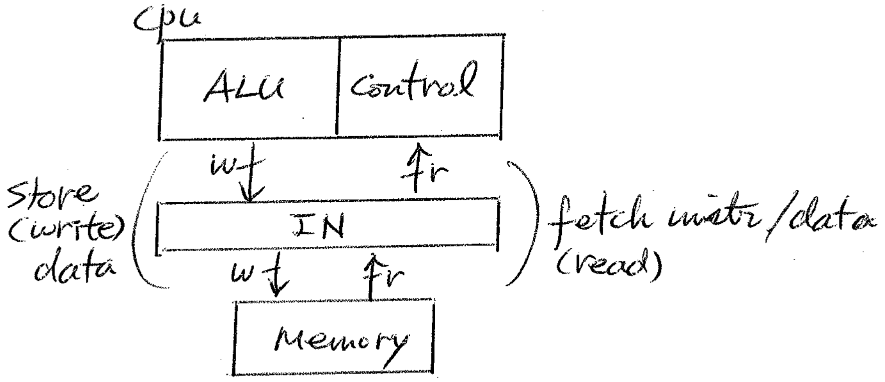
Ch2. Parallel Hardware and Parallel Software

— in order to write efficient parallel programs,
⇒ need to have knowledge of underlying HW & system SW.

— von Neumann Computation model

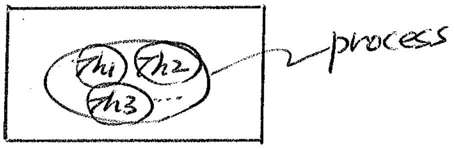


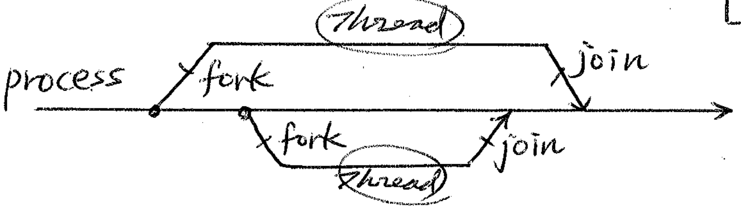
∃ von Neumann bottleneck — due to the separation of $\underbrace{\text{CPU}}_{\text{high exec. rate}}$ and $\underbrace{\text{memory}}_{\text{low transfer rate}}$

— $\underbrace{\text{process, thread}}_{\text{vs.}}$, $\underbrace{\text{multitasking}}_{\text{(op sys.)}}$

thread: light-weight
exec. switch between threads faster than between processes

multitasking: Capable of exec. multiple programs on a single processor
→ time sharing





— Threads belonging to the same process share resources (e.g., mem, I/O)

each thread has its own pc, stack.

— Solution approaches to von Neumann bottleneck

1. cache
2. Virtual memory
3. instruction-level parallelism (ILP) — fine-grain
4. Thread-level parallelism (TLP) — coarse-grain

1. Cache

Which memory item should be in the cache?

⇒ locality
- temporal — accessed again in the near future
  - ex) loop
- spatial — block movement (cache ⟷ mem)
  - block: contains nearby items (ex, array)

review

- block placement (mem → cache)
  - direct-mapped
  - set-associative
  - fully-associative
- when write-hit — WT vs. WB
  - write-miss — w. alloc. vs. no w. alloc.
- block replacement (no need for direct-mapped)
  - LRU, Random, ---

— cache access pattern and program exec. performance

C/C++ use row-major order (mem)

ex) 2-D array → row-major 1-D array layout

↓

↓

ex) 4x4 array A stored in memory

<mem-block>

Ø	A00 A01 A02 A03
1	A10 A11 A12 A13
2	A20 A21 A22 A23
3	A30 A31 A32 A33

block size=4

Assume: Cache
- direct-mapped
- 2 blocks total

<cache-block>

Ø	○○○○
1	○○○○

block size=4

1. row-major access

```
for (i=Ø ; i<4 ; i++)
    for (j=Ø ; j<4 ; j++)
        Y[i] += A[i][j]*S ;
```

⇒ 4 cache misses

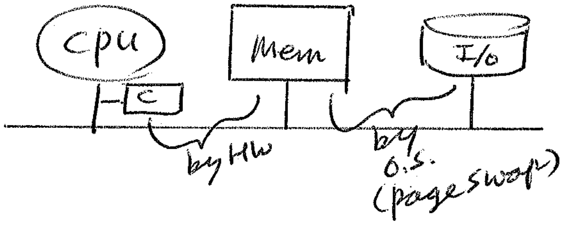
vs.

