2018年度（平成30年度）版

Ver. 2018-11-14a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

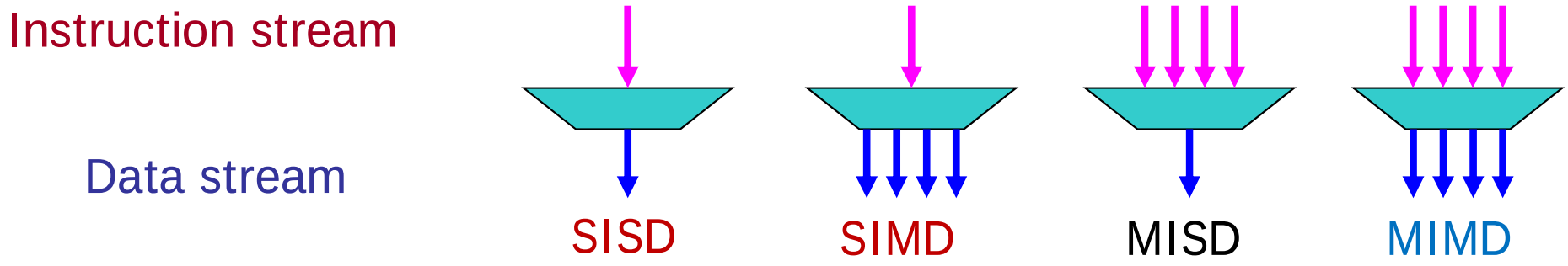
13. ベクタ、SIMDにおけるデータレベル並列性
Data-Level Parallelism in Vector and SIMD

www.arch.cs.titech.ac.jp/lecture/CA/
Room No.W321
Tue 13:20-16:20, Fri 13:20-14:50

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

アーキテクチャの異なる視点による分類

- Flynnによる命令とデータの流れに注目した並列計算機の分類(1966年)
  - **SISD (Single Instruction stream, Single Data stream)**
  - **SIMD (Single Instruction stream, Multiple Data stream)**
  - MISD (Multiple Instruction stream, Single Data stream)
  - **MIMD (Multiple Instruction stream, Multiple Data stream)**



SIMD Variants

- Vector architectures
- SIMD extensions
- Graphics Processing Units (GPUs)

- SIMD variants exploit data-level parallelism
- Instruction-level parallelism in superscalar processors
- Thread-level parallelism in multicore processors

Vector architecture

- Computers designed by Seymour Cray starting in the 1970s
- Basic idea:
  - Read sets of data elements into "vector registers"
  - Operate on those registers
  - Disperse the results back into memory

Cray Supercomputer

DAXPY in MIPS Instructions

Example: DAXPY (double precision a x X + Y)

```
        L.D       F0,a          ; load scalar a
        DADDIU    R4,Rx,#512    ; upper bound of what to load
Loop:   L.D       F2,0(Rx    )  ; load X[i]
        MUL.D     F2,F2,F0      ; a x X[i]
        L.D       F4,0(Ry)      ; load Y[i]
        ADD.D     F4,F2,F2      ; a x X[i] + Y[i]
        S.D       F4,9(Ry)      ; store into Y[i]
        DADDIU    Rx,Rx,#8      ; increment index to X
        DADDIU    Ry,Ry,#8      ; increment index to Y
        SUBBU     R20,R4,Rx     ; compute bound
        BNEZ      R20,Loop      ; check if done
```

- Requires almost 600 MIPS operations

DAXPY in VMIPS (MIPS with Vector) Instructions

- ADDV.D : add two vectors
- ADDVS.D : add vector to a scalar
- LV/SV : vector load and vector store from address

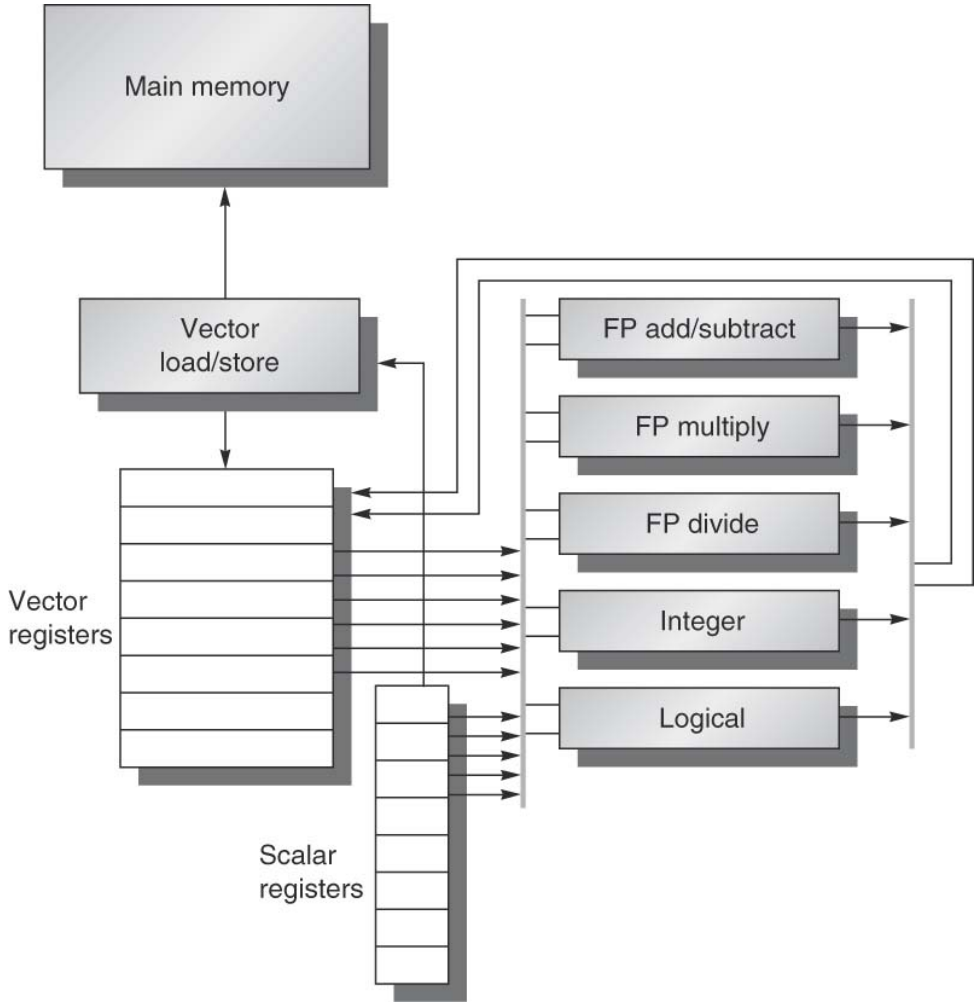
- Example: DAXPY (double precision a*X+Y)

```
L.D      F0,a      ; load scalar a
LV       V1,Rx     ; load vector X
MULVS.D  V2,V1,F0  ; vector-scalar multiply
LV       V3,Ry     ; load vector Y
ADDV.D   V4,V2,V3  ; add
SV       Ry,V4     ; store the result
```

