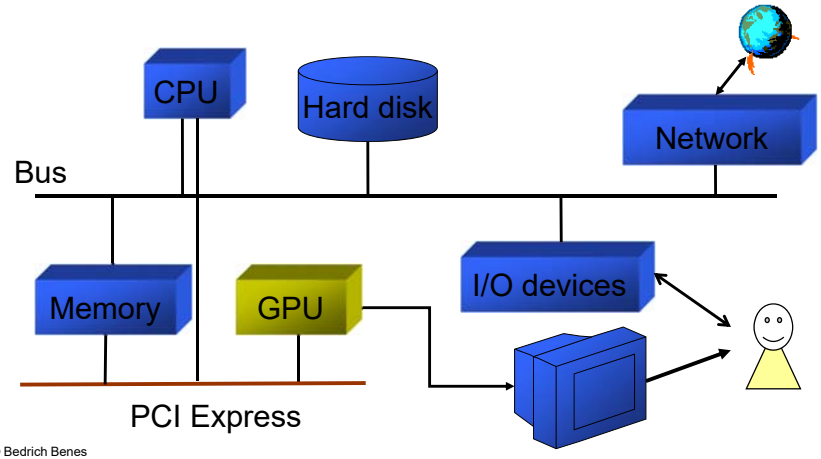
Parallel Computing Introduction

Bedřich Beneš, Ph.D.
Associate Professor
Department of Computer Graphics
Purdue University

PURDUE UNIVERSITY

von Neumann computer architecture



PURDUE UNIVERSITY

von Neumann computer architecture

- I/O devices communicate with the world
- memory stores data and instructions
- CPU
  - processes data,
  - issues GPU instructions via bus
- GPU reads data from the memory via PCIe
- GPU
  reads data and performs vertex
  and fragment operations

PURDUE UNIVERSITY

von Neumann computer architecture

- Traditionally:
  CPU does the calculation
- GPU does graphics
  But can be also used to do the calculations
- Different memory model, architecture, calculation power, etc.
- GPU is essentially parallel

Parallel Computing

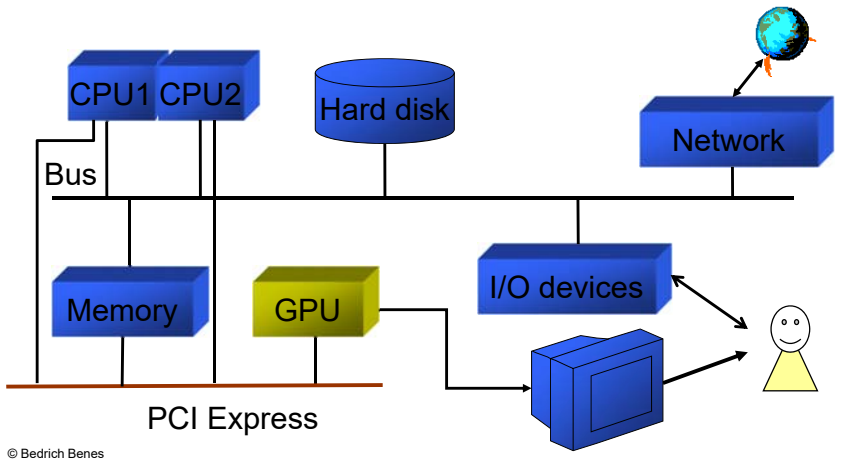
- Many calculations are essentially parallel

Example:

- vector addition on a single thread CPU:

```
for (int i=0;i<n;i++)
  c[i]=a[i]+b[i];
```


Vector Addition - Two CPUs



Parallel Computing

- vector addition two cores:
  `cores=2;`

```
core=0;
for (int i=core;i<n;i+=cores)
    c[i]=a[i]+b[i];
```

```
core=1;
for (int i=core+1;i<n;i+=cores)
    c[i]=a[i]+b[i];
```


Parallel Computing

- *Q:*
  Why is this interleaved vector addition (usually) faster than dividing the array into two equally long sequences?

