Lecture 4

Contents of Lecture 4

- Four steps in parallelizing a sequential algorithm for Java/C/C++ etc
- Multicore architectures

The three main issues

- Ignore for the moment that memories are slow and caches are useful.
- Now the two main issues are:
  - Program correctness, and
  - Load-imbalance: some processors might have less work to do and must wait for others before proceeding.
- Unfortunately, memories can be hundreds of times slower than e.g. adding two integer numbers in a CPU
  - Problem 1: communication usually creates new cache misses, and
  - Problem 2: caches in multiprocessors can introduce obscure bugs.
- All multiprocessors have some form of caches: optimizing performance usually means exploiting them better and this is the third main issue.

Akka

- Correctness: use good design without races between messages: nice
- Load-balancing: automatic by the Akka system in that a JVM thread can work for any actor when it has got a message
- There is a risk with Akka that data is copied between different cache memories which takes time
- And more overhead to send and receive messages than just writing and reading variables

Programming on a multiprocessor

- Global variables and data allocated with new or malloc are shared
- Data allocated on the stack is normally private to a thread (but not enforced by C/C++)
- The most convenient way to parallelize a program is to use a good parallelizing compiler
- The second most convenient way is to do it ***incrementally***, one loop at a time.
- OpenMP is a standard for this for C/C++/Fortran:

```
#pragma omp parallel
for (i = 0; i < n; i += 1)
        a[i] = b[i] * c[i];
```

- If it does not work for our program, we parallelize by hand.

An example: simulating ocean currents 1(2)

- For climate modelling of the earth, we can study interactions between oceans and the atmosphere.
- Our example program simulates the motion of water currents in an ocean.
- It was one of the main benchmarks in the 1990's for research in multicores
- Predicting the state of the ocean at any time requires simulating:
    - atmospheric effects,
    - wind,
    - friction with the ocean floor and walls.
- An ocean is represented as a set of cross-sections, like chess boards put on each other.

An example: simulating ocean currents 2(2)

- The Atlantic is about 2,000 km $\times$ 2,000 km.
- Using 100 $\times$100 points in a cross-section gives 20 km between each point.
- We actually want to simulate at a much finer granularity.
- Time is represented by steps at which all variables are re-calculated.
- Simulating e.g. five years with 8 hours steps requires 5,500 steps.
- The computations are enourmous but can be parallelized.

Terminology

- A **task** is a unit of work that should be performed, and which can be performed in parallel with other tasks.
- A **thread** is a software entity which runs on a processor and works on tasks, (one at a time)
- A **processor** (or core) is the real hardware which executes one thread.
- With simultaneous multithreading (explained in more detail later) multiple threads can share a core (up to 8 in POWER8)

Parallelization of a sequential algorithm

Four main steps in performance programming for a multiprocessor

1. Decomposition — dividing the work into parallel tasks.
2. Assignment — deciding which thread should do which tasks.
3. Orchestration — communication and synchronisation among the threads.
4. Mapping — deciding which threads should run on which processors.

- Decomposition and assignment are called **partitioning**.

Decomposition

- The number of tasks available at any time limits the amount of concurrency.
- We might want our program to have as many tasks as possible: for Ocean we can have
  - one task per set of cross-sections,
  - one task per cross-section,
  - one task for a subset of a cross-section, or
  - one task per grid point.
- What is best cannot be stated without knowing more details!
- ***We want our program to have as many tasks as can be efficiently handled.***
- Unfortunately, one partitioning may be best for one computer but not for others!
- The goal of ***performance portability*** is to have a program which is fast on most machines but is complicated due to varying cache sizes, number of CPU's and how the CPU's and memories are interconnected.

An example related to Ocean 1(3)

Assume a program has two phases.

- In phase 1 $n \times n$ operations are performed on grid points in parallel.
- In phase 2 the sum of values of the $n \times n$ points are accumulated.
- With p processors phase 1 can be done in time n^2/p.
- Without considering time for communication and synchronisation, phase 2 can be done such that each processor adds n^2/p values to the global sum.
- What will the speedup be if we consider synchronisation?

An example related to Ocean 2(3)

- Phase 1 can be done in parallel, taking time n^2/p.
- Since a single variable is incremented n^2 times in phase 2, it must be incremented in a critical region to avoid race conditions. This results in time n^2.
- The sequential execution time is $2n^2$.
- The parallelized execution time is $n^2/p + n^2$.
- The speedup will be $\frac{2n^2}{n^2/p+n^2} = \frac{2p}{1+p} < 2$.
- So, we need to find a better decomposition. See next slide.

An example related to Ocean 3(3)

- Let each processor sum up its n^2/p points to a private variable in phase 2.
- Add a phase 3 which adds all theses private variables, taking p time.
- New speedup becomes linear in the number of processors, p.

Assignment of tasks to threads

- Balance computation, I/O, data access, and communication.
- Easiest if it can be done well statically (at compile-time or just when the program starts and has read some specific input parameters), think e.g. of outer loops in matrix multiplication with a fixed amount of work for each processor.
- For other programs, e.g. a parallel chip simulator, it can be difficult to divide the work. A too sophisticated scheme for load balancing can also lead to too much run-time overhead.
- Run-time assignment is called dynamic assignment.
- Partitioning (decomposition and assignment) is an algorithmic step in the parallelization and is mostly independent of the details of the machine.

