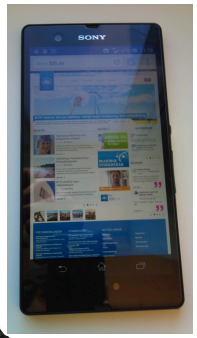
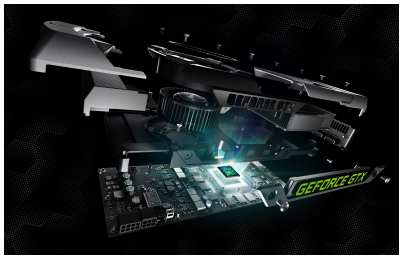
Multiprocessor programming, DV2597/DV2606

Håkan Grahn
Department of Computer Science
Blekinge Institute of Technology
Hakan.Grahn@bth.se





What do they have in common??

How to get more done?

- Work faster
  - Worked well until approx. 2005

- Work together and help each other
  - Since 2005 => Multicore systems

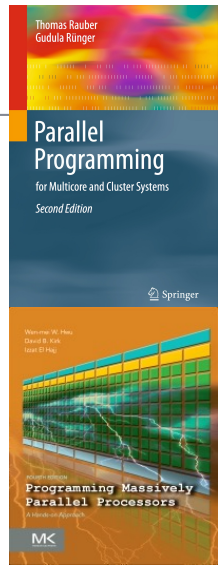
Main question

How do you get
many units (processors/cores)
to
work efficiently together???

Course information

- Text books:
  - T. Rauber and G. Rünger: “Parallel Programming for Multicore and Cluster Systems, 2nd ed.”
  - W.-m. W. Hwu, D.B. Kirk, and I. El Hajj, “Programming Massively Parallel Processors: A Hands-on Approach, 4th ed.”
- 16 lectures: **Håkan Grahn** + Lars Lundberg
- 2 (DV2606, 6hp) / 3 (DV2597, 7.5hp) laboratory assignments: Suejb Memeti

Examination: Passed written exam (3p) and assignments (2 x 1.5p / 3 x 1.5p)!



Lab information

- Lab 1 – Threads programming
  - Parallel Gaussian elimination
  - Parallel Quick Sort
- Lab 2 – CUDA 1
  - Parallel Odd-even sort (two versions)
  - Parallel Gauss-Jordan row reduction (extension of Gaussian elimination from lab 1)
- Lab 3 - CUDA 2 (only DV2597)
  - Accelerating deep learning for handwritten digits using a GPU (LeNet5)

Some practical matters

- Make sure you are formally registered on the course!
- **Mandatory** to enroll for the written exams
- Schedule, lecture slides, and lab exercises are published on Canvas
- Make sure you understand all practical issues concerning the lab assignments
- All labs are done in C/C++
- Lab assessment are done through an **oral discussion and a report + code submission** on Canvas

Aim and Learning outcomes (not complete)

- Thoroughly describe
  - the design and working conditions of different types of multiprocessors and parallel computer systems
  - different programming models for multiprocessors and parallel computer systems
  - different parallel program design principles to achieve high performance as well as how the architecture affects the performance of a program
- Practically apply different principles and techniques for developing and implementing parallel program
- Analyze the data flow in a problem in order to partition the program in parallel parts
- Perform simple performance measurements and performance analysis of parallel programs
- Implement a general algorithm / program for execution on a GPU
- Perform suitable optimizations of a parallel program, e.g., reducing communication and synchronization overhead, to achieve high performance

Contents (from the course descriptor)

- Introduction to multiprocessor systems and parallel programming
- Design principles for parallel computer architectures and computer systems
- Programming models and design principles for parallel programs to achieve high performance, e.g., partition of work to fit the target architecture, communication, and synchronization
- Libraries for parallel programming, e.g., pthreads and CUDA
- General-purpose computing on GPUs (GPGPU)
- Example algorithms for a number of interesting application domains
- Practical training in development of parallel programs

