The Multicore Software Challenge

Walter F. Tichy
We´re Witnessing a Paradigm Shift in Computing

- For 60 years, the sequential computing paradigm was dominant.
- Parallelism occurred in niches only:
  - Numeric computing
  - Distributed computing (client/server)
  - Operating systems, data base mgmt systems
  - Instruction level parallelism
- With multi/manycore, parallel computers have become affordable to everyone, and they will be everywhere.
- It is already difficult to buy a computer with a single main processor.
Important Parallel Computers

Atanasoff-Berry-Computer (1942)

30 adds/subtracts in parallel.
Not programmable.

30 add-subtract units
Important Parallel Computers

**Illiac-IV**: SIMD, distributed memory

- Only one built: 1976
  - 64 processors
- Worlds fastest computer until 1981
Important Parallel Computers

Cray-1 vector computer

- Shared memory
- Vector registers with 64 elements
- Vector instructions implemented by pipelining.
- First delivery 1976
- Second fastest computer after Illiac-VI
Important Parallel Computers

Transputer

- MIMD computer,
- Distributed memory
- Processors with 4 fast connections
- First delivery 1984
- Up to 1024 processors
- Occam programming language
- Developed by Inmos, Bristol, GB
Important Parallel Computers

CM-1 Connection Machine

- SIMD
- Distributed memory
- 65,536 processors (1-Bit)
- Hypercube interconnect
- First delivery 1986
- First massively parallel computer
Important Parallel Computers

Computer Clusters

- Off-the-shelf PCs connected by off-the-shelf networks
- MIMD, distributed memory
- Low cost because of mass market parts
- Today’s fastest machines
- Hundreds of thousands of processors
- See top500.org

Nasa Avalon with 140 Alpha processors, 1998
Your Laptop—a Parallel Computer?

Prices June 2007

**Inspiron™ 6400**
15" Notebook für vielseitige Unterhaltung & 1 GB RAM.

729 €
659 €
inkl. MwSt., zzgl. 78 € Versand

**Inspiron™ 1520**
Stylischer Denker, der es geniesst seine Qualitäten zeigen zu können und gerne im Mittelpunkt steht.

999 €
899 €
inkl. MwSt., zzgl. 78 € Versand

**Inspiron™ 1720**
Unterhaltung & Spaß garantiert! Technologie genau angepasst für Ihren Lifestyle. Jetzt in 8 Farben.

1.049 €
inkl. MwSt. und Versand

**Inspiron™ 1520**
Schlank, elegant und noch etwas schlauer und stärker. Jetzt in 8 Farben.

1.129 €
1.079 €
inkl. MwSt., zzgl. 78 € Versand

**Prozessor**

Intel® Pentium® Dual-Core T2080 Prozessor (1,73 GHz, 533 MHz, 1 MB L2-Cache)

**Prozessor**

Intel® Core™ 2 Duo T5450 Prozessor (1,66 GHz, 667 MHz, 2 MB L2-Cache)

**Prozessor**

Intel® Core™ 2 Duo T5250 Prozessor (1,5 GHz, 667 MHz, 2 MB L2-Cache)

**Prozessor**

Intel® Core™ 2 Duo T7100 Prozessor (1,8 GHz, 800 MHz, 2 MB L2-Cache)
Intel Dunnington
6 Processors on one Chip

3 x 2 Xeon Processors
(no HW multithreading)
2,6 GHz
130 W
45nm technology
Up to 4 of those on one board
Available 2008
Sun Niagara 2: 8 Processors on 3.42 cm²

- 8 Sparc Processors
- 8 HW-threads per processor
- 8x9 cross bar
- 1.4 GHz
- 75 W
- 65nm technology
- 4 per board
- Available 2007
- First Niagara 2005
Tilera’s TILE64

64 VLIW processors plus grid on a chip

For network and video applications

700 MHz

22 W

Available 2007
Nvidia GeForce 8 Graphics Processing Unit

128 cores altogether, each with 96 threads in HW
Total of 12288 HW threads!
SIMD, distributed memory

16 KB

processor, 1,35 GHz, 32-Bit FPU, 1024 registers
Intel’s Larrabee: 32 Pentiums on a Chip

- 32 x86 cores (45nm),
- (48 cores with 32 nm)
- Cache coherent,
- Ring interconnect
- 64 bit arithmetic
- 4 register sets per processor
- Special vector instructions for graphics
- Expected 2010

Source: www.pcgameshardware.com, May 12, 2009
What Happened?

Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Doubling the number of processors per chip with each chip-generation, at about the same clock rate.

Parallel computers will be everywhere in a short time.
What to do with all the Cores?

• „Who needs 100 processors for M/S Word?“
  • Lack of creativity, CS education?
  • Looking for applications that can use 100´s of cores.
• How could ordinary users of PCs, mobile phones, embedded systems benefit?
• Run faster!
• More compute intensive applications
• Speech and video interfaces
• Better graphics, games.
• Smart systems that model the user and environment and can predict what the user wants, therefore act more like a human assistant.
Example 1: Logistic Optimization (MS thesis with SAP)

Which deliveries?
On which trucks?
Which routes?

Generalization of travelling salesman problem

Given:
- Delivery orders
- Trucks
- Road network

Goal: optimal transportation routes
**Why parallelize?**

### Real logistics scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliveries</td>
<td>804</td>
<td>1177</td>
<td>7040</td>
</tr>
<tr>
<td>Load dimensions</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Loading stations</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Delivery points</td>
<td>31</td>
<td>559</td>
<td>1872</td>
</tr>
<tr>
<td>Intermediate stations</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Vehicles</td>
<td>281</td>
<td>680</td>
<td>2011</td>
</tr>
<tr>
<td>Vehicle types</td>
<td>7</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Time window (days)</td>
<td>1</td>
<td>2</td>
<td>64</td>
</tr>
</tbody>
</table>

Also important:
- Rest periods of drivers,
- Time to load, trailers,
- With/without refrigeration,
- Ferry schedules, ships,....

**Sequential implementation:**
- 150,000 lines of C++
- Evolutionary algorithm

Runs several hours for good solutions.
1. Start with initial solution

2. While cost bound not satisfied:
   - Improve solution with local changes (explore neighboring solutions)
   - Occasionally escape from local optimum with a jump in solution space
General Procedure

1 thread

- ZB
- TS
- ILS
- ZB
- TS
- ILS
- ZB
- TS
- ILS

Sequential time

- ZB: random move
- TS: depth first search
- ILS: iterated local search

Threads

- Thread 1
  - ZB
  - TS
  - ILS
- Thread 2
  - TS
  - TS
  - TS
- Thread 3
  - ILS
  - ILS
  - ILS

Parallel time

Replicate for more threads
Solutions examined in 2 Min.

Factor 23 improvement!!!

Computer with 4 Intel Dunnington Chips
(4 * 6 cores)
Speedup with 24 Threads

Average speedup: 17.37
Super-linear speedups if good candidates are found early

Random fluctuations

Speedup for reaching a cost bound

3 runs each:
- min
- avg
- max

Scenarios:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Speedup
What is the Basic Challenge in Parallel Software?

- Speedup, programmer productivity, and software quality must be satisfactory simultaneously.
  - Parallelization only interesting if there is a speedup
  - Programmer productivity and software quality should not get any worse!
- Current languages and tools are unsatisfactory (Thread $\approx$ Goto?)
- Most programmers are poorly prepared for parallelism.
Example 2: Metabolite Hunter

Algorithms pipeline searches for metabolites in spectrograms. Parallelization potential

Result: time-dependent graph of metabolites
Auto-Tuning

- **Problem**: Find parameter configuration that optimizes performance.
- Parameters are platform and algorithm dependent
- Parameters:
  - number of cores,
  - number of threads,
  - parallelism levels,
  - number of pipeline stages, pipeline structure,
  - Number of workers in master/worker, load distribution
  - size of data partitions,
  - choice of algorithm
- Manual adjustment is too time-consuming
- Let the computer find the optimum!
Auto-Tuning (2)

**Solution:** *Atune* Parameter Optimizer

- Library that searches for the optimum, given annotations about which parameters can be changed
  - specified with annotation language *Atune-IL*
- Search space can be huge, so sampling, learning, and other optimization techniques need to be explored.
- Difference between best and worst configuration in Metabolite Hunter: Factor of 1.9 (total speedup 3.1 on 8 cores)
- Gene expression application: Auto-tuning contributes a Factor 4.2 to a total speedup of 7.7 on 8 cores.

