NUMERICAL PARALLEL COMPUTING

Lecture 1, March 23, 2007: Introduction

Peter Arbenz

Institute of Computational Science, ETH Zürich
E-mail: arbenz@inf.ethz.ch

http://people.inf.ethz.ch/arbenz/PARCO/
Organization: People/Exercises

1. **Lecturer:**
   Peter Arbenz, CAB G69.3 (Universitätsstrasse 6)
   Tel. 632 7432
   Lecture: Friday 8-10 CAB G61
   arbenz@inf.ethz.ch

2. **Assistant:**
   Marcus Wittberger, CAB G65.1, Tel. 632 7547
   marcus.wittberger@inf.ethz.ch
   Group 1: Friday 14-16 IFW D31

3. **Assistant:**
   Cyril Flaig, CAB F63.2, Tel. 632 xxxx
   cyril flaig@inf.ethz.ch
   Group 2: Friday 10-12 IFW D31
What is parallel computing [in this course]

A parallel computer is a collection of processors, that can solve big problems quickly by means of well coordinated collaboration.

Parallel computing is the use of multiple processors to execute different parts of the same program concurrently or simultaneously.
An example of parallel computing

Assume you are to sort a deck of playing cards (by suits, then by rank). If you do it properly you can complete this task faster if you have people that help you (parallel processing). Note that the work done in parallel is not smaller than the work done sequentially. However, the solution time (wall clock time) is reduced.

Notice further that the helpers have to somehow communicate their partial results. This causes some overhead. Clearly, there may be too many helpers (e.g. if there are more than 52). One may observe a relation of speedup vs. number of helper as depicted in Fig. 1 on the next slide.
Figure: Sorting a deck of cards
Why parallel computing

▶ **Runtime**
want to reduce wall clock time [in e.g. time critical applications like weather forecast].

▶ **Memory space**
Some large applications (*grand challenges*) need a large number of degrees of freedom to provide meaningful results. [Reasonably short time step, discretization has to be sufficiently fine, c.f. again weather forecast]
A large number of small processors probably has a much bigger (fast) memory than a single large machine (PC cluster vs. HP Superdome)
The challenges of parallel computing

Idea is simple: Connect a sufficient amount of hardware and you can solve arbitrarily large problems. (Interconnection network for processors and memory?)
BUT, there are a few problems here...

Let’s look at the processors. By Moore’s law the number of transistors per square inch doubles every 18 – 24 months, cf. Fig. 2.

Remark: If your problem is not too big you may want to wait until there is a machine that is sufficiently fast to do the job.
Figure: Moore’s law
Moore's Law for Power

- Chip Maximum Power in watts/cm²
- Not too long to reach Nuclear Reactor
- Itanium - 130 watts
- Pentium 4 - 75 watts
- Pentium III - 35 watts
- Pentium II - 35 watts
- Pentium Pro - 30 watts
- Pentium - 14 watts
- I486 - 2 watts
- I386 - 1 watt

Source: Fred Pollack, Intel. New Microprocessor Challenges in the Coming Generations of CMOS Technologies, MICRO32 and Transmeta
How did this come about?

- Clock rate (+30% / year)
  - increases power consumption
- Number of transistors (+60-80% / year)
  - Parallelization at bit level
  - Instruction level parallelism (pipelining)
  - Parallel functional units
  - Dual-core / Multi-core processors
Instruction level parallelism

Figure: Pipelining of an instruction with 4 subtasks: fetch(F), decode(D), execute(E), write back(W)
Some superscalar processors

<table>
<thead>
<tr>
<th>processor</th>
<th>isuable instructions</th>
<th>clock rate (MHz)</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>ALU</td>
<td>FPU</td>
</tr>
<tr>
<td>Intel Pentium</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Intel Pentium II</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Intel Pentium III</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Intel Pentium 4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>AMD Athlon</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>AMD Opteron</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

ALU: integer instructions; FPU floating-point instructions; LS: load-store instructions; B: branch instructions; clock rate at time of introduction.

[Rauber & Rünger: Parallele Programmierung]
One problem of high performance computing (and of parallel computing in particular) is caused by the fact that the access time to memory has not improved accordingly, see Fig. 4. Memory performance doubles in 6 years only [Hennessy & Patterson]

**Figure:** Memory vs. CPU performance
To alleviate this problem, memory hierarchies with varying access times have been introduced (several levels of caches). But the further away data are from the processor, the longer they take to get and store.

Data access is everything in determining performance!

Sources of performance losses that are specific to parallel computing are

- **communication overhead**: synchronization, sending messages, etc.  
  (Data is not only in the processor’s own slow memory, but even on a remote processor’s own memory.)

