Parallel Computer Architecture I

Stefan Lang

Interdisciplinary Center for Scientific Computing (IWR)
University of Heidelberg
INF 368, Room 532
D-69120 Heidelberg
phone: 06221/54-8264
email: Stefan.Lang@iwr.uni-heidelberg.de

WS 14/15

Parallel Computer Architecture I

- Why parallel computing?
- Von-Neumann architecture
- Pipelining
- Cache
- RISC und CISC
- Scalable computer architectures
- UMA, NUMA
- Protocols for cache coherency
- Examples

Definition of Parallel Machine

What is a parallel machine?

A collection of processing elements that communicate and cooperate to solve large problems fast
(Almasi und Gottlieb 1989)

What is a parallel architecture?

It extends the usual concepts of a computer architecture with a communication architecture

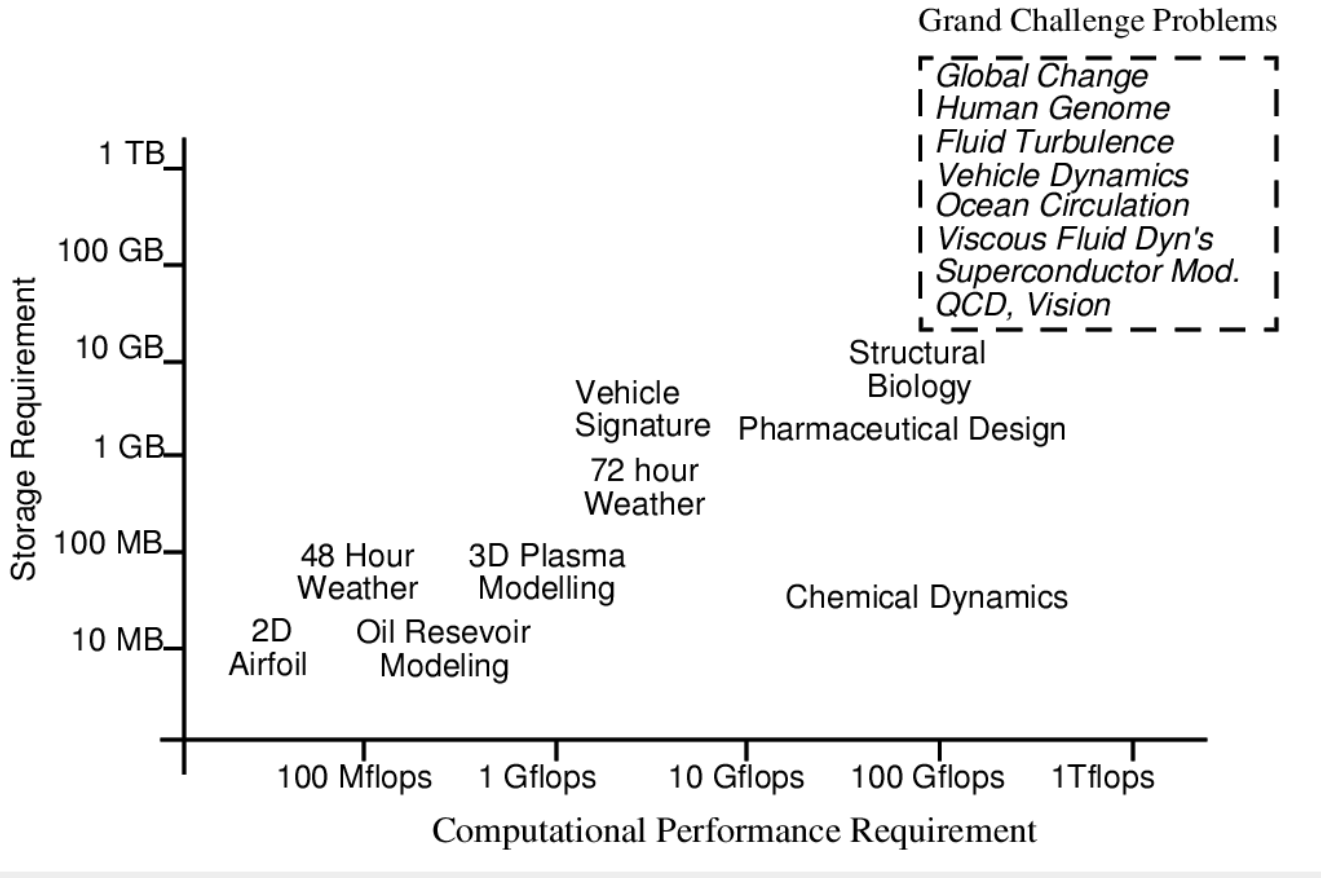
Why Parallel Computing?

3 flavours of parallel computing

- Solve a problem of fixed size fast
  Goal: Minimize time-to-solution and speedup r&d cycle
- Compute very large problems
  Goal: exact result, complex systems
- Simulate very large problems fast (respec. in adequate time)
  Goal: Grand Challenges

Single processor performance is not sufficient
$\rightarrow$ Parallel Architectures

What are Problems?

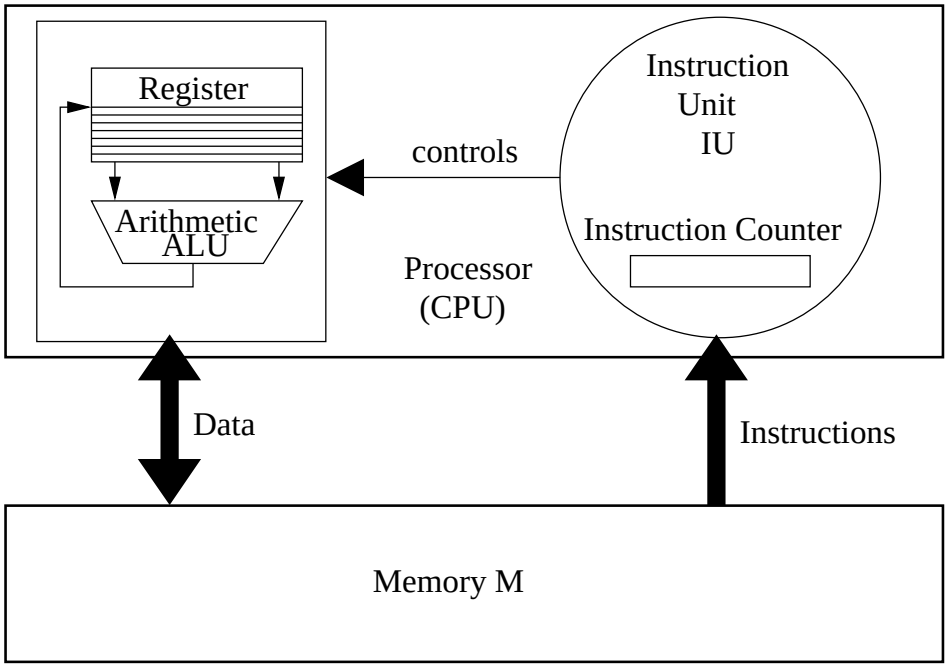


from Culler, Singh, Gupta: Parallel Computer Architecture

- Classification of problems according to memory and computing demands
- Categorisation in 3 types: memory limited, compute-time limited and balanced problems

Von Neumann Architecture

Schematical structure with instruction unit, arithmetic unit and memory



Instruction cycle:

- fetch instruction
- decode instruction
- execute instruction
- store results

- Memory contains program code and data
- Data transfer between processor and memory uses system bus
- Several devices (processors, I/O-Units, Memory) on bus