Terminology

- ***Task:***
  a set of instructions
  executed by a processor

- ***Parallel task:***
  a set of instructions
  executed by multiple processors

Terminology

- ***Execution***
  - serial – execution one instruction at a time
  - parallel – execution by more than one CPUs

Terminology

- ***Shared Memory***
  - memory with simultaneous access
    by more than one device

- ***Distributed Memory***
  - networked systems: CPU sees local
    memory, the rest is communications

Terminology

- ***Symmetric Multi-Processor (SMP)***
  - more processor share the same address
    memory (SM)

- ***Granularity***
  - measure of computation : communication
  - coarse – large computation, small comm.
  - fine – small computation, large communication

Terminology

- ***Parallel overhead***
  - the extra work needed to assure parallelism
- ***Massively parallel***
  - many CPUs
- ***Embarrassingly parallel***
  - doing many similar tasks simultaneously with little overhead (vector addition)

Terminology

- ***Scalability (hw or algorithm)***
  - proportional increase of speed with an increase of resources
- ***Multi-core processor***
  - more CPUs in one chip
- ***Cluster Computing***
  - commodity computers used to do a supercomputer (TeraGrid)

Flynn's Taxonomy

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

Michael J. Flynn

Flynn's Taxonomy

- SISD
  is a single unit machine working on a single data stream
- SIMD
  also *data level parallelism* is a single instruction being executed over a large data sets (vector data)

Flynn's Taxonomy

- MISD
  multiple instructions executed over the same data – theoretically *pipeline* can be viewed as this, however, the data changes once executed

- MIMD
  multiple units working over multiple data (e.g., multi-core CPU)

Memory Architecture

- ***Shared Memory***
  - Uniform Memory Access
  - Non-Uniform Memory Access

- ***Distributed Memory***

- ***Hybrid Shared/Distributed***

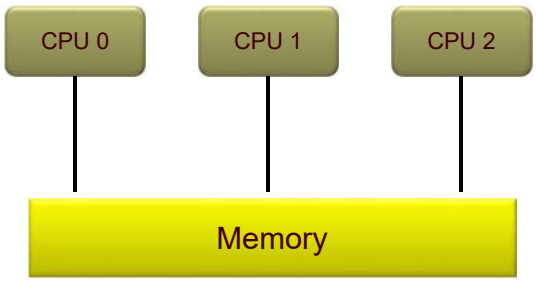
Memory Architecture

- ***Shared Memory***
  - Uniform Memory Access
  - Non-Uniform Memory Access

- ***Distributed Memory***

- ***Hybrid Shared/Distributed***

Shared Memory



Shared Memory UMA/NUMA

- ***UMA (uniform memory access)***
  - Identical Processors
  - The same speed of access r/w
- ***NUMA (non-uniform memory access)***
  - Different processors
  - Variable speed of access r/w
  - (e.g., network of computers with disk access)

cc-NUMA and cc-UMA

- cc – stands for cache coherent
- CPUs have local cache that are synchronized
- One CPU changes shared data all CPUs know about it

Shared Memory UMA/NUMA

- Advantages:
  - Shared memory – nice to program
  - Fast access
- Disadvantages
  - Does not scale
  - Write collisions

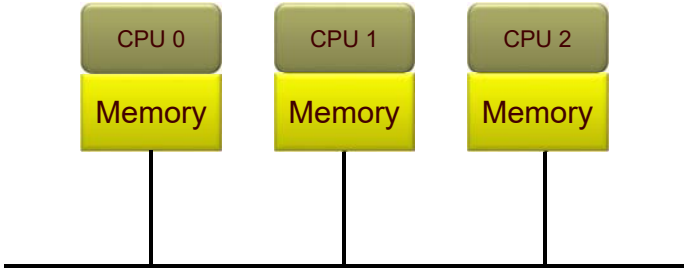
Memory Architecture

- ***Shared Memory***
  - Uniform Memory Access
  - Non-Uniform Memory Access
- ***Distributed Memory***
- ***Hybrid Shared/Distributed***

Memory Architecture

- ***Shared Memory***
  - Uniform Memory Access
  - Non-Uniform Memory Access
- ***Distributed Memory***
- ***Hybrid Shared/Distributed***

Distributed Memory



Distributed Memory

- Each CPU has a local memory
- CPUs operate independently
- Data passing (programer's task)

Distributed Memory

- Advantages
  - Scales well
  - Fast local calculations
  - Cost effective (computer network)
- Disadvantages
  - Difficult to program
  - Difficult to map data (can be slow)
  - Difficult when a lot of communication is necessary