- **unbalanced loads**: the different processors do not have the same amount of work to do.
Outline of the lecture

- Overview of parallel programming, Terminology
- SIMD programming on the Pentium (Parallel computers are not just in RZ, most likely there is one in your backpack!)
- Shared memory programming, OpenMP
- Distributed memory programming, Message Passing Interface (MPI)
- Solving dense systems of equations with ScaLAPACK
- Solving sparse systems iteratively with Trilinos
- Preconditioning, reordering (graph partitioning with METIS), parallel file systems
- Fast Fourier Transform (FFT)
- Applications: Particle methods, (bone) structure analysis

For details see PARCO home page.
Exercises’ Objectives

1. To study 3 modes of parallelism
   ▶ Instruction level (chip level, board level, ...). SIMD
   ▶ Shared memory programming on ETH compute server. MIMD
   ▶ Distributed memory programming on Linux cluster. MIMD

2. Several computational areas will be studied
   ▶ Linear algebra (BLAS, iterative methods)
   ▶ FFT and related topics (N-body simulation)

3. Models and programming (Remember Portability!)
   ▶ Examples will be in C/C++ (calling Fortran routines)
   ▶ OpenMP (HP Superdome Stardust/Pegasus)
   ▶ MPI (Opteron/Linux cluster Gonzales)

4. We expect you to solve 6 out of 8 exercises.
References


   http://www.oup.co.uk/isbn/0-19-851577-4

Complementary literature is found on the PARCO home page.
Flynn’s Taxonomy of Parallel Systems

In the taxonomy of Flynn parallel systems are classified according to the number of instruction streams and data streams.

SISD: Single Instruction stream - Single Data stream

The classical von Neumann machine.
- processor: ALU, registers
- memory holds data and program
- bus (a collection of wires) = von Neumann bottleneck

Today’s PCs or workstations are no longer true von Neumann machines (superscalar processors, pipelining, memory hierarchies)
During each instruction cycle the central control unit broadcasts an instruction to the subordinate processors and each of them either executes the instruction or is idle. At any given time a processor is “active” executing exactly the same instruction as all other processors in a completely synchronous way, or it is idle.
Example SIMD machine: Vector computers

Vector computers were a kind of SIMD parallel computer. Vector operations on machines like the Cray-1, -2, X-MP, Y-MP, ... worked essentially in 3 steps:

1. copy data (like vectors of 64 floating point numbers) into the vector register(s)
2. apply the same operation to all the elements in the vector register(s)
3. copy the result from the vector register(s) to the main memory

These machines did not have a cache but a very fast memory. (Some people say that they only had a cache. But there were no cachelines anyway.)
The above three steps could overlap: “the pipelines could be chained”.

A variant of SIMD: a pipeline

Complicated operations often take more than one cycle to complete. If such an operation can be split in several stages that each take one cycle, then a pipeline can (after a startup phase) produce a result in each clock cycle.

Example: elementwise multiplication of 2 integer arrays of length $n$.

$$\mathbf{c} = \mathbf{a} \ast \mathbf{b} \iff c_i = a_i \ast b_i, \quad 0 \leq i < n.$$ 

Let the numbers $a_i, b_i, c_i$ be split in four fragments (bytes):

$$a_i = [a_{i3}, a_{i2}, a_{i1}, a_{i0}]$$
$$b_i = [b_{i3}, b_{i2}, b_{i1}, b_{i0}]$$
$$c_i = [c_{i3}, c_{i2}, c_{i1}, c_{i0}]$$

Then

$$c_{i,j} = a_{i,j} \ast b_{i,j} + \text{carry from } a_{i,j-1} \ast b_{i,j-1}$$

This gives rise to a pipeline with four stages.
Flynn’s Taxonomy of Parallel Systems

Example SIMD machine: Vector computers

```
<table>
<thead>
<tr>
<th>Time</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A00*B00</td>
</tr>
<tr>
<td></td>
<td>A01<em>B01 A10</em>B10</td>
</tr>
<tr>
<td></td>
<td>A02<em>B02 A11</em>B11 A20*B20</td>
</tr>
<tr>
<td></td>
<td>A03<em>B03 A12</em>B12 A21<em>B21 A30</em>B30</td>
</tr>
<tr>
<td></td>
<td>A13<em>B13 A22</em>B22 A31<em>B31 A40</em>B40</td>
</tr>
<tr>
<td></td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td>A23<em>B23 A32</em>B32 A41<em>B41 A50</em>B50</td>
</tr>
<tr>
<td></td>
<td>etc.</td>
</tr>
</tbody>
</table>
```
Example SIMD machine: Pentium 4