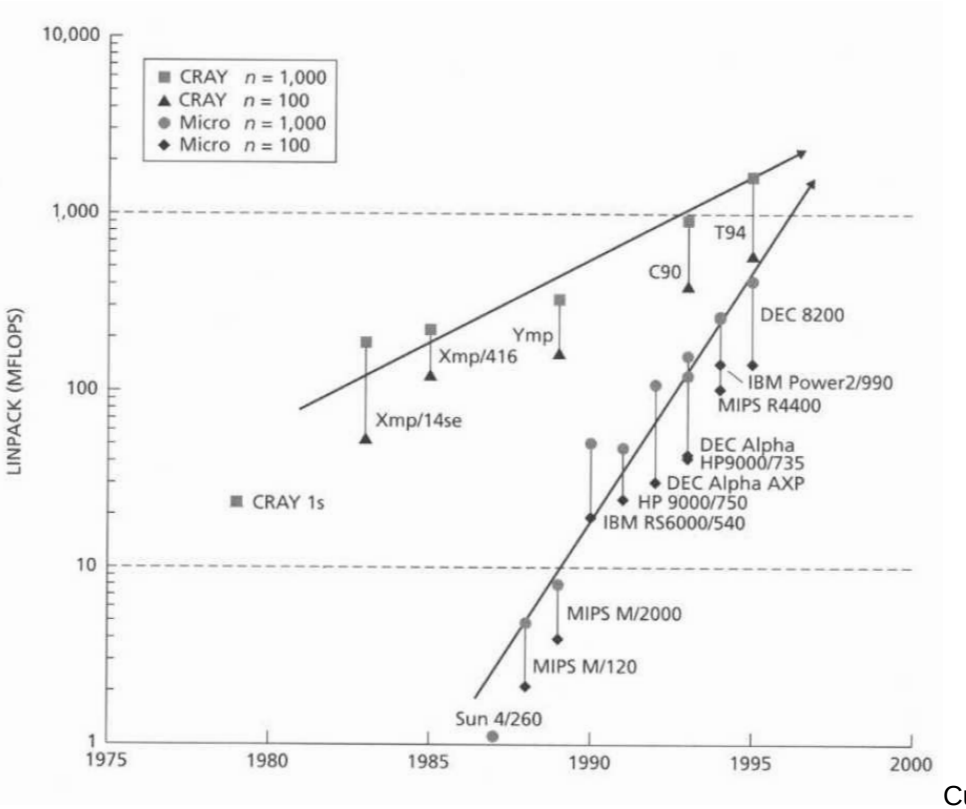
Generations of Electronic Computers

Distinction of 5 + 2 computer generations

Generation	Technology and Architecture	Software and Applications	Representative Systems
First (1945-54)	Vacuum tubes and relay memories, CPU driven by PC and accumulator	Machine/assembly languages, single user, no subroutinge programmed I/O using CPU	ENIAC, Princeton IAS, IBM 701
Second (1955-64)	Discrete transistors and core memories, floating-point arithmetic	HLL used with compilers, subroutine libraries, batch processing monitor	IBM 7090, CDC 1604, Univac LARC
Third (1965-74)	Integrated circuits, micro-programming, pipelining cache, lookahead processors	Multiprogramming and time-sharing OS, multiuser applications	IBM 360/370, CDC 6600, TI-ASC, PDP-8
Fourth (1975-90)	LSI/VLSI, semiconductor memory, multiprocessors, vector- and multicomputers	Multiprocessor OS, languages, compilers, enviroments for parallel processing	VAX 9000, Cray X-MP, IBM 3090
Fifth (1991-1997)	ULSI/VHSIC processors, mems and switches, high-density packaging, scalable archs	Massively parallel processing grand challenge applications heterogeneous processing	Fujitsu VPP-500, Cray/MPP, Intel Paragon
Sixth (1997-2003)	commodity-component cluster high speed interconnects	Standardized Parallel Environ-ments and Tools, Metacomputing	Intel ASCI-Red, IBM SP2, SGI Origin
Seventh (2004-present)	Multicore, Powersaving Extending memory hierarchy	Software for Failure Tolerance, Scalable I/O, Grid Computing,	IBM Blue Gene, Cray XT3

nach Hwang (with additions)

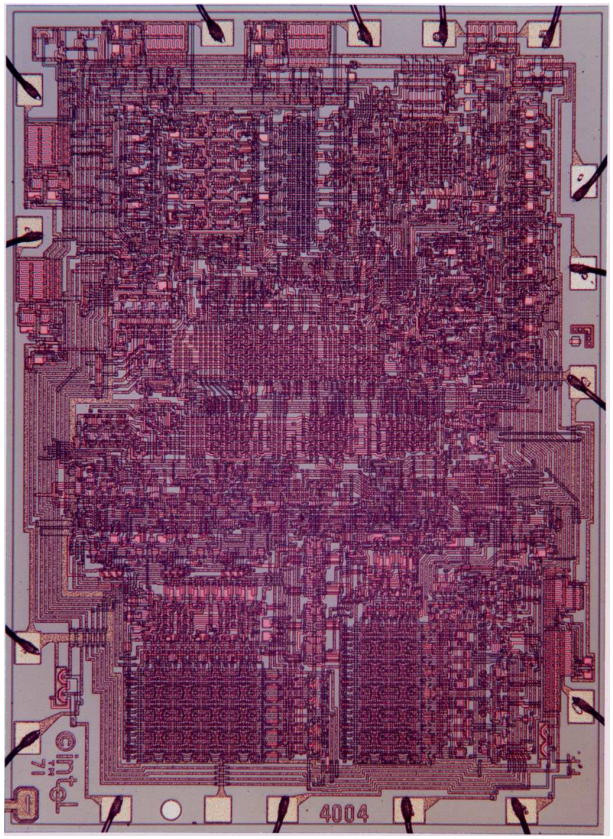
Single-core Processor Performance



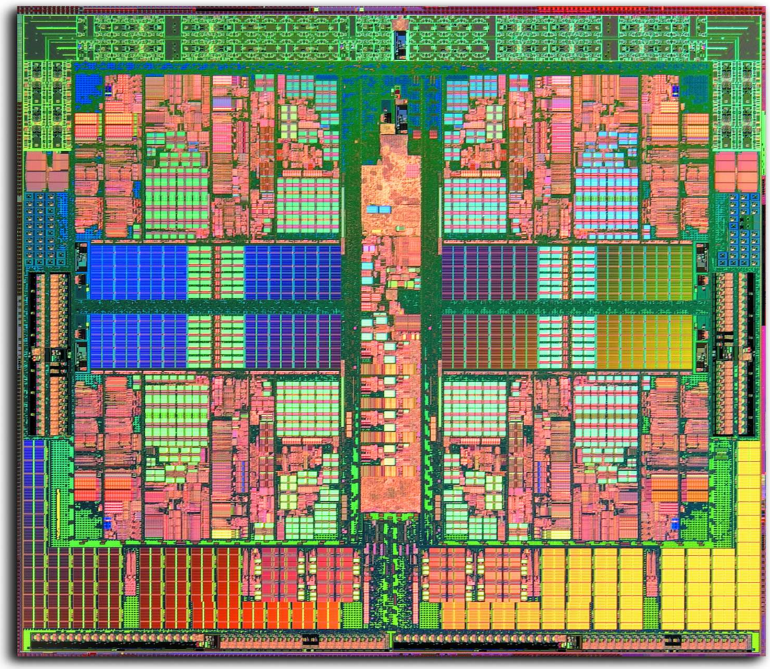
Culler, Singh, Gupta: Parallel Computer Architecture

- Performance development of vector- and superscalar processors
- Earlier: many manufacturers, now: some market leaders
- Speed advantage of vector processors shrinks

Single-core Processors: Two Examples



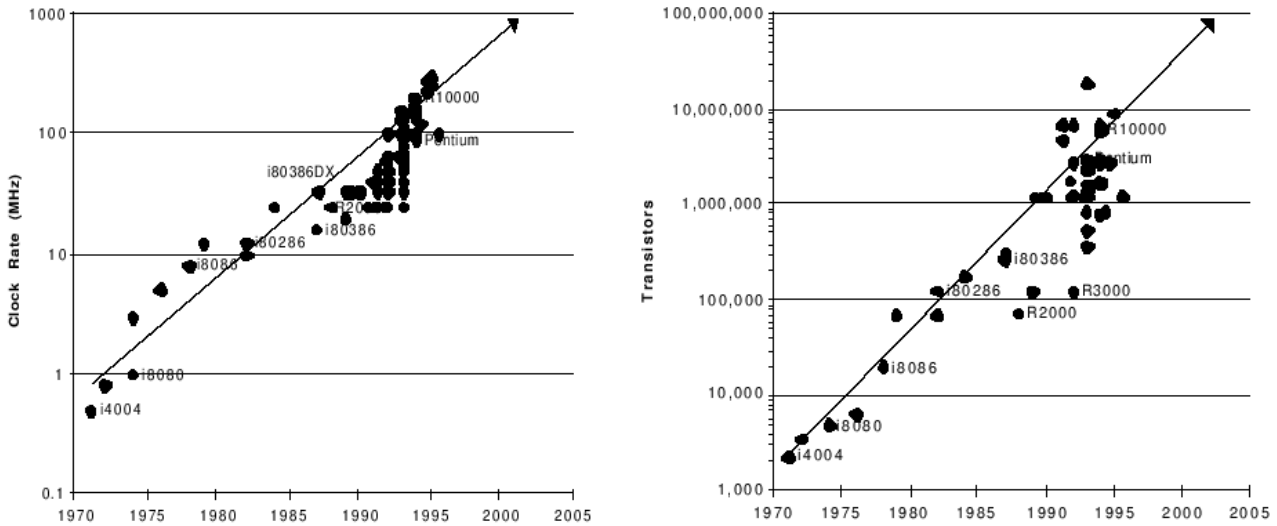
1971: Intel 4004, 2700 Trans., 4 bit, 100 KHz