Orchestration

- It is now we write source code.
- The programming model (message passing vs shared memory) determines how to do this.
- Questions in orchestration include:
  - How to organize data structures to reduce cache misses?
  - How to reduce the cost of communication and synchronisation?
  - How to reduce serialisation of access to shared data?
  - How to schedule tasks to satisfy dependences early?
- Orchestration is essentially equivalent to making an organization efficient (except that computers are machines without feelings...)

Mapping

- Mapping is done in cooperation with the operating system at run-time.
- Mapping specifies which thread should run on which processor.
- Sometimes the programmer can specify on which processor a thread should run, called to **pin** it.
- Pinning can be important for large multiprocessor with a non-trivial topology (topology = how the CPUs are connected together).
- It may very well be useful to have more threads than processors.

Owner computes

- So far we have parallelized a program with respect to computation.
- Sometimes it is better to parallelize it with respect to data (or both).
- The basic rule then is **owner computes** (and modifies) and other threads read the data.
- In Ocean, it is natural to use the owner computes rule.

Multicore architectures

Multicore architectures

- SIMD architectures
- Vector architectures
- Distributed memory architectures
- Shared memory architectures
- Multithreaded architectures
- Dataflow architectures
- Systolic arrays
- Cache-only memory architectures

Why is it important to study computer history?

Why is important to study computer history?

- You will notice that most of the fancy designs have been tried already
- You will understand why some designs survive and others die
- You will not be impressed when somebody presents the new breakthrough which will change everything and make parallel computing easy
- You will understand where today's machines come from which is important to help you to predict which architectures will be around in 10 or 20 years...

Technology in computer generations

1. -1954, vacuum tubes and relay memories
2. 1955-1964, discrete transistors and core memories
3. 1965-1974, integrated circuits, pipelining, cache memories
4. 1975-1990, DRAM memory, vector and multiprocessors
5. 1991-present scalable architectures

Zuse Z3 1(3)

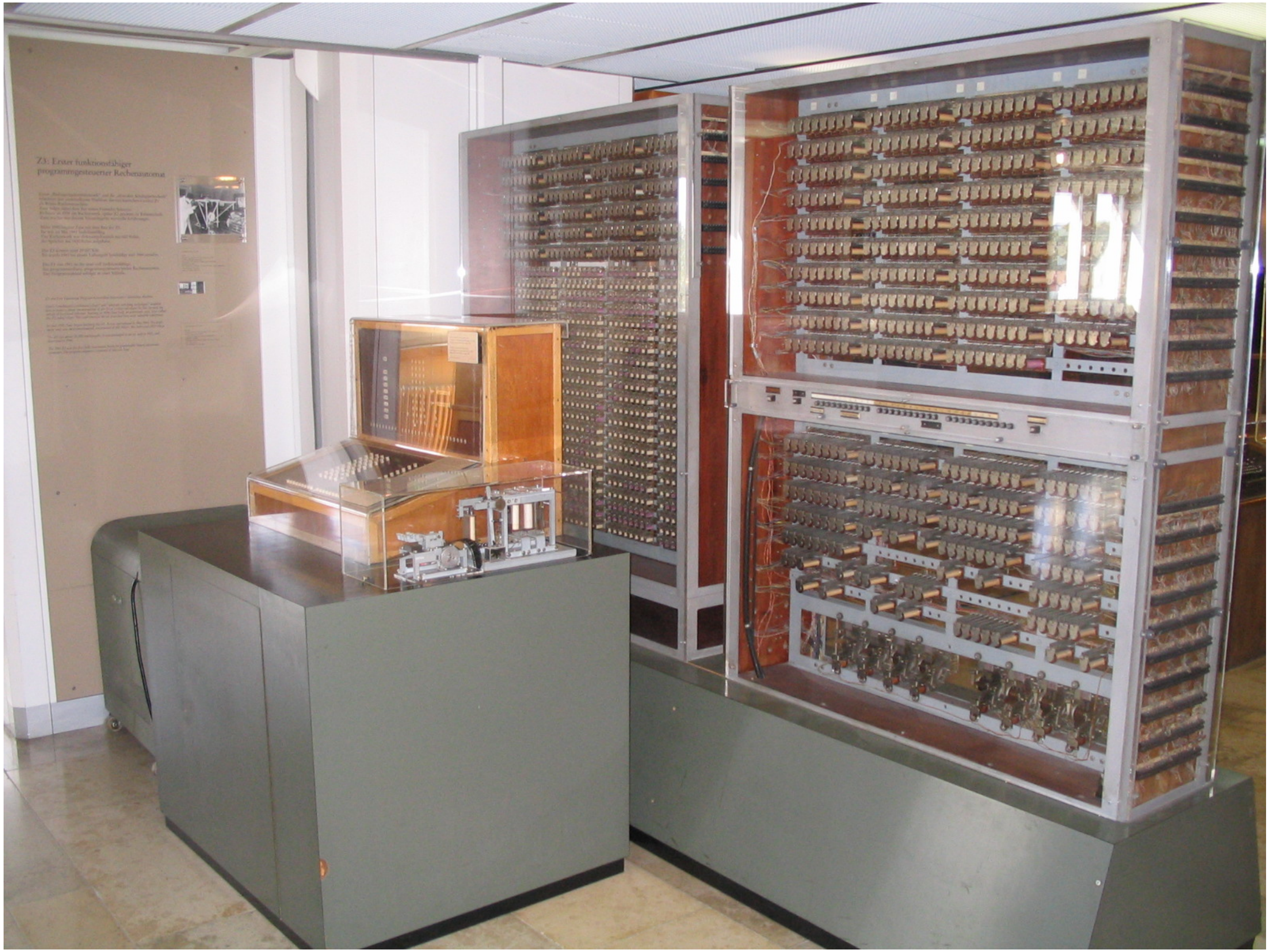
- Konrad Zuse was born in 1910 and became a civil engineer
- He found it boring and started at Henschel to design aircrafts
- He also found the many calculations made by hand boring and started to dream about a machine that could do them for him
- He wanted to automate his calculations and filed a patent for the computer in 1936, and then started to build several machines...Z3:

number representation	binary floating point with NaN and ∞
Turing complete	yes, shown 1998
clock frequency	5.3 Hz
average add delay	0.8 s
average multiplication delay	3 s
memory size	64 words of 22 bits
power consumption	4 kW
weight	1,000 kg
in operation	1941 - 1943 (when it was bombed)

Zuse Z3 2(3)

- After the war, in 1946, Zuse sold patents to IBM to fund his computer company Zuse KG which built a replica of Z3 now at Deutsche Museum in München, where also original machines by Pascal and Leibniz and others can be seen. Very inspiring museum of engineering dreams, successes and failures.
- According to Zuse, IBM didn't understand the coming computer revolution then but wanted his patents for other things.
- His company produced approximately 250 computers (which was a good number as we will see below)
- Siemens bought Zuse KG in 1968.
- When the Computer Science and Engineering program (i.e. D-Linjen), started in 1982, celebrated its 101 anniversay in 1987, Konrad Zuse accepted an invitation by Professor Lars Philipson to give a speech here but unfortunately he could not come. He died 1995.
- So, the inventor of the computer wanted to talk to the engineering students at LTH!

Zuse Z3 3(3)


Z3: Erster funktionsfähiger programmgesteuerter Rechenautomat

ENIAC

- A machine built from 1943 to 1945 at University of Pennsylvania
- Practical use in December 1945
- Used to calculate tables for artillery
- Designers Eckert and Mauchly left to found the EMCC company — see below

The Princeton IAS machine

- The Princeton IAS machine was built from 1942 to 1951 and became fully operational in 1952.
- The chief designer was John von Neumann.
- It used two's complement to represent negative numbers.
- Both instructions and data were stored in memory, so loops could be implemented by modifying the conditional branch instruction...
- An addition took 62 μs
- Many machines were built as derivates from this.
- The founder of the Department of Computer Science, Carl Erik Fröberg, was sent by Vetenskapsakademien 1947-1948 to the U.S. to study the development of electronic computers, and built the SMIL computer in Lund (Sweden's second after BESK).

Commercial computers

- The first commercial computer was in a sense a multicore machine: the BINAC.
- It had two bit serial CPUs each with a 512 word memory, and was built by the Eckert-Mauchly Computer Corporation in 1949.
- It's not regarded as a multicore machine, however. The first multicore was delivered by Burroughs to the US defense only 13 years later, in 1962.
- Compared with the Z3, it was **very much** faster.
- It could compute for over 31 hours without any error at EMCC.
- Then they disassembled and packaged it for delivery to the Northrop Aircraft Company.
- The customer was so concerned about security that no employee from EMCC was allowed to come and assemble it...

BINAC

- Unfortunately it never worked properly after delivery to the Northrop Aircraft Company.
- EMCC and Northrop blamed each other...
- If you buy the world's first commercial computer it might have been a good idea to let the seller assemble it!
- EMCC is now called Unisys and has about 20,000 employees.
- A few months later a mathematician at ETH in Zürich visited Zuse and asked him to program his new Z4 to solve a differential equation which he did on the spot and then ETH bought it. The Z4 was the only commercial computer in Europe during 1950 and 1951.
- Zuse invented the programming language Plankalkül for it.

IBM

- IBM machines were called electronic calculators and were programmed by manually plugging wires.
- Cuthbert Hurd instead wanted to store the program in memory (as in the von Neumann architecture) and convinced the new IBM president to make a commercial machine that also could be programmed using punch cards for that purpose.
- He hired the first team of programmers, including John Backus and Fred Brooks, as well as and John von Neumann as a consultant.
- This started the amazing story of IBM computers.

IBM 701

- The IBM 701 was IBM's first commercial scientific computer.
- Introduced on April 29 1952.
- Memory consisted of 2048 36-bit words.
- Two registers accessible to programmers.
- 19 systems were installed.

