Overview of the Class
Signal Processing System Design

Wonyong Sung
School of Electrical Engineering
Seoul National University

2009. 3

Contents

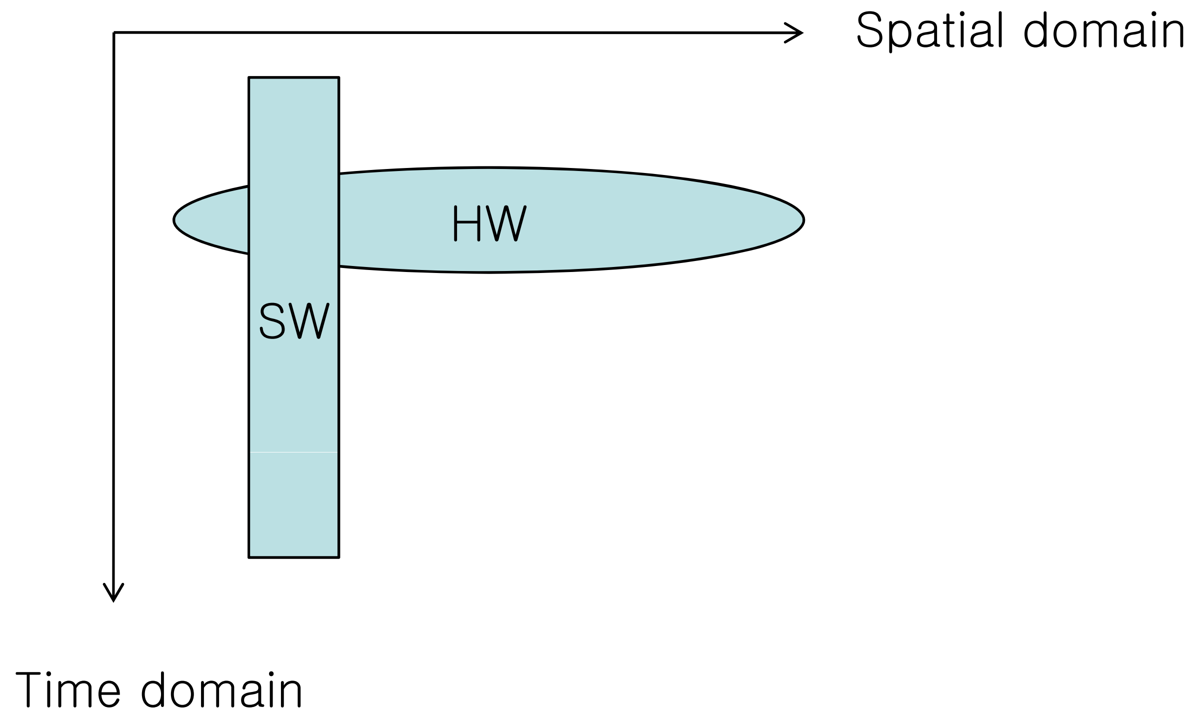
1. Introduction
2. High Performance DSP Architecture
3. Programming Models and Methods
4. Conclusion and Course Outline

1. Introduction

- Signal processing system
  - Many are based on computation intensive real-time DSP (digital signal processing) algorithms. (MPEG 1, 2, H.264, JPEG, CELP vocoder, CDMA, Wibro, software defined radio)
  - Main kernels: filtering, transform, motion estimation, object recognition, Viterbi decoding
  - Usually dedicated purpose, e.g. DVR, PDA, digital camera, cellular phones, printers
  - Software compatibility is not much required as is for PC, but more optimization expected in terms of power consumption (less than 1/10) and cost.
  - Implementation approaches: hardware vs software based

Two different approaches in DSP implementation

- SW does everything in sequential
- HW tries to do everything in parallel



Hardware or Software? – HW system

- Fixed interconnection of multipliers, adders, and memory blocks
  - Pros: High throughput at low power consumption possible
    - Low chip size (economic for mass production)
  - Cons: Difficult to change the functionality
    - Dedicated chip design needed (large NRE)
  - Design method: HDL, high-level synthesis (scheduling and binding), logic-synthesis, …
- The weakest point when considering scaling
  - (scaling reduces the chip size, but increases NRE)
  - Large NRE
  - How to overcome it? FPGA (Field Prog Gate Array)

Hardware or Software? – SW system

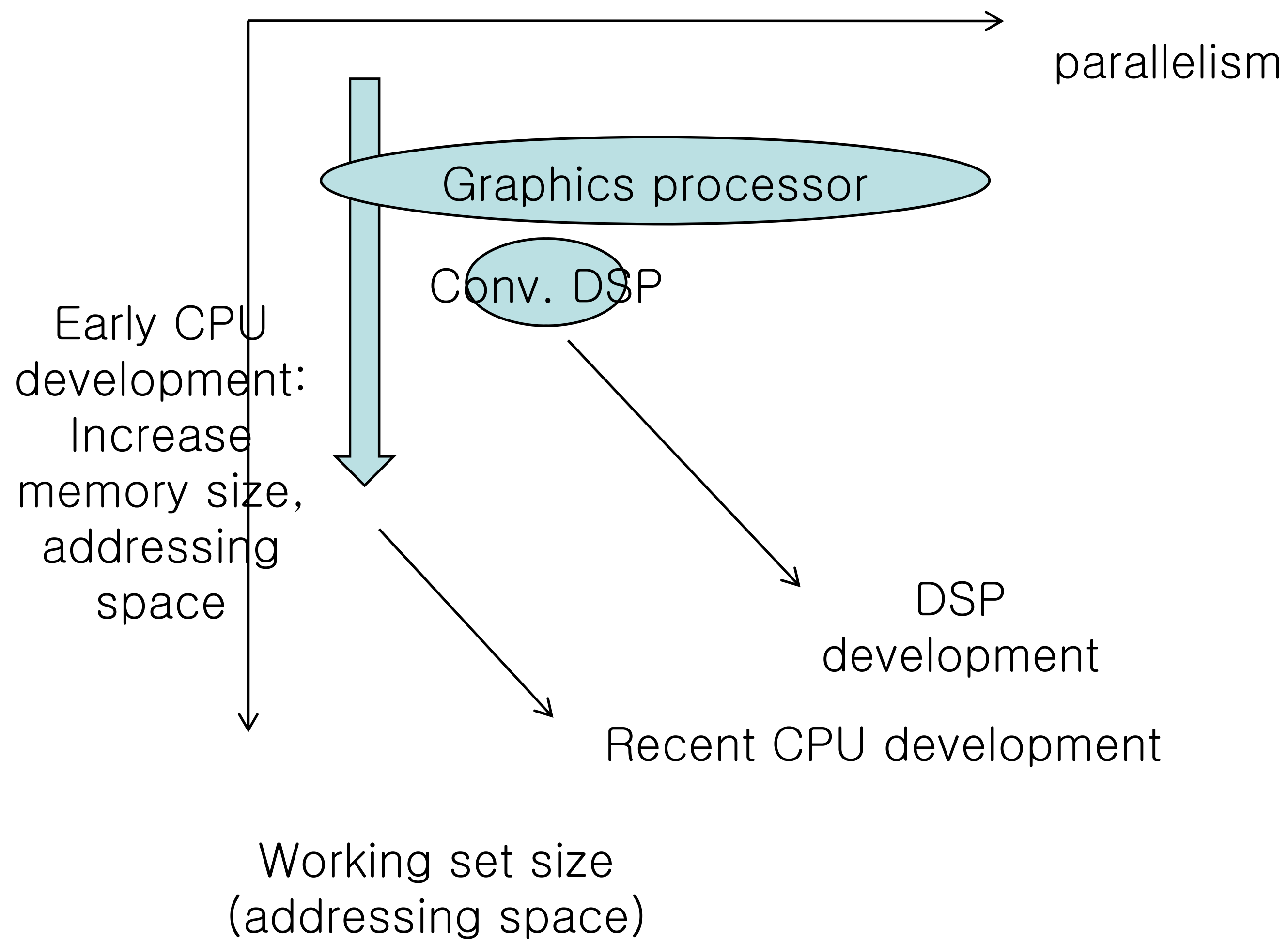
- Contains "ALU – memory (for program and data) – instruction decoding logic"
  - Pros: Easy to program/change the functionality
  - Cons: large chip size, larger power consumption
  - Design method: High level language (C, C++), parallel programming library, assembly language
- The weakest point when considering scaling
  - Small number of execution blocks (ALUs) – limited execution throughput
  - How to overcome it? Parallel computer architecture

Spectrum of SW based systems

- Parallelism: the number of operations executed simultaneously
  - 1: sequential processor (conventional DSP, RISC, ..)
  - Data level parallelism (SIMD): partitioned data-path SIMD (Intel MMX), vector processors
  - Instruction level parallelism (closely coupled MIMD): superscalar processors, VLIW processors
  - Thread level parallelism (loosely coupled MIMD): SMP (symmetric multiprocessors), GPU, interconnection of systems
- Size of working set (or addressing range)
  - Large: usually general purpose computing
    - Allows easy access of large main memory (DRAM)
    - Inefficient in terms of energy consumption, speed
  - Small: usually for special purpose computing

Why the size of working set important?

