Introduction into Multicore Programming

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

Content of the Course

- Challenges of Multicore Programming
- Multicore Designs from some of the Leading Chip Manufacturers
- Roles of the Operating Systems
- Process Programming
- Thread Programming
- Communication and Synchronization
- Introduction into some Thread-Safe C++ Component Libraries
- Multicore Software Design

- No GPU (CUDA) programming on this course
- No *"Reduced Instruction Set Computing" (RISC)* programming (e.g.: ARM MIPS, etc.)

Organization/Requirements

Karoly.Bosa@jku.at

- Lectures (**english**)
- Required knowledge: **C/C++** Programming Skill, Basic knowledge about Linux/Unix, UML
- There will be a **written final exam**.

- (Solving some practical exercises may be also required.)

Slides, exercises and other information are/(will be) available at:

http://moodle.risc.uni-linz.ac.at/course/view.php?id=66

Literatures

The course material will be based on among others the following book:

Cameron Hughes, Tracey Hughes
Professional Multicore Programming: Design and Implementation for C++ Developers

ISBN: 978-0-470-28962-4

http://www.amazon.de/Professional-Multicore-Programming-Implementation-Developers/dp/0470289627/ref=sr_1_1?ie=UTF8&s=books-intl-de&qid=1268746754&sr=8-1-catcorr

Slides, exercises and other information are/(will be) available at:

http://moodle.risc.uni-linz.ac.at/course/view.php?id=66

What Is a Multicore?

- A *multicore* is an architecture design that places multiple processors on a single computer chip (*Chip Multiprocessors or CMPs)*.
- Each processor is called a core.
- Way to achieve gains in overall system performance:
  - After increasing the clock frequency has started to hit its limits in terms of cost effectiveness.
  - the CMP design has recently become the preferred method of improving overall system performance.
- But regular desktop software have not been designed to take advantage of the new multicore architectures.
- Desktop softwars have to be redesigned **(to see any real speedup).**

The Promise of Multicore I.

Dr. Edward Lee,
an electrical and computer engineering professor,
At the University of California-Berkeley:

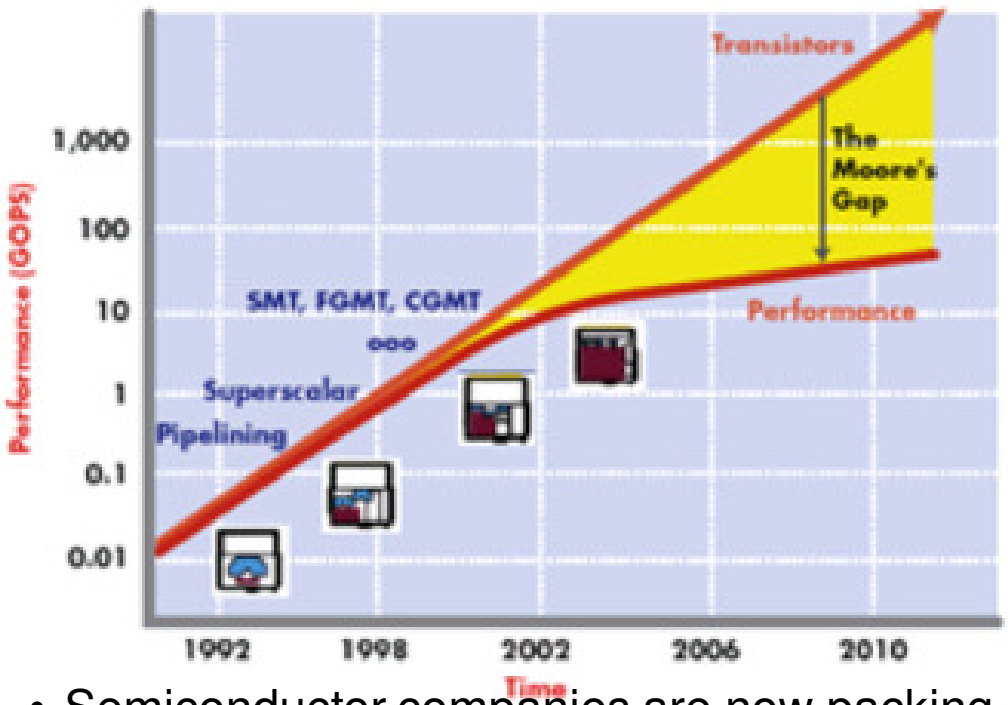
“Many technologists predict that the end of the Moore’s Law will be answered with increasingly parallel architectures. If we hope to continue to get performance gains in computing, programs must be able to exploit this parallelism.”

