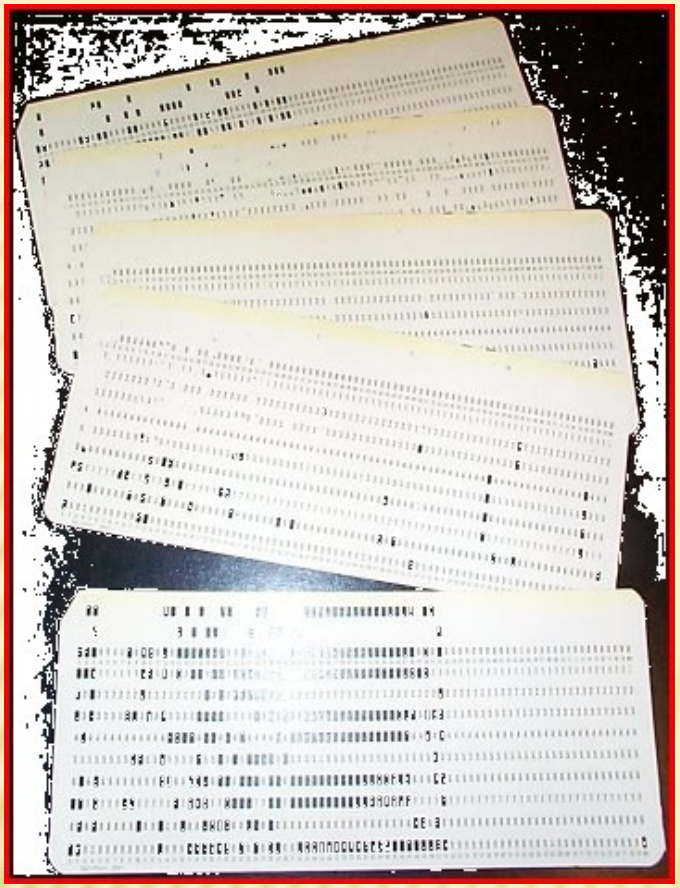


PhD/Master course, Uppsala 2011

SCIENTIFIC PROGRAMMING

NIKOLAI PISKUNOV

RATIONALE

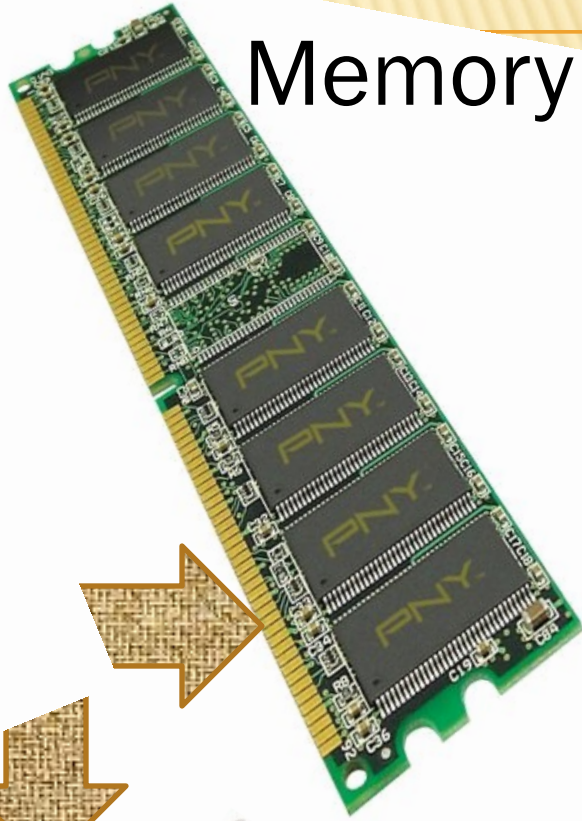
- Understanding the interaction between your program and computer
- Structuring the code
- Optimizing the code
- Debugging strategies
- Parallelization
- A complementary course to *numerical methods* and *programming languages*

COMPUTERS: HARDWARE

Peripheral devices

CPU

Memory



GPU

COMPUTERS: HARDWARE

- Computers perform any data transformation on registers or in pipelines!
- Registers are typically **64-bit** long. Intel Core I series has four 14 stage pipelines. Each stage is 32-bit long. One important conclusion is that double precision register operations are not much slower than single precision!
- Bus operates at rates of a few hundred MHz. **How does this work with a 3 GHz CPU?**
- Bus width×speed determines memory access performance.