2007: AMD Quadcore, 465 mill. trans., 64 bit, 2 GHz

Intel founder Andy Grove, Robert Noyce, Gordon Moore in 1978

Integration Density and Clock Frequency



Culler, Singh, Gupta: Parallel Computer Architecture

- Increase according to Moore's law: Doubling within 18 months
- Moore's law is NOT related to performance but to integration density
- Divergence of speed and capacity in storage technologies

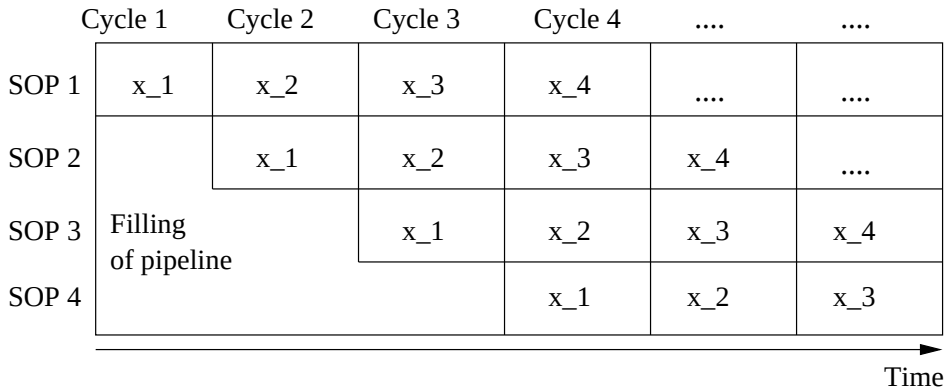
Architecture of Single-core Processors

Techniques to increase single-core processor performance

- deep pipelining
- speculative branch prediction
- out-of-order execution
- clock frequency scaling
- superscalar design (instruction level parallelism ILP)
- speculative execution
- thread-level parallelism
- multi-core design

Pipelining I: Principle

Synchronous, overlapping processing of operations
Pipeline with 4 stages:



Requirements:

- An operation $OP(x)$ has to be applied onto many operands $x_1, x_2, \ldots$ in sequence.
- The operation can be divided into $m > 1$ sub-operations (or also stages), that can be executed in (preferably) equal time.
- An operand x_i may be with restrictions only a result of former operations.

Gain with pipelining: The time demand for processing of N operands is

$$T_P(N) = (m + N - 1)\frac{T_{OP}}{m}$$

Pipelining II: Speedup

The Speedup is therefore

$$S(N) = \frac{T_S(N)}{T_P(N)} = \frac{N * T_{OP}}{(m + N - 1)\frac{T_{OP}}{m}} = m\frac{N}{m + N - 1}$$

For $N \to \infty$ the speedup converges towards m.

Utilization inside processors:

- Instruction pipelining: fetch, decode, execute, write back
- Arithmetic pipelining: adapt exponents, add mantissa, norm mantissa

Further applications:

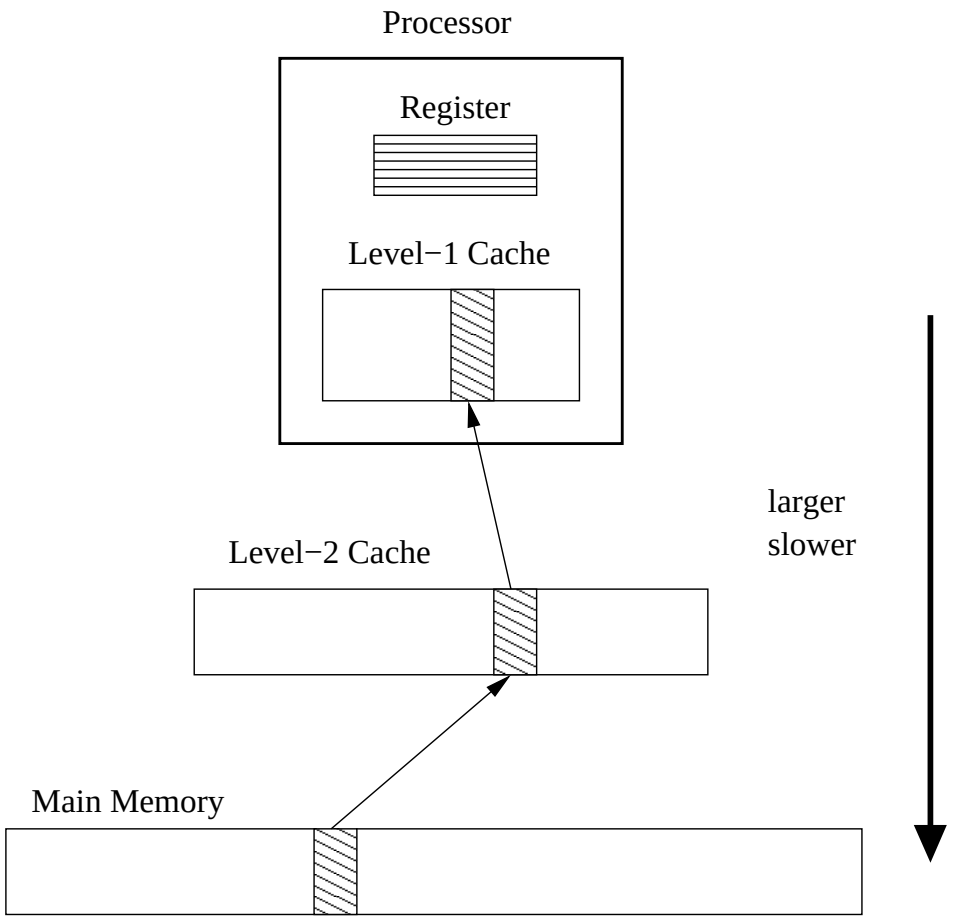
- Memory interleaving
- Cut-through routing
- Wavefront algorithms: LU-decomposition, Gauß–Seidel
- ...

Cache I: Memory Hierarchy

Speed gap:

- Processors are fast: 2-3 GHz clock, $\geq$ 1 instruction/cycle due to pipelining
- Memory is slow: MHz clock, 7 cycles to read 4 words

Way out: Hierarchy of always slower but larger memories



Cache II: Cache Organisation

Memory contains respectively least recently used data of the next higher hierarchy level

Transfer is managed in blocks (Cache Lines), typical size: 16...128 bytes

Cache organisation:

- Direct mapping: main-memory block i can only be positioned in place $j = i \mod M$ inside the cache (M: size of cache).
  Advantage: easy identification, Disadvantage: aliasing.
- Assoziative cache: main-memory block i can be positioned at each location inside the cache.
  Advantage: no aliasing, Disadvantage: costly identification (M comparisons).
- Combination: k-way assoziative cache.

Replacement: LRU (least recently used), random

Storage: write through, write back

Cache III: Locality Principle

Up to now we have assumed, that all memory words can be accessed equally fast.

But with cache least recently fetched data can be accessed faster. This has implications on the implementation of algorithms.