The Promise of Multicore II.

Bill Gates,
founder of Microsoft, Inc.:

“The fully exploit the power of processors working in parallel… software must deal with the problem of concurrency. But as any developer who has written multithreaded code can tell you, this is one of the hardest tasks in programming.”

Gap in the Moore's Law



- Performance kept pace till 2002 due to technologies like pipelining, caching and superscalar designs.
- Increasing the clock rate of CPUs is approached the limit of cost effectivenes (*The Power Wall*)

- Semiconductor companies are now packing more cores in the processor, instead of increasing the speed of the processors.
- There are predictions that the processors will support 100's of cores in near future.
- **The responsibility of improving performance has now moved from hardware to software** and the big question is whether the software world prepared for it?

Multicore and Multiprocessor

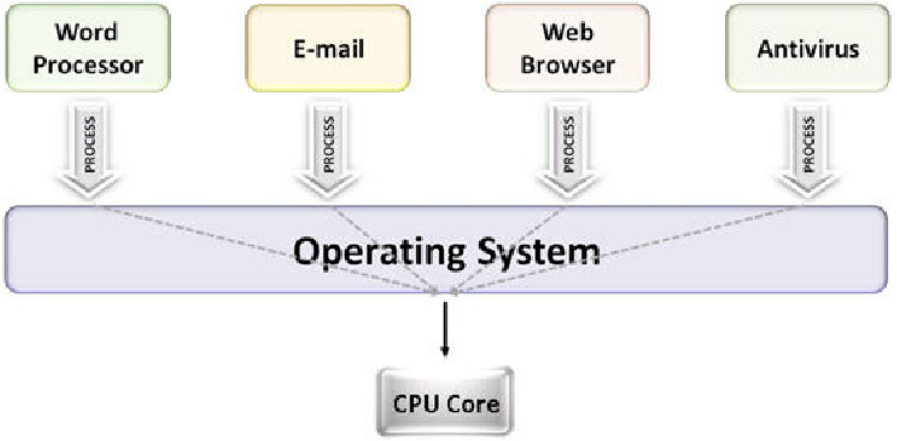
Karoly.Bosa@jku.at

- A muticore system is a single-processor CPU that contains two or more cores.
- Multicore systems share some resources that are often duplicated in multiprocessor system (e.g.: L2 cache and FSB).
- Multicore systems provide performance that is similar to mutliprocessor systems, …
- … but often at a significantly lower cost.

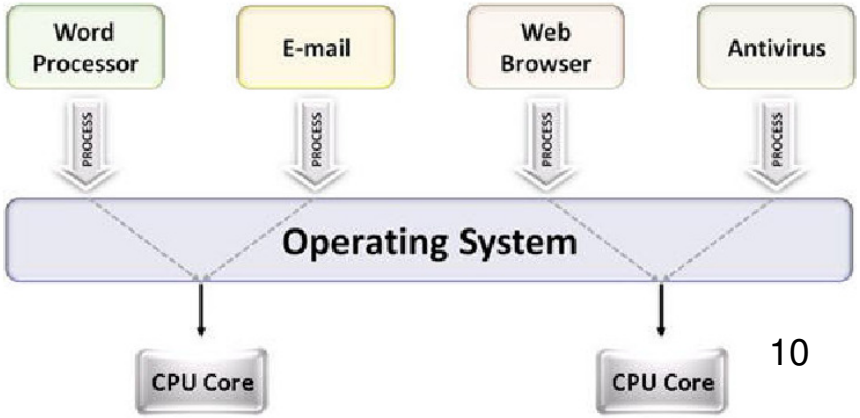
Multicore and Multitasking

Karoly.Bosa@jku.at

- Multitasking refers to the ability of the OS to quickly switch between each computing task and
- Give the impression the different applications are executing simultaneously.
- **In single core:** only one task runs at any point in time, but OS apply task scheduling.



- **In multicore**: Multitasking OSs can truly execute multiple tasks concurrently



Multicore and Multithreading

- Multithreading (and multiprocessing too) extends the idea of multitasking
- You can subdivide operations into individual threads/processes
- Each of them can run in parallel
- The OS divides processing time not only among different applications but also among each thread/process within an application.

Multicore Slow Done I.

- The widespread adoption of multi-core hardware is in many cases actually slowing down computing.