2. Column-major access

```
for( j=Ø ; j<4 ; j++)
    for( i=Ø ; i<4 ; i++)
        Y[i] += A[i][j]*S ;
```

⇒ 16 cache misses

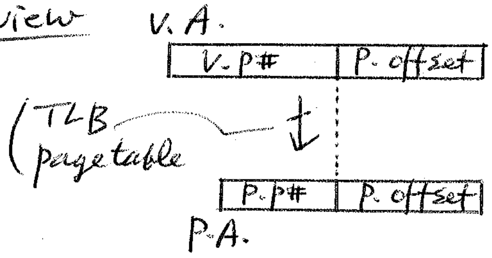
2. Virtual memory



very big program/data doesn't fit into memory.

⇒ pages by keeping temporal/spatial locality

review



— page fault
(page is stored in HD only

3. <u>ILP</u> (Instr. level parallelism) — pipelined / multiple-issue

- scalar pipeline with N stages

  tot. exec. time = $N + (\underset{IC}{n}-1)*1$ cycles $\approx$ N-fold

  ↓

- multiple issue

  VLIW (Very Long Instr. Word) — static $\underset{\text{compiler}}{\text{(SW)}}$ speculation.

  superscalar — dynamic $\underset{\text{Using buffer}}{\text{(HW)}}$ speculation.

<u>speculation</u>

determining the instructions to be exec. in parallel.

⇒ possibly incorrect

```
eg) Z=X+Y;
    if (Z>Ø)
        W=X;
    else
        W=Y;
```

— fetch group/dispatch/issue

$\left[\begin{array}{l} Z=X+Y; \\ W=X; \end{array}\right]$ 2 in parallel — by assuming $Z>\emptyset$

if speculation is incorrect, go back and exec. <u>W=Y;</u>

```
eg) Z=X+Y;
    W=*ap;
```

— fetch group/dispatch/issue

$\left[\begin{array}{l} Z=X+Y; \\ W=*ap; \end{array}\right]$ 2 in parallel

if *ap points to Z, go back and reexecute W=*ap;

— if ∃ long seq. of dependent statements
⇒ difficult to exploit ILP.

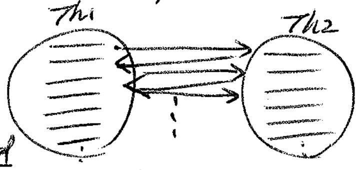
4. TLP (Thread level parallelism)

— simultaneous execution of different threads — coarse grained (than ILP)

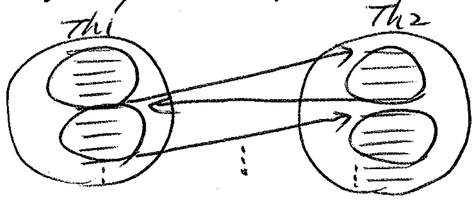
— 2 implementations of TLP

- multicore — multiple independent exec. cores with all resources onto a single processor chip.
- HW multithreading (time slice way)
  switching a single processor between different threads

vs. fine-grained — instr. by instr. in each thread

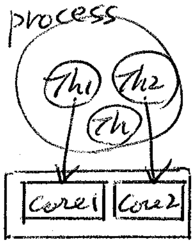


coarse-grained
chunk-by-chunk

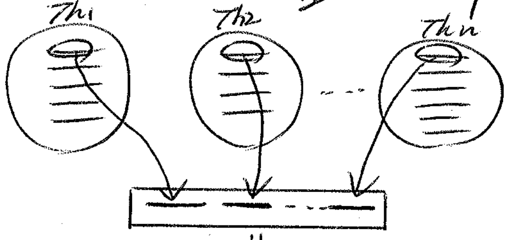


—multicore

process



※ SMT (Simultaneous MultiThreading)
a variation of fine-grained multithreading
— multiple threads utilize FU's of a processor



