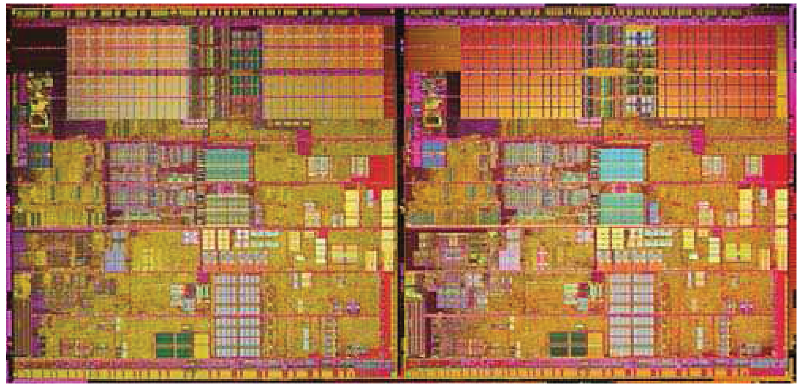
Overview

2. A Short Overview: Multicore Processors
   Evolution of Micro Processors
   Parallelism on Processor Level
   Architecture of Multicore Processors
   Examples

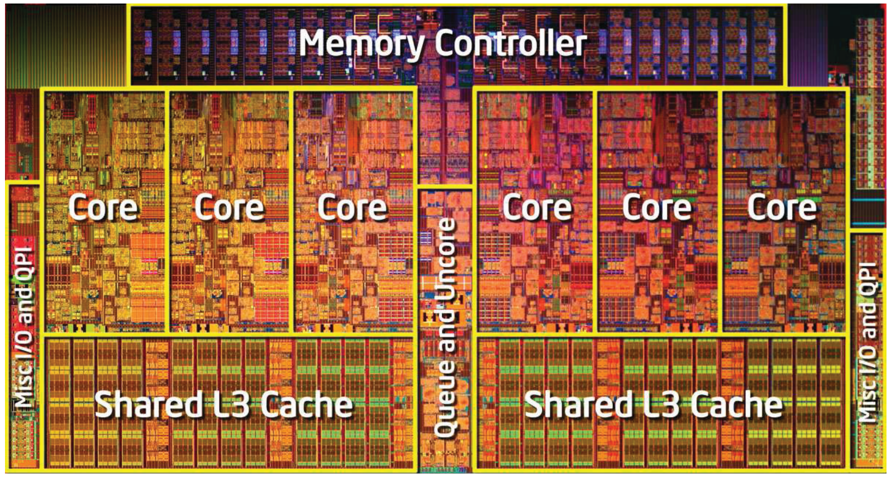
Design Options for Multicore-Processors

Illustration of a **Pentium D** processor (dual-core) from 2005



Design Options for Multicore-Processors

Illustration of a **Intel Core i7** processor (hexa-core) from 2010



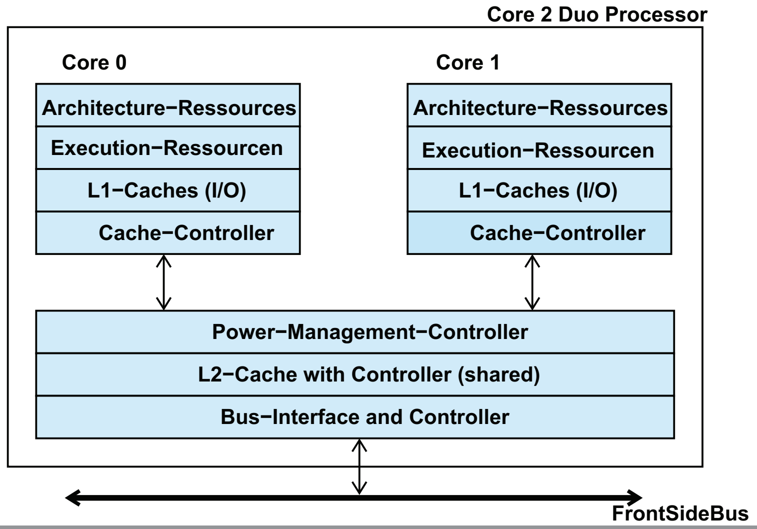
Intel Core 2 Processors (I)