- How can this be? Don't more cores mean more computing power? Doesn't more computing power mean software that runs faster? **The simple answer is no**.

- The software must be written specifically for the multi-core paradigm.

- The transition from a programming model based on one processor to the one based on many processors involves a huge leap in complexity and skill.

- With the multi-core paradigm, the application itself must be aware that many processors are available.

Multicore Slow Done II.

- The low cost and wide availability of CMPs bring parallel programming design into the everyday life of software developers.
- But not every software application requires multiprocessing or multithreading.
- **Parallelism and multiprocessing come at a cost**:
  - Some software solutions and computer algorithms are better implemented using sequential programming techniques.
  - In some cases, introducing the overhead of parallel programming techniques into a piece of software can degrade its performance.
- If the amount of effort (for computer) required to solve the problem sequentially in software is less than
  - the effort required to **create additional threads** and processes or
  - the effort required to **coordinate communication** between concurrently executing tasks,

  then the sequential approach is better (developer must decide).

For software developers…

From a logical point of view,

there is no significant difference between programming for **multiple processors in separate packages** and programming for **multiple processor core contained on a single chip**.

From practical point of view,

there may be performance differences, because the new CMPs are using advances in bus architectures and in connections between processors.

Nevertheless, the potential performance gains, the design and implementation are very similar.

The possible gains are limited by the fraction of the software that can be **parallelized** to run on multiple cores simultaneously (Amdahl's law).

No Platform Independent Design

- In order to take full advantage of a multicore platform, you need to understand what you can do to access the capabilities of a particular type of CMP.

- You need to understand what elements of a particular type of CMP you have control over.

- You need a basic understanding of the processor architectures.

Multicore Architectures

- CMPs come in multiple flavors:
  - two processors (dual core),
  - four processors (quad core),
  - eight processors (octa - core), etc.

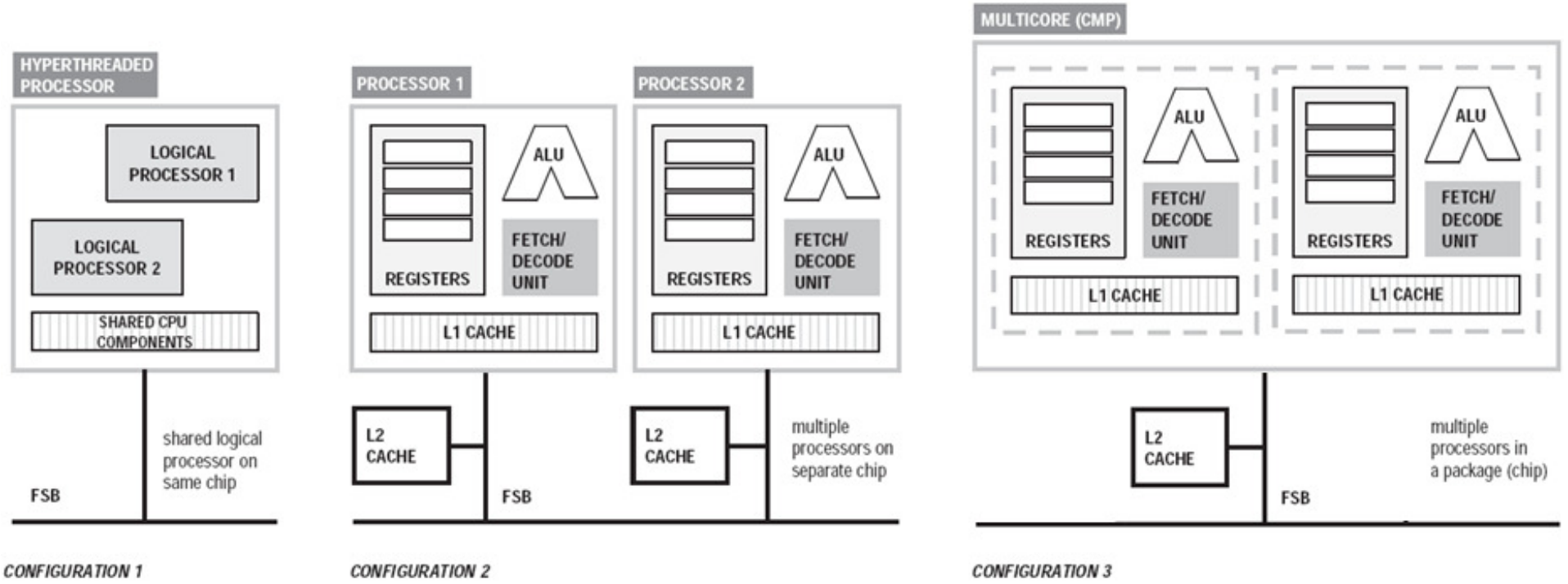
- There are several variations in how cache and memory are approached.

