Multiprocessor programming, DV2597/DV2606

Parallel Computer Architecture, part 1 (Chapter 2)

Håkan Grahn
(some slides by G. Karypis, T. Rauber and G. Rünger)

Topic overview

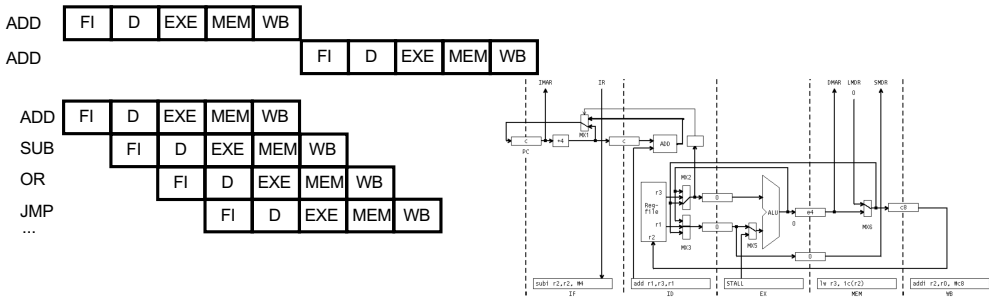
- Trends in Microprocessor Architectures
- Memory System Performance
- Parallel Computing Platforms

Implicit Parallelism: Trends in Microprocessor Architectures

- High microprocessor clock speeds
  (two to three orders of magnitude over two decades).
- Higher levels of device integration
  => a large number of transistors (> 1billion transistors).
- How do we best utilize these resources?
- Current processor cores (super-scalar + deep pipelines)
  - Have multiple functional units
  - Execute multiple instructions in the same cycle.
  - The precise manner in which these instructions are selected and executed provides impressive diversity in architectures.
- Simultaneous multithreading (a.k.a. Hyperthreading)
- Chip-multiprocessors (a.k.a. multicore processors)

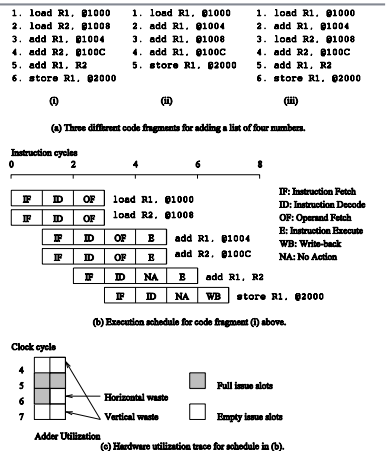
Pipelining and Superscalar Execution

- Pipelining overlaps various stages of instruction execution to achieve performance (Fetch, Decode, Execute, Data access, Write back).
- At a high level of abstraction, an instruction can be executed while the next one is being decoded and the next one is being fetched.
- This is akin to an assembly line for manufacture of cars.



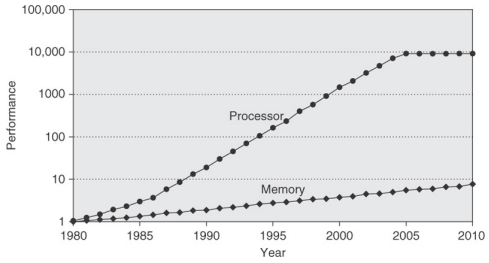
Superscalar Execution: An Example

- Instruction-level parallelism (ILP)
- An example of a two-way superscalar execution of instructions.
- Some waste of resources due to *data dependencies* (RAW, WAR, WAW).
- Different instruction mixes with identical semantics can take significantly different execution time.
- *In-order* or *out-of-order* execution of instructions.



Memory System Performance

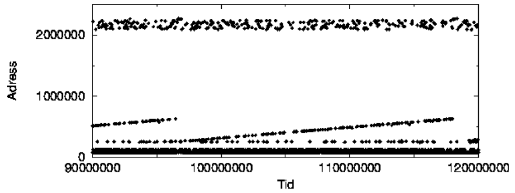
- Memory system, and not processor speed, is often the bottleneck for many applications.
- Memory system performance: latency and bandwidth.
  - ***Latency*** is the time from the issue of a memory request to the time the data is available at the processor.
  - ***Bandwidth*** is the rate at which data can be pumped to the processor by the memory system.