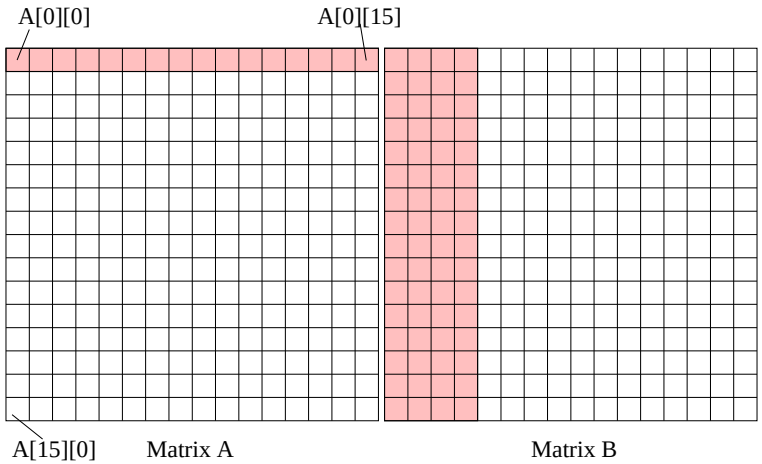
Example: Multiplication of two $n \times n$-matrices $C = AB$

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

Assumption: Cache-line is 32 bytes = 4 floating point numbers.

Cache III: Locality Principle

After calculation of $C[0][0]$ there are the following words stored inside the cache:



A, B, C completely in cache: $2n^3$ arithmetic operations but only $3n^2$ memory accesses

If fewer than $5n$ numbers fit into the cache: slow

Tiling: Process matrix in $m \times m$ blocks with size $3m^2 \leq M$

```
for (i = 0; i < n; i+=m)
    for (j = 0; j < n; j+=m)
        for (k = 0; k < n; k+=m)
            for (s = 0; s < m; s++)
                for (t = 0; t < m; t++)
                    for (u = 0; u < m; u++)
```

RISC und CISC

RISC =„reduced instruction set computer“
CISC=„complex instruction set computer“

Development of processors with increasingly complex instruction sets (i.e. adressing methods): Costly decoding, instructions with variable length

Begin of 1980s : „Back to the roots“. Simple instructions, aggressive usage of pipelining.

The idea was not new: Seymor Cray has always build RISC machines (CDC 6600, Cray 1).

Design principle of RISC machines:

- All instructions are coded in hardware, no micro programming.
- Aggressive usage of instruction pipelining (parallelism on instruction level ILP).
- Preferably execute one instruction/cycle (or more for superscalar machines). This requires a preferably simple and homogeneous instruction set.
- Memory accesses only with special load/store–instructions, no complicated addressing methods.
- Provide many general purpose register to minimize memory access. The saved chip area in the instruction unit is used for registers or caches.
- Follow the design principle „Make the frequently occuring case fast“.

Today predominantly RISC processors. Intel Pentium is CISC with RISC-core.

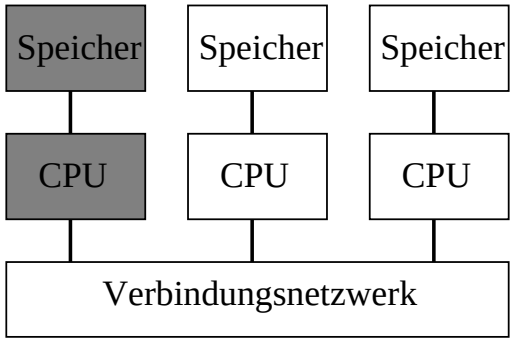
Scalable Computer Architecture I

Classification of parallel machines according to FLYNN (1972)

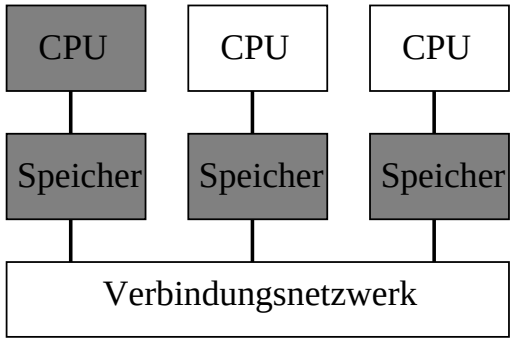
Distinction with regard to data streams and control pathes

- *SISD – single instruction single data*: The Von Neumann Computer
- *SIMD – single instruction multiple data*: The machines, also called array processor, possess an instruction set and multiple independent arithmetic units each is connected to its own memory. The arithmetic units are controlled clock synchronous by the instruction unit and execute the same operation on different data.
- *MISD – multiple instruction single data*: This category is empty.
- *MIMD – multiple instruction multiple data*: This correlates to a collection of self-contained computers, each equipped with its own instruction– and arithmetic unit.

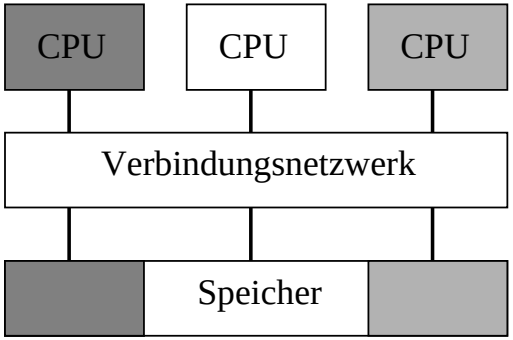
Scalable Computer Architecture II



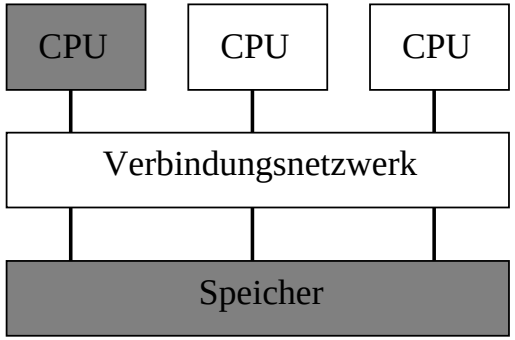
(a) verteilte Speicherorganisation, lokaler Adressraum



(b) verteilte Speicherorganisation, globalerAdressraum



(c) zentrale Speicherorganisation, lokaler Adressraum



(d) zentrale Speicherorganisation, globalerAdressraum

Distinction according to physical address space and physical memory organisation

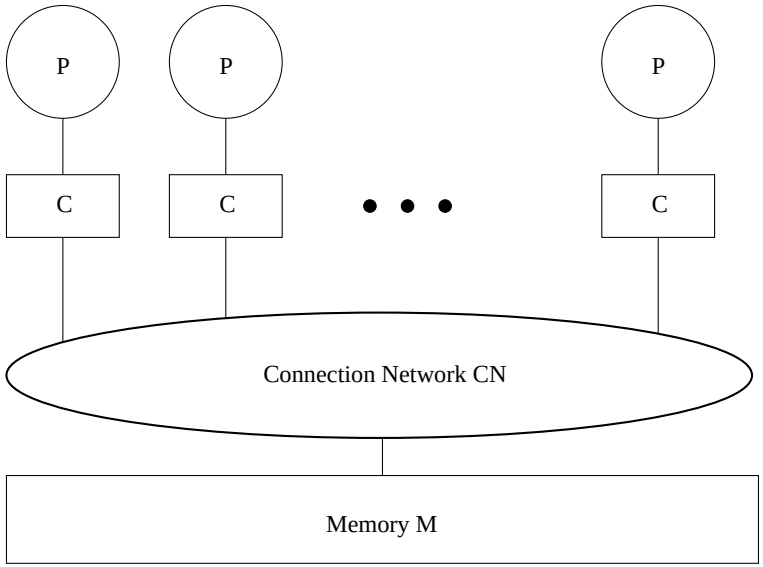
Scalable Computer Architecture III

Classification according to type of data exchange:

- Shared Memory
  - *UMA – uniform memory access*. Shared memory with uniform access time.
  - *NUMA – nonuniform memory access*. Shared memory with *non*-uniform access time, with cache-coherency we speak of *ccNUMA*.
- Distributed Memory
  - *MP – multiprozessor*. Private memory with message passing.

We will consider predominantly MIMD–machines. The SIMD approach exists still in the data parallel programming model (OpenMP, CUDA/OpenCL).