- The approaches to processor - to - processor communication vary among different implementations, too.

- The CMP implementations from the major chip manufacturers each handle the I/O bus and the Front Side Bus (FSB) differently (see later).

Three Common Configurations that Support Multitasking

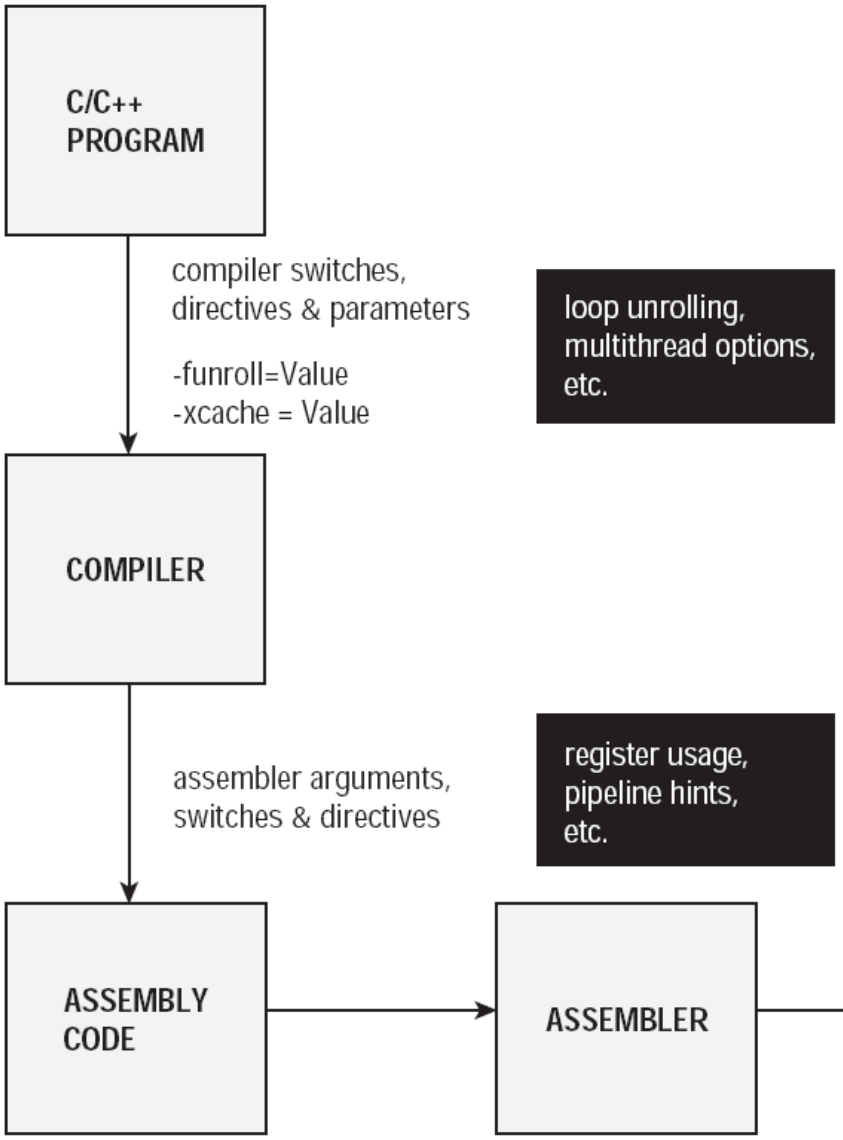
Karoly.Bosa@jku.at



Remark: Some multicore designs support hyperthreading within their cores. For example, a hyperthreaded dual core processor could present itself logically as a quad core processor to the operating system.

The Compilation and the CPU Instruction Set

Karoly.Bosa@jku.at



The compiler can have options for how to produce the object code:

- how registers are used,
- whether to perform loop unrolling or
- how much L1 or L2 cache is present, etc.

Compiler Switches I.

