Introduction to Parallel Computing

Alexandre David

1.2.05

http://www.cs.aau.dk/~adavid/teaching/MVP-08/

Welcome.

Web page: schedule, book, exercises, slides, everything about the course. Also accessible from my personal web page.

E-mail: adavid@cs.aau.dk, also from my web page.

Presentation of the Course

- Parallel Computing
  - Little on parallel hardware
  - Mostly on parallel algorithms and design
- Models for Parallelism (PRAM...)
- Tools for Parallelism (MPI, pthreads...)
- 15 lectures, 3x30 min + exercises
- TA: Claus Thrane crt@cs.aau.dk

Updated from last year, changed content according to feedback and discussions with other lecturers.

No overlap with other courses.

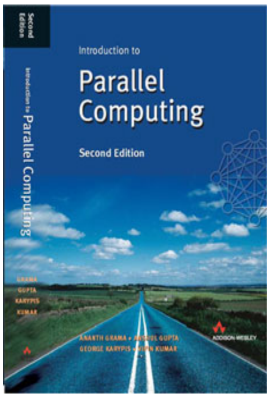
Only place in the curriculum where you have a chance to learn about parallel programming.

Meaning of the course:

•Models for parallel machines and programs, programming paradigms, etc...

•Tools for parallelism: standard API such as MPI, pthreads, OpenMP (not in the course but you have materials in the book).

The Course Book



- *Introduction to Parallel Computing*
- Covers many aspects on parallel computing.
- Both basic and advanced topics.
- I will follow the book but not cover it all.
- You'll be lost if you don't follow the lectures.

Claims to be modular and suitable for a wide variety of undergraduate and graduate level courses.

Covers traditional algorithms (sorting, graph, searching) and scientific computing algorithms (matrix, FFT).

Practical, code examples.

MPI, pthreads, also OpenMP (not in this course): 3 most widely used standards for writing portable parallel programs.

Recent (2003) and solid book.

Show the plan of the chapters (Figures/chap1.pdf).

Course & Assignments

- Lectures will be alternated between theory & practice.
- Assignments will be half theory, half practice.
  - 6-7 assignments, all compulsory.
  - Model: complete them until they are good.
  - Careful: Do not accumulate delay.
  - 2 weeks for completing every assignment.
  - Examination through assignments.

Goals of the Course

- Design, analysis, and implementation of parallel algorithms.
  - Principles of parallel algorithm design.
  - Analytical modeling of parallel programs.
  - Tools such as MPI & pthreads.
  - Some examples.

Main goal (design, analysis, implementation) that needs important notions (sub-goals).

Short motivation:

•Hardware: Parallel computing has changed from tightly scalable message passing platforms to today's inexpensive clusters and multiprocessor machines.

•Software: Programming models have evolved from custom to standard APIs. MPI = standard message passing library, pthreads = thread library, OpenMP = directive based models.

Impact on process of **design**, **analysis**, and **implementation** of parallel algorithms: What this course is about.

Do We Need Parallelism?

- Complexity of parallel programs: How to *specify* and *coordinate* **concurrent** tasks?

  race/deadlock/livelock

- Are there standards?
- Do you need to accelerate applications?

There is another course dealing with the question of *concurrency*, in particular problems of deadlocks, livelocks, synchronization, condition variables, etc… I won't spend much time on the needed concepts, only what I need for this course (more pragmatic and practical approach).

•Specify: How to decompose a problem (most often sequential) into a set of parallel tasks to execute?

•Coordinate: Efficiency (control overhead in extra communication) and correctness issues (race conditions, deadlocks, livelocks).

Recall of race condition (several execution orderings may yield different results with the same program), deadlock (system not responding or doing anything), livelock (infinite loop without progress).

Algorithm standards: Actually in terms of API, now there are some.

Need to accelerate: Think of spending 2 years of development when the platform is going to be obsolete by the time you are finished.

Trends in Hardware

- Everything points towards parallelism from multi-core, hyper-threading, multi-threads, superscalar, ... technologies.
- Because
  - Limits to continue to increment performance with single processors.
  - Other constraints like heat, complexity, yields, etc...

Put a product name on all of these technologies to make it more concrete:

•Multi-core: X2 Athlon64, Intel's dual-core (P4 & new mobile CPU).

•Hyper-threading: Intel's technology to utilize the CPU better (switch thread instead of staying stalled on data).

•Multi-thread: Microsoft Xbox CPU, multithread triple core.

•Superscalar: Every modern GPU and CPU, several instructions in the same clock cycle (pipeline, different execution units).

Single processors have implicit lack of parallelism and have bottlenecks such as critical data paths and limited memory sub-system.

Example: multi-core recent design adopt simpler architectures replicated several times (no out-of-order execution), why? Complexity, efficiency/watt, die-size, price transistor/OPS (operation per second).

Arguments for Parallelism

- Computational power:
  - Moore's law.
  - Translating transistors into useful OPS.
- Memory/disk speed:
  - Performance/yr: CPU +40%, DRAM +10%.
  - How to feed data?
  - What are the problems?
- Parallel platforms: larger aggregate cache+bandwidth+IPC...

•Computational power: The demand for higher and higher computation power always grows, Moore's law is only the technological answer to that demand. We *want* to pursue Moore's law, that's why it still holds. The question is: How to continue from now on? (with the problems mentioned before).

•Memory/disk speed: Overall system performance is defined by CPU speed and the ability of the system to feed data to it. We have cheated so far by bridging the speed gap with caches that work thanks to the **data locality** property of almost all programs. Still we have both problems of **latency** and **bandwidth**. The same applies to disks (you are familiar with RAID technology).

•Parallel platforms: Linear increase in the number of processors of cache, bandwidth, etc… in total. The question is: How to use the increased resource such that the performance has a linear increase as well?

•Other arguments: Data communication (SETI@home a.k.a. search for extraterrestrial intelligence), constraints on location of data & resources that require distributed/parallel algorithms.

Scope of Parallel Algorithms

- Engineering & design.
- Scientific computing.
- Commercial (web) applications.
- Embedded systems.
- Gaming industry.

•Engineering & design: complex physical processes, geometric & mathematical modeling in context of parallel computers.

•Scientific applications: human genome sequencing, computational physics & chemistry.

•Commercial applications: multiprocessor & cluster machines for web & database servers.

•Embedded systems: cars, planes, etc… have many computer systems communicating via some network. 90% of computer systems are embedded systems.

•Gaming industry: Xbox 360 (triple core CPU + general multi vertex/pixel shader engine), PS3 with Cell processor (8 simple computational units + 1 G5 on one core).