- If you need to do a large number of small jobs (requiring only small memory size), it is easy to parallelize and execute at a high speed.
- But, if you need to do a large number of large jobs (requiring a large working set), it requires a large communication cost and the clock speed is slow.
  - -> Localize means reducing the working set size.
- Usually,
  - Video processing requires a large working set (MBs memory)
  - However, speech or communication processing require a small working set (KBs memory)


parallelism
Graphics processor
Conv. DSP
Early CPU development: Increase memory size, addressing space
DSP development
Recent CPU development
Working set size (addressing space)

Is the boundary (between SW and HW) clear?

- Someone design SW based systems (Microblaze) using FPGA. What is this?
- A HW system can be a kind of SW based systems by
  - Increasing the parallelism as much as possible
  - Allow heterogeneous parallelism
  - Reduce the working set size as small as possible

How to speed-up SW based DSP systems? (1)

- Reduced complexity DSP algorithms
  - FFT (fast Fourier transform)
  - Full search vs three-step search motion estimation
  - Skip-zero input filtering in interpolation
- Use specialized instructions (highly efficient instructions for DSP)
  - Programmable digital signal processors
  - ASIP (application specific instruction set processors)
- Process multiple data at a time (SIMD)
  - Partitioned data-path ALU
  - 8 data processing of 16 bit data with 128 bit ALU
  - Economic, but low flexibility (large data arrange overhead)

How to speed-up SW based DSP systems? (2)

- Execute multiple instructions at a time (closely coupled MIMD)
  - Superscalar or VLIW
  - More flexible than SIMD
  - Mostly operates in single thread
- Multi-thread approach (divides a program into multiple execution threads)
  - Shared memory systems (e.g. SMP)
  - Distributed memory multi-processor systems

The expected speed-up (or parallelism) from each technology

- Special instruction set
  - (max 10) Mostly 2~10. The frequency of the special instruction usage determines the efficiency (Amdahl's law)
- Pipelining
  - (max 5) Mostly 3~5. The number of possible pipelining stages, pielining hazards limit the gain.
- Partitioned data-path
  - (max 8) Mostly 2~4. The degree of data-level parallelism, the overhead of data rearrangement hinders the gain.
- Superscalar or VLIW
  - (max 8 for VLIW)
- Multi-core
  - (max 64? We do not know yet) Can be much more potential.

Scope of this course

- Real-time digital signal processing system design (mainly embedded system) in SW
  - Special topic on communication system design using FPGA (2010, spring – covers HW based approach)
- Covers
  - Fast DSP algorithms
  - Programmable digital signal processors
  - SIMD processors
  - VLIW processors
  - SMP
  - GPU and many-core systems

Scope of this course

Real-time embedded DSP

- For portable devices, wireless telephone, robotics, ..
- Focus: low-power, low-cost (single chip SoC)
- Architecture:
  - ARM (RISC) CPU
  - Programmable DSP
  - VLIW DSP
- SW technique
  - Instruction level parallelism
  - C programming & optimization

High-performance DSP

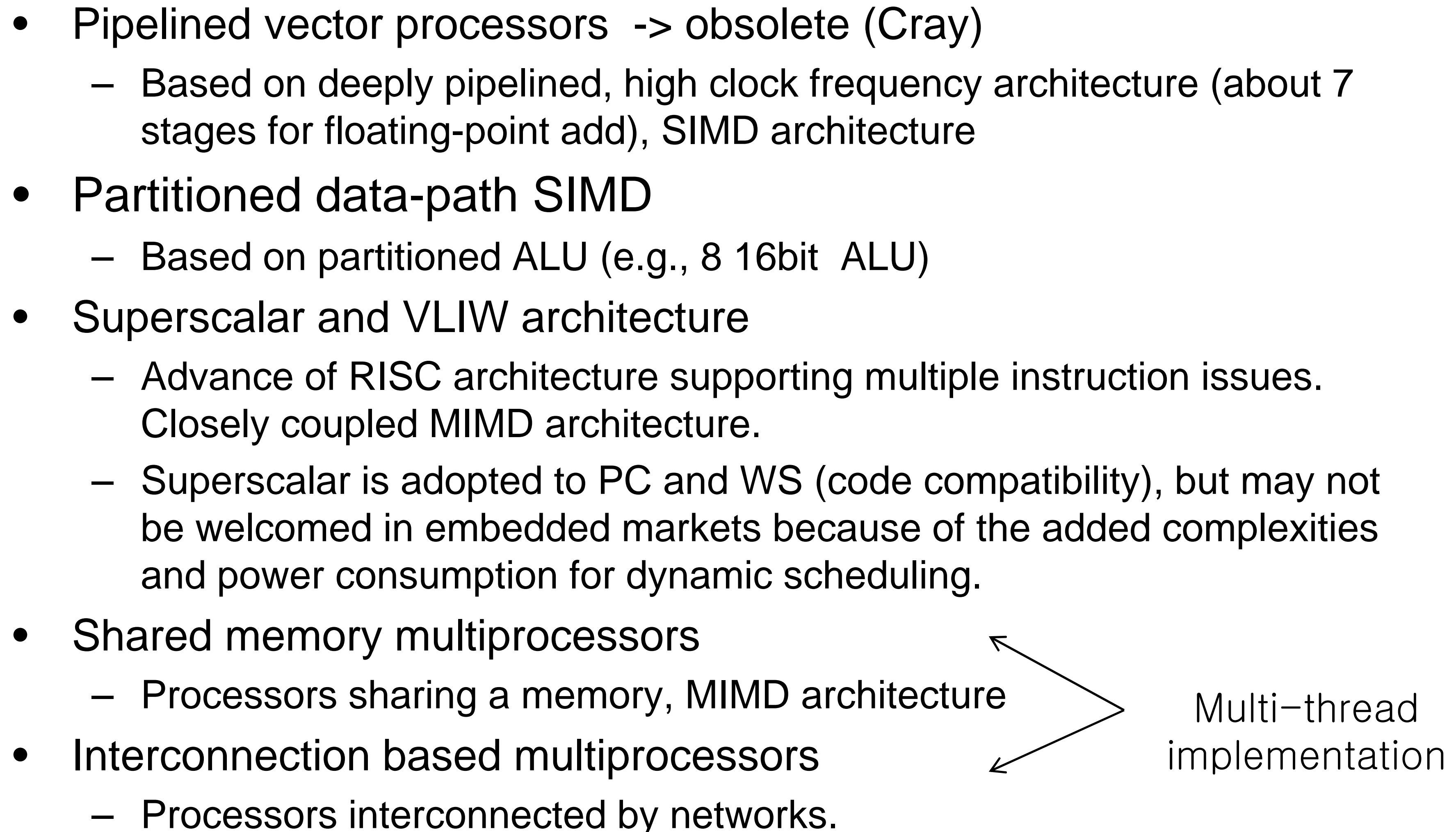
- For object recognition, medical imaging, ..
- Focus: high-performance,
- Architecture:
  - Multi-core PC (SMP)
  - Graphics CPU
- SW technique
  - Parallel partitioning
  - OpenMP, MPI, CUDA

2. High Performance DSP Architecture

- Traditional DSP architecture
  - Texas Instruments C2x, C50, C54, C55
  - Equip specialized instructions for digital filtering, FFT, and Viterbi decoding
  - One tap of FIR filtering requires only one instruction, while a RISC CPU consumes almost 10 instructions.
  - Difficult to develop good compilers
  - Support a small memory space (support applications with a small working set, such as audio, speech codec, communication modem)
- New DSP architecture
  - Based on parallel computer architecture

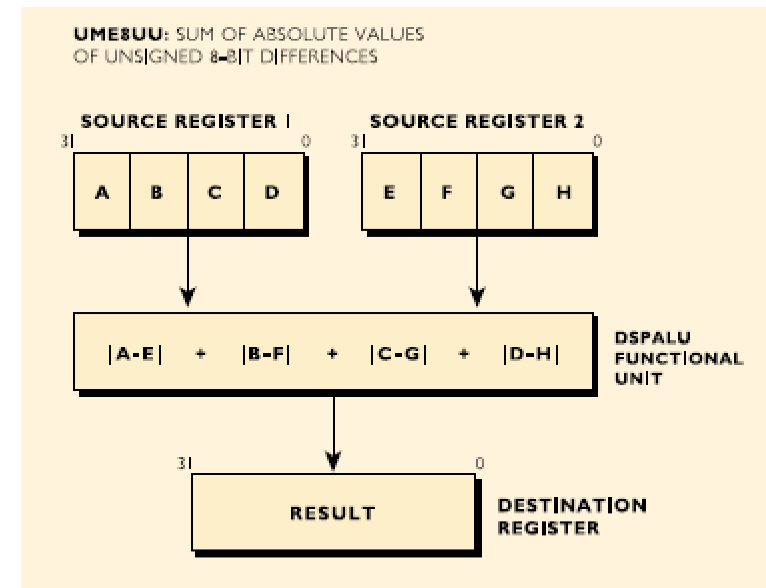
Parallel Computer Architecture

- Pipelined vector processors -> obsolete (Cray)
  - Based on deeply pipelined, high clock frequency architecture (about 7 stages for floating-point add), SIMD architecture
- Partitioned data-path SIMD
  - Based on partitioned ALU (e.g., 8 16bit ALU)
- Superscalar and VLIW architecture
  - Advance of RISC architecture supporting multiple instruction issues. Closely coupled MIMD architecture.
  - Superscalar is adopted to PC and WS (code compatibility), but may not be welcomed in embedded markets because of the added complexities and power consumption for dynamic scheduling.
- Shared memory multiprocessors
  - Processors sharing a memory, MIMD architecture
- Interconnection based multiprocessors
  - Processors interconnected by networks.