Title	Description	Examples of Usage
Fast optimization	This option enables the best possible generic switch combination to optimize for speed (e.g choose a CPU specific instruction set).	-fast Enables the vectorizer. (Intel)
Optimization (involves many other compiler switches)	This option detects incompatible processors; error messages are generated during execution.	**-O1** Optimized to favor code size and code locality and disables loop unrolling, software pipelining, and global code scheduling. **-O2** Turns pipelining ON.
Parallelization with OpenMP	With this option the compiler generates multithreaded code based on OpenMP directives in the source code added by the programmer.	**#pragma omp parallel** **{** **#pragma omp for** // **your code** **}**

Compiler Switches II.

Karoly.Bosa@jku.at

Title	Description	Examples of Usage
Loop unrolling	These options enable loop unrolling. This option makes code larger, and may or may not make it run faster.	**-funroll-loops** Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. (**-funroll-all-loops**)
Floating point	Set of switches that allows the compiler to influence the selection and use of floating-point instructions.	**-fschedule-insns** Tells the compiler that other instructions can be issued until the results of a floating-point instruction are required (included in **-O2**). **-ffloat-store** Tells the compiler that when generating object code do not use instructions that would store a floating-point variable in registers.

Compiler Switches III.

Karoly.Bosa@jku.at

Title	Description	Examples of Usage
Loop unrolling	This option enables loop unrolling. This applies only to loops that the compiler determines should be unrolled. If n is omitted, lets the compiler decide whether to perform unrolling or not.	**-unroll<n>** Enables loop unrolling; <n> sets the maximum time to unroll the loop. **n = 0** Disables loop unrolling, only allowable value for 64-bit architectures. (Intel)
Memory bandwidth	This option enables performance tuning and heuristics that control memory bandwidth use among processors. It allows the compiler to be less aggressive with optimizations that might consume more bandwidth, so that the bandwidth can be well-shared among multiple processors for a parallel program. This option is used for 64-bit architectures only.	**-opt-mem-bandwidth<n>** **n = 2** Enables compiler optimizations for parallel code such as pthreads and MPI code. **n = 1** Enables compiler optimizations for multithreaded code generated by the compiler.

Compiler Switches IV.

Karoly.Bosa@jku.at

Title	Description	Examples of Usage
Code generation	With this option code is generated optimized for a particular architecture or processor; if there is a performance benefit, the compiler generates multiple, processor specific code paths; used for 32- and 64- bit architectures.	**-ax<processor>** Generates optimized code for the specified processor. **-axS** Generates specialized code paths using SIMD Extensions 4 (SSE4) vectorizing compiler and media accelerators instructions. (Intel)
Auto parallelization	This option identifies loop structures that contain parallelism and then (if possible) safely generates the multithreaded equivalent executing in parallel.	**-parallel** Triggers auto parallelization. (Intel)

Compiler Switches V.

Karoly.Bosa@jku.at

Title	Description	Examples of Usage
Thread library (auto parallelization)	This option causes the compiler to include code from the Thread Library; The programmer needs to include API calls in source code.	**-pthread** Uses the pthread library for multithreading support. (-ftree-parallelize-loops=n)

- Platform independent general code optimization with GCC:
  - **-O -O1** Optimizing compilation takes somewhat more time, and a lot more memory for a large function.
  - **- O2** Optimize even more. GCC performs nearly all supported optimizations that do not involve a *space-speed tradeoff*.
  - **-O3** Optimize yet more.
  - **-Ofast** Disregard strict standards compliance (from gcc 4.6).
  - **-O0** Reduce compilation time and make debugging produce the expected results. This is the default

- For more information see: **http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html**

Examples: Floating Point Switches

- GNU gcc compiler:
  gcc -ffloat-store my_program.cc

- SUN C++ compiler:
  CC -fma=used my_program.cc

Cross-Platform Compatibility

- The **UltraSparc, Opteron, Intel Core 2 Duo**, and **Cell processors** are commonly used multicore configurations. These processors each support high-speed vector operations and calculations.

- They have support for the **Single Instruction Multiple Data (SIMD)** model of parallel computation. This support can be accessed and influenced by the compiler.