- Family of Intel processors with a common architecture
- Further development of the Pentium M (mobile) processor
- Features of the architecture:
  - Reduction of the pipeline length to max. 14 stages
  - Reduction of the clock rate $\rightsquigarrow$ lower energy consumption
  - Power management unit for temporary shutdown of unused processor parts
    (reduction of energy consumption by reduction of clock rate or shutdown of L2 cache)
  - New SIMD extension (SSE)

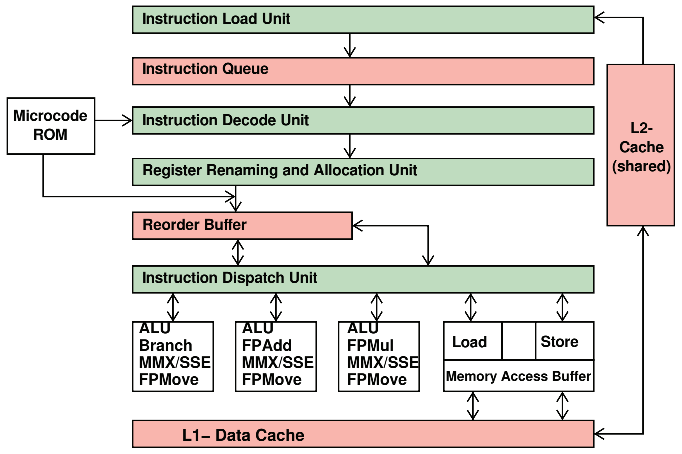
Intel Core 2 Processors (II)

- Current versions: Core 2 Duo, Core 4 Duo (with 2 or 4 independent cores)
- No Hyperthreading (since Pentium M based)
- Separate L1 caches (32 KiB for instructions and 32 KiB for data)
- Shared L2 cache (4 MiB) for instructions and data
- Accessing the L2 cache (from cores or bus) is manged by the L2 controller.
- MESI protocol ensures coherency on the different levels of memory hierarchy
  (MESI – Modified, Exclusive, Shared, Invalid)
- Bus controller for data and I/O requests to/from the external bus
- Computations:
  - Individual instruction stream per core (incl. load/store)
  - Up to 4 instructions per core in parallel

Core 2 Duo Architecture



Instruction Processing and Memory Organization of a Core 2 Processor Core



Overview

Parallel Programming Concepts

- Parallel programming techniques are being used for many years, e.g. in
  - High-performance computing
  - computational science

  ⇝ Multithreaded Programming is no new programming style
- Ubiquity of multicore processors in all areas of software development
- Essentials in multicore programming:
  - Transfer of parallel programming concepts
  - Providing of several instruction streams
- Subtopics of parallel programming:
  Classification of parallel architectures, parallel programming models, parallel programming environments, parallel performance metrics

Basic Concepts

- How to proceed when designing a parallel program?
- What properties of the parallel hardware should be made use of?
- What programming model should be used?
- How can the performance gain of the parallel program be quantified in comparision to the sequential program?
- Which parallel programming environment or language should be used?

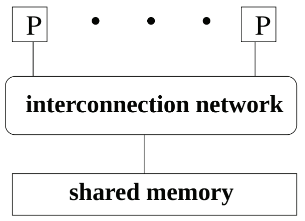
Overview

Flynn’s Taxonomy

- Flynn’s taxonomy characterizes parallel computers according to the **global control** and the resulting **data and control flows**.
- Four categories of parallel systems can be distinguished:
  a) SISD – **S**ingle **I**nstruction, **S**ingle **D**ata: in each step **a single instruction** is applied to **a single data item**;
  b) MISD – **M**ultiple **I**nstruction, **S**ingle **D**ata: In each step **several different instructions** are applied **to the same data item**;
  c) SIMD – **S**ingle **I**nstruction, **M**ultiple **D**ata: In each step **the same instruction** is applied to **several data items**;
  (Many former parallel computers used that approach)
  d) MIMD – **M**ultiple **I**nstruction, **M**ultiple **D**ata: In each step **different instructions** are applied to **different data items**;
  (Most of today’s parallel computers)
  Further characterization by **memory organization**

Memory Organization: Shared Address Space

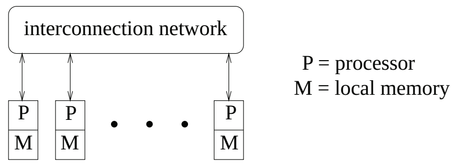
- Multi-processor systems use an **interconnection network** for connecting the processors to a **global memory** (**shared memory machines**):



- **Data exchange** using **shared variables** and coordinated reading and writing $\rightarrow$ **explicit synchronization operations**
- **Local and non-local memory accesses** are handled implicitly
- Variant: **symmetric multi-processors (SMP)**: The time for each **access to the shared memory** is the same for each processor and independent from the memory location (address).