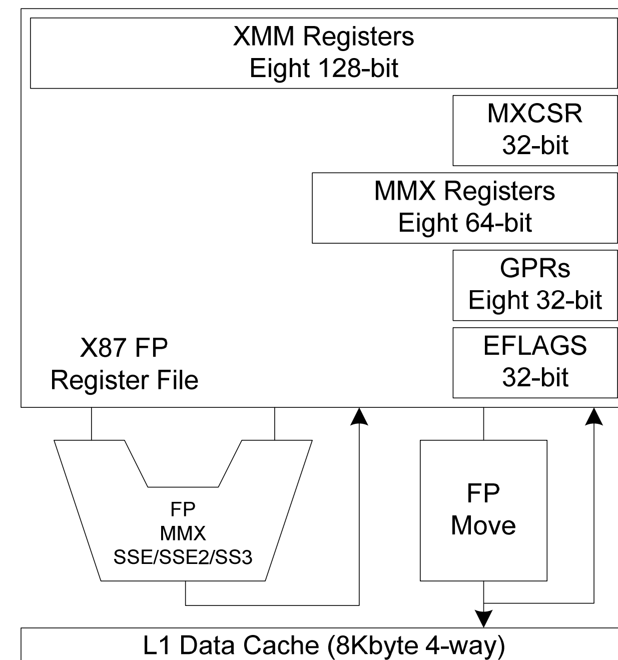
Partitioned datapath

- X8, x4, x2 partitioning of the datapth
  - Especially efficient for video coding (8bit or 16 bit precision)
  - Main ALU modification
    - Sun, HP, ARM
  - Floating-point unit modification
    - Intel (SSE, SSE2)
- Characteristics:
  - small overhead, relatively small speed-up, pack/unpack overhead can be dominant for irregular data structure
  - Automatic SIMDizer not developed yet



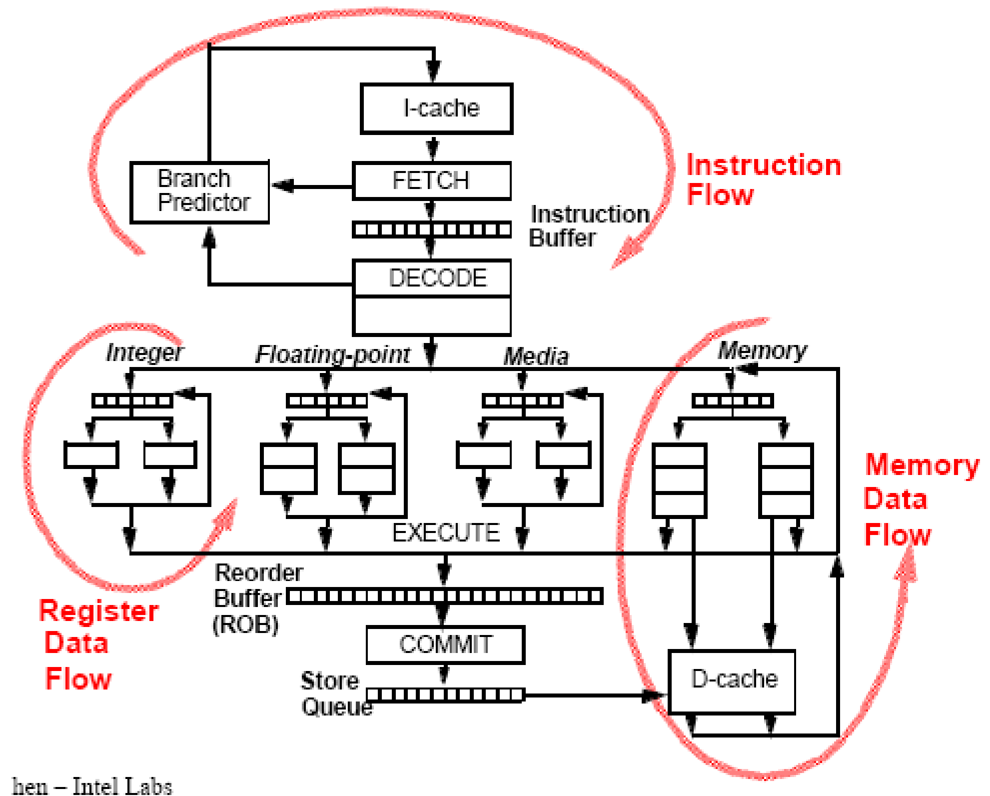
Partitioned data-path architecture (SIMD extension)

- Divide a long datapath (e.g. 64bits) into multiple small words (e.g. 8 8bits) for multipixel processing of image.
- Can be considered an extension of vector processors, but with more HW, instead of higher stage pipelining.
- Intel MMX MMX, MMX-SSE (multiple floating-point operations supported)



Superscalar and VLIW

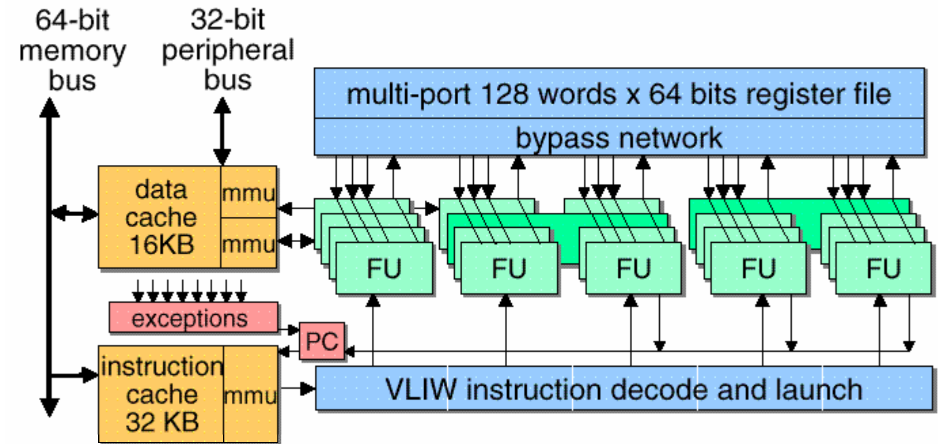
- Superscalar: execute multiple instructions at a time, the scheduling is conducted by the CPU. So, existing binary codes will not be modified, they will run just faster. But, HW is complex.
  E.g. Intel Pentium
- VLIW: execute multiple instructions at a time, but the scheduling is conducted in SW at the compilation stage. So, no binary code compatibility. HW is a little simpler (compared to superscalar).
  E.g. TI C6x (upto 8 issues), Philips Trimedia (5)