BUT:

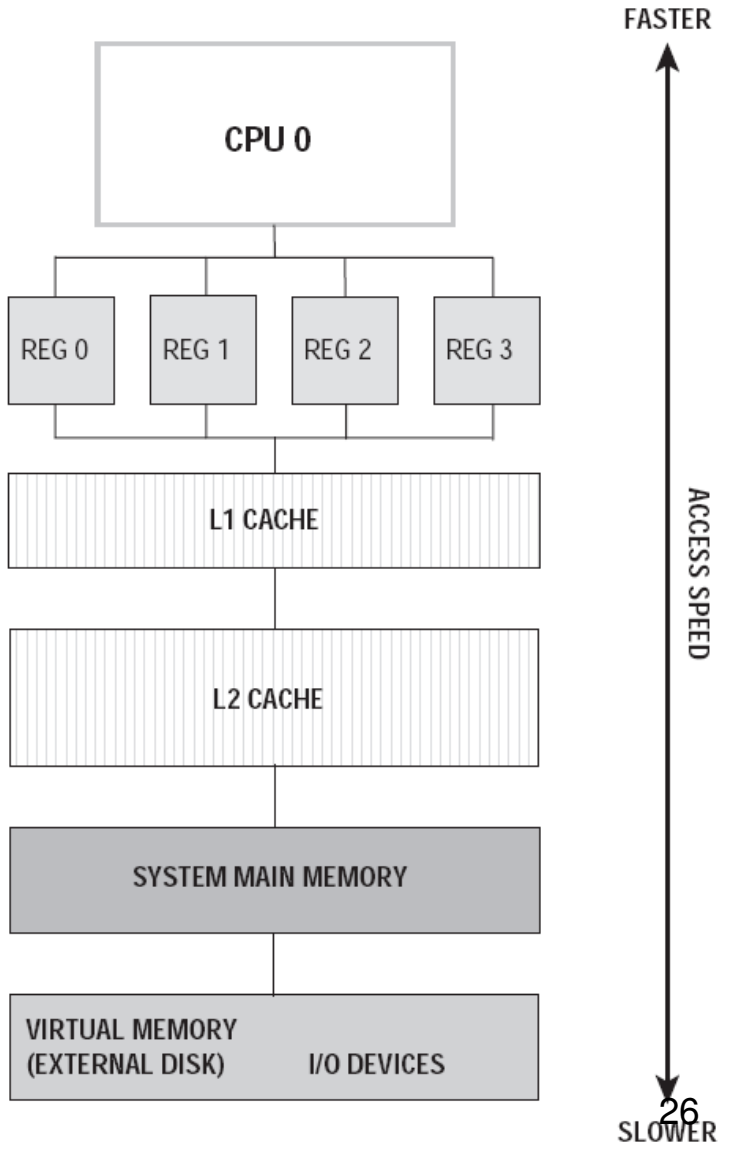
- Using many of these types of compiler options cause the compiler to optimize code for a particular processor(!).

- If cross-platform compatibility is a design goal, then compiler options have to be used very carefully.

Memory I.

Karoly.Bosa@jku.at

- A typical CPU operates only on data stored in its registers.
- Prior processing data pass through some kind of memory.
- Most things pass through many levels of memory.
- Each stage the memory performs at a different speed(!).
- But memory size is also a factor:
  - registers: 32, 64 bits
  - L1 and L2 caches: Megabytes
  - Main memory: Gigabytes
- The **connections between the memory types** also have a major impact on overall system performance.



Registers

Registers are:

- Small but fast memory that are directly accessed by the core,
- Volatile when the program exits any data that it had in its registers are gone,
- The registers are usually located inside the processor and, therefore, have **almost zero latency(!),**
- Most C/C++ compilers have switches that can influence register use.

Remark: In addition to compiler options that can be used to influence register use, C++ has the asm{ } directive, which allows assembly language to written within a C++ procedure or function.

Cache

- *Cache* is memory placed between the processor and main system memory(RAM).

- While cache is not as fast as registers, it is faster than RAM. But its capacity is greater than the registers, but less than the RAM.

- Cache is used to contain copies of small chunks of memory which may be needed by the processor(s) in the foreseeable future. This guessing(!) works with following methods:
  - **Temporal locality** is the tendency to reuse recently accessed instructions or data.
  - **Spatial locality** is the tendency to access instructions or data that are physically close to items that were most recently accessed.

- Cache increases the effective memory transfer rates and, therefore, overall processor performance (only in case of correct guessing).

