Computer Hardware Architecture
Lecture 2

Manfred Liebmann
Technische Universität München
Chair of Optimal Control
Center for Mathematical Sciences, M17
manfred.liebmann@tum.de

November 3, 2015

Motivation

Rationale: Why parallel computing?

- **Before the year ≈2000**
  - Computing power was increased by increasing clock frequency
  - Wait for the next generation of hardware if a program is too slow
  - Increasing the clock frequency increases the power consumption
  - Heat dissipation got problematic for smaller and smaller transistors

- **After the year ≈2000**
  - Duplicate computing units to increase performance: multicore CPU
  - Ongoing trend multicore CPUs: 18 cores and rising
  - Accelerators with dozens, hundreds, or thousands of cores.
  - Drawback: Sequential software does not take advantage of additional cores
  - Parallel software requires often a complete redesign of algorithms
  - Rethink algorithms to fit massively parallel hardware architectures

Writing Parallel Programs

Write a parallel program: Decompose the problem at hand using two main approaches:

- **Task-Parallelism**
  - Decompose the problem into many different unrelated task and put them in a pool
  - Distribute tasks to the computing cores
  - Schedule new work if a core has completed its task and there is remaining work

- **Data-Parallelism**
  - Partition the data of a single task and distribute the work to the computing cores
  - Every core executes similar instructions on a part of the data
  - Vectorization of an algorithm is a prime example

Communication is often needed between the computing cores to exchange data or wait for events to coordinate the execution of the program.

Shared and Distributed-Memory Architectures

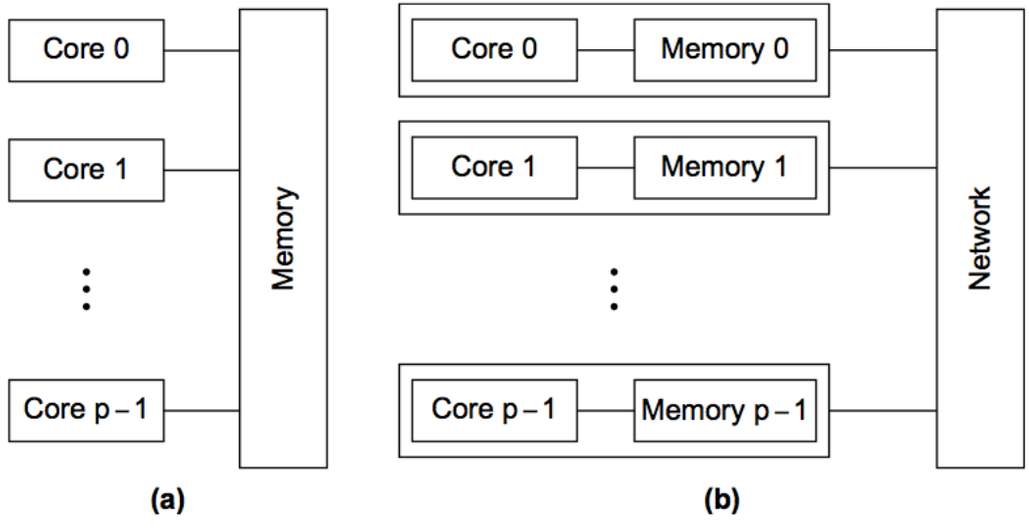

Figure 1: (a) A shared-memory system and (b) a distributed-memory system

On a shared memory system every core has access to the a single memory. On a distributed memory system every core has only access to its memory and data must be exchanged over the network explicitly.

Parallel Programming Interfaces and Extensions

- **Message Passing Interface (MPI)**
  - MPI provides a programming interface to exchange data over a network
  - Works on *both* distributed and shared memory systems
  - The underlying network technology is abstracted away
  - Compatible with Ethernet and Infiniband and proprietary networks
- **OpenMP, OpenACC**
  - OpenMP provides a programming API and compiler extensions
  - Source code is decorated with pragma directives to implement parallel constructs
  - Works *only* on shared memory systems
  - OpenACC extends the programming model for accelerators (Xeon Phi, GPU)
- **CUDA**
  - CUDA implements a single instruction multiple thread (SIMT) parallelization model
  - Sequential code is mapped by the compiler to the vector hardware of a Nvidia GPU