— max. 1 instr. from each Th. at same cycle

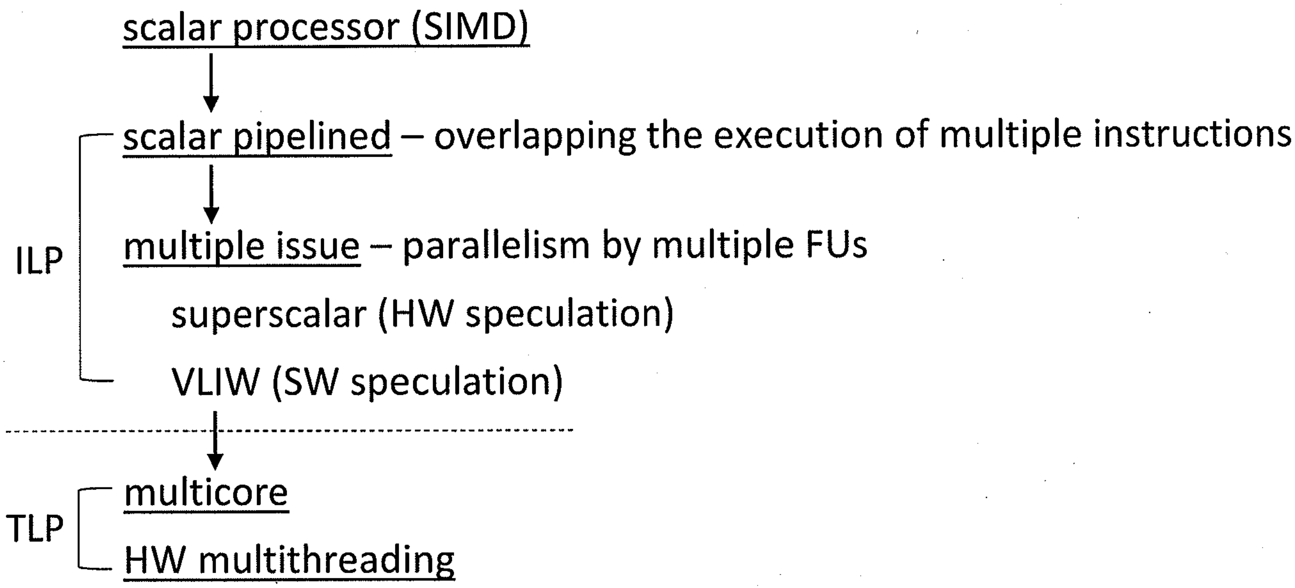
⇓
compete for resources (FU's) in a processor

as if ⇒ ∃ multiple logical processors
each has pc and replicated registers
HW supports

— multicore + SMT
ex) Intel Core i3, i5, i7
2 logical processors
— IBM Power7
4 L.p.
— Sun/Oracle T4
8 L.p.

— using more than 2 L.p. is not efficient

Processor Design Migration



scalar processor (SIMD)

scalar pipelined – overlapping the execution of multiple instructions

multiple issue – parallelism by multiple FUs

superscalar (HW speculation)

VLIW (SW speculation)

multicore

HW multithreading

- ILP (Instruction Level Parallelism) → TLP (Thread Level Parallelism)

ILP: Achieving HP via overlapped instruction execution;
Serial control flow;
Limited degree of parallelism (sequence of dependent instructions)
high energy consumption, HW complexity (space).

TLP: Achieving HP via multiple cores on a processor chip (a thread on a core)
or, via HW multithreading (e.g., SMT);
Multiple independent control flow;
Parallelism at process/thread level.