- Requires 6 instructions

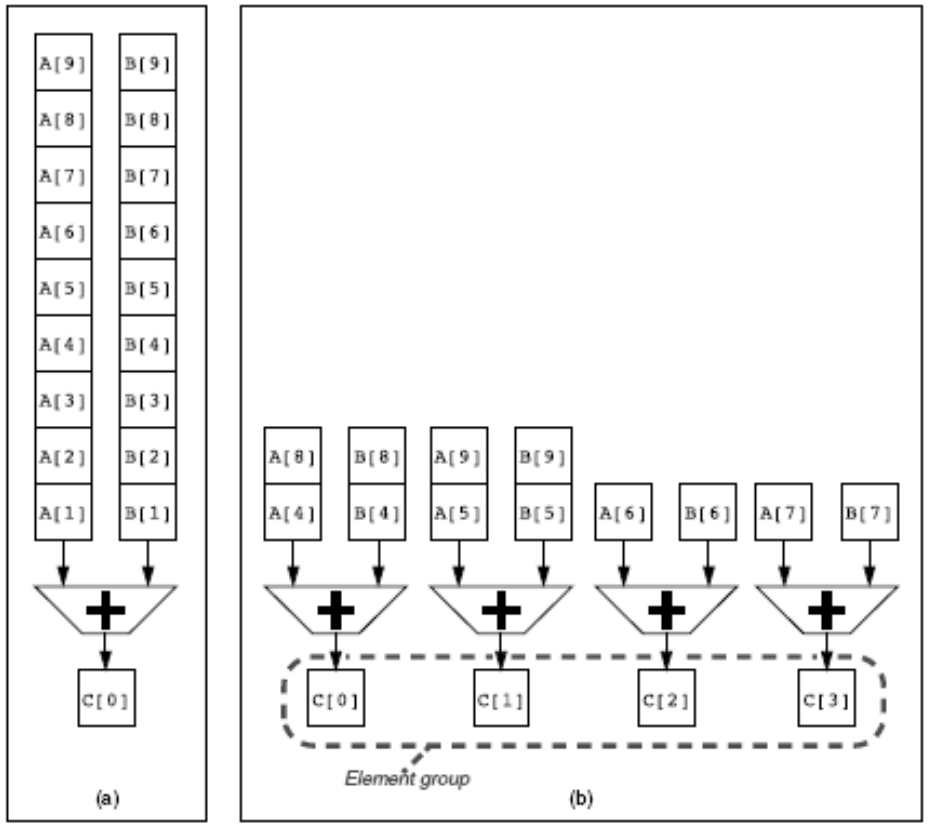
The basic structure of a vector architecture, VMIPS

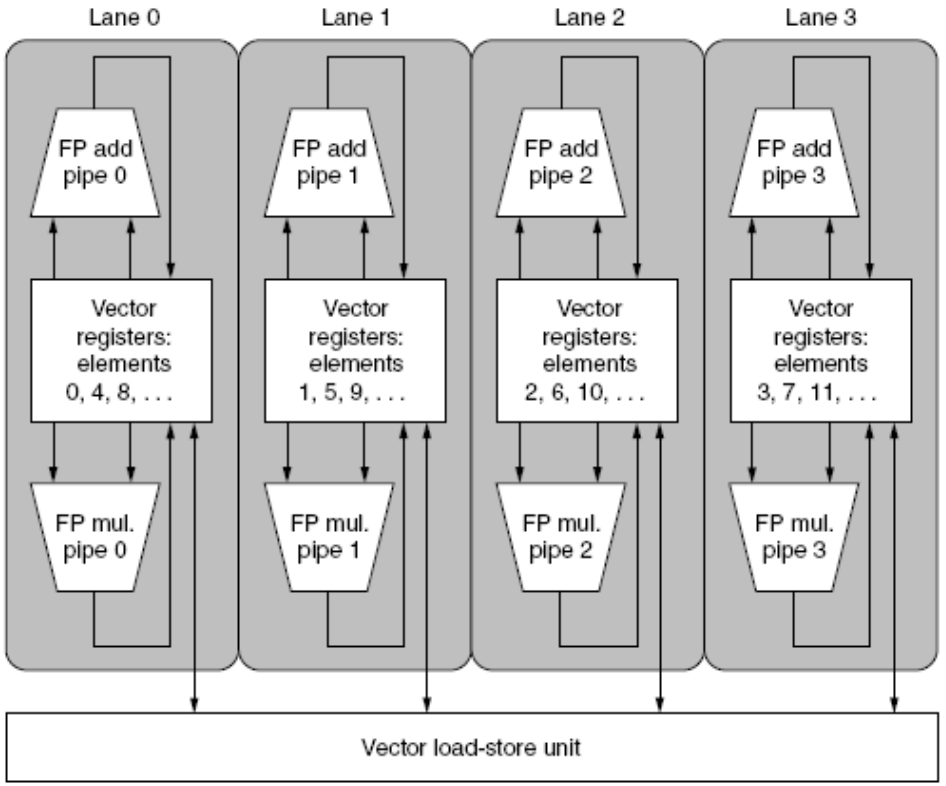
- Eight 64-element vector registers
- All the functional units are vector functional units.