Week	What	Who	Theme	Chapter
W 45	F1	HGR	Course intro., Parallel computer architecture I	R&R: Chap 2
	F2	HGR	Parallel programming models	R&R: Chap 3
	F3	HGR	Thread programming - pthreads	R&R: Chap 6
W 46	F4	HGR	Performance analysis of parallel programs	R&R: Chap 4
	F5	HGR	Principles for parallel algorithm design	
W 47	F6	HGR	Dense matrix algorithms	R&R: Chap 8
	F7	HGR	Parallel computer architecture II	R&R: Chap 2
	Lab 1	SCM	Submission lab assignment 1, Threads programming	Deadline: 2025-11-23
W 48	F8	LLU	Introduction to CUDA and GPU programming	HK&E: Chap 1-2
	F9	LLU	Fundamental concepts 1	HK&E: Chap 3-4
	F10	LLU	Fundamental concepts 2	HK&E: Chap 5-6
W 49	F11	LLU	Convolution and Stencil	HK&E: Chap 7-8
	F12	LLU	Parallel histogram and Reduction	HK&E: Chap 9-10
	F13	LLU	Sparse matrix computation	HK&E: Chap 14
W 50	F14	LLU	Application Case Studies	HK&E: Chap 17-18
	F15	LLU	Heterogeneous computing	HK&E: Chap 20(-21)
	F16	HGR/LLU	Summary, repetition	
	Lab 2	SCM	Submission lab assignment 2, CUDA 1	Deadline: 2025-12-14
W 2	Lab 3	SCM	Submission lab assignment 3, CUDA 2 (only DV2597)	Deadline: 2026-01-11

The big picture of the course

Why is parallelism interesting?

How can we exploit parallelism in order
to achieve higher performance?

Gain theoretical knowledge *and* practical experience from
multiprocessor systems and parallel computing!

Why are parallel computers interesting?

- There is a limit to sequential processing, mainly due to fundamental limitations in hardware (power, thermal, energy efficiency).
- Today (and in a foreseeable future) *multicore processors* is the main building block for computer systems.
- There are always problems that need more computational power to be solved (weather forecast, simulations, databases, AI/ML, computer games, etc.).
- Software for parallel computers is moving forward for each year.

Now, and in the future, there is a need for
software developers with parallel programming skills!

Elements of a parallel computer system

- Hardware
  - Multiple processors or cores (+memory and networks)
- System software
  - Parallel operating system
  - Programming constructs to express concurrency
- Algorithms and application software
  - Decomposition of work into subtasks

Utilize the parallel computer system to:

- Achieve speedup, $T_p = T_s/P$ **<= Our focus**
- Solve problems with large memory requirements

Two large challenges in parallel computing

Limited parallelism in the application

- Amdahl's law

$$Speedup = \frac{1}{\frac{Fraction_{parallel}}{NoProc} + (1 - Fraction_{parallel})}$$

Long remote memory access latencies

- Important to obtain communication locality, e.g.,

$$CPI = CPI_{base} + remote\_req\_rate * remote\_req\_cost$$

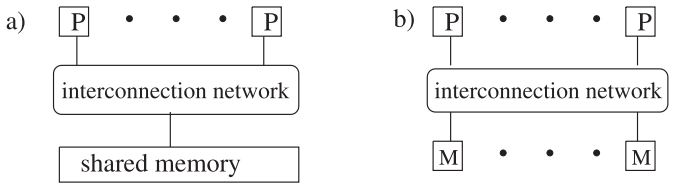
Parallel computer systems

Shared address space:

- Local and non-local memory accesses (implicit)
- example: contemporary multicore processors

Distributed address space:

- Message passing (explicit)
- example: cluster systems



Parallelization of algorithms

Necessary steps to parallelize an algorithm:

1. **Decomposition** of the computations into tasks that can be executed in parallel
2. **Distribution** of the computations and the corresponding data on the processors;
   - The tasks can have a different **granularity** depending on the platform (e.g., instruction-level parallelism, data parallelism, function-level parallelism).
   - Goal: balanced distribution of computations (load distribution);
3. **Data exchange** between the subtasks (distributed address space), i.e., communication/coordination planning;
   - Goal: minimization of communication costs;
4. Definition of **synchronization points**;
   - Goal: avoidance of inconsistent states (shared address space) and minimization of waiting times.

Parallel programming models – the programmer view on a parallel system

- Which types/levels of parallelism are used?
  - Instruction-level parallelism, data parallelism, function-level parallelism, mixed parallelism
- How is the parallelism specified?
  - explicit (e.g. message-passing)
  - implicit (e.g. functional languages)
- In which form is the parallelism being processed?
  - Fork-Join, SPMD or SIMD, Master-Slave, Pipelining
- How is data exchange implemented?
  - shared variables
  - explicit communication (point-to-point messages and collective communication operations)
- Does a cost model exist?

Software for parallel program development

- Parallelizing compiler
- Parallel programming languages: Example: HPF - High Performance Fortran, UPC, X10, Fortress, Chapel
- Message-passing programming (for a distributed address space): Usage of communication libraries (with Fortran, C, C++, Java)
  - MPI - Message Passing Interface
- Thread programming (for a shared address space):
  - Pthreads (library)
  - OpenMP (controlling the parallelism using pragmas)
  - C++ / Java (multithreading is part of the programming language)
  - CUDA (define kernels for parallel execution on the GPU)