Multicore Programming

Prof. Dr. Gudula Rünger
Dr. Jens Lang

Professur Praktische Informatik
Technische Universität Chemnitz

Winter Semester 2014/2015

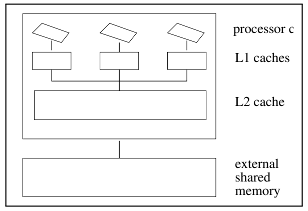
Organizational Information

- Time of **lecture**:
  Monday 15:30 to 17:00, Room 1/305,
- Time of **tutorials**:
  Wednesday 17:15 to 18:45, Room 1/375,
  Wednesday 19:00 to 20:30, Room 1/375,
  Tutor: Thomas Jakobs, M.Sc.
- **Exam**:
  - Date: 1st February, during the lecture hours.
  - There will be homework sheets for every tutorial.
  - **You have to solve 8 out of 12 homework sheets. A homework is considered as solved if 50 % of the answers are correct.**
- Requirements to obtain credit points:
  - Attendance and passing of the exam
- Website for the lecture:
  http://www.tu-chemnitz.de/informatik/PI/lehre/plan/WS15/teach_mp.php.en

Overview

Multicore Processor

- Multiple execution cores are placed on one chip.
- Each core typically has a private level-1 cache.
- The cores of a multicore processor typically share a level-2 cache.
- The cores have access to a shared memory (off-chip).
- The cores use shared resources (cache, TLB, bus etc.)



Multicore Processor

- Delivery by Intel, AMD, SUN, IBM since 2005
- In 2015 typically: processors with 2 (dualcore) or 4 (quadcore) cores, sometimes, 6 or 8 cores
- Highly parallel multicore processors (MIC – many integrated cores):
  - Intel's Xeon Phi co-processor with 61 cores (based on 80586 Pentium architecture).
  - Phytium's Mars processor with 64 cores (based on ARMv8).

Software for Multicore Processors

- Programs should take advantage of the cores of a multicore processor, i.e. all cores should compute tasks in parallel
  $\rightsquigarrow$ **Parallel Programming**
- The cores use a shared memory
  $\rightsquigarrow$ **Shared-Memory Programming**
  (in contrast to message-passing programming)
- The cores interact by using shared ressources
  $\rightsquigarrow$ **Threading Technology**
- Programming environments:
  Pthreads, Java Threads, OpenMP, C++11

Contents of the Lecture

- Architecture of Multicore Processors
- Overview: Parallel Programming
- Concepts of Shared Memory Programming
- Programming with OpenMP
- Scheduling Methods for Threads
- Cache, Cache Consistency and Cache Coherency
- Methods for Optimizing Cache Accesses (Loop Fusion, Loop Tiling)
- GPU Programming with OpenCL and CUDA
- Pthread Programming
- Software Transactional Memory

Literature

1. Parallel Programming for Multicore and Cluster Systems Second Edition. Rauber, Rünger, Springer, 2013.
2. Parallele Programmierung 2. Auflage. Rauber, Rünger, Springer, 2013.
3. Multicore: Parallele Programmierung. Rauber, Rünger, Springer, 2008.
4. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism, Reinders, O'Reilly 2007.
5. OpenMP, Informatik im Fokus, Hoffmann, Lienhart, Springer, 2008.
6. Multi-Core Programming – Increasing Performance through Software Multi-Threading. Akhter, Roberts, Intel Press, 2006.
7. Using OpenMP – Portable Shared Memory Parallel Programming. Chapman, Jost, van der Pas, MIT Press, 2008.
8. OpenMP Application Program Interface, Version 4.0, 2013, OpenMP Architecture Review Board, `http://www.openmp.org`
9. CUDA programming – A developer's guide to parallel computing with GPUs, Shane, Cook, Waltham, Morgan Kaufmann, 2013.
10. Pthreads programming, Dick Buttlar et al., O'Reilly, 2013.

Detailed Literature

Chapter	English Literature	German Literature
Architecture: A short overview on multicore processors	[1] 2.1 -2.4	[2] 2.1 - 2.4 [3] 1.1-1.4
Parallel Programming Concepts	[1] 3.1, 3.2, 3.3.1-3.3.5 [3] 2.1 bis 2.4, 2.9	[2] 3.1-3.5, 4.2.1-4.2.2.1
Thread Programming	[1] 3.6-3.7	[2] 6.1 [3] 3.1-3.5
OpenMP Programming	[1] 6.3	[2] 6.4 [3] 6
Cache, Cache-consistency, -Coherency	[1] 2.7	[2] 2.7
Synchronization Methods	[1] 6.1,6.2.6	[2] 6.3.1 [3] 5.1,5.6
Software Transactional Memory	[1] 6.2.6	[3] 7

Note: The numbers enclosed in brackets [] refer to the list of books on the literature slide.