Multiple functional units to improve the performance

- (a) can complete one addition per cycle
- (b) can complete four addition per cycle
- The vector register storage is divided across the lanes





SIMD extensions

- Media applications operate on data types narrower than the native word size
  - Example: disconnect carry chains to "partition" adder
- Implementations:
  - Intel MMX (1996)
    - Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Eight 16-bit integer ops
    - Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector Extensions (AVX 2010)
    - Four 64-bit integer/fp ops
    - 256 bit vectors -> 512 -> 1024
  - Operands must be consecutive and aligned memory locations

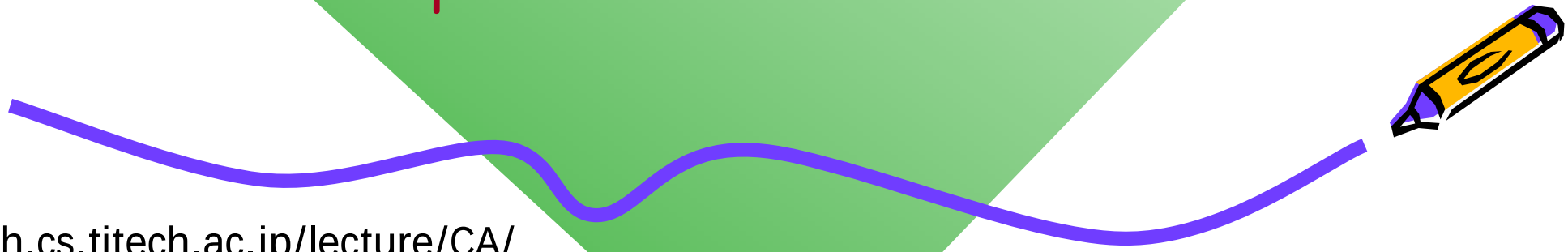
2018年度（平成30年度）版

Ver. 2018-11-14a

Course number: CSC.T363

コンピュータアーキテクチャ
Computer Architecture

14. マルチプロセッサ、マルチコア
Multiprocessors and Multicore

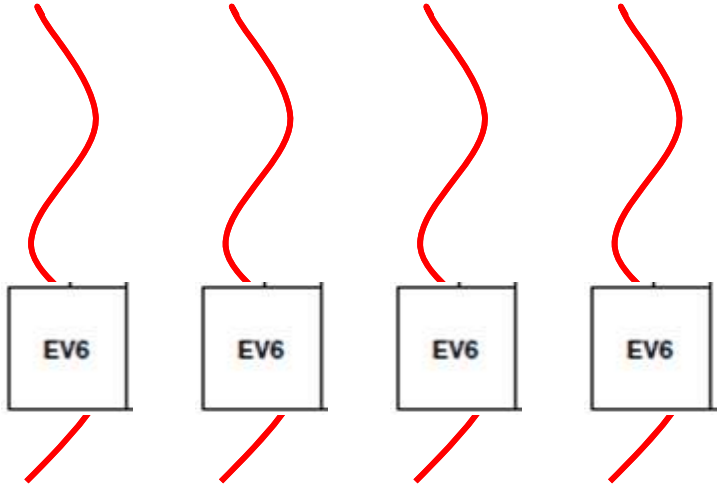


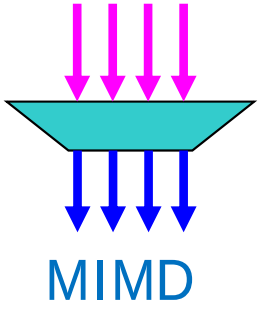
www.arch.cs.titech.ac.jp/lecture/CA/
Room No.W321
Tue 13:20-16:20, Fri 13:20-14:50

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

アーキテクチャの異なる視点による分類

- Flynnによる命令とデータの流れに注目した並列計算機の分類(1966年)
  - **SISD (Single Instruction stream, Single Data stream)**
  - **SIMD (Single Instruction stream, Multiple Data stream)**
  - MISD (Multiple Instruction stream, Single Data stream)
  - **MIMD (Multiple Instruction stream, Multiple Data stream)**