Shared Memory: UMA



- *Global adress space*: Each memory word has its global unique number and can be read and written by all processors.
- Memory access occurs over a *dynamic* connection network that connects processor and memory (therefrom later more).
- Memory organisation: *Low-order interleaving* – consecutive adresses are in consecutive modules. *High-order interleaving* – consecutive adresses are in the same module.

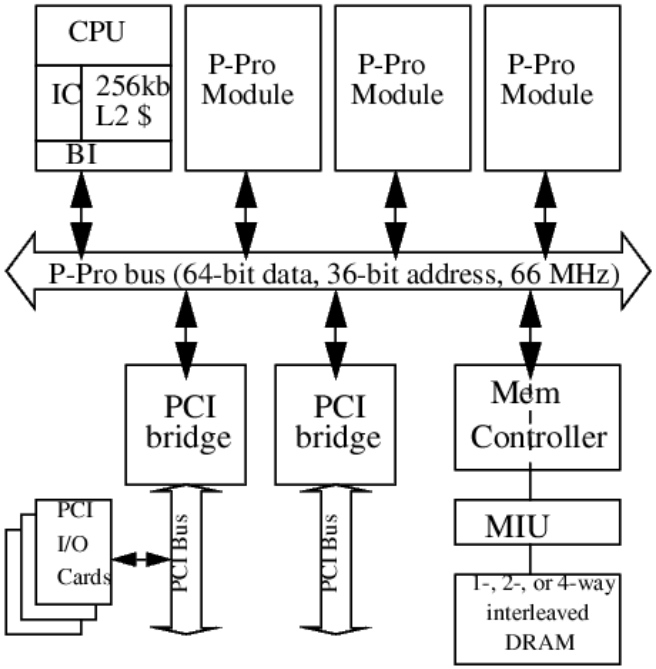
Shared Memory: UMA

Cache is necessary to

- avoid slow down of the processor, and
- to remove load from the connection network.

- Cache coherency problem: A memory block can be stored in several caches. What happens, if a processor writes?
- Write access onto the same block in different caches have to be serialized. Read accesses have to provide up-to-date data.
- UMA enables the usage of up to few 10th of processors.

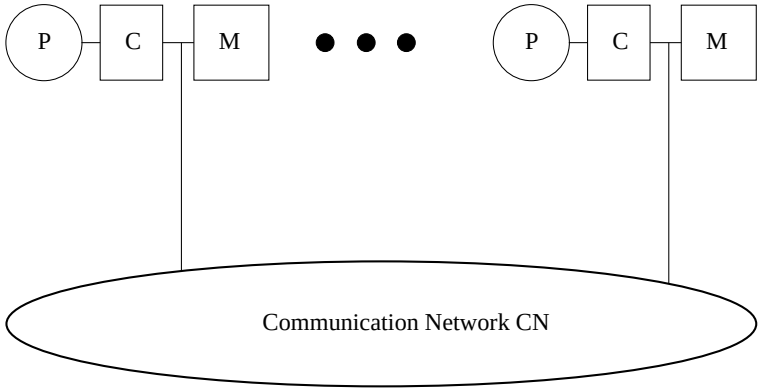
Shared Memory Board: UMA



Quad-processor Pentium Pro Motherboard

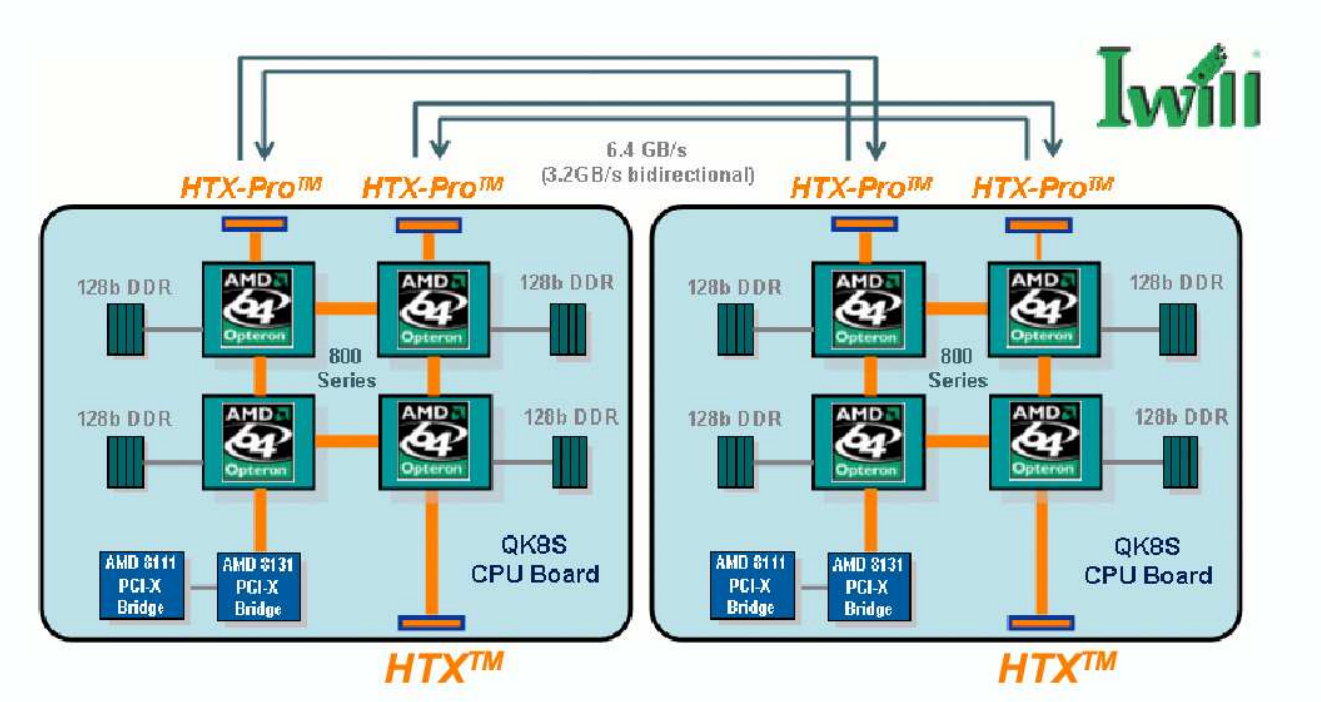
- Symmetric multi processing (SMP)
- Access to each memory word in equal time
- Implementation of cache coherency protocols (MESI)

Shared Memory: NUMA



- Each component consists of processor, memory and cache.
- *Global address space*: Each memory word has a global unique number and can be read and written from all processors.
- Access onto local memory is fast, access onto other memory is (considerably) slower, but transparently possible.
- Cache-coherency problem as in the UMA case
- Extreme memory hierarchy: level-1-cache, level-2-cache, local memory, remote memory
- Scales up to about 1000 processors (SGI Origin)

Shared Memory Board: NUMA

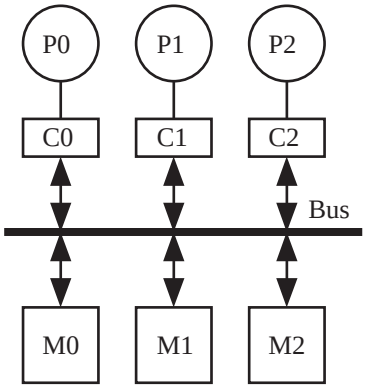


Quad-processor Opteron Motherboard

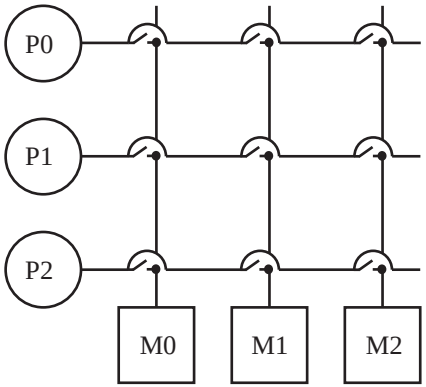
- Non-uniform memory access (NUMA)
- Intra/Interboard connection with Hypertransport HTX-technology

Dynamic Connection Networks

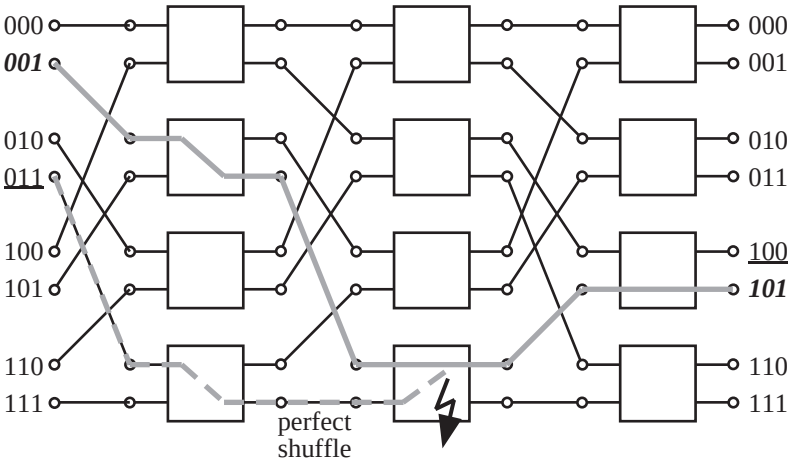
Line transmission: Truely electric connection from source to target.