Overview

Overview

Evolution of Micro Processors

- The development of modern computer architectures is strongly pushed by the rapid technological development of electrical circuits;
  VLSI circuits (very large scale integrated circuits)
  $\rightarrow$ high complexity of modern computer systems.
- The reason for the rapid development is the need for **higher performance** of processors.
- Around 2005: transition from **megahertz era** to **multicore era**
  $\rightarrow$ processors do not become faster any more, but more parallel.
- Technological evolutions include parallelism on different levels.

Trends in the Development of Processor Chips

- A rough measure of the complexity and performance of a processor is the **number of transistors**
- **Moore's Law**: The **number of transistors** per chip **doubles** every **18 to 24 months**.
  - Observation by Gordon Moore (Co-Founder of Intel), now valid for about 40 years.
- **More detailed**:
  The number of **transistors per chip area** increases by about 35 % per year. Additionally, the chip area increases between 10 % and 20 % per year.

Performance Evolution of Processors

- Increase of **processor performance** by 55% (integer operations) or 75% (floating point operations) per year (1980-98)
- Increase of **memory capacity** of DRAM memory chips by 40% to 60% per year since 1977.
- Decrease of **memory access latency** of DRAM memory chips on average by about 25% per year.
  ⇝ Problem of memory wall

Reasons for the previous performance increase

(1) The **clock rate** of micro processors increases on average by about 30% per year

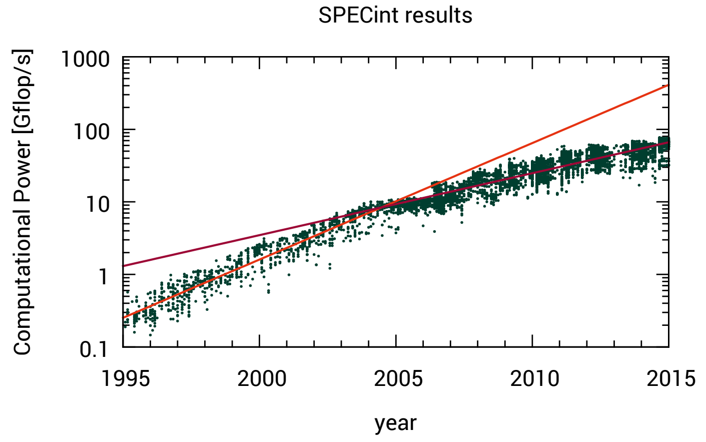
(2) Architectural enhancements: Make use of the increasing number of transistors for **additional functionality** and **parallelism** on the instruction level (ILP - instruction level parallelism)

Architectonic Enhancements

Additional functionality and parallelism on the instruction level due to

- Increase of the **internal word length** to 32 or 64 bits;
- Make use of **internal pipelining** by decomposing the instruction processing into pipeline stages
  ⇝ Reduction of the **latency** per instruction (more instructions per time unit);
- Usage of **multiple functional units**, which receive instructions dynamically from a dispatch unit
  ⇝ Independent machine instructions can be executed in **parallel**
- Integration of **multi-level caches** onto the die of the processor
  ⇝ Reduction of the **average memory access latency**;
  - reduces the Memory Wall Problem

Chart of Performance Increase



Each dot stands for a result of the SPECint benchmark with a CPU presented in the respective year.

Why a further performance increase is impossible

(1) Clock Rate:

- The **memory access latency** is not reduced to the same extent as the clock rate increases. This leads to a relative increase of the memory access time.
  1990: Intel i486, 6–8 clock cycles per main memory access,
  2006: Intel Pentium, more than 220 clock cycles per main memory access
- The increased transistor count is achieved by a higher packaging density, which results in an increased **heat generation** per chip area.
- An increased clock rate (time per clock cycle decreases) results in a **decreased distance** a signal can travel in one clock cycle. Therefore, several clock cycles are required for the complete processing of an instruction.
  Layout avoiding long distances on the chip are more complex.

$\rightsquigarrow$ A further increase of the clock rate is neither possible nor useful.

Why a further performance increase is impossible

(2) Instruction-level parallelism (pipelining; several functional units):

- Could be increased from a technological point of view;
- **Dependencies between instructions** prevent utilization of several functional units and deeper pipelines;
- The **control logic** requires a **high hardware complexity** and results in high energy requirements.

Note:
A lot of these (older) architectural enhancements require no or only minor **support from the compiler or operating system**

- Dynamic instruction scheduling managed by hardware;
- Cache management with reload strategy handled by hardware.