Multi-threading

- Multi-threading processor
- Simultaneous multi-threading (SMT) processor

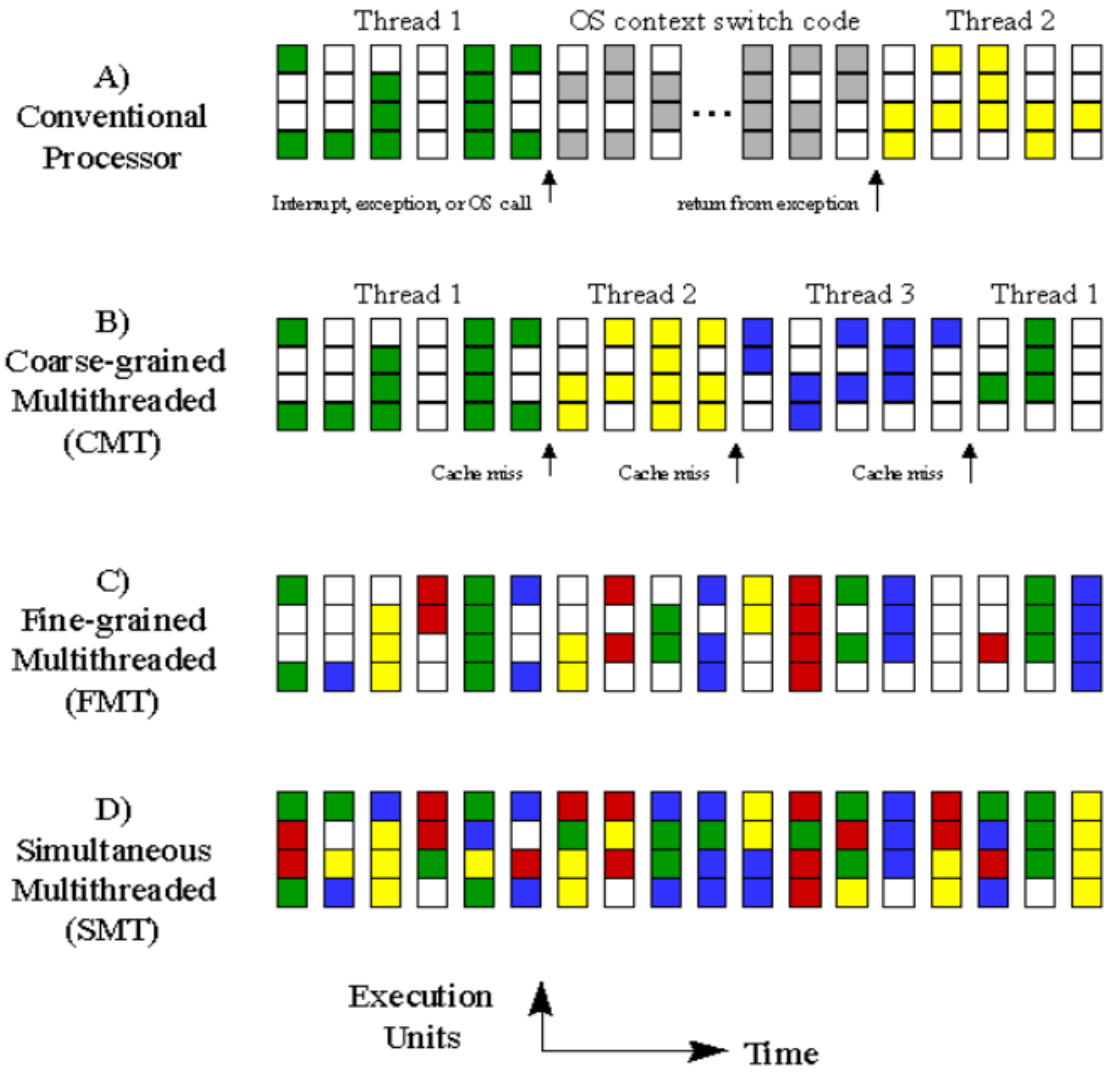
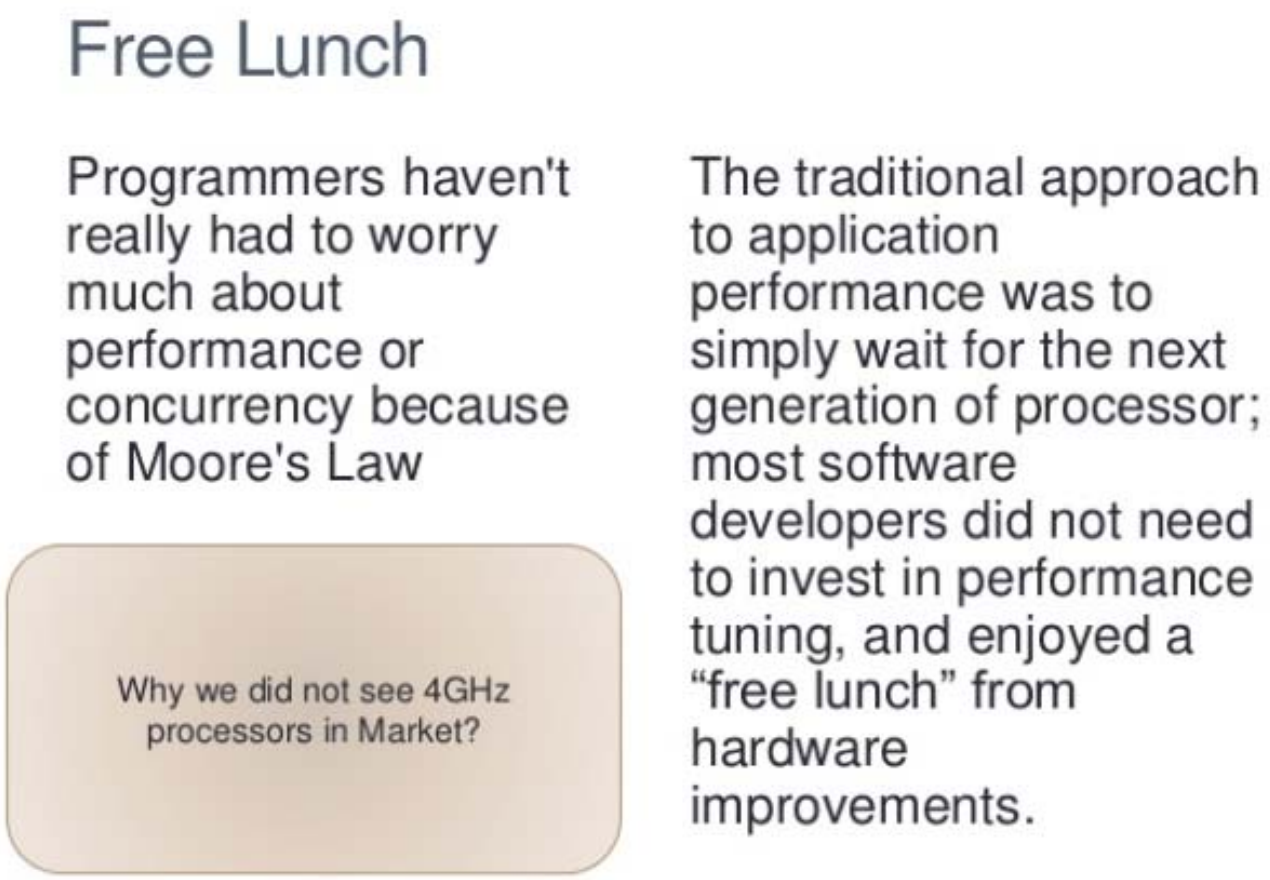