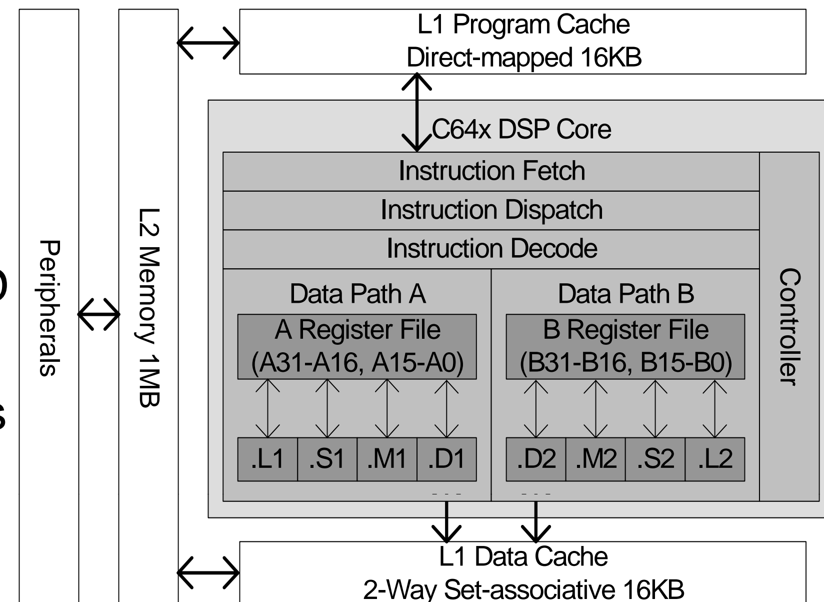


Superscalar architecture

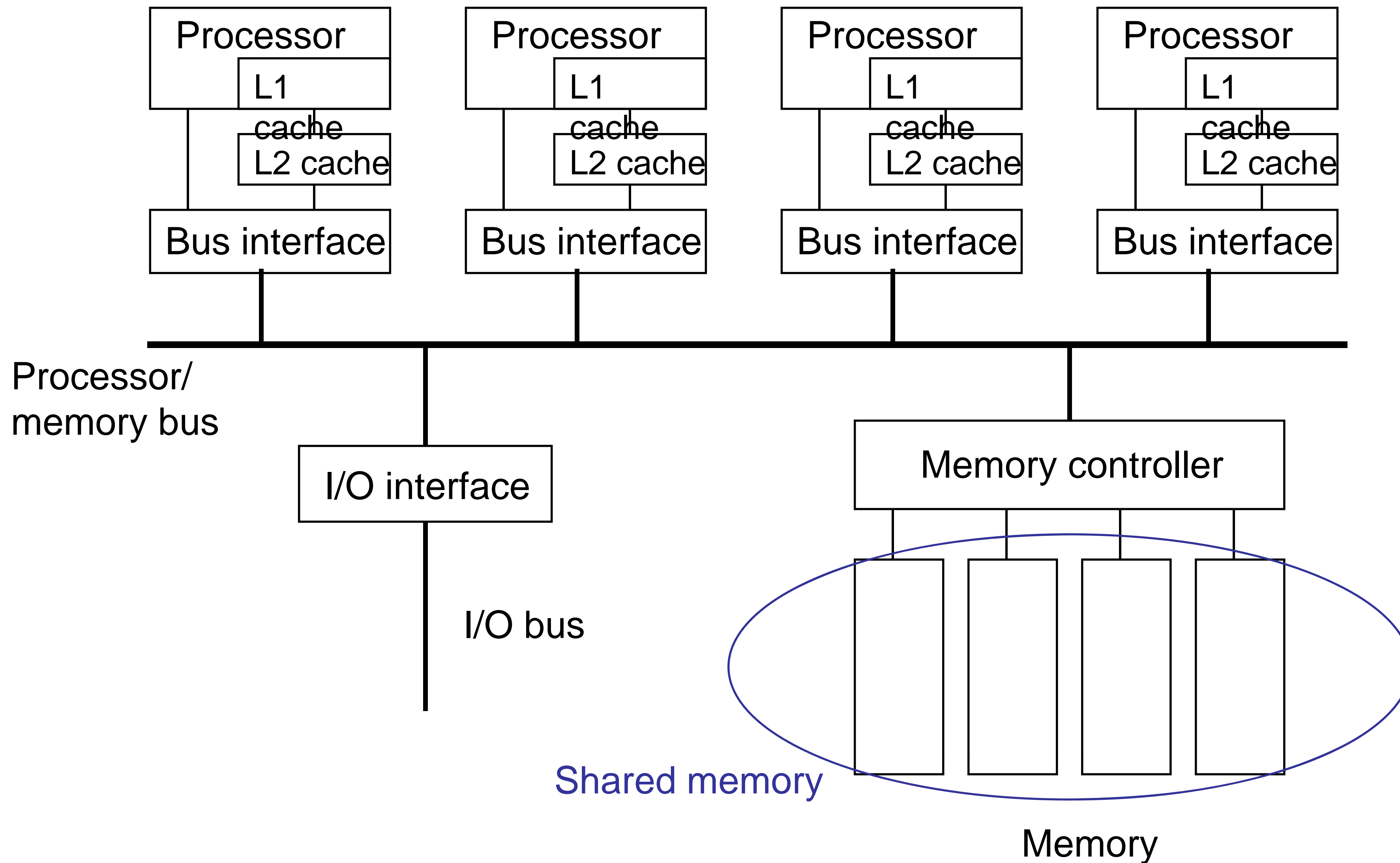
VLIW (Very Long Instruction Word)

- Now popular as commercial multimedia processors
  - TI C6x series, Philips Trimedia, Carmel DSP
    - C6x has 8 functional units (not general)
    - Trimedia has 5 general functional units
  - VLIW compiler (automatic parallelization), scheduler
  - Upto x8 speed-up for C6x (usually around x5), can combine with SIMD processing.
  - Good memory systems, peripherals are implemented