- The Pentium III and Pentium IV support SIMD programming by means of their Streaming SIMD extensions (SSE, SSE2).
- The Pentium III has vector registers, called Multimedia or MMX Registers. There are 8 (eight) of them! They are 64 bit wide and were intended for computations with integer arrays.
- The Pentium 4 additionally has 8 XMM registers that are 128 bit wide.
- The registers are configurable. They support integer and floating point operations. (XMM also supports double (64 bit) operations.)
- The registers can be considered vector registers. Although the registers are very short they mean the return of vector computing on the desktop.
- We will investigate how these registers can be used and what we can expect in terms of performance.
Flynn’s Taxonomy of Parallel Systems

Example SIMD machine: Pentium 4
Example SIMD machine: Cell processor

- IBM in collaboration with Sony and Toshiba
- Playstation 3
- Multicore processor: one Power Processing Element (PPE)
- 8 SIMD co-processors (SPE’s)
- 4 FPUs (32 bit), 4 GHz clock, 32 GFlops per SPE
- 1/4 TFlop per Cell, 80 Watt
- SPE’s programmed with compiler intrinsics

Next page: EIB = element interface bus; LS = local store; MIC = memory interface controller; BIC = bus interface controller
Flynn’s Taxonomy of Parallel Systems

Example SIMD machine: Cell processor

Synergetic Processing Elements

SPU  SPU  SPU  SPU  SPU  SPU  SPU
LS   LS   LS   LS   LS   LS   LS

EIB (bis 96 B/Zyklus)

64-Bit Power Architektur

L2

L1

PPU

MIC

BIC

Dual XDR

RRAC I/O

[Rauber & Rünger: Parallele Programmierung]
## Flynn’s Taxonomy of Parallel Systems

Example SIMD machine: Cell processor

<table>
<thead>
<tr>
<th></th>
<th>Sony Emotion Engine</th>
<th>Cell-Prozessor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU-ISA</strong></td>
<td>MIPS64</td>
<td>64-Bit Power</td>
</tr>
<tr>
<td><strong>Issue-Rate</strong></td>
<td>dual</td>
<td>dual</td>
</tr>
<tr>
<td><strong>Taktfrequenz</strong></td>
<td>300 MHz</td>
<td>ca. 4 GHz</td>
</tr>
<tr>
<td><strong>Instruktionspipeline</strong></td>
<td>6 Stufen</td>
<td>21 Stufen</td>
</tr>
<tr>
<td><strong>L1 Cache</strong></td>
<td>16KB I-Cache + 8KB D-Cache</td>
<td>32KB I-Cache + 32KB D-Cache</td>
</tr>
<tr>
<td><strong>zusätzl. Prozessorspeicher</strong></td>
<td>16KB Skratch</td>
<td>512KB L2-Cache</td>
</tr>
<tr>
<td><strong>Vektoreinheiten</strong></td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td><strong>Vektorregister</strong></td>
<td>32 (128Bit) + 16 (16Bit)</td>
<td>128 (128Bit)</td>
</tr>
<tr>
<td><strong>lokalener Vektorspeicher</strong></td>
<td>4K/16KB I-Cache + 4K/16KB D-Cache</td>
<td>256KB unified</td>
</tr>
<tr>
<td><strong>Speicherbandbreite</strong></td>
<td>3.2 GB/s max.</td>
<td>ca. 25.6 GB/s max.</td>
</tr>
<tr>
<td><strong>Peak FLOPS</strong></td>
<td>6.2 GFLOPS</td>
<td>256 GFLOPS</td>
</tr>
<tr>
<td><strong>Anzahl Transistoren</strong></td>
<td>10.5 Millionen</td>
<td>235 Millionen</td>
</tr>
<tr>
<td><strong>Leistungsaufnahme</strong></td>
<td>15 Watt</td>
<td>ca. 80 Watt</td>
</tr>
<tr>
<td><strong>Chipfläche</strong></td>
<td>240mm²</td>
<td>235mm²</td>
</tr>
</tbody>
</table>
MIMD: Multiple Instruction stream - Multiple Data stream

Each processor can execute its own instruction stream on its own data independently from the other processors. Each processor is a full-fledged CPU with both control unit and ALU. MIMD systems are asynchronous.
Memory organization

Most parallel machines are MIMD machines. MIMD machines are classified by their memory organization:

- **shared memory machines** (multiprocessors)
  - parallel processes, **threads**
  - communication by means of **shared variables**
  - data dependencies possible, race condition
  - multi-core processors

![Interconnection network diagram]
Interconnection network

- network usually dynamic: *crossbar switch*

Crossbar switch with \( n \) processors and \( m \) memory modules. On the right the possible switch states.