Figure 1. Multithreaded Execution with Increasing Levels of TLP Hardware Support

The Free Lunch Is Over

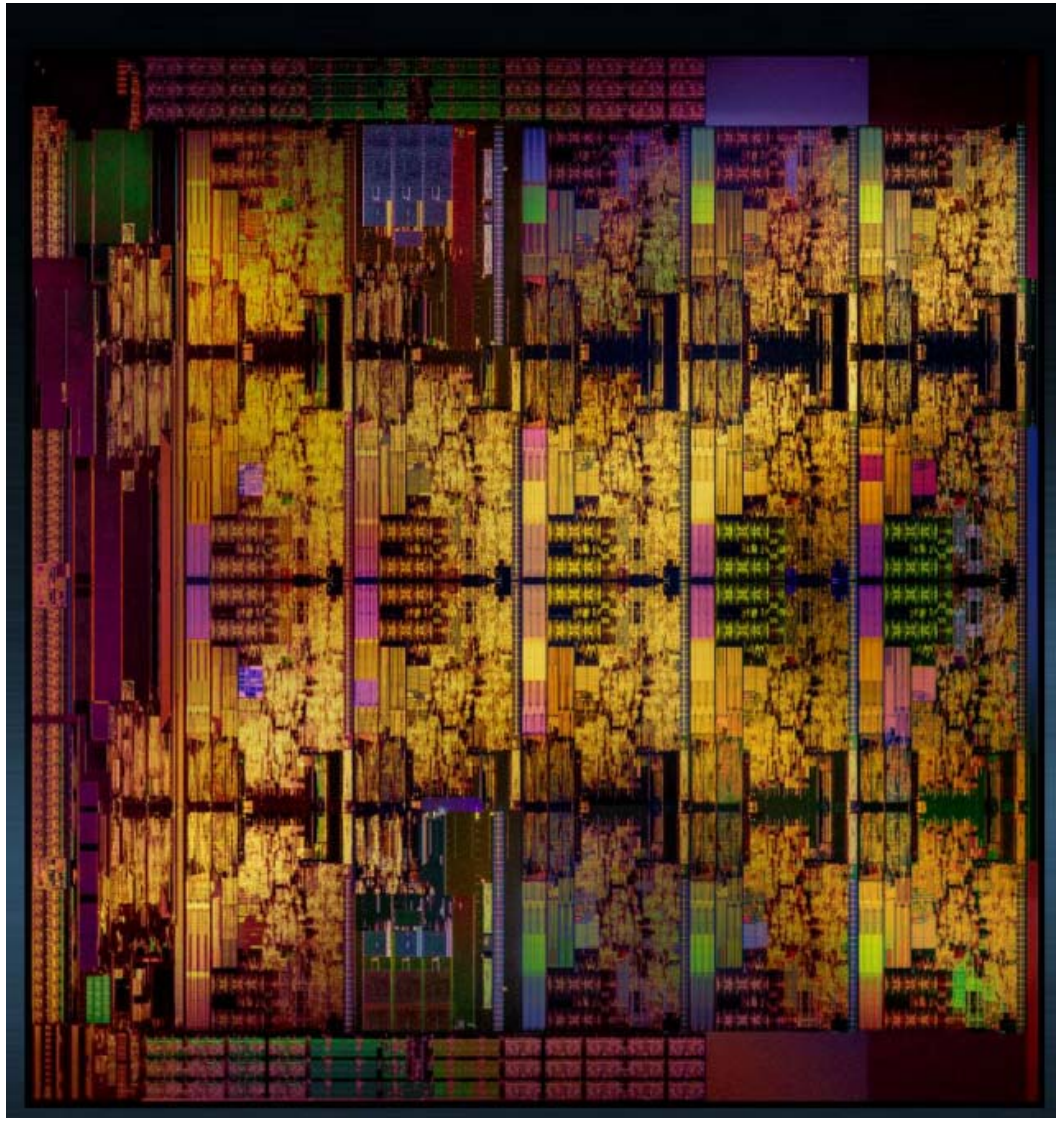
- Tuning, Optimization, and Parallel processing (Concurrency)



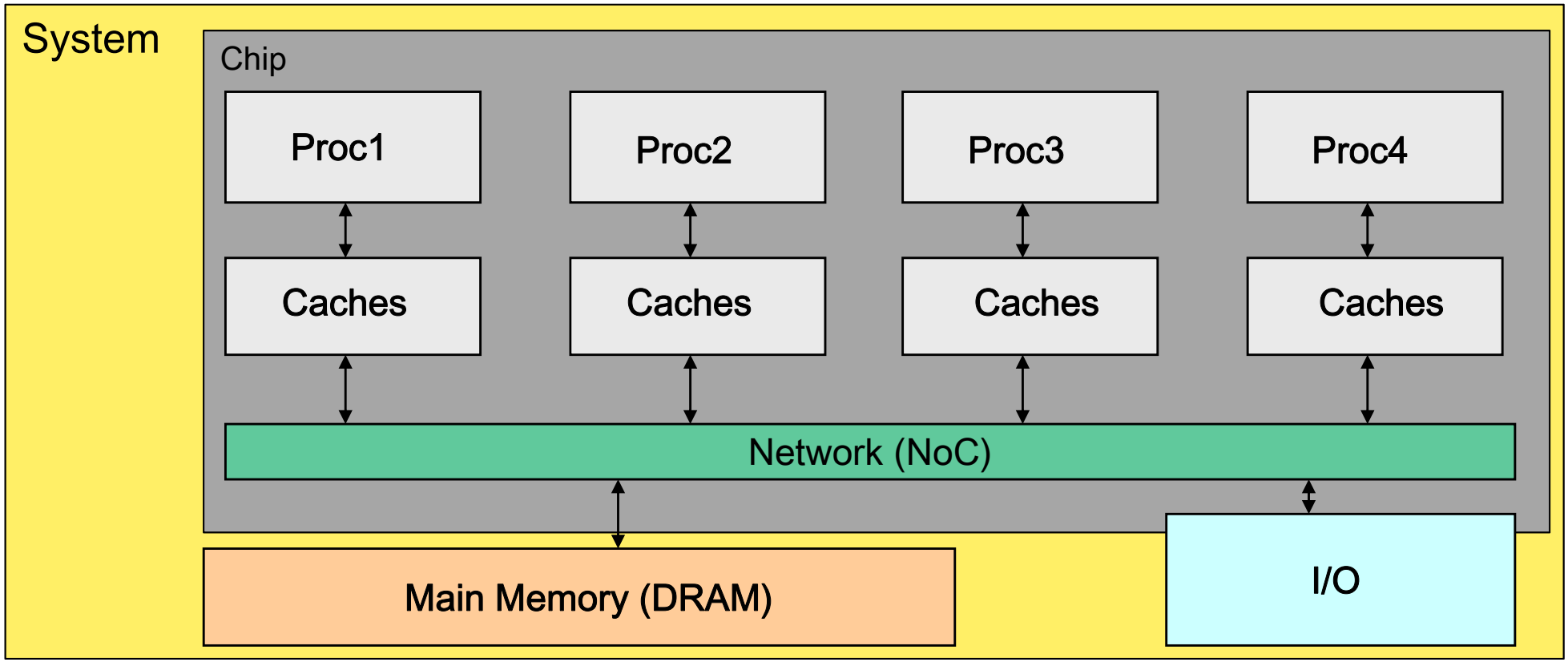
The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software by Herb Sutter, 2005