(a)



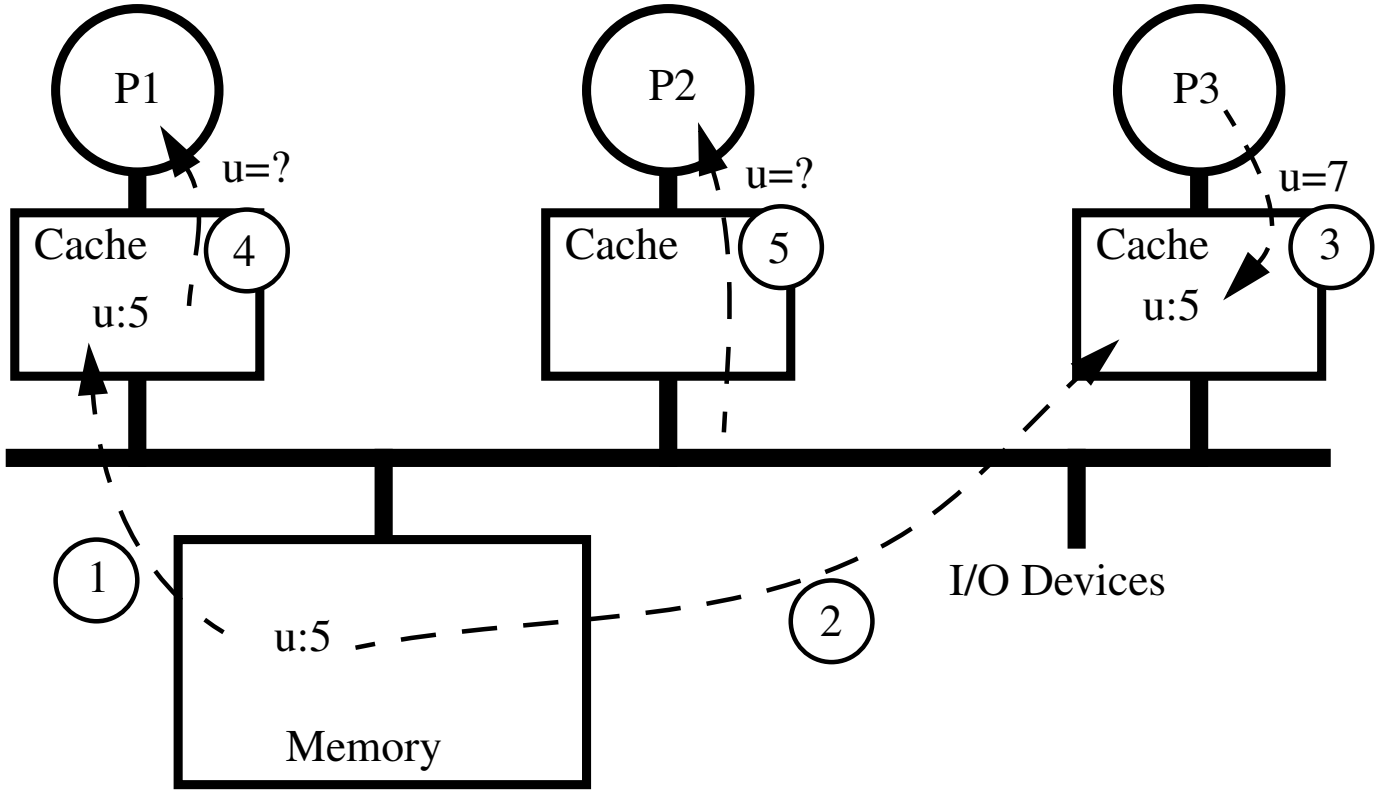
(b)



(c)

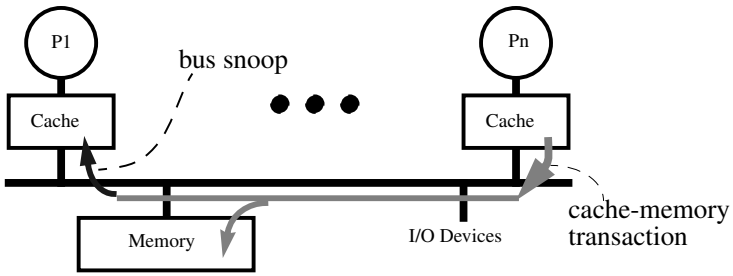
(a) Bus: connects only two units at a time, thus is not scalable. Advantages: cheap, cache coherency by snooping.
(b) Crossbar: Complete permutation realisable, but: P^2 switching units.
(c) Ω network: $(P/2)\,\mathrm{ld}\,P$ switching units, no complete permutation possible, each stage is *perfect shuffle*, simple routing.