IBM 704

- The IBM 704 was introduced 1955.
- Chief architect was Gene Amdahl (who built a computer as PhD thesis in 1952).
- It was the world's first mass produced computer (i.e. more than 100 machines) with floating point hardware.
- It had 5 programmer accessible registers.
- 123 systems were sold until 1960.
- Both LISP and FORTRAN were created on the IBM 704.
- 40,000 instructions per second could be executed.
- Floating point performance was 5 kFLOPS.
- Software was incompatible with IBM 701.

Amdahl

IBM 709 and IBM 7090

- IBM researchers John Cocke and Daniel Slotnick writes a memo discussing **parallel computing**. Slotnick proposed SIMD processing.
- The IBM 709 was introduced 1958 as an improvement over IBM 704.
- Again, software was incompatible with the previous machine.
- An emulator was provided which could run IBM 704 software.
- On the IBM 704, I/O was done from the CPU but the IBM 709 improved I/O by introducing separate processors called I/O channels for this.
- The IBM 7090 was introduced 1959 and was similar in design as the IBM 709 but was implemented with transistors instead of vacuum tubes.

IBM 360 — the first computer architecture

- One of the main annoyances with previous machines was that every new machine required new software.
- The IBM 360 was instead an **architecture**, introduced in 1964.
- IBM wanted to sell ”inexpensive” slow machines and scale the performance and price but they should be **compatible** for software.
- The price was initially set based on the performance.
- Later machines include the IBM 370 series, and the IBM 3090.
- A recent compatible machine, Z14, was announced 2017 and has 10 core processors clocked at 5.2 GHz and can have up to 10 TB of **RAM** memory (for Z13 — I could not find the max memory for Z14).

More about the IBM 360

- Introduced the 8-bit byte.
- It had 32-bit words.
- It was byte addressable.
- 16 general purpose registers.
- 24-bit addresses.
- First TLB was in IBM 360/Model 67 (and GE 645)
- First data cache was in IBM 360/Model 85
- Tomasulo's algorithm for out-of-order execution was first used in IBM 360/Model 91

IBM Stretch

- IBM 7030, or IBM Stretch, was the fastest machine from 1961-1964.
- The design goal was to be 100 times faster than the previous machine!
- It was neither technically nor commercially successful but important technologies were developed for it, including:
    - Instruction pipelining: also available to some extent in both of Z1 and Z3, and Illiac IV.
    - The pipeline stage names **fetch, decode, execute** come from Stretch
    - Memory interleaving

Control Data Corporation

- Control Data Corporation, founded 1958, was a small competitor with IBM during early 1960's.
- One of their engineers, Seymour Cray, set in 1960 out to build the CDC 6600, a machine which should be 50 times faster than their previous, CDC 1604 (a machine similar to the IBM 7090).
- After four years of development, the management was quite worried about what was happening...
- Cray's response was that he wanted his own lab in his home town, near Minneapolis, and nobody was allowed to go there except by invitation — otherwise he would quit.
- The CDC's president accepted his demand and Cray with some engineers moved to the new lab.
- He chose the location so that it was too distant for a one-day visit by car.

Supercomputing

- In 1964 the CDC 6600 was released and Control Data Corporation then dominated the supercomputer industry during the 1960's.
- The CDC 6600 became the fastest computer in the world and started the supercomputing era.
- The CDC 6600 is regarded as the first superscalar computer — execution was controlled by a scoreboard.
- Now the IBM president asked approximately:
  How is it that this tiny company of 34 people including the janitor can be beating us when we have thousands of people?

Cray's reply

You just answered your own question.

The IBM response

- IBM's response was to also set up a remote lab of 200 engineers, to build an even faster machine.
- It was intended to be able to issue seven instructions per clock cycle.
- It turned out to be impossible to achieve the desired performance levels if binary compatibility with the 360 architecture was to be maintained.
- The project was cancelled in 1969.
- It is extremely important to have a good instruction set architecture, because all future CPUs and optimizing compilers will have to live with it.
- This cannot be overstated. Compare X86 and Itanium, and AMD's X86_64

Vector supercomputers

- Daniel Slotnick (who wrote the memo at IBM about parallel computing with John Cocke) had moved to the company Westinghouse where he designed a parallel machine called Solomon.
- Solomon was intended to have one CPU called the control unit and 256 processing units with their own memories
- The control unit specified the instruction that all processing units should perform, i.e. it's a SIMD vector machine.
- The Solomon machine was built with funding from the US Air Force but they withdrew from the project in 1964, and it was cancelled.

ILLIAC IV

- When Solomon was cancelled, Slotnick convinced Burroughs and the University of Illinois to build a similar machine, the ILLIAC IV.
- It was built from 1965 - 1975.
- It was intended to reach 1 GFLOPS using 256 SIMD units clocked at 13 MHz, but a smaller machine could only be built which reached 200 MFLOPS.
- As Solomon, it relied on the application to be suitable for SIMD processing. The ILLIAC IV was the fastest machine from 1975 until 1981 for ***suitable*** applications.
- A lot of research on optimizing compilers was done at the University of Illinois.