Intel Skylake-X, Core i9-7980XE, 2017

- 18 core



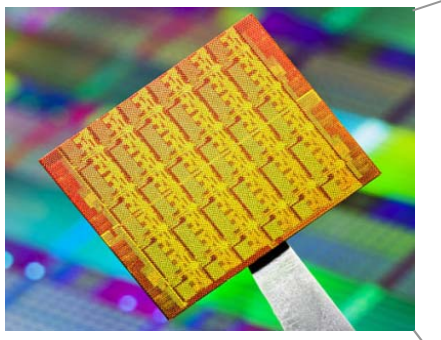
Multicore, Shared Memory System

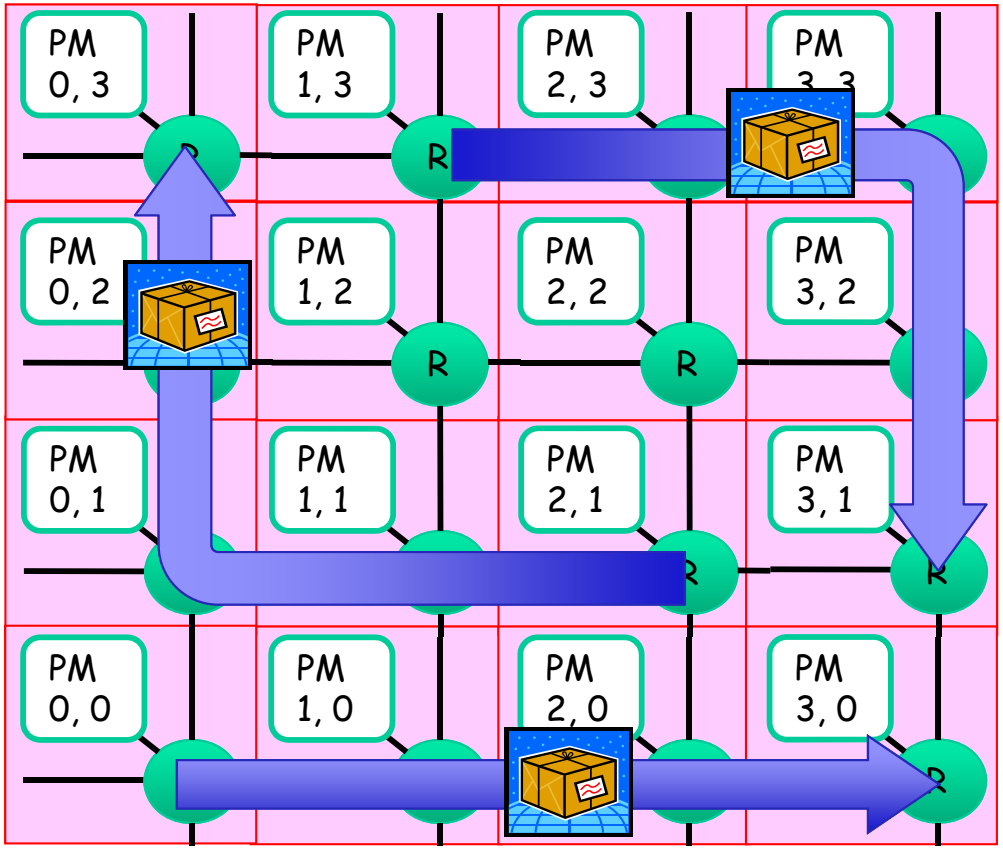


- Caches are used to reduce latency and to lower network traffic
- Must provide hardware to ensure that caches and memory are **consistent** (cache coherency)
- Must provide a hardware mechanism to support process (thread) synchronization

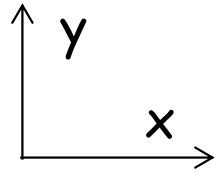
NoC and Many-core

- NoC requirements: low latency, high throughput, low cost
  - Focus on mesh topology
- Packet based data transmission via NoC routers and XY-dimension order routing