WHAT DOES THE HARDWARE DO?

Simple example: **addition**

We want to add two numbers:

A (located in memory M_A)

+

B (located in memory M_B)

| |

The result is stored to memory M_C

WHAT DOES THE HARDWARE DO? ADDITION

1. Load content of memory M_A to register R_1
2. Load content of memory M_B to register R_2
3. Add R_1 and R_2 and store result in R_3
4. Save the content of register R_3 to memory M_C

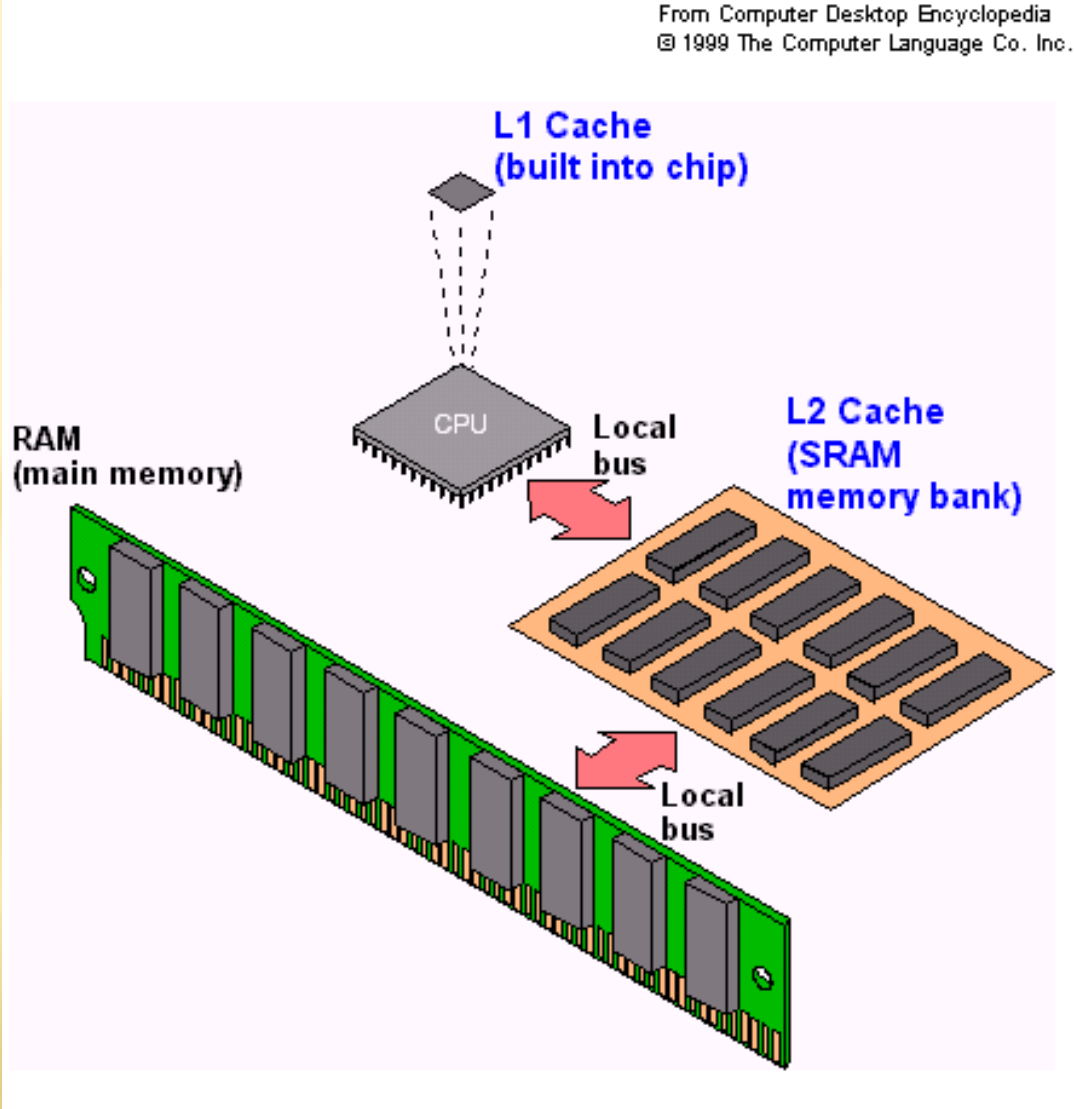
Registers are part of CPU. Modern CPUs are so fast that addition is takes negligible time compared to memory access