Additional machines at CDC and Cray's departure

- The next machine from Control Data Corporation, the CDC 7600, was about five times faster than the CDC 6600
- These machines were very expensive but fastest in the world and good investments for some customers
- Development costs of each of 6600 and 7600 almost bankrupted CDC.
- The next CDC machine, CDC 8600, was too complex and Cray told the president they had to redesign it from scratch... :-(
- In the meantime CDC was working on another, less complex machine.
- The president, Norris, didn't want to prioritize Cray's redesign.
- He left to fund Cray Research in 1972.
- Norris invested USD 300,000 in Cray's startup.
- The new CDC machine was not a success due to it relied too much on fast parallel parts but was slow at sequential parts — Amdahl's Law!

Cray Research

- First a word from a CTO of HP, Joel Birnbaum:
  "It seems impossible to exaggerate the effect he had on the industry; many of the things that high performance computers now do routinely were at the farthest edge of credibility when Seymour envisioned them."
- If it was too expensive to develop the next machine at CDC, what could he do with his startup???
- It turned out that Seymour Cray was well known and respected at Wall Street and they got all funding they needed.

The Cray-1

- In 1975 the Cray-1 was announced with a clock rate of 80 MHz — it was very fast both for scalar and vector parts of an application.
- Vector machines before (and sometimes after) the Cray-1 were keeping the vectors in memory, this allows flexible vector sizes.
- Cray instead used vector registers of fixed sizes.
- Due to the fact that vectors often were used several times this increased performance.
- After the announcement of their machine, there was an auction for who would get the first Cray-1.
- Seymour Cray was not impressed by the ideas behind ILLIAC IV: he said ***"If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?"***
- Many years later, high performance microprocessors changed that, of course.

Cray-1

The first real multicore was released in 1962

- Stepping back to 1962, Burroughs released the Burroughs D825.
- This is the first multicore machine. It was called a multiprocessor.
- It had up to four CPUs and 16 memory modules.
- It was symmetric in that all CPUs could access any memory module.
- It's operating system had a shared ready queue of tasks.

Multics operating system

- AT&T, General Electric and MIT set out to build the Multics operating system in 1965.
- It was intended to run on multiprocessors.
- In 1969 it ran (to some extent) on an 8 CPU multiprocessor built by Honeywell.
- It was a failure but out of it came UNIX and later Mac OS X and Linux.
- The same year Dijkstra formulated the critical regions problem.

Summary so far

- What have we got so far?
- Software is FORTRAN
- Optimizing compilers which can parallelize and vectorize numerical codes are being developed, they are very good at vectorization but less at parallelization (and still are).
- By the end of 1970's and beginning 1980's supercomputers with few CPU plus vector instructions are the fastest machines, either from Japanese companies or Cray.
- A typical multiprocessor was bus-based.

CISC processors

- Before the 1960's hardware was more expensive than software.
- There was a software crisis in the 1960's when it was realized that software was becoming the expensive part.
- Most of the computer industry went in the direction of creating CPUs which they thought would help programming in high-level languages.
- For instance instructions to set up a stack frame and save registers and later restore registers were common, e.g. in the VAX.
- Instructions to copy memory were taken for granted. With virtual memory and page faults such instructions are tough to implement efficiently.
- The VAX even had an instruction to evaluate a polynomial.
- Fancy instructions were ''selling features''

IBM 801

- After finishing his PhD in mathematics, John Cocke worked for IBM from 1956 to 1992.
- In 1979 John Cocke designed a new processor which went completely against the mainstream of more complex instruction sets.
- This was the foundation for the RISC revolution — all your mobile phones use it and some of the fastest computers.
- RISC = reduced instruction set computer (reduced as in simpler, not fewer)
- The commercialization of superscalar RISC was in the POWER architecture.
- POWER = performance optimization with enhanced RISC

POWER, ARM, MIPS, RISC-V

- ARM is similar to POWER and the ARM business model is to sell designs and licenses — not physical chips
- MIPS was developed at Stanford and went open source in yearly 2019
- RISC-V is an open source architecture introduced 2010 and competes with ARM and MIPS
- RISC acronym comes from UC Berkeley
- Since a few years Google, Samsung, IBM, Canonical (Ubuntu) and many others are collaborating on POWER — see `openpowerfoundation.org`.
- In August 2019 IBM made POWER open source in the sence that there are no license fees to use it
- Strike against both ARM and RISC-V (and possibly MIPS)

Many new parallel computing principles

- With increased transistor budgets many new ideas could be realized.
- Today we know which one won the battle: shared memory multiprocessors with coherent caches, i.e. essentially the Burroughs D825 but with added cache memories.
- It's relatively easy to invent new faster parallel machines, if you can ignore the constraint of having happy paying commercial customers to make your company survive (by instead letting the US Department of Defence pay).
- Many niche market machines were developed by companies which spent huge amounts of money before cancelling their projects.
- Typical killing features for **hardware companies** include
    - insufficient performance
    - requiring new programming languages
    - requiring nonstandard extensions to C or FORTRAN
    - taking too long to reach market so your technology becomes obsolete
- New programming languages are of course not bad but need time to have impact which companies often cannot afford.