H. Grahn/DV2597/DV2606

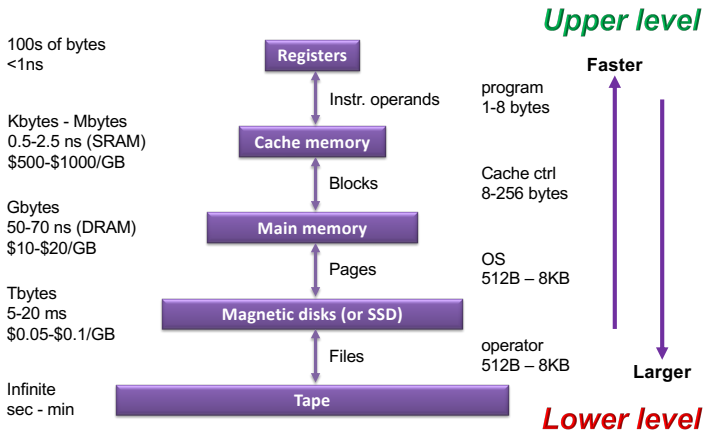
Reference Locality

- A program accesses only a relatively small portion of the address space at a given point in time



  - **Temporal locality:** If a memory location is referenced, it will tend to be referenced again soon (locality in time).
  - **Spatial locality:** If a memory location is referenced, locations whose addresses are close by will tend to be referenced soon (locality in space).

Memory Hierarchy



Multiprocessor programming, DV2597/DV2606

Parallel Computer Architecture, part 1 (Chapter 2)

Håkan Grahn
(some slides by G. Karypis, T. Rauber and G. Rünger)

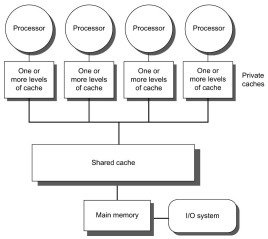
Flynn's Taxonomy

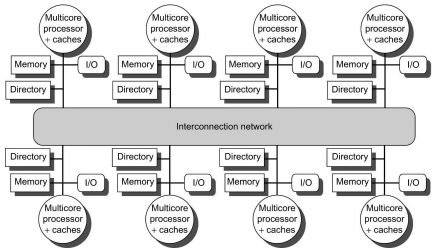
- Flynn's classification characterizes computers by the **organization of the global control** and the resulting **data and control flows**.
- The following four classes are distinguished:
  1. **SISD** - Single Instruction stream, Single Data stream:
     in each processing step one instruction is applied to one value;
  2. **MISD** - Multiple Instruction streams, Single Data stream:
     in each processing step different instructions are applied to one value;
  3. **SIMD** - Single Instruction stream, Multiple Data streams:
     in each processing step the same instruction is applied to a number of values;
  4. **MIMD** - Multiple Instructions streams, Multiple Data streams:
     in each step a number of instructions is applied to a number of values;
- A lot of computers in the past had a SIMD architecture. GPUs are based on the SIMD concept; also SSX and AVX extensions

Most of the parallel computers used today are MIMD

Shared-Address-Space Platforms

- The memory is accessible to all processors.
  - Processors interact by modifying data objects stored in this shared-address-space => Needs synchronization!
- If the access time of any memory word in the system is equal, the platform is classified as a uniform memory access (UMA), else, a non-uniform memory access (NUMA) machine.



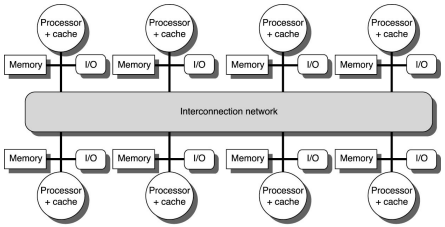


NUMA and UMA Shared-Address-Space Platforms

- Programming these platforms is easier since reads and writes are implicitly visible to other processors.
  - However, read-write data to shared data must be coordinated (discussed in more detail when we talk about threads programming).
- The distinction between NUMA and UMA platforms is important from the point of view of algorithm design.
  - *NUMA machines require locality* from underlying algorithms for performance.
- Caches in such machines require coordinated access to multiple copies. This leads to the *cache coherence problem*.
- A weaker model of these machines provides an address map, but not coordinated access. These models are called non cache coherent shared address space machines.

Message-Passing Platforms

- These platforms comprise of a set of processors and their own (exclusive) memory.
  - Examples: Clustered workstations and non-shared-address-space multicomputers.
- These platforms are programmed using (variants of) send and receive primitives.
  - Libraries such as MPI and PVM provide such primitives.



Message Passing vs. Shared Address Space Platforms

- Message passing requires little hardware support, other than a network.
- Shared address space platforms can easily emulate message passing. The reverse is more difficult to do (in an efficient manner).

