Parallel Systems

Conclusions

Jan Lemeire
Dept. ETRO
December 2014

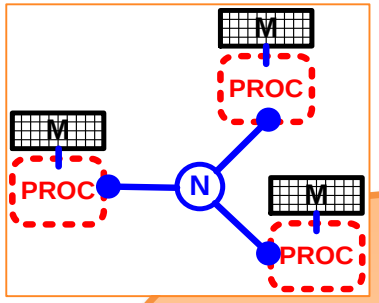
Vrije Universiteit Brussel

Goal of course Part I

- Learn the basic techniques
  - Of the 2 basic paradigms: MP & MT
  - Plus one recent hit: GPU computing
- All are quite low-level
  - Higher-level techniques rely on them
  - Can be learned easily when understanding the lower level
  - E.g. OpenMP for multithreading
- Understand specific parallel issues
  - Indeterminism
  - Race conditions
  - Synchronization
  - deadlocks

Parallel Systems

Distributed memory

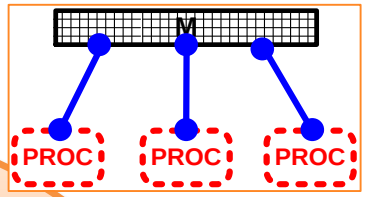


Message-passing MPI

coarse-grain parallelism

Shared memory

coarse-grain parallelism



Explicit multi-threading

OpenMP

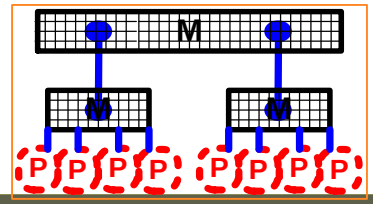
OpenCL/ CUDA

SIMT

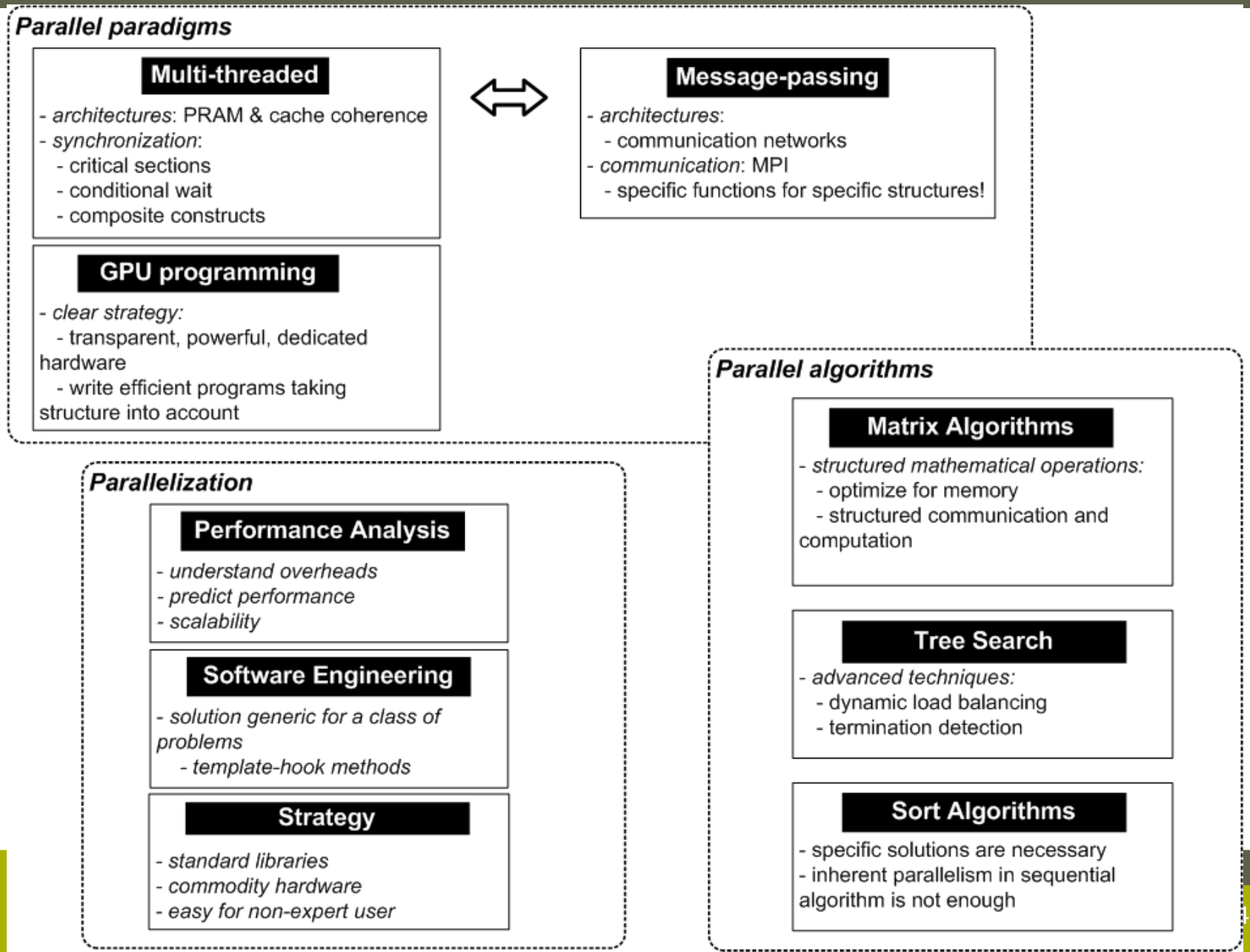
Vector instructions

SIMD

fine-grain parallelism



Course Overview



The Current Big Question

Will Parallel Computing become widely used?

Parallelism

- In the need of processing power?
- Combine resources!

- Cumbersome!!

  Parallelism is not for free…
  - *No compiler that transform a sequential algorithm into high-performant parallel code*

Some comment on blogs

- “A Programmer had a problem. He thought, 'I know, I'll use threads‘. Now the programmer has two problems.”
- “Using multiple threads adds a whole new layer of complexity to your code.”
- “I don't think any extension to C or C++ will suddenly make all the multithreading headache goes away.”
- “I suspect GPUs will suffer from the same problem facing today's multicore: too difficult to program, and too few programmers.”

Parallelism is not for free

Sequential world

- Separation hardware – program (3GL)

With one abstract model for architecture: Von Neumann

- Java: platform-independence
- .net: language-independence

Parallel world

- Ultimate goal: match hardware – program