Examples of parallel architectures

- MPP (massively parallel machines)
- Dataflow machines are developed at MIT by Jack Dennis and Arvind
- Systolic arrays are described by H.T. Kung and Charles Leiserson
- The C.mmp multiprocessor consisting of 16 PDP-11 connected by a crossbar to a shared memory was built at CMU.
- The multithreaded machine Denelcor HEP developed by Burton Smith
- Stanford DASH
- KSR-1 and DDM

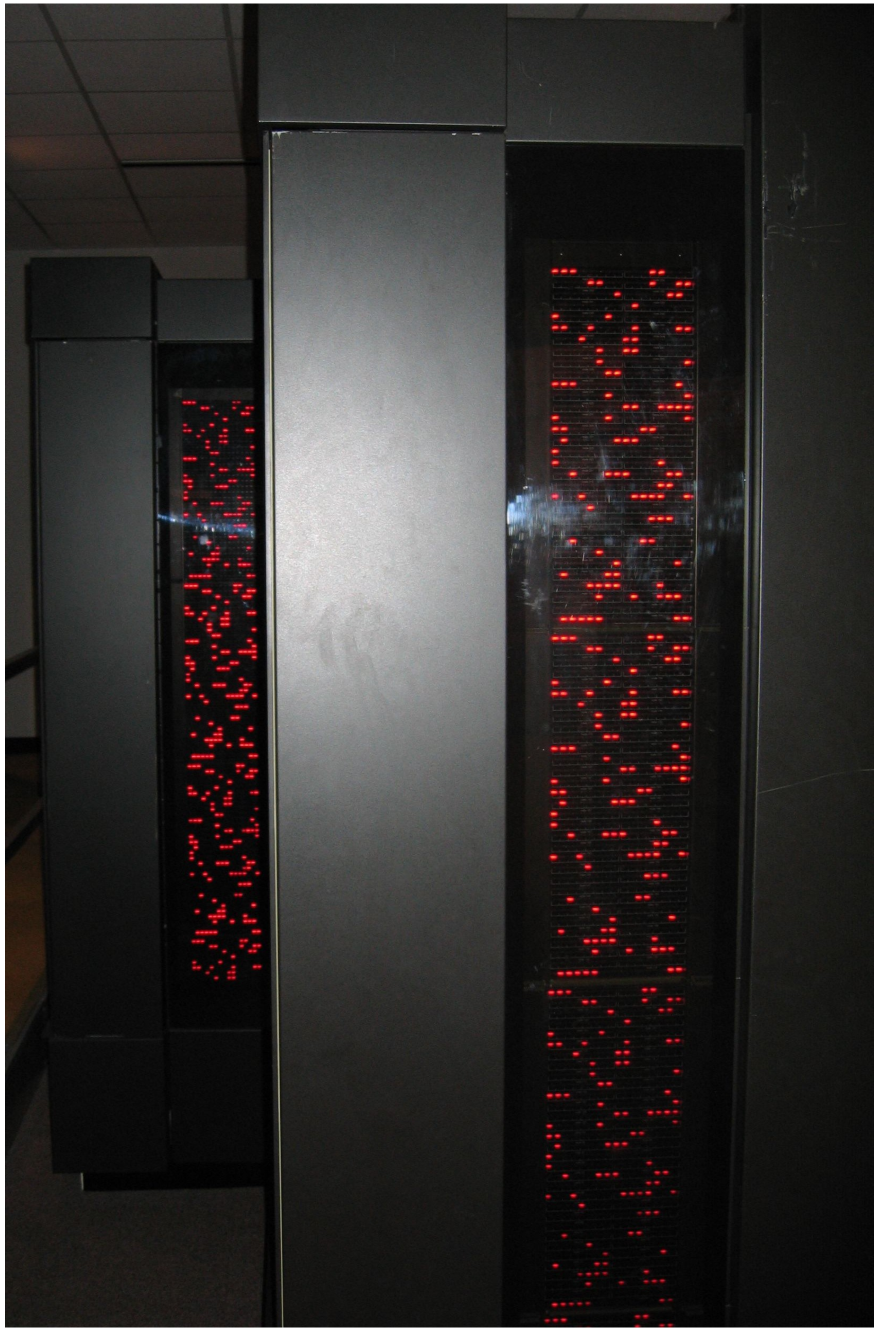
MPP machines

- By Massively Parallel Machines are meant distributed memory machines where each node has a CPU and a private memory.
- The number of such nodes are typically hundreds or thousands.
- To communicate, programmers (or optimizing compilers) must create messages to send to some other node.
- MPI (message passing interface) is an industry standard for high performance computing and is actively used today as well.

Thinking Machines, Inc

- It was founded 1983 and they made some of the coolest machines in the 1980's.
- Famous persons like Guy Steele worked for them (who coauthered the Java Language Specification among **many** other things).
- The first **connection machine**, the CM-1, consisted of up to 65,536 one-bit processors, each with 4 KB RAM.
- To improve performance, the CM-2, used normal floating point processors (Weitek), and the last model, the CM-5, used normal SPARC processors from Sun.
- Sun later bought the company.
- They were programmed in FORTRAN, as well as a parallel version of Lisp, called *Lisp (star Lisp).