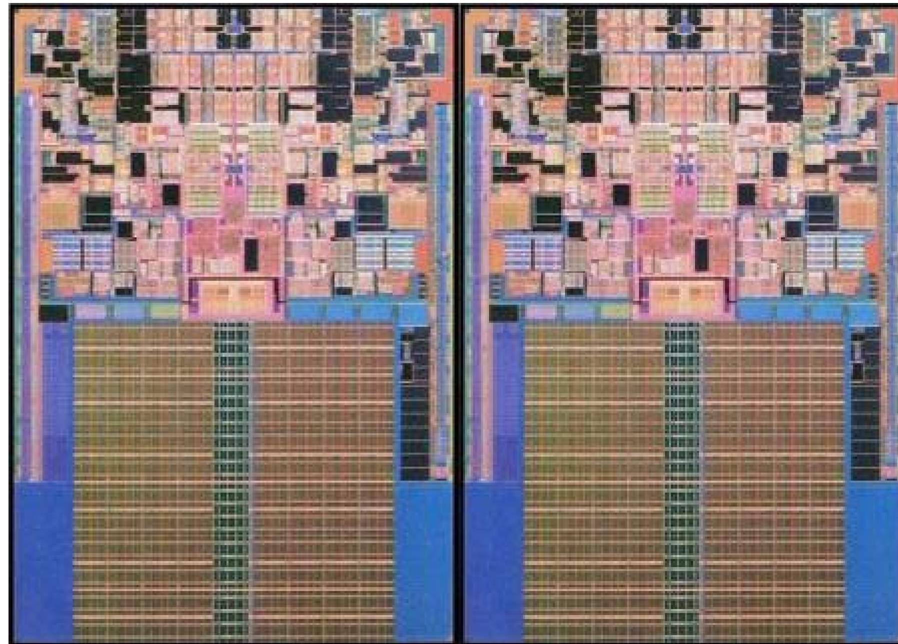




Quad Pentium shared memory multiprocessor

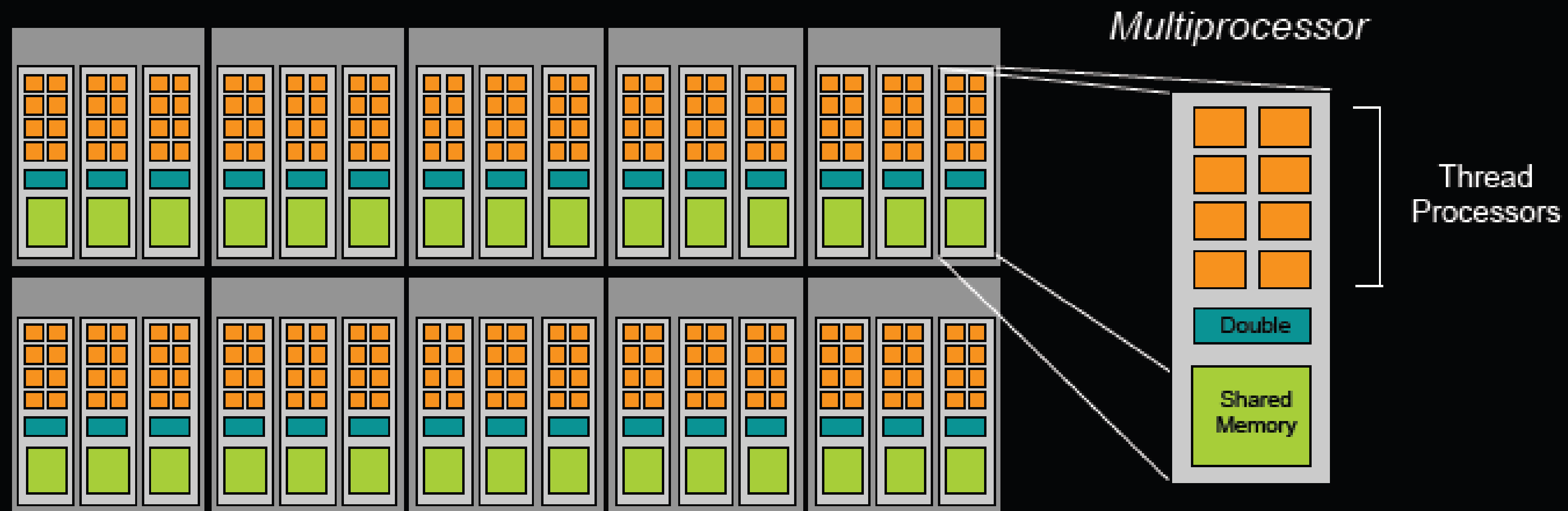


Intel’s quad core



GPU: nVIDIA 10 series

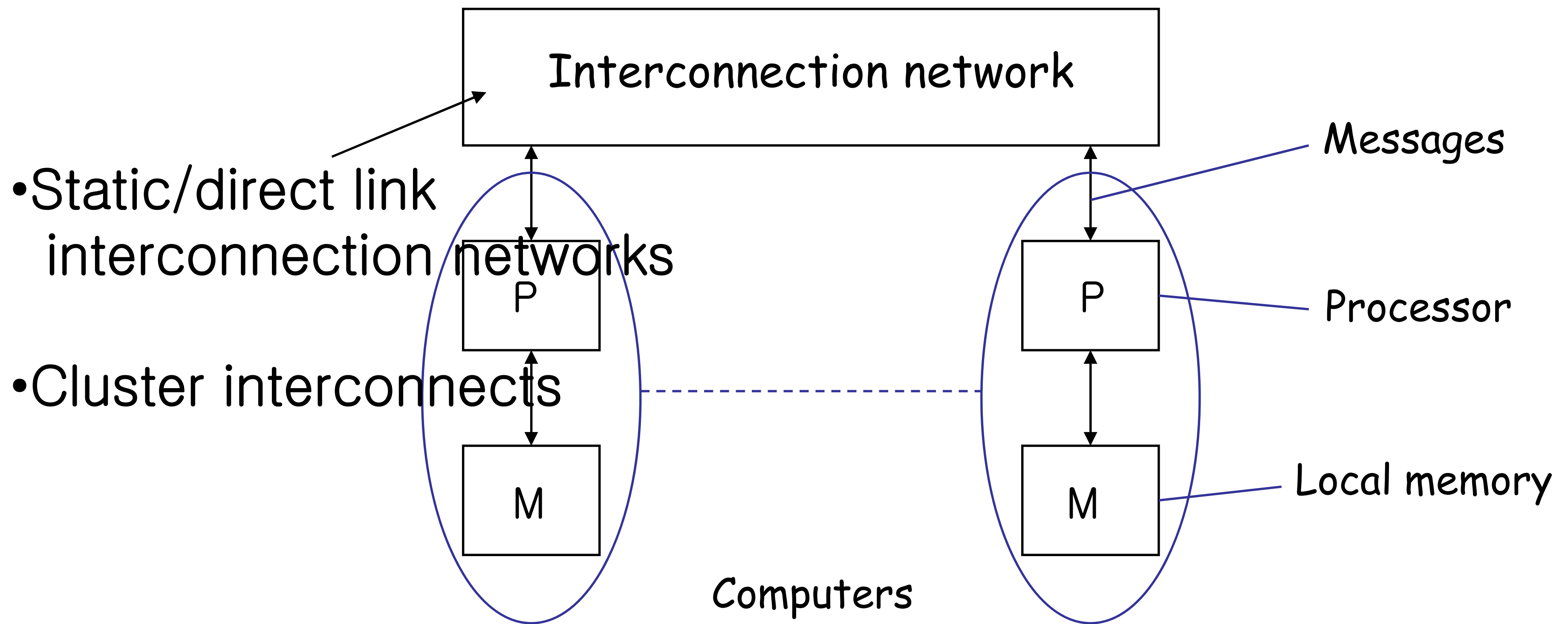
- 240 *thread processors* execute kernel threads
- 30 *multiprocessors*, each contains
  - 8 *thread processors*
  - One *double-precision* unit
  - *Shared memory* enables thread cooperation



Distributed memory multicomputers - interconnection based multiprocessor systems

Complete computers linked by some type of Interconnection network.

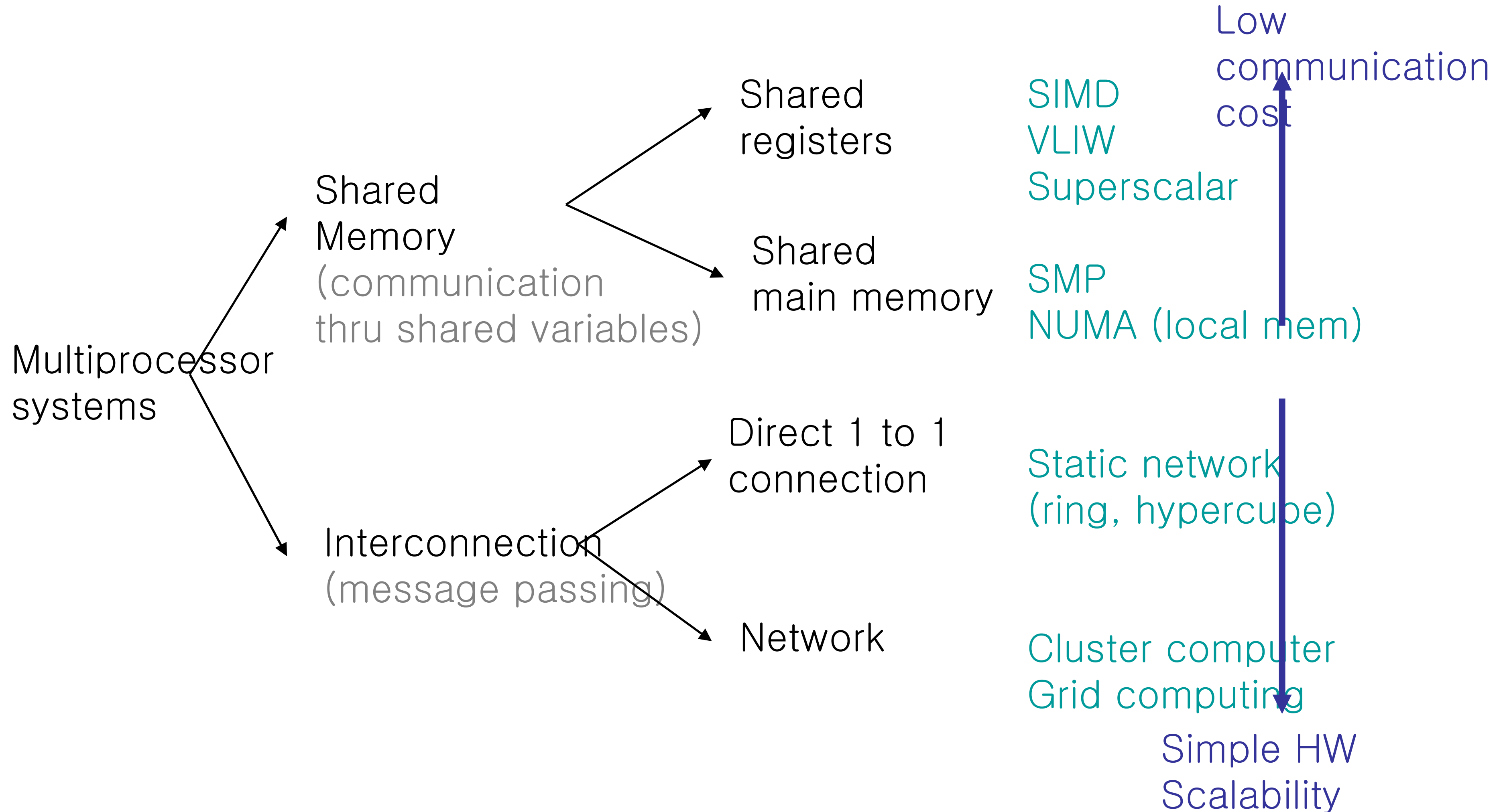
- Static/direct link interconnection networks
- Cluster interconnects



Cluster interconnects - computers connected by network (e.g. ethernet, optical)

•Static link interconnects fell out of favor during the 1990s - too expensive!
•A network of workstations (NOWs) became a very attractive alternative to the expensive supercomputers and parallel computer systems for high performance computing
•Very high performance workstations and PCs readily available at high cost and the latest processors can easily be incorporated into the system as they become available (future-proof)
•Existing software can be used or modified

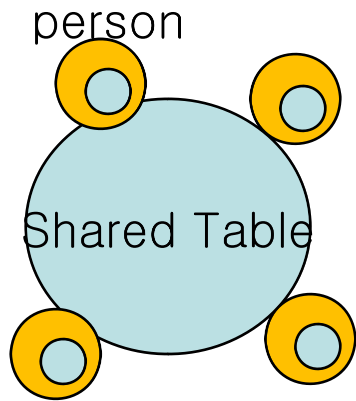
Taxonomy of multiprocessor systems according to the communication method

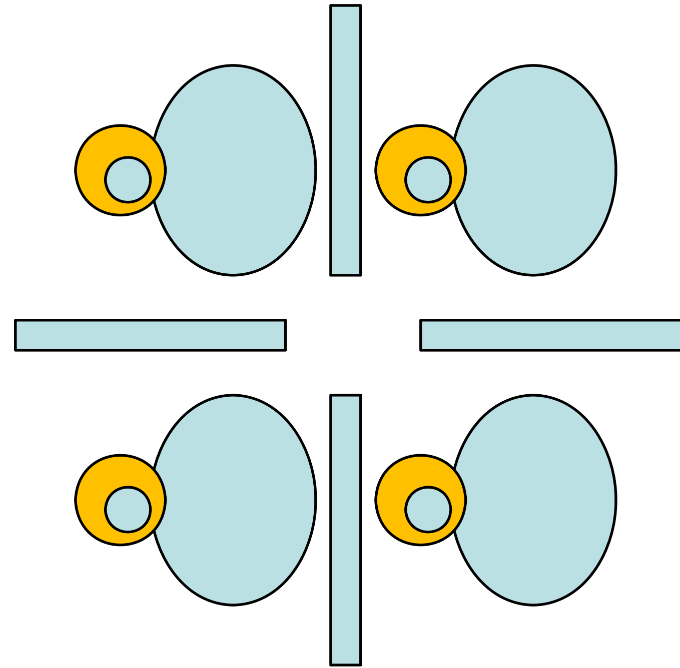


Classification: Shared or distributed memory?