- uniform access, scalable, very many wires \( \Leftrightarrow \) very expensive, used for only limited number of processors.
- **distributed memory machines** (multicomputers)
  - all data are local to some processor,
  - programmer responsible for data placement
  - communication by *message passing*
  - easy / cheap to build \(\rightarrow\) (Beowulf) clusters
Interconnection network

- network usually static: Array, ring, meshes, tori, hypercubes

- processing elements usually connected to network through routers. Routers can pipeline messages.
HP Superdome (Stardust / Pegasus)

The HP Superdome systems are large multi-purpose parallel computers. They serve as the Application Servers at ETH. For information see

http://www.id.ethz.ch/services/list/comp_zentral/

Figure: HP Superdome Stardust (3 cabinets left) and Pegasus (right)
Superdome specifications

- Stardust: 64 Itanium-2 (1,6GHz) dual-core processors, 256GB main memory
- Pegasus: 32 Itanium-2 (1,5GHz) dual-core processors, 128GB main memory
- HP/UX (Unix)
- Shared memory programming model
- 4-processor cells are connected through crossbar
- ccNUMA: Cache-coherent, Non-Uniform Memory Access
- Organisation: batch processing. Jobs are submitted to LSF (Load Sharing Facility)
  Interactive access possible on Pegasus.
- System manager: tonko.racic@id.ethz.ch
- We will use Pegasus for experiments with shared memory programming with C and compiler directives (OpenMP).
Gonzales Cluster

(Speedy) Gonzales is a high-performance Linux cluster based on 288 dual-processor 64-bit AMD Opteron 250 processors and a Quadrics QsNet II interconnect.

Figure: An image of the old Linux cluster Asgard
Cluster specifications

- One master node, two login nodes (Gonzales) and three file servers, of 288 compute nodes.
- 1 node = two 64-bit AMD Opteron 2.4 GHz processors, 8 GB of main memory (shared by the two processors).
  Global view: distributed memory
- All nodes connected through Gb-Ethernet switch (NFS and other services).
- Compute nodes inter-connected via a two-layer Quadrics QsNet II network. Sustained bandwidth 900 MB/s. Latency 1μsec between any two nodes in the cluster.
Each 64-way switch is based on a 3-stage fat-tree topology. The top-level switch adds another 4th stage to this fat-tree.

Nodes run SuSE Linux (64-bit) with some modifications for the Quadrics interconnect.

The login nodes have a more or less complete Linux system, including compilers, debuggers, etc., while the compute nodes have a minimal system with only the commands and libraries necessary to run applications.

The AMD Opteron runs both 32-bit and 64-bit applications.

Compilers: C/C++, Fortran 77/90 & HPF.

Note: all parallel applications must be recompiled (in 64-bit) and linked with the optimized MPI library from Quadrics.

Jobs are submitted from the login nodes to the compute nodes via the LSF batch system, exclusively. Users are not allowed to login or execute remote commands on the compute nodes.

System manager: olivier.byrde@id.ethz.ch
Cray XT-3 at CSCS in Manno

- The Cray XT3 is based on 1664 2.6 GHz AMD Opteron single-core processors, that are connected by the Cray SeaStar high speed network.

- The computer’s peak performance is 8.7 Tflop/s.

- Names: The XT3 is called “Red Storm”. The actual machine in Manno is called **horizon**.

- In number 94 on the list of the top 500 fastest machines (7.2 Tflop/s).

  [http://www.top500.org/list/2006/11/100](http://www.top500.org/list/2006/11/100)

Behind 2 Blue Gene (EPFL, IBM Research) and Intel Cluster (BMW Sauber)
Cray XT3 supercomputer is called “Red Storm” in the US. The CSCS model has been baptized “Horizon”.
SeaStar router: The high-speed interconnection network exchanges data six neighbouring knots in a 3D-torus topology.
Bone structure analysis

Computation of stresses in loaded human bone. FE application with $1.2 \cdot 10^9$ degrees of freedom.
IBM Blue Gene BG/L

- Presently fastest parallel computer.
- Blue Gene/L at Lawrence Livermore National Laboratory has 16 Racks (65’536 nodes, 131’072 processors): 280 TFlop/s
- Simple, cheap, processors (PPC440), moderate cycle time (700MHz), high performance / Watt, small main memory (512MG/node)
- 5 Networks
  - 3D - torus for point-to-point messages (bandwidth 1.4 Gb/s, latency < 6.4μs)
  - broadcast network for global communication, in particular reduction operations (bandwidth 2.8 Gb/s, latency 5μs)
  - barrier network for global synchronization (latency 1.5μs)
  - control network for checking system components (temperature, fan, . . .)
  - Gb-Ethernet connects I/O - nodes with external data storage.
IBM Blue Gene BG/L

Examples of MIMD machines

IBM Blue Gene BG/L

NUMERICAL PARALLEL COMPUTING
Examples of MIMD machines
IBM Blue Gene BG/L

IBM Blue Gene BG/L at Lawrence Livermore NL