Cache Coherency: An Example

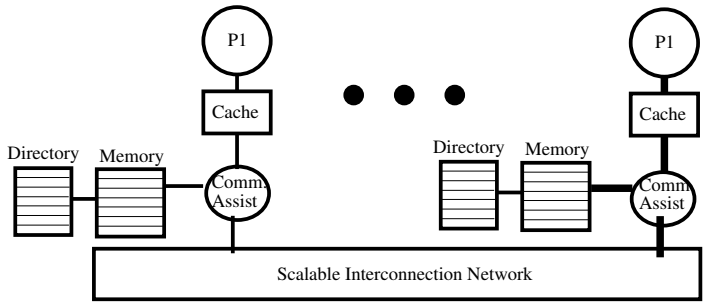


Cache Coherency: Protocol Types

Snooping based protocols



directory based protocols



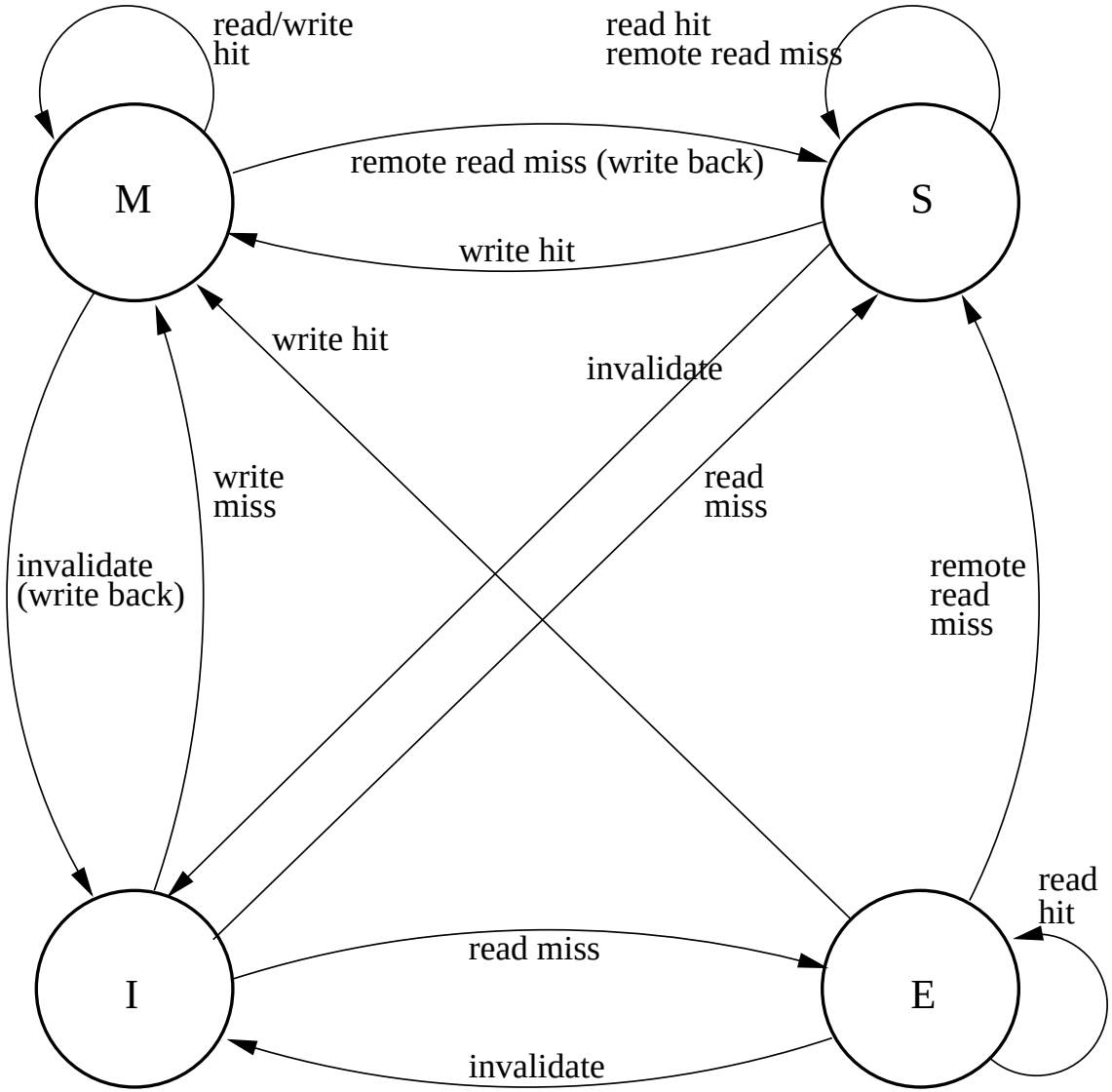
Cache Coherency: Bus Snooping, MESI

- Bus enables simple, efficient protocol for cache-coherency.
- Example MESI: Each cache block has one of the following states:

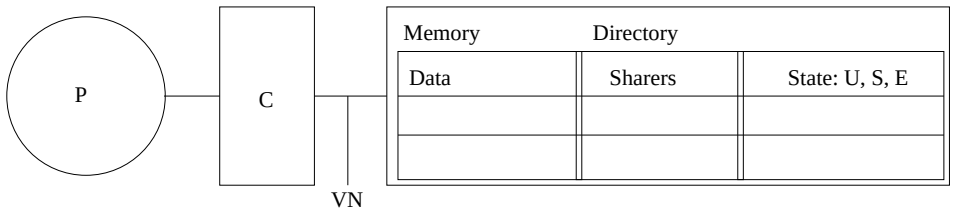
Status	Meaning
E	Entry valid, memory up-to-date, no copies exist
S	Entry valid, memory up-to-date, further copies exist
M	Entry valid, memory invalid, no copies exist
I	Entry is not valid

- Extends write-back protocol by cache coherency.
- Cache controller monitors the bus traffic (snoops) and performs the following state transitions (from the point of view of *a* controller):

Cache Coherency: Bus Snooping, MESI



Directory-based Cache Coherency I



States:

Cache–Block		Main Memory Block	
State	Description	State	Description
I	Block invalid	U	noone has the block
S	$\geq$ 1 copies exist, caches and memory are up-to-date	S	see left
E	exactly one has written the block (equals M in MESI)	E	see left

Directory-based Cache Coherency II

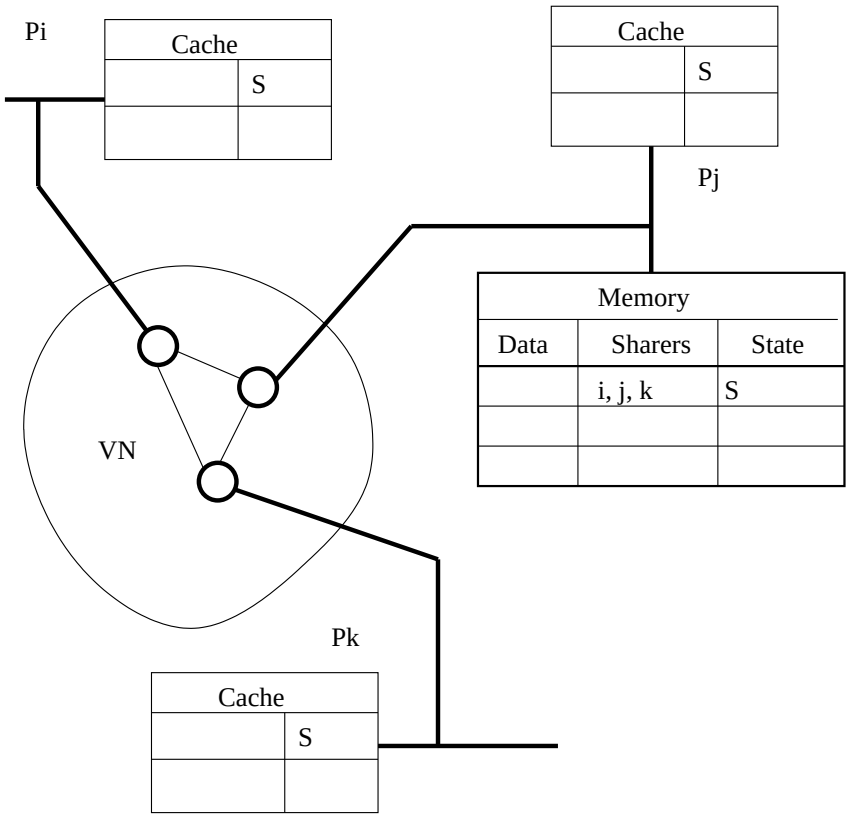
State transitions (view of directory):

Z	Action	Succ.	Description
U	read miss	S	Block is tranmitted to cache, bit vector stores who has the copy.
	write miss	E	Block ist transmitted to the requesting cache, bit vector contains who has the valid copy.
S	read miss	S	requesting cache gets copy from the memory and is registered in bit vector.
	w miss/hit	E	Requester gets (if miss) a copy of the memory, directory sends invalidate to all remaining owners of a copy.
E	read miss	S	Owner of the block is informed, this sends block back to home mode and changes to state S, directory sends block to requesting cache.
	write back	U	Owner wants to replace cache block, data are written back, noone has the block.
	write miss	E	Owner changes. Previous owner is informed and sends block to home node, this sends the block to new owner.

Variant: COMA (Cache Only Memory Architecture)

Directory-based Cache Coherency III: Example

Situation: Three processors P_i, P_j, und P_k have a cache line in state shared. Home node of this memory block is P_j



Actions:

1. Processor P_i writes into the cache line (write hit): Message to directory, this informs caches of P_j and P_k, succeeding state is E in P_i
2. Processor P_k reads from this block (read miss): Directory fetches block of P_i, directory sends block to P_k

Directory-based Cache Coherency IV: problem cases

Problems of ccNUMA architectures:

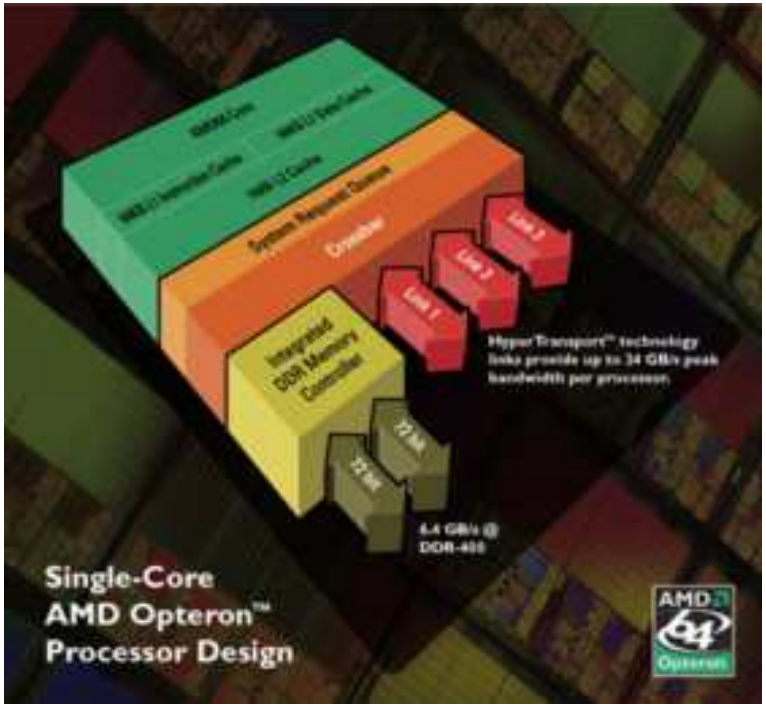
- false sharing: Two processors read and write different memory locations, that are by accident in the same block (probability increases with block size, Origin: 128 byte)
- capacity miss: Data amount, that a processor handles (working set), does not fit into the cache and the data are in the main memory of another processor

Solution for the capacity problem: Cache Only Memory Architecture (COMA), Software Distributed Shared Memory. Pages of the main memory (i.e. 4-16 KB) can be migrated automatically, combination with virtual memory mechanism.

Examples I: Intel Xeon MP

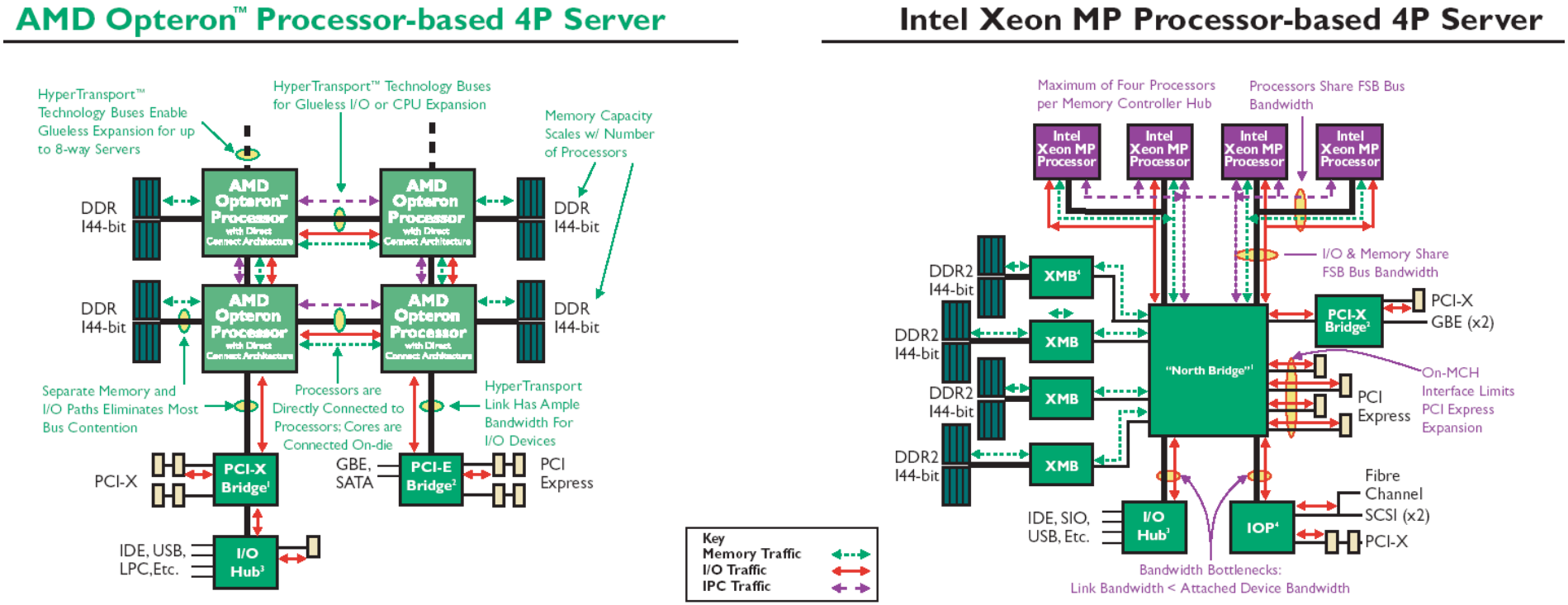
- IA32 architecture (as P4)
- Cache coherency protocol MESI
- Hyperthreading technique (2 logical CPUs)
- Integrated 3-level cache architecture (-1 MB L2, -8 MB L3)
- Machine Check Architecture (MCA) for external und internal buses, cache, translation look-aside buffer and instruction fetch unit
- Intel NetBurst Microarchitecture

Examples II: AMD Opteron



- Direct connect architecture
- On-Chip DDR memory controller
- HyperTransport technology
- Cache coherency protocol MOESI
  MESI states + 5th state
  direct data transfer between CPU caches via Hypertransport
- 64-bit Data/Address path, 48-bit virtual address space
- ECC for L1/L2 and DRAM with hardware scrubbing
- 2 additional pipeline stages
- many IPCs (instructions per cycle) through advanced branch prediction

Examples III: Server Architecture AMD vs INTEL



www.amd.com/us-en/assets/content_type/DownloadableAssets/AMD_Opteron_Streams_041405_LA.pdf

Examples III: Server Architecture AMD vs INTEL

Server System Comparison	AMD Opteron™	Intel Xeon[1]	Intel Xeon[2]	Intel Xeon MP[3]	Intel Itanium 2[4]
Modular, glueless scalability	Yes	Requires Northbridge	Requires Northbridge	Requires Northbridge	Requires Northbridge
SMP Capabilities	Up to 8-way	Up to 2-way	Up to 2-way	Up to 4-way	Up to 4-way
Direct Connect Architecture	Yes	No	No	No	No
Dual-Core technology	Yes	No	No	No	No
High Performance 32-bit and 64-bit computing	AMD64	No	EM64T	EM64T	No
HyperTransport™ technology	Yes	No	No	No	No
Integrated DDR memory controller	Yes	No	No	No	No
Front Side Bus frequency	1.4 – 2.6GHz†	533MHz	800MHz	667MHz	400MHz
Front Side Bus bandwidth	11.2 – 20.8GB/s†	4.3GB/s	6.4GB/s	10.6GB/s	6.4GB/s
Maximum Inter-processor bandwidth	8.0GB/s	4.3GB/s	6.4GB/s	10.6GB/s	6.4GB/s
Memory support	DDR200/266/333/400	DDR266	DDR333/DDR2-400	DDR266/333/DDR2-400	DDR200
Memory Bandwidth 2P System	12.8GB/s††	4.3GB/s	6.4GB/s	12.8GB/s	6.4GB/s
Memory Bandwidth 4P System	25.6GB/s†††	N/A	N/A	12.8GB/s	6.4GB/s
L1 cache size (max.)	64KB (Data) + 64KB (Instruction) per core	8KB + 12k mop	16KB + 12k mop	16KB + 12k mop	32KB
L2 cache size (max.)	1MB per core	512KB	2MB	1MB	256KB
L3 cache size (max.)	N/A	2MB	N/A	8MB	9MB
Maximum I/O bandwidth 2P System	24.0GB/s††	3.2GB/s	12.3GB/s	14.0GB/s	6.4GB/s
Maximum I/O bandwidth 4P System	32.0GB/s†††	N/A	N/A	14.0GB/s	6.4GB/s
SIMD Instruction Set Support	SSE, SSE2, SSE3	SSE, SSE2	SSE, SSE2, SSE3	SSE, SSE2, SSE3	N/A

Examples IV: Board Level Protocol

Hypertransport: Low latency chip-to-chip interconnect up to 8 CPUs with I/O aggregate bandwidth 8 GB/s (22.4), link width 16 bit (32), clock 1 GHz (1.4)
Priority request interleaving