The CM-5



Dataflow machines

- Dataflow machines have a completely different hardware architecture than anything you are used to.
- Instead of a program counter which addresses the next instruction to execute, a dataflow program is a dataflow graph.
- The dataflow graph is worked on in parallel. An operation is called a token and as soon as an operation's operands have been computed that operation can be computed by a processor, and the result then stored back to a memory in a special way (see below).
- The memory is called a **token store**.
- In a normal computer the data is stored in memory and an instruction accesses that memory location.
- In a dataflow machine, the data is stored in the instruction!

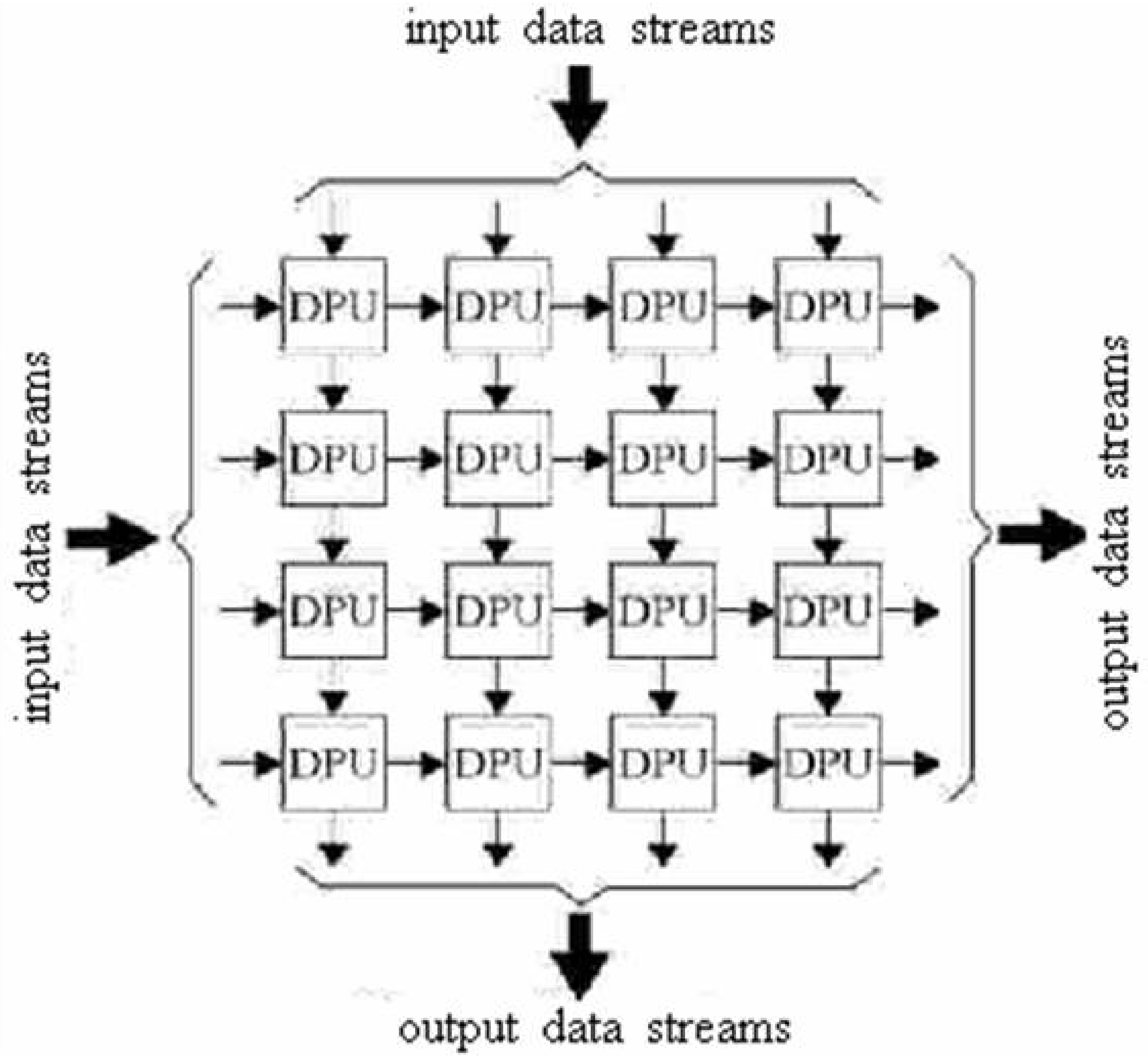
More about dataflow architectures

- When an operation has been computed, and the result should be stored in memory, the result is stored in each instruction which needs the data as an operand.
- Instead of shared memory and a program counter, a token matching mechanism is needed to propagate data to the instructions (tokens).
- The main architecture and machine designer at MIT has been Arvind, who built the Monsoon Dataflow machine together with Motorola, but it was never a commercial machine.