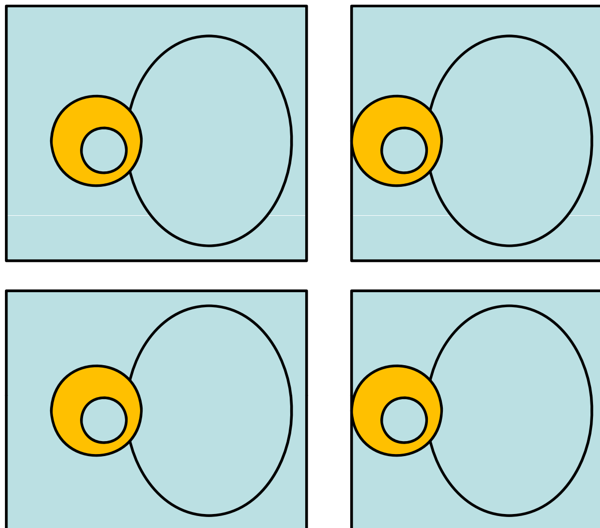
- Shared memory (or register) based multiprocessor
  - Shared (common) register based: VLIW, Superscalar, Vector or MMX architecture
  - Shared memory based: SMMP (shared memory multiprocessor)
  - Do not need explicit interprocessor communication, just synchronization and private data protection are needed.
  - Fine grain parallel processing, automatic vectorization possible
- Distributed memory multicomputer
  - Static/direct link interconnection networks, cluster interconnects
  - Need message passing for communication (higher cost communication)
  - Coarse grain parallel processing needed
  - Easily scalable, low-cost large scale implementation possible

Analogy





Shared living room



Separate house

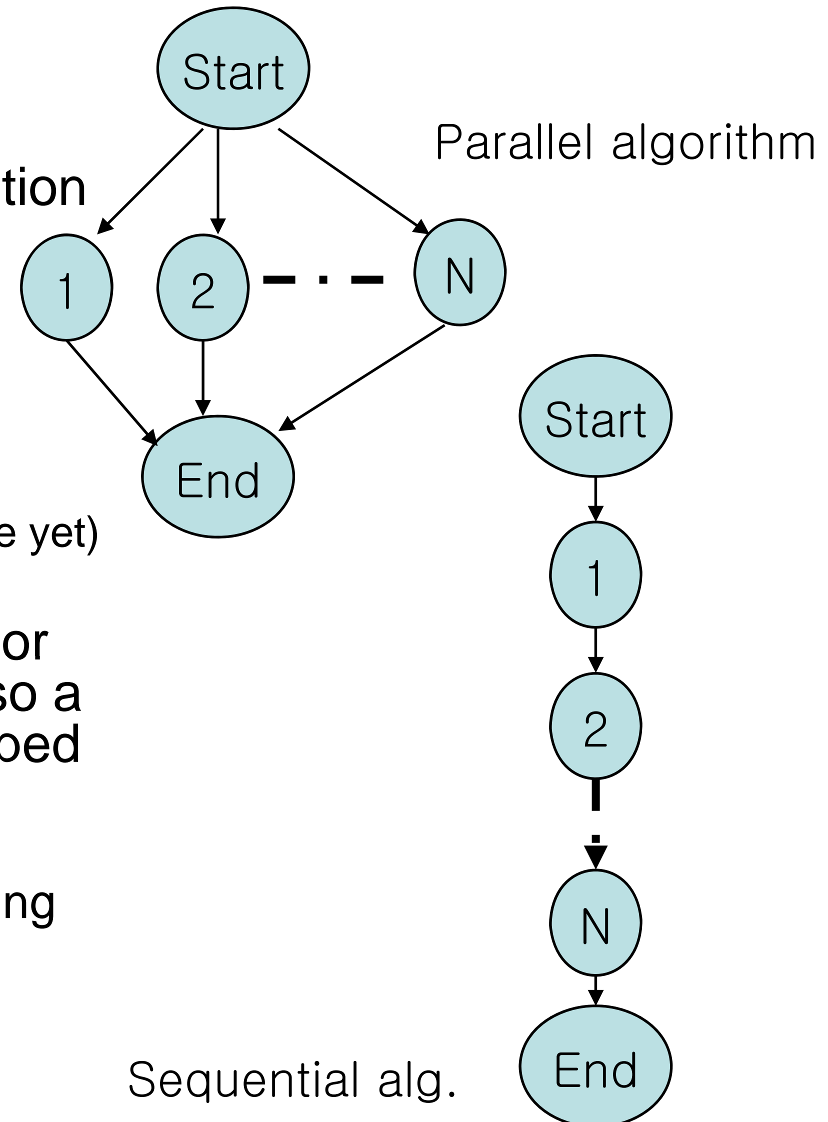
3. Programming Models and Methods

- Wish list for parallel processor programming
  - Use a conventional language (C or Fortran, not VHDL..)
  - Automatic parallelization (I do not care the number of processors, interconnection topology, memory model…). Just develop a C/C++ or Fortran program and, compile, go!
  - Efficient and scalable hardware, just replication of a widely used CPU's. If the speed is not satisfactory, just add more. (not much investment to communication network.)
- <- Nothing can satisfy all!

Partitioning and scheduling

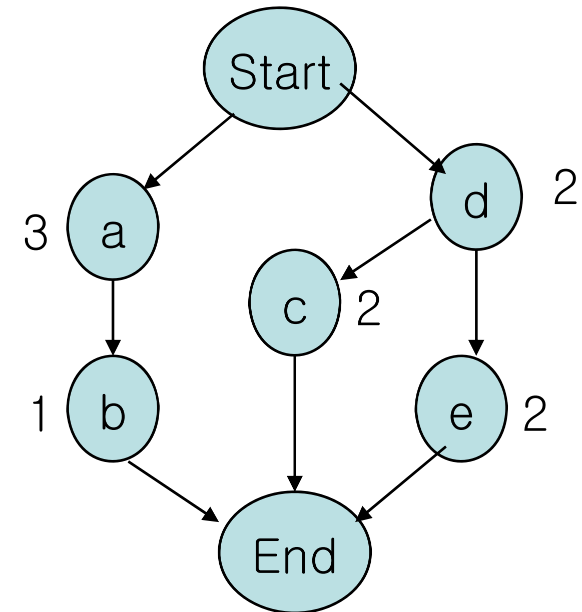
- Partitioning: how we divide a job
- Scheduling: how we order the execution of the job.

- What did we learn from pipelining?
  - Pipelining hazards
    - Structural – resource limitation
    - Data – dependency (data unavailable yet)
    - Control – conditional jump
- Same thing happens in multiprocessor scheduling (because pipelining is also a kind of parallel processing – overlapped in time)
  - We need to consider the COMMUNICATION cost for scheduling



Example of scheduling and allocation

- Communication cost affects the optimal scheduling results!
- When the communication cost is high, we need to combine operations.