- *No universal abstract model for parallel architectures!*

  No common HW model can be used, since parallel programming has to break abstractions (for performance) by preventing us from hiding low level details (PPP 70)
- Trade-off performance – genericity

  Performance is program and hardware dependent, and their match

  Efficient programs should be developed specifically …

  Less impact in the sequential case (overhead of OOP or java is small)

Goal of course Part II

- Understand limitations of current technology so that better solution can be build tomorrow
- Provide answers to the following, deeper questions:
  - How make parallelism successful?
  - Which strategies to follow?
  - What does prevent us for exploiting resources in parallel successfully?
  - What should a software developer know?
  - How can we offer parallel processing power to the non-expert user?

Strategy

- Ask yourself:
  - Is it worth it?
  - Is it possible?
  - Is it easy?
  - ➡ Effort ~ Benefit
- Our strategy for make parallelism used
  - Use commodity hardware
  - Use standard approaches which are close to the sequential world: MPI & Multi-threading
  - Develop solutions that are as generic as possible
  - Put them in domain-specific libraries
- For specific, critical solutions, however, parallel technology is mature.

More lessons…

- Sequential program is always (!?) more efficient, simpler and less error-prone
  - What about maintenance of parallel programs? Like keeping documentation up-to-date…
- What do we learn from the GPU case…
  - Economical aspects guide technology
  - NVIDIA's approach: Transparency, good documentation, easy installation, automatic GPU detection an compilation
  - ➡ Already widely used for specific algorithms

  Will it remain and become a standard??

Feedback is welcome

Feedback on course content & organization?

Fill in the online course evaluation!