Systolic arrays

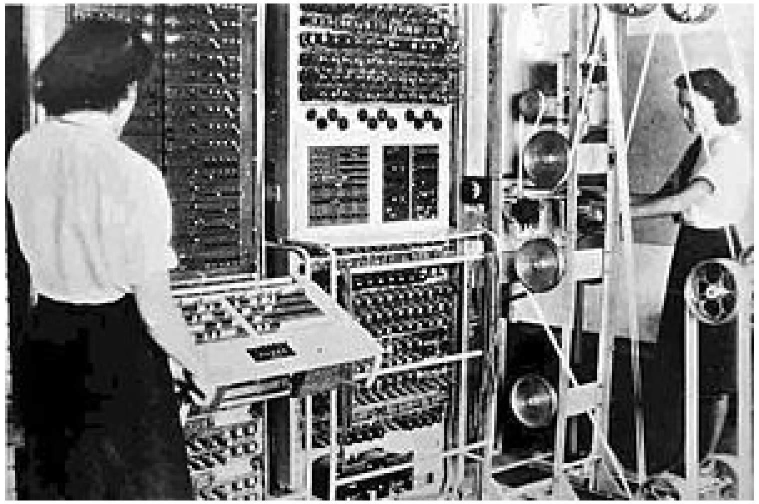
- The name systolic arrays was coined by H.T. Kung and Charles Leiserson (one of the authors of Introduction to Algorithms) who wrote the first paper on this architecture.
- By systolic is meant computing at a certain pace, like heart beats.
- The idea is to put processors in an array and let a processor get data from its left neighbour and produce data to its right neighbour.
- Alternatively input comes from two sources and are written to two destinations.
- The key idea then is to have very efficient communication since the data is just copied from one processor to the next.
- The potential performance is huge.
- The main problem is to map algorithms to this architecture.

Systolic array idea



A somewhat earlier systolic array from 1944

- The British Collosus code breaking machines also used the idea of systolic arrays. 10 such machines were used until the end of WWII.



The iWarp machine

- CMU and Intel set out to explore the possibilites of building a parallel supercomputer based on the CMU Warp systolic array machine.
- An iWarp machine had 64 compute cells each of which was a 32-bit processor.
- They were produced around 1990 and in 1992 Intel created a supercomputing division (now no longer existing).
- One result from the systolic array project at CMU is the software pipelining algorithm called modulo scheduling which is now widely used in optimizing compilers also for normal CPUs.

Multithreaded architectures

- To avoid the problem of long latency operations, instead of waiting an alternative is to switch thread to execute.
- Obviously this must be done in hardware.
- There are many different ways to design such multithreaded machines.
- Some switch threads every clock cycle, others when there is a long latency operation such as a cache miss.
- After designing the HEP machine for Danelcor, Burton Smith designed the Tera machine in his own company.
- The Tera parallel machine had no caches but some interesting ideas, e.g. with each memory word was a full or empty bit that could be used for synchronization. Writing a word set the bit and reading a word cleared it.
- They had very few commercial customers.

Cache coherent shared memory multiprocessors

- From the few samples we have seen, there are many ways to build parallel computers.
- In the 1980's multiprocessors with cache memories were becoming commercial products.
- The main question was how to make them scalable to a large number of CPUs.
- A single bus is of course out of the question.
- A mesh is one acceptable topology and was used in the Stanford DASH machine.
- We will look at the details in a later lecture, but consider a multiprocessor with caches where a memory word may be copied to any number of caches.
- Sometimes we need to tell every node with a copy to forget about its copy — and fetch a new value from memory next time.
- How should this information be organized?

A directory

- A so called directory at the memory in a node keeps track of which nodes have a copy.
- Note that a directory can become large if there are many nodes and we allow any number of copies.
- An alternative is to have a fixed maximum number of copies stored in the directory and move the directory information for a certain memory block to the operating system kernel.
- So each memory address has a **home location** i.e. a node in whose memory it is located.
- The home location, or node, is typically static.

COMA

- Cache-Only Memory Architectures, COMA.
- Consider two threads in different nodes accessing the same data in a third node.
- If both threads read and write the data, the home node of the data will be involved.
- The latency is increased by having to go to that home node for telling the other thread (i.e. cache) to write it's modified copy back to memory.

Two machines: KSR-1 and DDM

- Researchers at SICS (Swedish Institute of Computer Science) as well as a company called Kendall Square Research (close to MIT) invented a different design.
- Instead of having a static home location, data can ”diffuse” to a suitable node that is currently using it and let that be the new home. That way only two nodes need to be involved in the above scenario.
- The Swedish machine was called the Data Diffusion Machine.
- For some applications this is useful while for others it is not very important.

Summary: the most important lessons from the past

- With the difficulty of increasing the clock rate chip companies want us to use multicores everywhere
- Programmability and mass market are essential
- Amdahl's Law is extremely important
- Cache coherent shared memory multiprocessors are here now and we must write general purpose applications in Java and C/C++ for them.
- It is essential to understand that all the fancy ideas have been around many decades and we should be sceptical when somebody tells us they have the ultimate solution for faster parallel machines...
- GPU's are SIMD with multiple threads and work very well for some application types