Memory Organization: Distributed Address Space

- Multi-computer systems consist of **nodes** (**processors** with **local memory**), connected to each other by an interconnection network:



- **Exchange of data** is performed by **sending messages – message passing**;
  $\rightarrow$ **Data exchange by explicit communication operations**
- **Examples:** Blue Gene Project, IBM (gene research)
  PC clusters: Myrinet interconnect, Gigabit ethernet, Infiniband

Interconnection Networks of Parallel Computers

- Used for **communication** in multi-computer systems
- Used for **memory access** in multi-processor systems

Features:

- **Topology** of the interconnection network: **geometric structure** of the connections between the processors or memory modules;
  - **Static** or **direct** interconnection networks: **The nodes are connected directly by using point-to-point wires.**
  - **Dynamical** or **indirect** interconnection networks: **The processors and/or memory modules are indirectly connected using several wires and switches inbetween (bus based or switch based networks).**
- **Routing techniques** – way of message transmission
  - **Routing algorithm**: path selection
  - **Switching strategy**: transmission mode

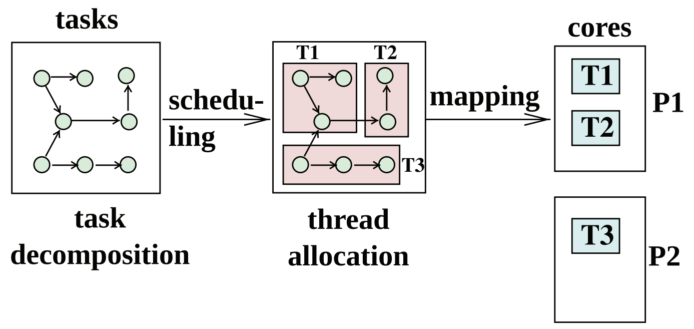
Overview

Creation of Parallel Instruction Streams (I)

Basic idea: Creation of independent instruction streams for a parallel execution on several cores

- Division into **tasks**
  - Tasks are the basic units of parallelism
  - Task structure with task and control dependencies
  - The size of tasks is called **granularity**
- Mapping of tasks to threads or processes: **Scheduling** (static or dynamic)
- Mapping of processes or threads to cores (explicitly in the program or by the operating system)

Creation of Parallel Instruction Streams (II)



Execution of Parallel Tasks or Threads

- **Multitasking:** One processor, multitasking operating system, processes get time slices on the same processor, hiding of latencies (e.g. due to I/O)
- **Multiprocessing:** several physical processors, true parallel execution
- **Simultaneous Multithreading:** several logical processors on a physical processor, can decrease execution time of the tasks
- **Chip-Multiprocessing:** Fusion of multiprocessing and simultaneous multithreading:
  Multithreaded programs are not only executed concurrently, they are really executed in parallel on the processor

Differences Between Multithreaded Programming for Multicores and Simultaneous Multithreading

- Performance improvements:
  SMT processors: Additional concurrent thread, e.g. for handling user requests
  Multicore: Whole application is parallelized.
- Impact of caches: **False Sharing** – two cores with separate caches work with data items that are different, but located in the same cache line.
- Thread Priorities:
  SMT: Thread with highest priority is executed first
  Multicore: Threads with different priorities can be executed in parallel
  $\rightsquigarrow$ Can lead to varying results

Overview

Parallel Programming Models (I)

Abstract view of the parallel system, which executes the parallel software, i.e.
programmers' view of the parallel hardware and system software like the operating system, parallel programming languages or libraries, compilers and runtime libraries

- On which level of a program should parallel program parts be executed?
  - Parallelism on instruction level
  - Parallelism on statement level
  - Parallelism on loop level, parallel loops
  - Parallelism on function level

Parallel Programming Models (II)

- Should parallel program parts be specified explicitly?
  - Implicit parallelism: no specification of parallelism
    (e.g. parallelizing compilers, functional programming languages)
  - Explicit Parallelism
    - Implicit decomposition (Fortran Extensions)
    - Explicit decomposition (e.g. coordination languages like Linda)
    - Explicit communication and synchronization (e.g. MPI, Pthreads)
- How are parallel program parts specified?
  - **Thread:** Stream of instructions, which can be executed in parallel with other instruction streams (threads).
  - Processes which are executed on a separate physical processor or logical processor