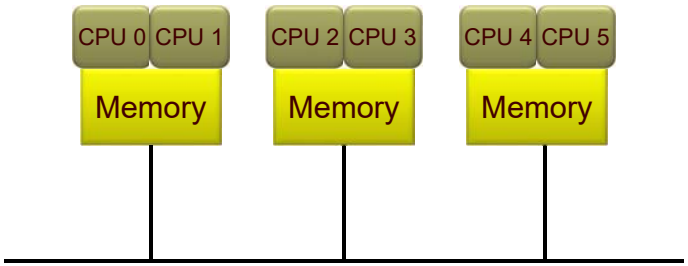
Memory Architecture

- ***Shared Memory***
  - Uniform Memory Access
  - Non-Uniform Memory Access
- ***Distributed Memory***
- ***Hybrid Shared/Distributed***

Memory Architecture

- ***Shared Memory***
  - Uniform Memory Access
  - Non-Uniform Memory Access
- ***Distributed Memory***
- ***Hybrid Shared/Distributed***

Hybrid Memory Architecture



Hybrid Memory Architecture

- The most common…
- e.g., a network of multi-core computers
- advantages and disadvantages depend on the architecture itself

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

© Bedrich Benes

Parallel Progr. Models - SM

- More processes (CPUs) share the same address space
- Problem with asynchronous write
- Locks, semaphores, critical sections…
- No or little communication between tasks is usually required

© Bedrich Benes

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

© Bedrich Benes

PURDUE UNIVERSITY

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

PURDUE UNIVERSITY

Parallel Progr. Models - threads

- A process has multiple paths (threads)
  it runs multiple times
  controlled by the input data
- Threads communicate via global memory, message passing, etc.
- There must be synchronization (rendez-vous, fork, joint)
- POSIX, OpenMP (open multi processing), Microsoft

PURDUE UNIVERSITY

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

PURDUE UNIVERSITY

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

Parallel Progr. Models - MP

- Multiple tasks with local memory
- Sends messages to each others
- Transputers, hybrid systems, cluster computing
- MPI (message passing interface)

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

Parallel Programming Models

- ***Shared Memory (SM)***
- ***Threads***
- ***Message Passing (MP)***
- ***Data Parallel***
- ***Hybrid***

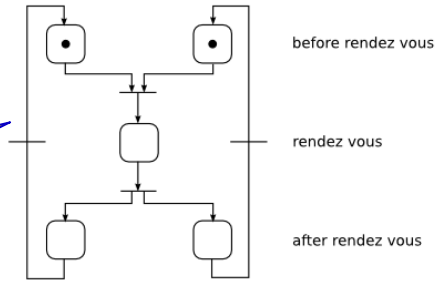
Par. Progr. Mod. –data parallel

- Most of the calculations are dictated by the data structure (array, cube, matrix)
- Tasks perform the same on each portion of data
- Shared memory - given access
- Distributed memory - divide and conquer

Synchronization

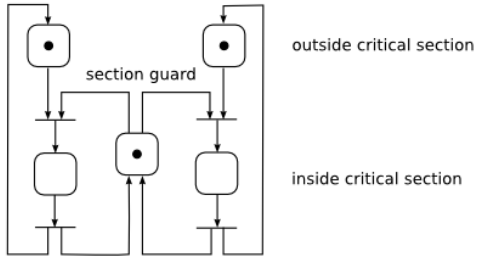
- ***Barrier***
  - all processes involved must stop until all reach this point
  - All enter at the same time
  - rendez-vous

Petri Nets are useful for depicting parallel processes and their synchronization



Synchronization

- *Critical Section, Lock*, (mutual exclusion)
  - A process that enters locks the entrance and enables it when finished
  - e.g., printer



Synchronization

- ***Semaphore***
  - A lock that allows *n* processes enter the critical section
- ***Mutex*** (mutual exclusion)
  - A semaphore with capacity one (binary)

Load Balancing

- Means distributing work to keep everything as busy as possible
- ***Static load balancing***
  - Determined beforehand
- ***Dynamic load balancing***
  - Work is redistributed as necessary

Amhdal's law

- potential program speedup is depends on the fraction of code *P* that can be parallelized

$$speedup = \frac{1}{1-P}$$

- P=0 speedup=1
  P=0.5 speedup=2
  etc.

Amhdal's law

- Having:
  - *n* the # of CPUs
  - *p* the parallel fraction as above
  - *s* serial fraction of the program

$$speedup = \frac{1}{\frac{P}{n} + s}$$

Amhdal's law

- for P=0.5
  speedup goes to 2 with increasing *n*

- for P=0.9
  speedup goes to 10 with increasing *n*

Reading

- Blaise Barney:
  Lawrence Livermore National Lab

https://computing.llnl.gov/tutorials/parallel_comp/#Neumann