - PVM, MPI.
    - A library to extend C/Fortran programs.
- 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.

Some OpenMP code

Sequential code

```c
double res[10000];
for (i=0 ; i<10000; i++)
  compute(res[i]);
```

Parallel Open-MP code

```c
#pragma omp parallel for
for (i=0 ; i<10000; i++)
  compute(res[i]);
```

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.

`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`
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 complex system.
    - These parameters reflect the program/machine.
  - Abstract
    - I.e. generic
    - You do not want to change your model each 18 months (see Moore’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.
  - Very 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 homogeneous, 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 bandwidth (LogP)**
  - Okay for a cluster
- Consider a dynamic, heterogeneous machine (Grid)...  
  - No one knows how 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 functional programming.
- The dependencies can be explicit (e.g.: depend on messages) or implicit (e.g.: arguments).
- 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).

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: > 3 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