Parallel Programming Models (III)

- How is the parallel execution of program parts organized in relation to each other?
  - SIMD or SPMD (Single Program, Multiple Data)
  - Synchronous, asynchronous
  - Programming pattern: e.g. Pipelining, Master-Worker or Producer-Consumer-Model,
- How is the information exchange organized between parallel program parts?
  - Communication
  - Shared variables
- Which kind of synchronization can be used?
  - Barrier-Synchronization causes all threads to wait for each other
  - Coordination of the threads to avoid conflicts

Overview

Parallel Runtime of a Program

- The **parallel runtime** $T_p(n)$ of a parallel program P with input size n on p processors is the **time between the start of program** P and the **termination** of the computations of P on **all** processors.
- For computers with physically **distributed memory** $T_p(n)$ consists of:
  - Time for **local computations**
  - Time for **data exchange** with communication operations;
  - **Waiting time** of processors e.g. because of load imbalance
  - Time for **synchronization** of the executing processors or a subset of the executing processors
- For computers with shared memory the time for **data exchange** is replaced by the time for the **access to global data**.

Costs of Parallel Programs

- The **costs** $C_p(n)$ of a parallel program P with input size n on p processors is the **total time**, that the involved processors require for the execution of P:

$$C_p(n) = T_p(n) \cdot p$$

- The cost of a parallel program is **a measure for all the computations performed**.
- A parallel program is **cost optimal** if $C_p(n) = T^*(n)$ holds, where $T^*(n)$ is the runtime of the fastest sequential program;
  A cost optimal program requires **as many computations** as the **fastest sequential program**.
  Difficulty: The fastest sequential program or method is possibly **not known** or can only be determined with hight efforts.
- The costs is often called **work** or **Processor-Time-Product**;

Speedup of a Parallel Program

- The **Speedup** $S_p(n)$ of a parallel program P with input size n on p processors is defined as:

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \qquad \frac{\text{sequential time}}{\text{parallel time}}$$

  - $T_p(n)$ = Parallel runtime of a parallel program P on p processors;
  - $T^*(n)$ = Runtime of an optimal sequential implementation for the solution of the problem;
- The speedup is a **measure of the relative speed increase compared to the best sequential implementation**
- Typically at most **linear speedup** can be reached: $S_p(n) \leq p$ (theoretical upper bound)
- In practice, due to cache effects a **super linear speedup** can occur.

Efficiency of a Parallel Program

- The **Efficiency** $E_p(n)$ of a parallel program P with input size n on p processors is defined by

$$E_p(n) = \frac{T^*(n)}{C_p(n)} = \frac{S_p(n)}{p} = \frac{T^*(n)}{p \cdot T_p(n)}$$

  - $C_p(n)$ = Parallel program cost
  - $T_p(n)$ = Parallel runtime of a parallel program P
  - $T^*(n)$ = Runtime of the best sequential implementation
- The efficiency is a **measure** for the portion of the runtime, that is required for computations that are also present in the sequential program.
- The ideal speedup $S_p(n) = p$ is equivalent to $E_p(n) = 1$.

Amdahl's Law

- When the parallel implementation requires a (constant) fraction $f, 0 \leq f \leq 1$, to be computed **sequentially**, the runtime of the parallel implementation is composed of:
  - The runtime $f \cdot T^*(n)$ of the **sequential part** and
  - The runtime of the **parallel part**, which is at least $(1-f)/p \cdot T^*(n)$
- Attainable speedup

$$S_p(n) = \frac{T^*(n)}{f \cdot T^*(n) + \frac{1-f}{p} T^*(n)} = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f}.$$

- **Example:** $f = 0.2 \rightarrow S_p(n) \leq 5$ independent from the number p of processors;
  $\rightarrow$ The sequential part has a big influence on the attainable speedup
  for the efficient utilization of a **high number of processors** a

Scalability

- The **scalability** of a parallel program on a parallel computer is a **measure** for the property **to achieve a performance increase proportional to the number p of processors**.
- Common observations:
  - For a fixed problem size n and an increasing number of processors p a **saturation of the speedup** occurs;
  - For a fixed number of processors p and an increasing problem size n an **increas of the speedup** occurs.
- **Concretization:** Scalability means, that the efficiency of a parallel program is constant when the number of processors p and the problems size n is increased