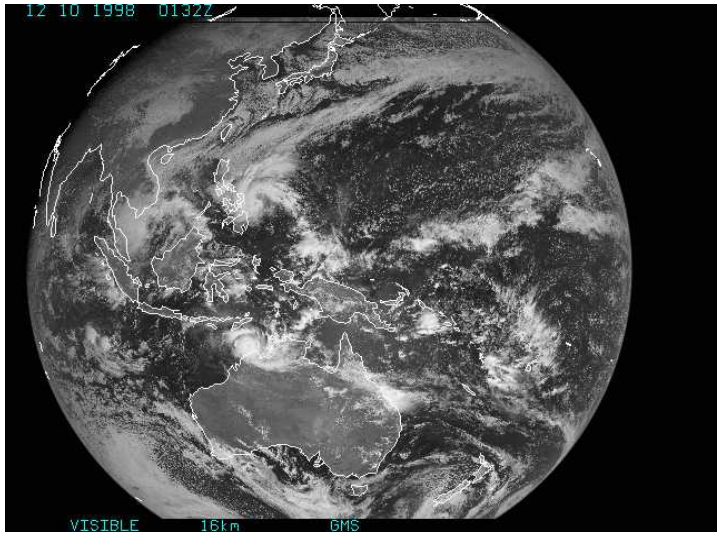


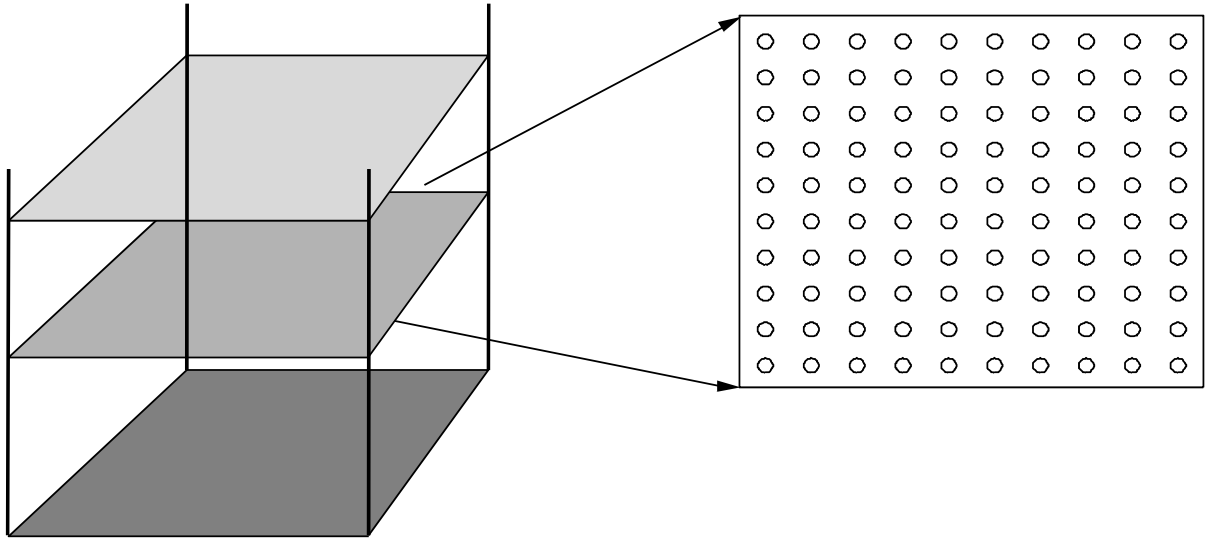
packet (tag + data)



PM: Processing Module or Core,
R: Router

Simulating Ocean Currents



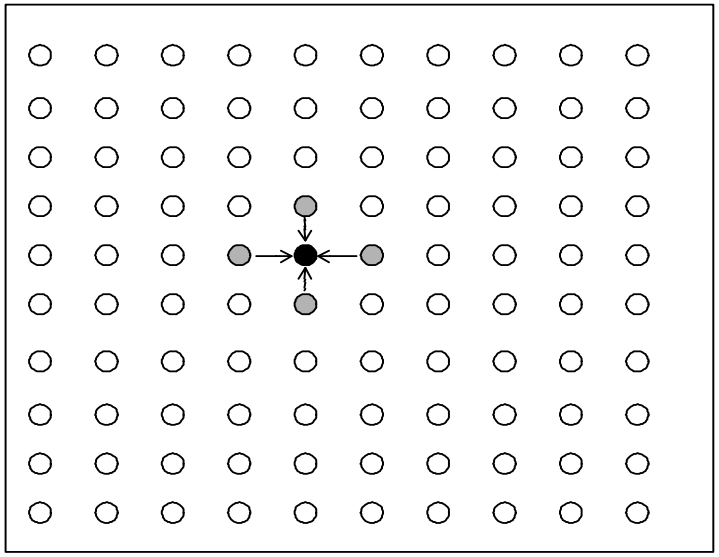


(a) Cross sections

(b) Spatial discretization of a cross section

- Model as two-dimensional grids
  - Discretize in space and time
  - finer spatial and temporal resolution => greater accuracy
- Many different computations per time step
  - Concurrency across and within grid computations
- Static and regular

Grid Solver



Expression for updating each interior point:

$$A[i,j] = 0.2 \times (A[i,j] + A[i,j-1] + A[i-1,j] + A[i,j+1] + A[i+1,j])$$

- Gauss-Seidel (near-neighbor) sweeps to convergence
  - interior n-by-n points of (n+2)-by-(n+2) updated in each sweep
  - updates done in-place in grid
  - difference from previous value computed
  - accumulate partial diffs into global diff at end of every sweep
  - check if it has converged to within a tolerance parameter