SOLVING MEMORY ACCESS BOTTLENECK

Solution is **cache memory**
CACHE (French) *hidden*
Very fast and very close

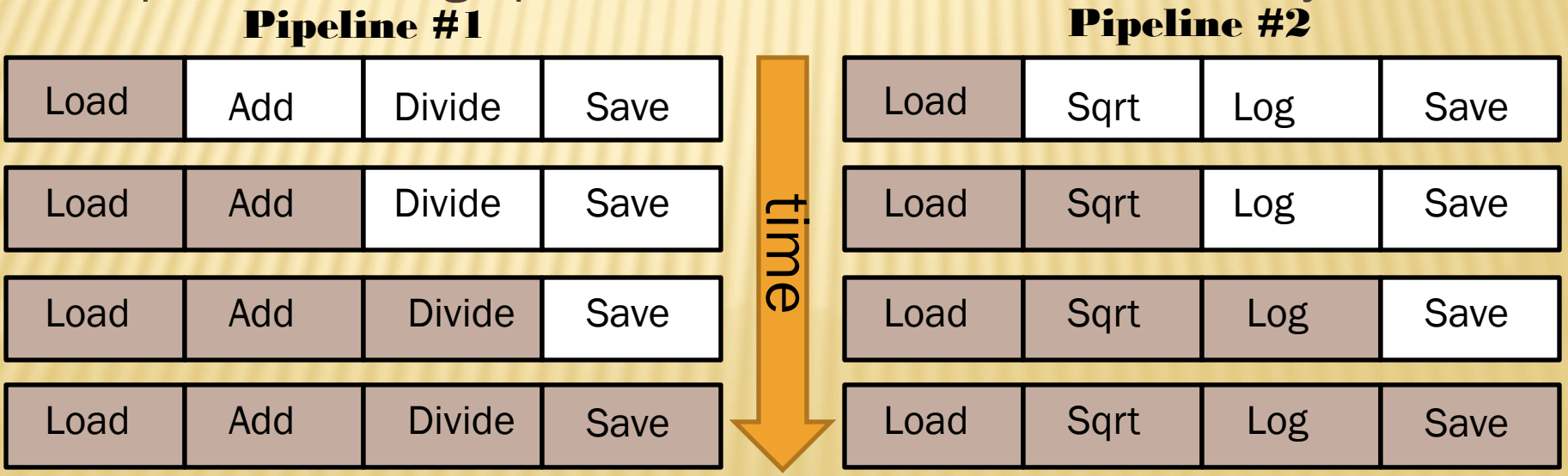
Loading and saving between main memory and cache memory is done by a "separate" CPU which takes an advance look at the program

Multi-core CPUs will have three levels of cache memory with only L3 shared by all cores



CACH MEMORY: ADVANTAGES

- Cache memory can be loaded/saved in blocks, very fast
- Multiple registers can be loaded simultaneously
- Multiple operations performed in the same time – pipeline(s). Intel Core i has 4 pipelines per core each capable of doing up to 14 instructions simultaneously.



CACHE MEMORY: PROBLEMS

Main memory is typically read within 40-80 nanoseconds. Cache memory must operate close to the CPU clock rate: 3 GHz CPU requires 3 nanoseconds cache (10 CPU cycles)!!!

- To be that fast cache memory must be small and static
- Cache memory is very expensive
- Branching in the code can be deadly!!!