With no comm cost

	1	2	3	4	5
P1	a			c	
P2	d		e		b

With comm cost =2

	1	2	3	4	5	6	7
P1	a					c	
P2	d		e				b

With comm cost =2

	1	2	3	4	5	6	7
P1	a			b			
P2	d		e		c		

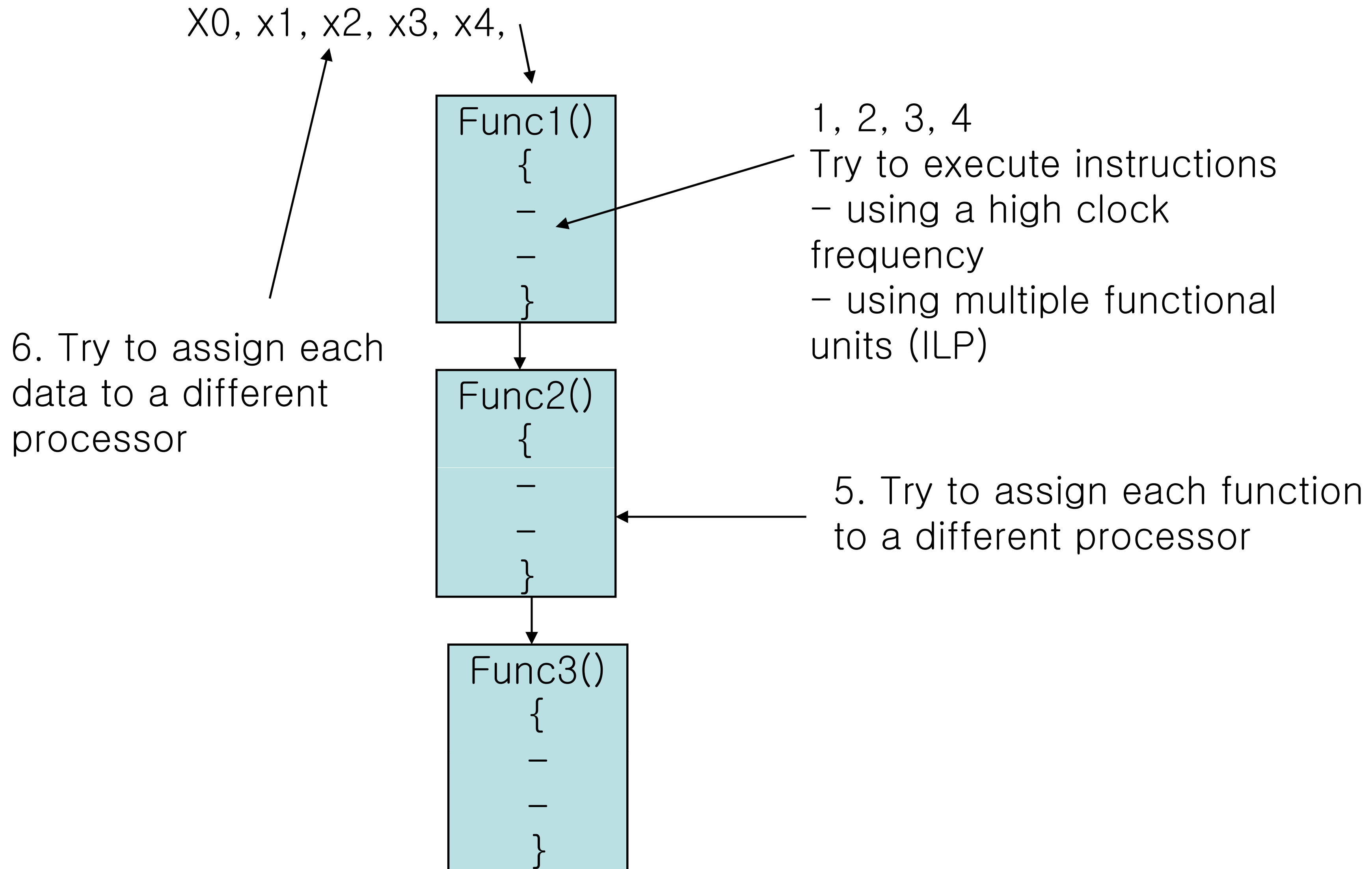
* Bigger operation size means less communication needed, but not good for job balancing.

How to do something faster?

- Goal of high performance computing: usually do it faster!

1. Use more efficient instructions (better ISA). This may need a special purpose data-path.
    1. The idea of CISC. Configurable instruction set.
    2. Programmable digital signal processors (MAC, search unit, ..)
2. Execute instructions at a faster speed.
    1. RISC processors. Pipelining and high bandwidth memory (registers, cache).
3. Execute multiple data with one instruction (at a time).
    1. The idea of SIMD (or MMX).
4. Execute multiple instructions at one clock cycle
    1. The idea of superscalar, VLIW
    2. Need to explore the ILP (instruction level parallelism)
5. Divide a whole program into some small parts, and assign them to different processors
    1. Coarse grain control level parallel processing. (Shared memory multiprocessor?)
6. Divide a large set of data into some small parts, and assign them to different processors
    1. Coarse grain data level parallel processing. (Message passing multiprocessor?)

Speed-up method



Fine grain (local) parallel processing

- Extract parallelism from only a small part of an entire program and maps the instructions to different execution units.
  - Grain size: instruction
  - Scope: basic block, function, or loop
    - Superscalar and VLIW, vector processors (loop)
    - Superscalar conducts execution time scheduling for code compatibility.
  - Disadv.: The architecture should support immediate communication among the execution units (shared registers or direct interconnection)
  - Adv.: Good for automatic parallelization. Not much need of profiling for load balancing purpose.
      - Vector Fortan: automatic vectorization of a loop
      - VLIW: software pipeline optimizer

Coarse grain parallel processing

- Divides a program (or data) into big parts as long as the load is fairly evenly divided. If the load balancing is not good, it needs to be divided finer.
  - Grain size: function, sometimes loop, a block of data
  - Scope: entire program or multiple-jobs
    - Shared memory multiprocessor systems, interconnection based multiprocessor systems
    - Partitioning may not be automatic, it may need application or program specific knowledge.
    - Needs profiling for job partitioning and load-balancing
  - Good for systems with high communication cost
    - As the grain size increases, the total number of communication decreases, but the load-balancing can be poorer.

What should be considered for coarse grain parallel processing?

- Load balancing
  - Distribute a job into the processors as evenly as possible.
  - The processor with the heaviest load may determine the overall processing time.
- Communication overhead
  - There is some cost for inter-processor communication. So, it is necessary to divide a program into bigger parts so that there are not much communication operations.
- Dependency relation (precedence relation)
  - The job for concurrent processing should not have dependency relation. This complicates the parallel programming.

Control level parallelization

- Divides a program intro smaller pieces and assign them into different processors
  - Grain size: instruction – fine grain
  - Grain size: function, thread level– coarse grain
- Procedure
  - Divides a program into small pieces
  - Assign each piece to an available processor
    - Should consider the precedence relation
    - We need “synchronization”

Data-level parallel processing

- Divides a big data into small parts and assign them to each processor
  - Load balancing is easier
  - Each processor may have all the needed programs.
    - But may need less data memory space for each processor.
  - Example:
    - Mesh based modeling for weather forecast, oil well modeling
      - Each processor conducts computing for only a part of the grids. 10,000 grids for 100 processor -> about 100 grids for each processor
    - Block based processing of DSP algorithms
      - Round-robin scheduling method.
      - Divides an input data into blocks, and 1st block assigned to the 1st processor, 2nd block to the 2nd processor, and so on.

	Fine grain	Coarse grain
Control	Superscalar, VLIW	Shared memory multiprocessor
Data	Vector processor SIMD	Interconnection Based multiproc. Good job balancing

Fine grain parallel processing: vector (array) code parallelization

- Most scientific programs contain computation intensive loops
  - Loop execution takes much portion in scientific computing
  - Use deeply pipelined (high clock freq) or array type data-path (do multiple data execution in one cycle)
  - SIMD architecture based. Pipelined vector processors, partitioned data-path (MMX) architecture, sometimes SMP can use vectorization
  - Easy to vectorize (dependency check and go!)
  - The performance (speed-up) is dependent on the portion of loop and scalar codes. Loop with dependency cannot be vectorized (it's similar to the RAW hazards in the pipelined architecture).
  -

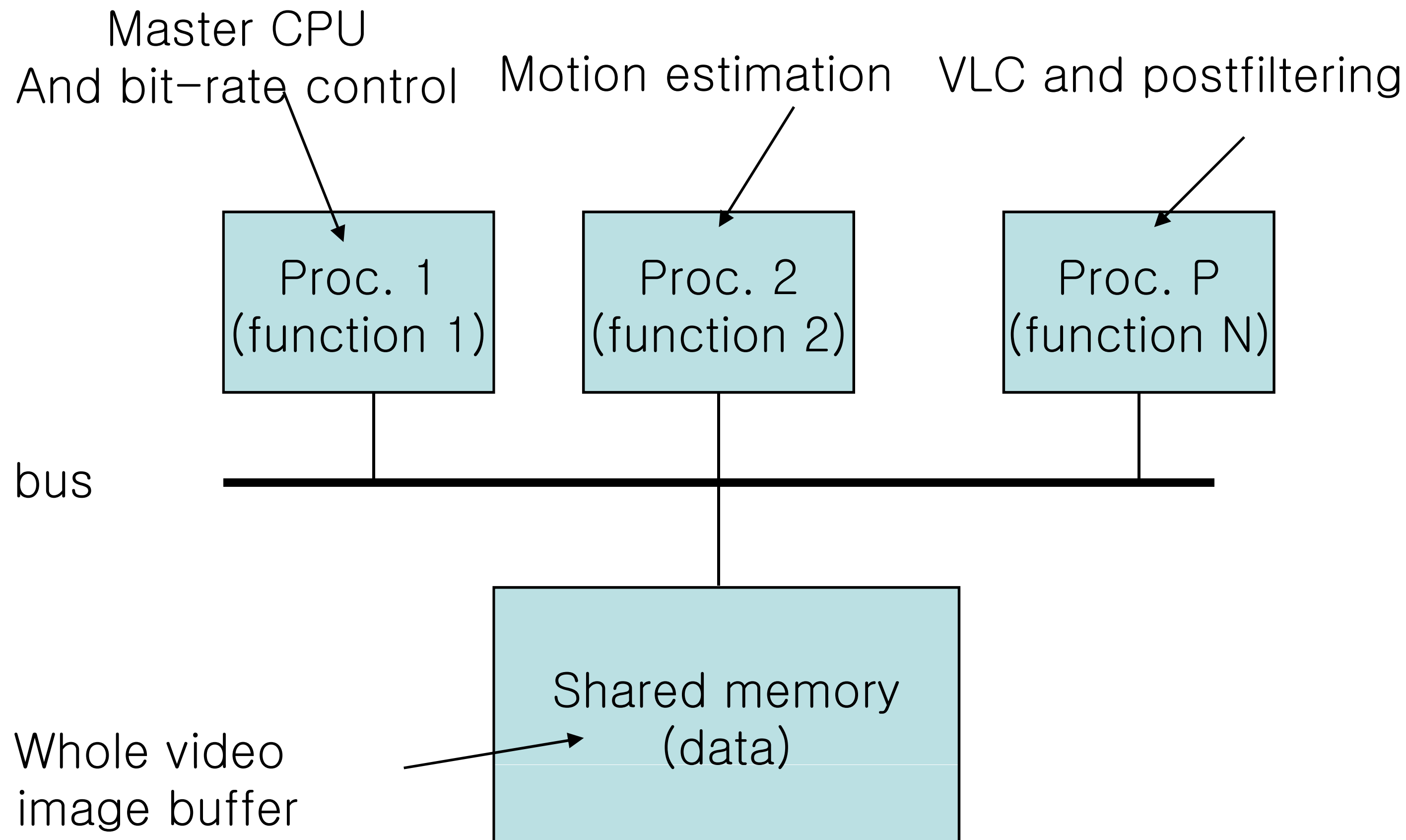
```
C  Vectorizable loop
   DO 10 I = 1, 100
10 C(I) = 0.7*A(I) + B(I)
```

```
C   Loop with dependency
     DO 10 I = 1, 100
10   A(I) = 0.7*A(I-1) + B(I)
```