VLIW Processors (Very Long Instruction Word)

- **Basis**: The functional units are controlled by **static scheduling** by letting the **compiler** arrange independent instructions into long instruction words; $\rightsquigarrow$ the **parallel** processing of instructions is **explicitly** visible in the instruction words;
  $\rightsquigarrow$ EPIC (explicitly parallel instruction computers)
- **Examples**: Intel IA-64-Architecture (Itanium, Itanium 2);
  Trimedia TM32-Architecture (processor for embedded systems);
  Texas Instruments TMS320c6x (Digital Signal Processor);
- The **micro-processor hardware** is **simpler**, because the complexity for **dynamic scheduling** by the processor is omitted. The **compiler** can ensure by analysis methods that the functional units are efficiently utilized.
- **Requirement:** The compiler has to recognize independent instructions and has to arrange them into the instruction word

Overview

Parallelism on Processor Level

Simultaneous Multithreading or Hyperthreading Processors

- **Basis**: Hyperthreading is based on a **duplication of the processor region for the storage of the processor state** on the chip of the processors;
  This includes: **general purpose registers, machine status registers, interrupt controller and its registers**.
- **Result:** The physical processor is seen as **two logical processors**
  $\rightsquigarrow$ The operating system is able to assign two processes or threads to the processor; those may come from **the same** or **different applications**.
- The logical processors **share almost all resources** of the physical processor (caches, functional units, control logic, buses etc.)
  $\rightsquigarrow$ Hyperthreading-Technology leads to an **increased chip area by only 5%**.

Hyperthreading Processors

- Advantage of hyperthreading: In case of **stalls** of the first logical processor, the **processing resources** can be used by the second logical processor.
  $\rightsquigarrow$ **Increased utilization** of the processing resources.
- **Reasons for stalls**: cache misses, wrong branch prediction, dependencies between instructions, pipeline hazards etc.
- Depending on the application, Hyperthreading can **increase the runtime by 15% to 30%**.

Hyperthreading-Processors

- The operating system has to be able to control **two logical processors**. On systems with more than one processor, the operating system has to distinguish between **logical** and **physical** processors.
- More than one thread has to be available for execution;
  $\rightsquigarrow$ the program has to be prepared accordingly;
- Case 1: **Multithreaded programming** (Pthreads, Java Threads)
- Case 2: **OpenMP** with compiler support
  (parallel loops, task queues, parallel sections)

Overview

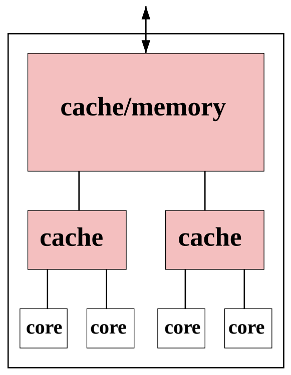
Multicore Processors

- Several **execution cores** are integrated into the chip of a processor.
  Every execution core appears to the operating system as a separate **logical processor** with separate execution resources, which have to be controlled separately.
- **Design options** for Multicore-Processors:
  - Hierarchical design,
  - Pipeline design,
  - Network-based design.

Design Options for Multicore Processors (I)

Hierarchical Design

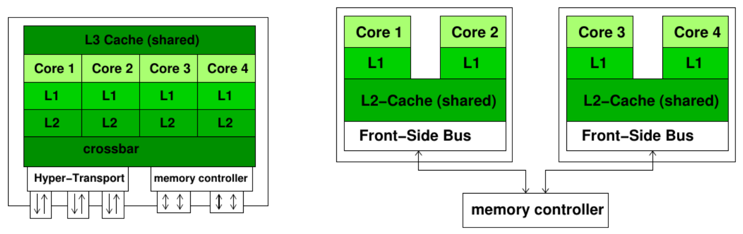
Several processor cores share serveral caches, with a **tree like** organization;
Examples: IBM Power6, Intel Core i7, AMD Opteron, Sun UltraSparc T1 (Niagara);



hierarchical design

Design Options for Multicore Processors (II)

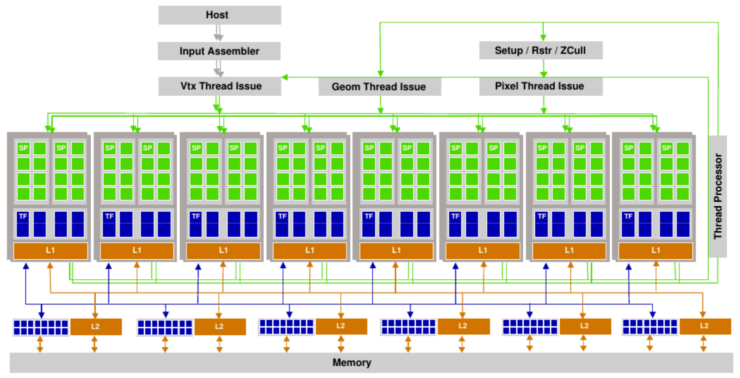
Hierarchical Design – Example I



Quad-core AMD Opteron (left) vs. quad-core Intel Xeon architecture (right) as examples for a hierachical design.