BRANCHING

- Two cases: explicit and conditional
- Explicit is easy: go to <address of the command>
- Conditional: **if** *true* do <next command>
  else *skip it* and <do the one after>
  is not so bad either
- Combination of the two is bad:
  if *true* **go to** <far, far away>
  else …
- Why is this bad? Examples of such situation

BRANCHING PREDICTION

- What happens with conditional branching is that the content of your cache may become inconsistent with the next part of the code
- In this situation (known as cache miss) you need to save cache entries that are not saved yet and load new memory parts
- A special part of a modern CPU (branch predictor) tries to guess the branching direction
- Loops are converted to count-down scheme helping with prediction

INTERACTION WITH THE SOFTWARE: MAMORY ALLOCATION/PROTECTION

- Computer codes need variables, arrays etc. How this is done in hardware? There are two concepts: *stack* and *heap*.
- Stack is characterized by its start, length and current position. Current position is stored in a special register. Many stacks can co-exist but cannot be accessed simultaneously (**dynamic variables**)
- Heap is a stack with the current position moving from the largest memory address backwards (**static variables**)
- Hardware provides memory protection: two processes (programs) cannot read/modify each others memory
- No protection is offered within a single program space!!!

ADDRESSING MEMORY

- Memory can be addressed directly or by segments (segment:offset). 64-bit CPUs can address up to 2^{64}-1 memory mode while 32-bit memory model is restricted to 4 Gb of address space.
- Modern computer memory bus is 64-bit wide so it can carry the address or the content of 8 memory bytes in one go.
- Memory is accessed in the following sequence:
  - CPU sends memory address to the memory controller through the bus (segment and offset are sent together).
  - Memory controller fetches the content and sends it back to the CPU

INTERACTION WITH THE SOFTWARE: CONTEXT SWITCHING

Programs often use subroutines. How does this work?

- Special stack is organized for this purpose
- Before a subroutine is called, all the registers and a pointer to the next command to be executed are stored in this stack
- New stack is created or an existing one is pulled up for the subroutine and the execution starts
- On exit the content of the registers is restored from the stack and the execution continues in the calling code
- This process is called **context switching**
- Context switching is the base of multi-processing

EXCEPTIONS

- What happen when a code tries to perform a division but the content of the divider register is zero? Actually, nothing. The result is marked by a special combination of bits called NaN. No other action is normally taken
- Modern processors have a special mode called **Exception Tracking**. When activated, the processor (hardware) initiates context switching calling exception handling program. This allows locating the place in the code where the problem occurred.

HOW DO WE GET IN AND OUT?

- CPU is also connected to the peripheral devices (keyboard, mouse, hard drives, network, printers) and to the graphics card
- Expensive graphics cards are powerful computers all by themselves with GPU, fast bus and huge memory. In small laptops the main CPU acts (at least partially) as a GPU. A special dedicated bus connects the CPU and the graphics

DIRECT MEMORY ACCESS

- External memory (HD, SSD) is much slower even than the main memory (10 microseconds versus 50 nanoseconds)
- Bus controller allows direct memory access
  - the CPU specifies that certain information from a disk must be copied to a certain memory area
  - The copying does not involve the CPU
  - It works in both directions HD⟶RAM or RAM⟶HD
- Transferring an optimal amount of data in one such operation results in large gain of speed

MORE ABOUT HD I/O

- To make the I/O operations more efficient and take maximum advantage of the direct access the operating system often uses RAM to simulate HD.
- The synchronization between the memory buffer and the HD is done asynchronously, once in a while.
- It speeds up disk access but creates a potential threat to the integrity of the file system.

NEXT LECTURE: PROGRAMMING LANGUAGES

Literature:

The C programming Language
Kernighan and Ritchie

FORTRAN 90/95 explained
Metcalf

Python http://www.python.org/doc/

HOME WORK

Find out what are you going to be using for the exercises:

- What computer (Intel, PowerPC)
- What operating system (Windows, Linux, Mac OS X)
- What programming languages do you have available (C, C++, FORTRAN, Python ...)