Multicore processor design issues

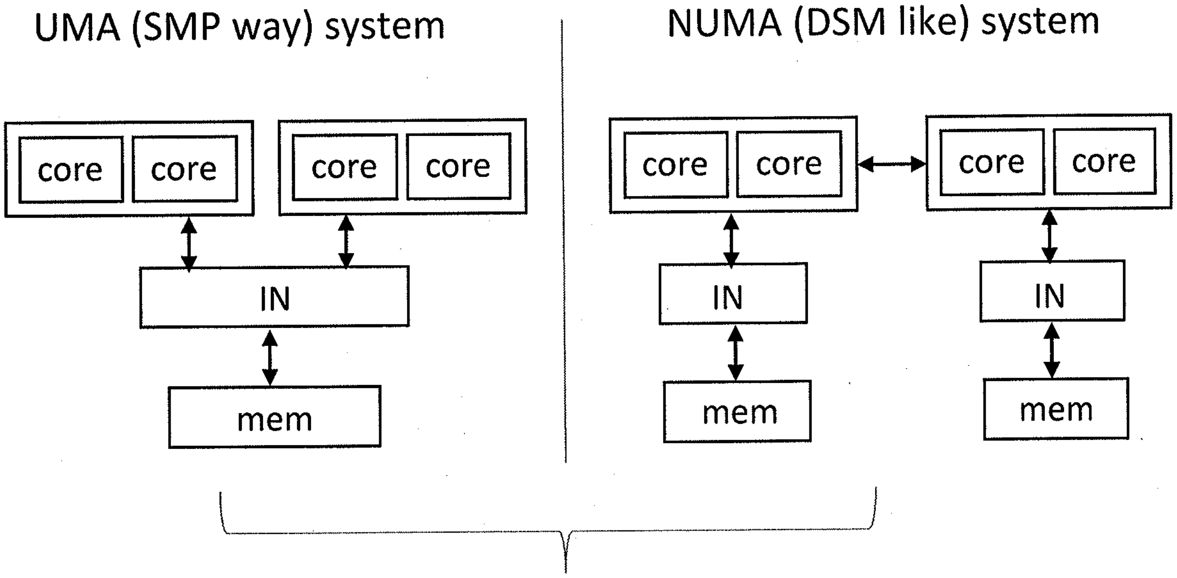
- data transfer between cores – on-chip interconnection provides enough bandwidth;
- IN should be scalable to increasing number of cores;
- fault tolerance of the entire system;
- low power consumption – desired (through IN);
- efficient memory (L1, L2, L3 caches), I/O systems for fast data transfer

  (to avoid idle core);

  more cores → more cache levels to fulfill bandwidth requirements;

  ex) Core i7: L1,L2 local to each core; L3 shred among all cores;

ex) shared-memory MIMD with multiple multicore processors



UMA (SMP way) system

NUMA (DSM like) system

both shared-memory system

Multicore chip architecture – 3 design choices

- Hierarchical design

  Multiple cores share multiple caches

  ex) Intel Core i7 – 4 cores/chip; 2 logical cores (hyper threading)/core

  IBM Power7 – 8 cores/chip; 4 logical threads/core

  AMD Opteron – 8 cores/chip

- Pipelined design

  Data elements are processed by multiple cores in a pipelined way;

  Each core performs a specific processing step;

  ex) Network processors in routers – Xelerator X10, X11 (800 cores, logically arranged for a pipeline)

  graphics processors

- Network based design

  Msg-passing (distributed-memory) way;

  ex) SUN Ultra SPARC T4 – 8 cores/chip; 8 threads (SMT)/core → total 64 threads; IN: Xbar