Level 1 and Level 2 Caches

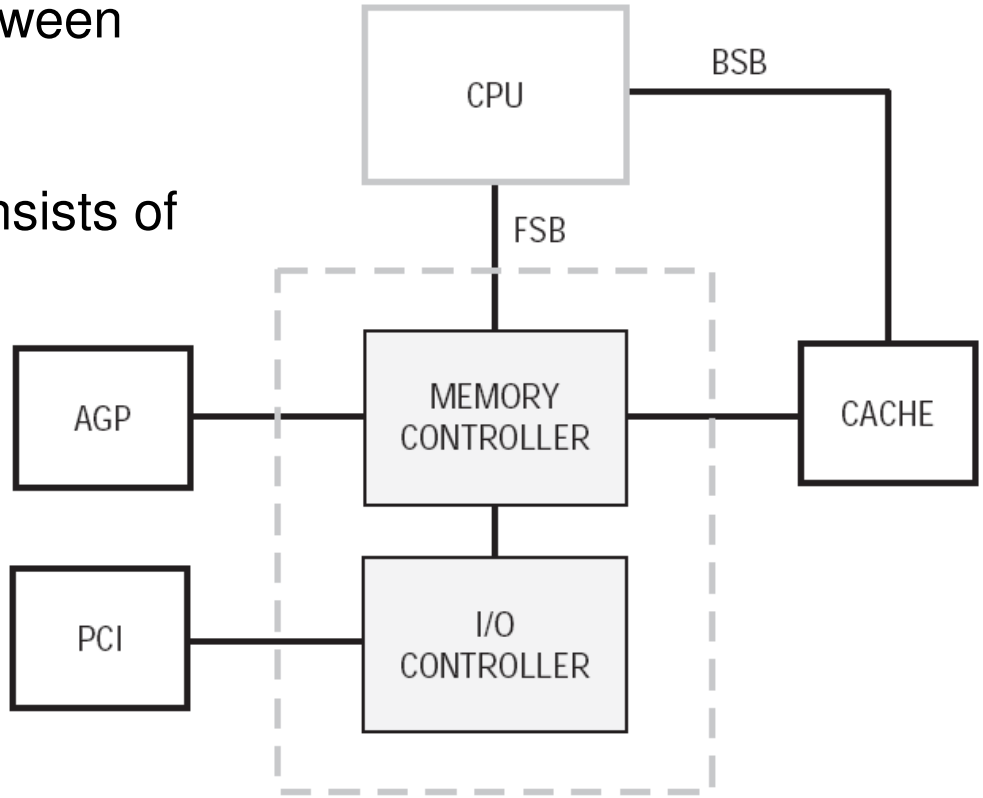
- ***Level 1 Cache:***
  - small in size sometimes as small as 16K,
  - usually located inside the processor and
  - used to capture the most recently used bytes of instruction or data.

- ***Level 2 Cache:***
  - bigger and slower than L1 cache (L2 cache measured in megabytes),
  - stored on the motherboard (outside the processor), but this is slowly changing,

Remarks: By compiler switches it can be given some hint as to how much L1 or L2 cache is available or a hint about the properties of the L1 or L2 cache.