Parallelism at the Processor Level

Simultaneous MultiThreading (SMT) a.k.a. HyperThreading (HT)

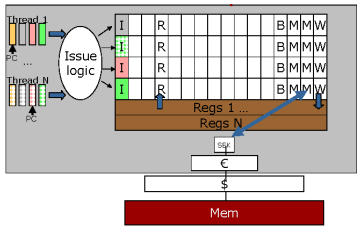
- **Basis:** SMT is based on a **duplication of the processor region for the storage of the processor state** on the processor chip:
  => This includes general purpose registers, machine state registers, interrupt controller and its registers
- **Result:** The physical processor is seen as **two logical processors**
  => The operating system is able to assign two processes or threads to the processor; those may come from the same or different applications
- **Note:** The logical processors **share almost all resources** of the physical processor (caches, functional units, control logic, buses, etc.)

Simultaneous MultiThreading (SMT)

- **Advantage:** In case of **stalls** of the logical processor, the *processing resources* can be used by the second logical processor;
  => 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 **decrease the runtime by 15% to 30%.**
- 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; the program has to be prepared accordingly!

Simultaneous MultiThreading (SMT)

- Exploit parallel computing and thread-level parallelism at the processor chip level
- Allow several threads to execute in parallel in the same processor pipeline
  - Resource sharing!
- Increase throughput performance by ~30% at ~10% higher hardware cost

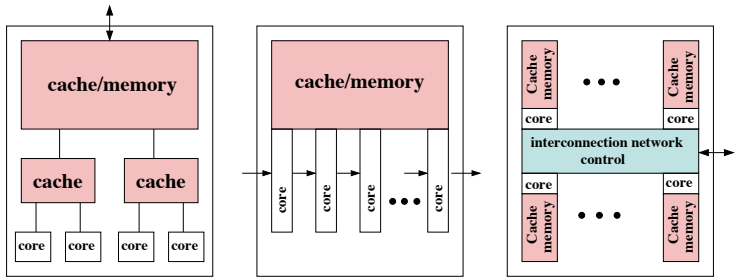


- Some examples:
  - Intel i7 / i9 / Xeon
  - IBM Power8 (12 cores, 8 threads/core)
  - Oracle SPARC M8 (32 cores, 256 threads)

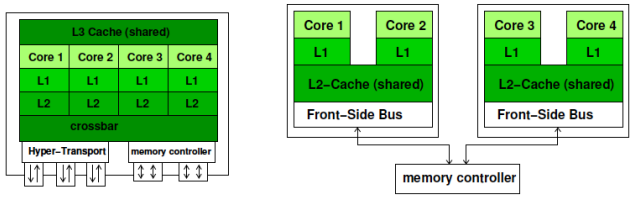
Multicore Processors

- 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



Design options for multicore processors - Hierarchical design (example)

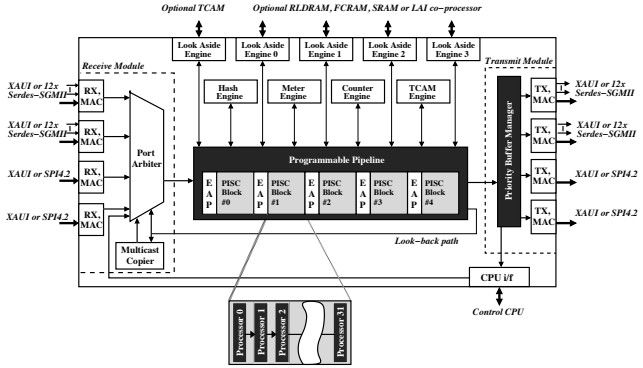


Quad-core AMD Opteron (left) and Intel Quad-core Xeon (right)

Design options for multicore processors - Hierarchical design (example)

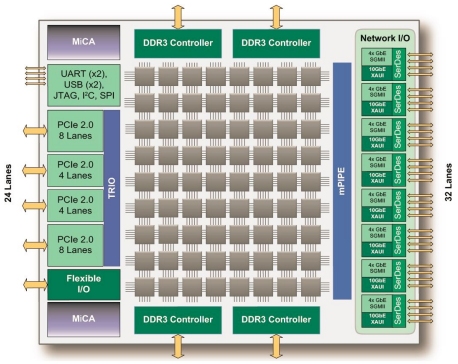
Nvidia Pascal (GP100)

Design options for multicore processors - Pipeline design (example)



Xelerator X11 Network Processor

Design options for multicore processors - Network-based design (example)



TILE-Gx8072 Processor Block Diagram

Intel Teraflop-chip; 80 cores in 8x10 configuration
Tilera TILE-Gx72; 72 cores

End of part 1