  IBM BG/Q processors

  Intel Teraflops research chip – 80 cores; IN: 2-D (8x10) mesh

  Intel SCC – single chip cloud computer

Springer book

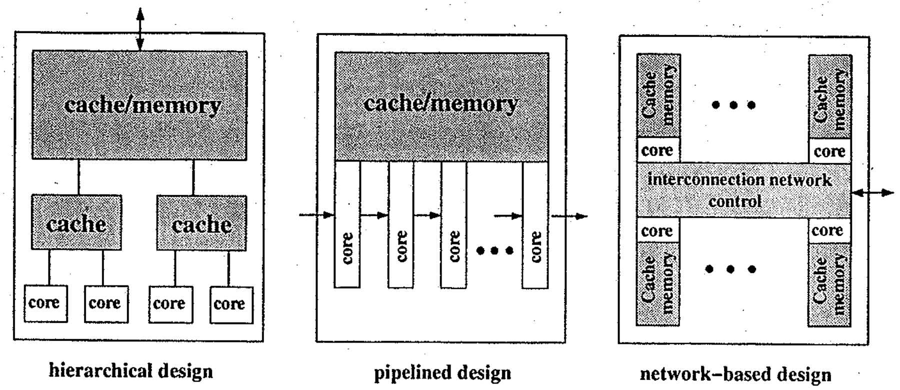


Fig. 2.6 Design choices for multicore chips according to [121].

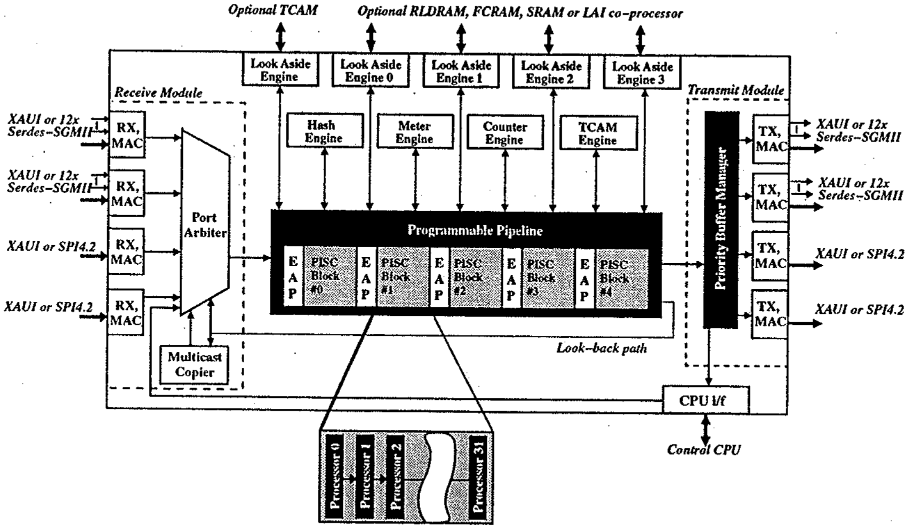
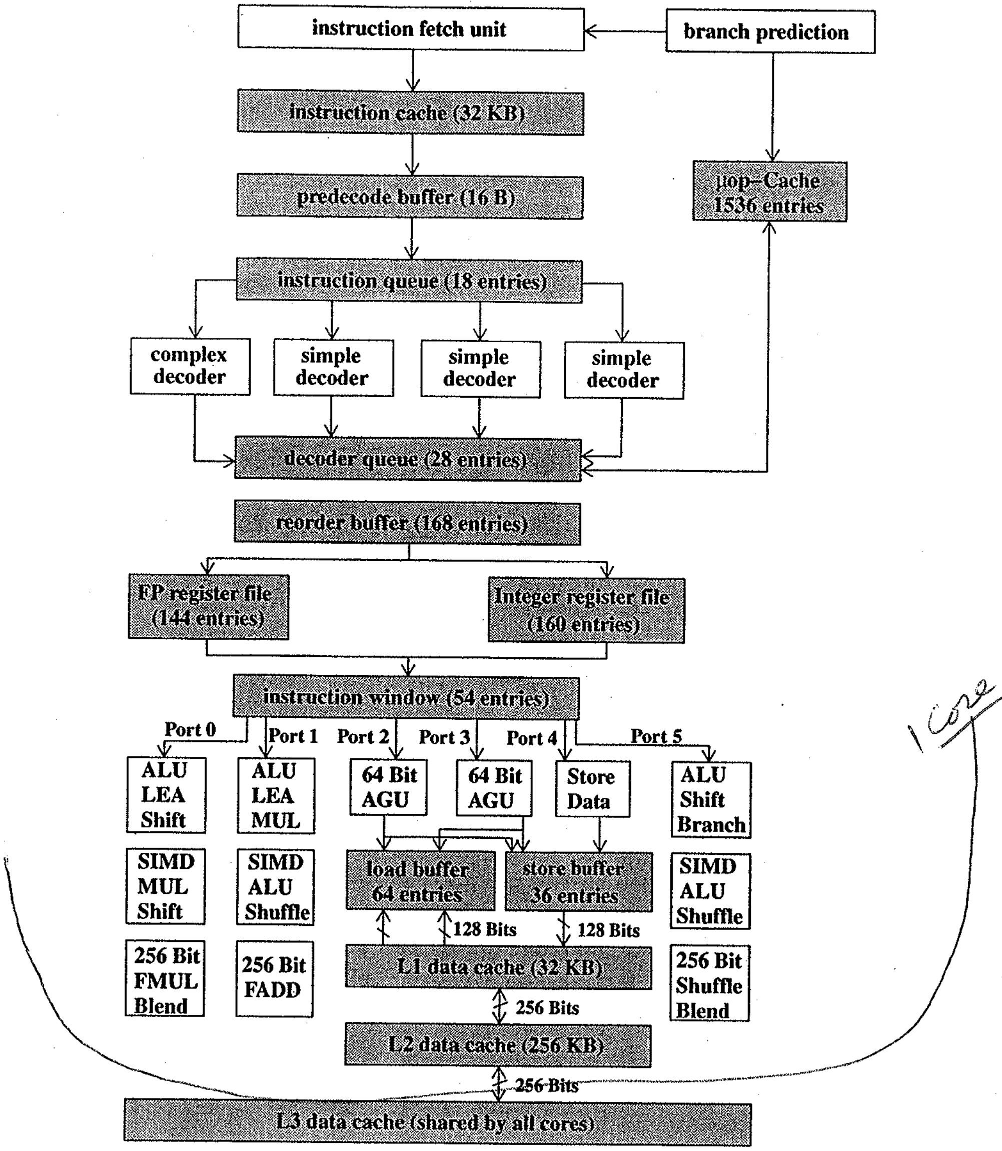