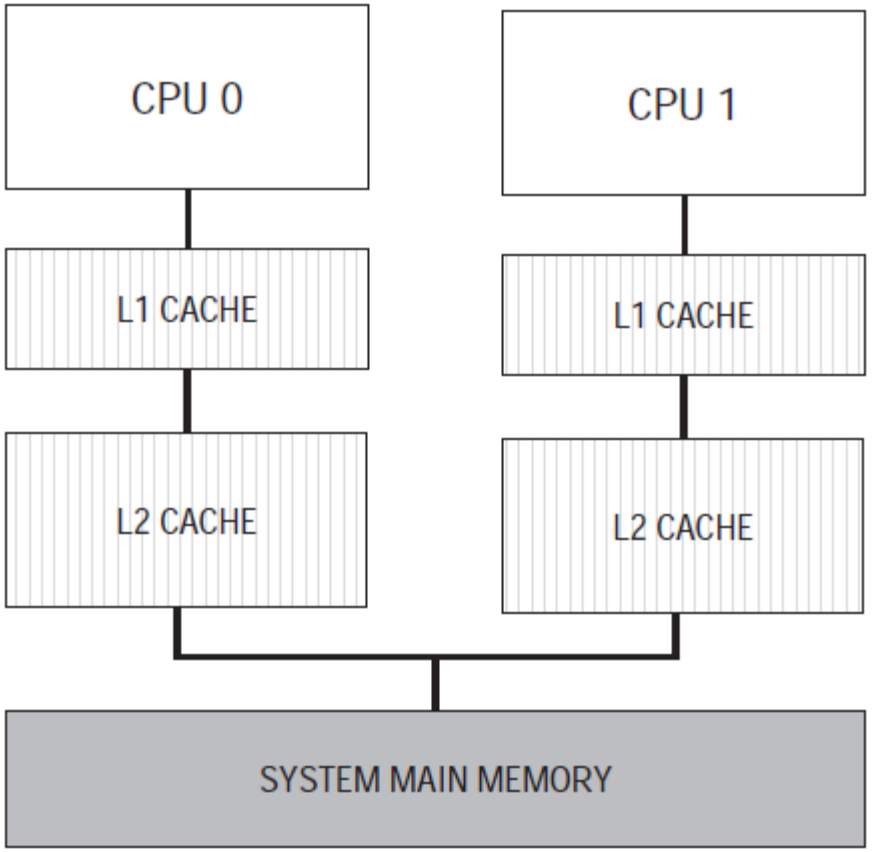
The BUS Connection

- The bus is a channel or path between components in a computer.
- A basic system configuration consists of two main buses:
  - a system bus or the Front *Side Bus (FSB)* and
  - and an *I/O bus*.
- If the system has cache, there is also usually a *Back Side Bus (BSB)* connected to the processor and the cache.
- Buses have the potential for throughput bottlenecks(!).



Multicore Architectures: UMA

- Uniform Memory Access (UMA)
- Processors share a single memory
- Shared address space is used



UMA configurations are often symmetric multiprocessor (SMP) architectures (all processors are the same type and they have a uniform latency from memory).

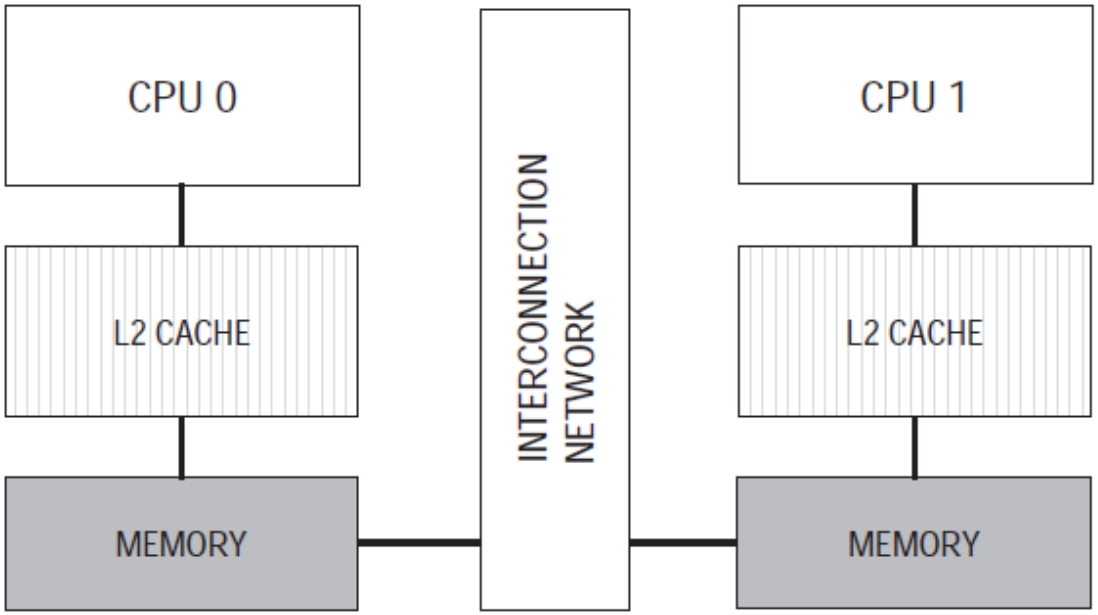
Multicore Architectures: NUMA

Karoly.Bosa@jku.at

- Non-Uniform Memory Access (UMA)

- Each processor has access to its own fast local memory through the processor's on chip memory controller.

- each block of memory shares a single address space(logically shared).



- NUMA architecture has a distributed but shared memory (DSM) architecture(!).

- The NUMA architecture primarily addresses the scalability bottleneck of the SMP architecture (where CPUs accessing the same memory bus).

Multicore Application Design I.

Some cases the optimization of software to a particular architecture has higher priority than platform independency feature, e.g.:

- High transaction software servers
  - Database
  - Financial transaction servers
  - Application servers
- Kernels
- Game engines
- Device drivers
- Large-scale matrix and vector computations
- Compilers
- Database engines
- High-definition computer animation
- Scientific visualization modeling, etc

Multicore Application Design II.

- Major Focus : taking advantage of CMP architecture.

- Multicore application design and implementation uses parallel programming techniques.

- The design process specifies the work of some task:
    - as either two or more threads,
    - two or more processes, or
    - some combination of threads and processes.