Fine grain parallel processing: scalar code parallelization

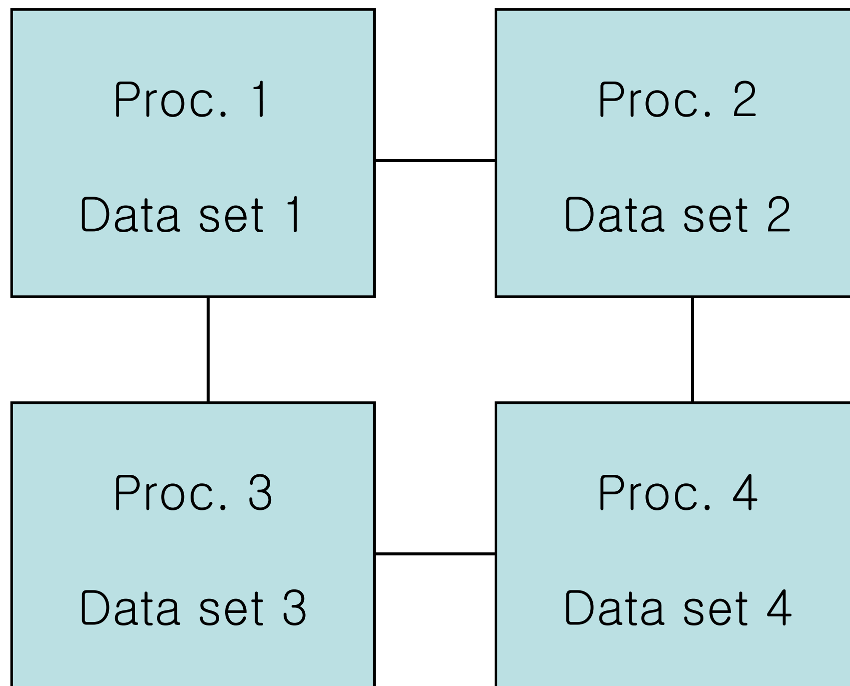
- Ordinary control intensive programs consist of small basic blocks
  - It's possible to execute multiple instructions at a time (especially when they are in the same basic block).
  - Some times code sequence may be changed.
  - Superscalar (on-line hardware based scheduling) and VLIW (off-line software based scheduling)
  - Software pipelining technique for loop part
  - The maximum achievable degree of parallelism is not high (a few for superscalar model) in general.

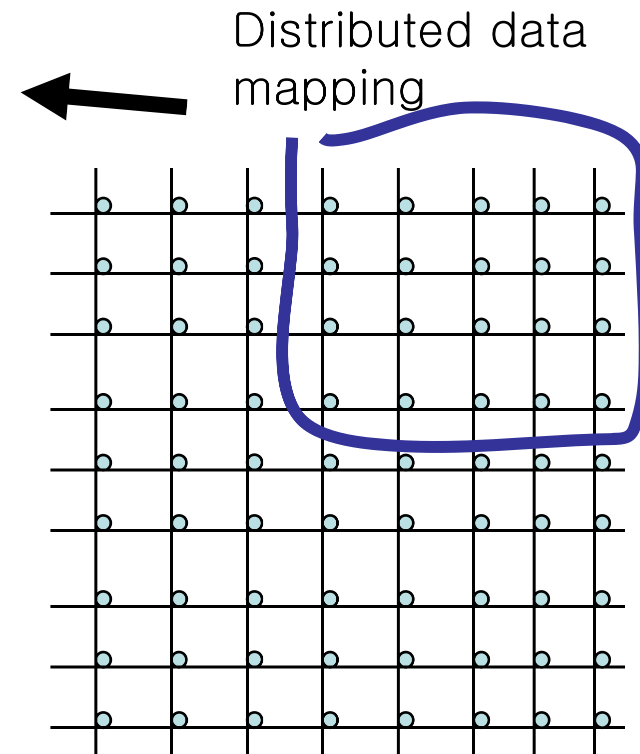
Coarse grain processing: shared data (control level parallel processing) model



Coarse grain parallel processing: distributed data (data-level parallel processing) model

Example: weather forecast
finite element analysis





Parallel programming tools

- Vectorizer, Simdizer: conducts (automatic) parallelization on loop kernel.
- VLIW compiler: conducts instruction level parallel processing using VLIW scheduling algorithm
- OpenMP: The Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures.
  - Consists of directives for job synchronization. Begin with master thread.
  - “parallel” directive creates a team of threads and specifies a block of code that will be executed by the multiple threads in parallel.
- MPI (Message Passing Interface): Message passing library (usually) for SPMD (Single Program Multiple Data) computation model.
  - MPI_Send(..), MPI_Recv(..), MPI_Bcast(), …
- CUDA

4. Conclusion

- New generation of embedded processors will be based on parallel processing architecture
  - Advanced semiconductor fabrication technology (enough silicon area) and low-power requirements
  - Better development tools (harder than silicon duplication)
  - Application requirements of software based high performance computing (H.264, software defined radio, DVR, ..)
- Good application development tools are essential!
  - New device does not mean just better processing speed
  - New program development methods are needed.
    - Success example: VLIW based, partitioned datapath
    - Failed example: C4x, C8x
  - Program development methods or tools are 10 years behind the architecture or hardware. We can spy the tool trend by studying the current and past supercomputers.
  - OpenMP for shared memory multiprocessors and MPI for interconnection based multicomputer would be a good candidate.

Course outline

- DSP system design using embedded devices (track1)
  - Floating-point to integer conversion
  - RISC CPU based embedded system design
  - Traditional programmable DSP
  - SIMD/VLWI DSP (C6x)
  - SIMDizer, VLIW compiler & scheduler, Matlab to C
  - Application to emerging embedded devices
  - Important concepts: fine-grain parallel processing, hazards(data, control, structural), demand ratio, scheduling, ILP, loop-carried dependency, …
- High-performance signal processing system design (track2)
  - Vector and SIMD processors (Cray, Intel SSE, ARM11)
  - Shared memory multiprocessor (cache coherency, bus arbitration, load balancing)
  - Interconnection based multicomputers (interconnection and performance, parallel matrix computation)
  - OpenMP, MPI, CUDA
  - Parallel algorithm development - parallel matrix computation, FFT, digital filter, multimedia algorithm programming
  - Important concepts: coarse-grain parallel processing, load balancing, communication overhead, synchronization, fork-join model, working-set minimization

Text and …

- Text
  - Parallel Programming in C with MPI and OpenMP (Michael J. Quinn)
  - Computer architecture: a quantitative approach.

- Several small homework and tiny program exercises (using VLIW, OpenMP, MPI): 20%
- Each student will have a chance (duty) of 20 minutes seminar presentation with a relevant topic (after reading a few papers or based on new embedded CPU): 10%
- One quiz
  - One around in the middle of September 27th (Ch. 3, 4 of CAQ book): 10%
- One test (mid-term period before or Nov. 1): 40%
- Final term project: 30% (report 20%, presentation 10%)

- For other lab students: you can choose lab either track1 or track2

Embedded processors today and tomorrow

- Parallel processing architecture is definitely advantageous for high performance applications in terms of power and cost.
  - Video encoding and decoding (H.264), multi-channel voice, communication (4G, software radio)
- But, the hurdle is
  - Development environment
    - C8x, multiprocessor with switched local memories failed!
    - New chips with reasonably good programming supports are emerging
      - C6x: VLIW
      - Cradle, Picochips, Sandbridge technology, …
    - Need to learn from previous generation PC and supercomputers