Design Options for Multicore Processors (III)

Hierarchical Design – Example II

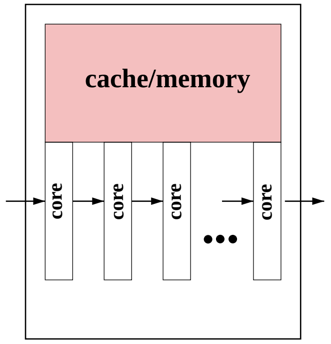


Nvidia GeForce 8800

Design Options for Multicore Processors (IV)

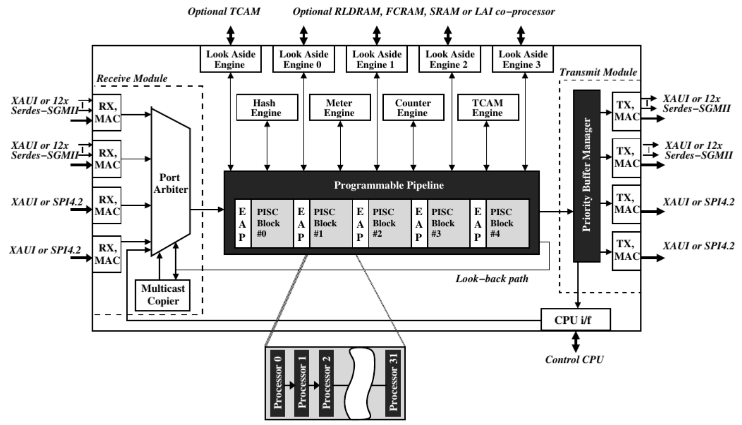
Pipeline Design

The data is processed **stepwise** core by core until the data reaches the last core, where the data is stored into memory.
Examples: GPUs, router processors for processing network packets; X10 and X11 processors from Xelerator: up to 200 separate VLIW cores;



pipelined design

Pipeline design - example

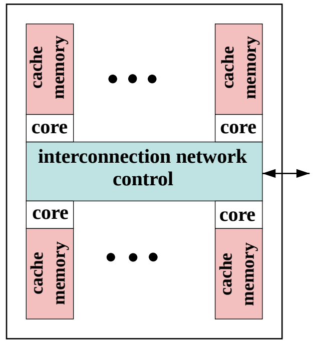


Xelerator X11 Network Processor

Design Options for Multicore Processors (V)

Network-based Design

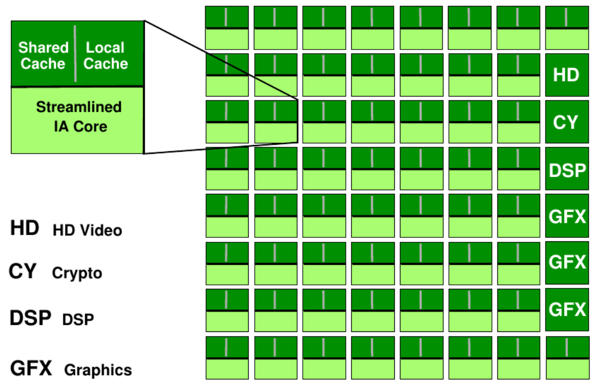
The cores and their caches or memories are connected by an **interconnection network**;
→ all **data transfer** between processor cores is handled by the interconnection network.
Example: Intel Teraflop chip with 80 processor cores organized in a 8 × 10 grid.



network-based design

Design Options for Multicore Processors (VI)

Network-based Design – Example



Intel Teraflop processor

Design Options for Multicore Processors (VII)

- **Evolution**: Intel Pentium D (without Hyperthreading) and Intel Pentium Extreme Edition (with Hyperthreading) introduced as dual-core in 2005;
  AMD Dual Opteron also introduced in 2005;
  2006: quad-core processors Intel Xeon 5300 (Clovertown);
  Since 2010: oct-core processors.
- Multicore processors with **several heterogeneous processor cores** on a chip (e.g. ARM's big.LITTLE architecture);
- **Programming:** the processor cores of a chip can only by used by **threads executed in parallel**;
  $\rightsquigarrow$ **Parallel programming techniques** have to be used.
- Difference to classical parallel systems: **Communication and synchronization** of the processor cores requires only **a small number of machine cycles** $\rightsquigarrow$ **fine grained parallelism**;