Fig. 2.7 Xelerator X11 Network Processor as an example for a pipelined design [198].

Table 2.1 Examples for multicore processors in 2012.

processor	number cores	number threads	clock GHz	L1 cache	L2 cache	L3 cache	year released
Intel Core i7 3770K "Ivy Bridge"	4	8	3.5	4 x 32 KB	4 x 256 KB	8 MB	2012
Intel Xeon E5-2690 "Sandy Bridge EP'	8	16	2.9	8 x 32 KB	8 x 256 MB	20 MB	2012
AMD Opteron 3280 "Bulldozer"	8	8	2.4	8 x 16 KB	4 x 2 MB	8 MB	2012
AMD Opteron 6376 "Piledriver"	16	16	2.3	16 x 16 KB	8 x 2 MB	2 x 8 MB	2012
IBM Power7	8	32	4.7	8 x 32 KB	8 x 256 KB	32 MB	2010
Oracle SPARC T4	8	64	3.0	8 x 16 KB	8 x 128 KB	4 MB	2011

Springer book



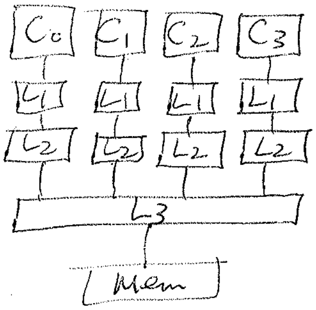


Fig. 2.8 Block diagram to illustrate the internal architecture of one core of an Intel Core i7 processor (Sandy Bridge).