Tested on 2x Intel Xeon E5320 Quad-Core 1.86 GHz
Example: BZip2

- BZip2
  - Compression program
  - Used on many PCs worldwide
  - 8000 LOC, open source
- Parallelized in student competition
  - 4 teams of 2 students each
  - Preparation in 3 month lab course on OpenMP and Posix threads
  - Competition in the final 3 weeks (course project)
Winners reached a ten-fold speedup on Sun Niagara T1 (8 processors, 32 HW-threads).
How did they do it?

• Massive restructurings of the code
  • Teams who invested little in restructuring were unsuccessful.
  • Winners parallelized only on the day before submission; they spent the preceding 3 weeks on refactoring to enable parallelization.
  • Dependencies, side effects, sequential optimizations needed to be removed before parallelization became possible.
What did not work?

• Adding parallelization incrementally did not work for any team.
• Parallelizing the critical path only was not enough.
• Parallelizing inner loops did not work.
  • Parallel steps must encompass larger units (coarse grained parallelization)
• BZip2 contains specialized algorithms, so help from algorithms libraries is unlikely.
The Good News: Parallelization is not a Black Art

- Have a plan. Trial and error does not work.
- Develop hypothesis were parallelization might produce the most gains.
- Consider several parallelization levels.
- Use parallel design patterns.
  - Producer/consumer, pipeline, domain decomposition, parallel divide and conquer, master/worker.
- Don't despair while refactoring!
- Build tools that help.
How Can We Use all this Computing Power?

- Intuitive interfaces with speech and video,
- Applications that anticipate what users will do and assist them,
- Extensive modeling of users, their needs, and their environments for truly smart applications,
- New types of applications that are too slow today,
- Improved reliability and security
  - Run all kinds of checks in parallel with applications
Some Research Topics for Parallel Software Engineering

- Better programming languages for clear and explicit expression of parallel computations
- Compilation techniques
- Processor/process scheduling
- Parallel design patterns and architectures
- Parallel algorithms and libraries
- Testing, debugging
  - Automated search for data races, synchronization bugs.
- Performance prediction for parallel architectures
- Auto-tuning, auto-scaling, adaptability
- Tools for sequential-to-parallel refactoring
- New classes of applications
- Your favorite research topic/technique/expertise applied to parallel software.
XJava: Parallelism Expressed Compactly

Operator "=>" links processes in pipeline, as in Unix

```java
compress(File in, File out) {
    read(in) => compress() => write(out);
}
```

- Reads file, outputs blocks
- Buffered stream
- Reads blocks, compresses them
- Buffered stream
- Writes blocks in file

All filters run in parallel, until end of input.
Streams are typed and typesafe.
Also suitable for master/worker, producer/consumer
Operator „|||“ runs processes in parallel:

```
compress(f1, f1out) ||| compress(f2, f2out);
```

Methods executed by their own threads, implicit barrier at the end.

For process and data parallelism.

Multilevel (nested) parallelism.
Master/Worker in XJava

- One master, three workers:
  ```java
  void => X master() { /* master*/ }
  X => void w() { /* worker*/ }
  X => void gang() { w() ||| w() ||| w(); }
  ```

- i workers (dynamic):
  ```java
  X => void gang() { w():i; }
  ```

- `master() => gang()`
  Master passes Elements of type X to workers in round-robin fashion.

- `master() =>* gang()`
  Broadcasts elements to all workers.
XJava Extensions

- Easy to understand
- Fully integrated in Java
- Typesafe
- Easier to handle than threads or libraries
- Less code, fewer „opportunities“ for bugs
- Specialized autotuning possible
  - Example: tune stages in a pipeline in such a fashion that they take about the same time.
Summary

• Future performance gains by parallelism
• Goal: Faster, intelligent applications
• … of the same quality and at the same programmer productivity as sequential applications now.
• … while the number of processors per chip doubles every two years.
• Lots of the basics of computer science need to be reinvented.

„Reinventing Software Engineering“
http://www.multicore-systems.org/iwmse

Working Group Software Engineering for parallel Systems (SEPARS)
http://www.multicore-systems.org/gi-ak-sepas

Allons!
Vamos!
Gehn´mas an!
Let´s go!

Papers:
http://www.ipd.uka.de/Tichy