Sequential Version

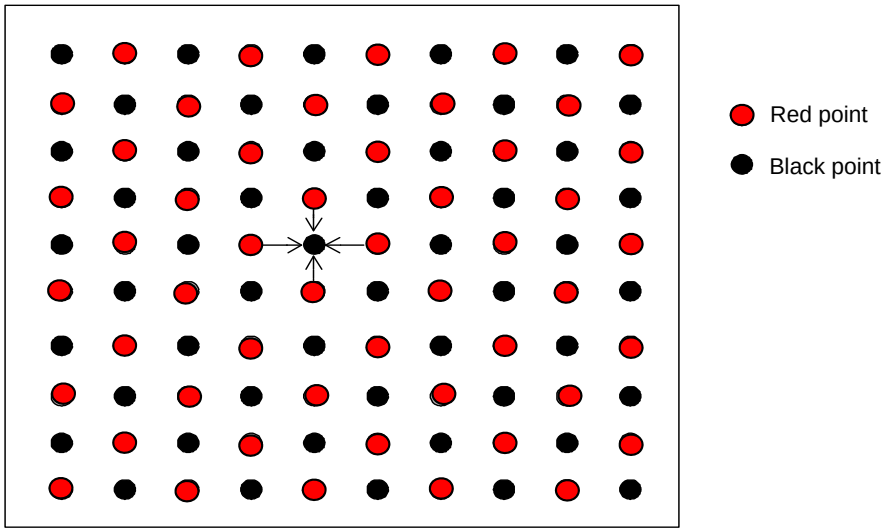
```
1.  int n;                                  /*size of matrix: (n + 2-by-n + 2) elements*/
2.  float **A, diff = 0;

3.  main()
4.  begin
5.     read(n) ;                            /*read input parameter: matrix size*/
6.     A ← malloc (a 2-d array of size n + 2 by n + 2 doubles);
7.     initialize(A);                       /*initialize the matrix A somehow*/
8.     Solve (A);                           /*call the routine to solve equation*/
9.  end main

10. procedure Solve (A)                     /*solve the equation system*/
11.    float **A;                           /*A is an (n + 2)-by-(n + 2) array*/
12. begin
13.    int i, j, done = 0;
14.    float diff = 0, temp;
15.    while (!done) do                     /*outermost loop over sweeps*/
16.       diff = 0;                         /*initialize maximum difference to 0*/
17.       for i ← 1 to n do                 /*sweep over nonborder points of grid*/
18.          for j ← 1 to n do
19.             temp = A[i,j];             /*save old value of element*/
20.             A[i,j] ← 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
21.                A[i,j+1] + A[i+1,j]);   /*compute average*/
22.             diff += abs(A[i,j] - temp);
23.          end for
24.       end for
25.       if (diff/(n*n) < TOL) then done = 1;
26.    end while
27. end procedure
```

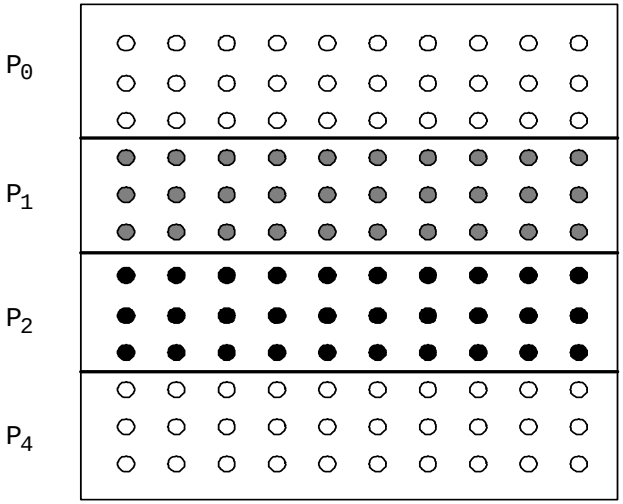
Exploit Application Knowledge

- Reorder grid traversal: red-black ordering



  - Different ordering of updates: may converge quicker or slower
  - Red sweep and black sweep are each fully parallel:
  - Global synch between them (conservative but convenient)
  - Ocean uses red-black
  - We use simpler, asynchronous one to illustrate
    - no red-black, simply ignore dependences within sweep
    - parallel program *nondeterministic*

Assignment

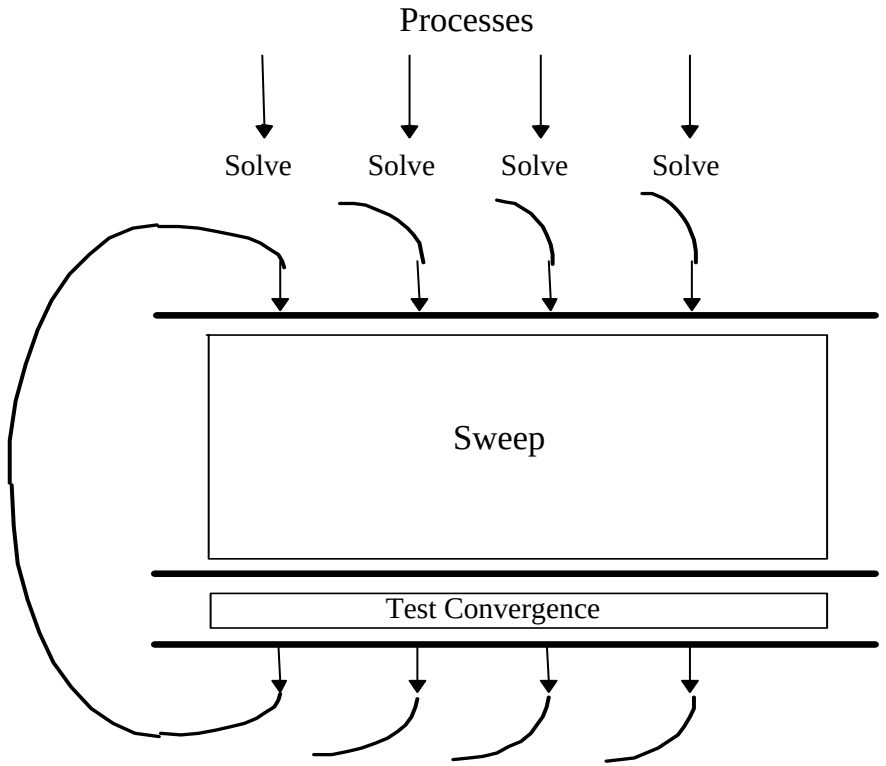


block assignment

- Static assignment: decomposition into rows
  - block assignment of rows: Row *i* is assigned to process $\left\lfloor \frac{i}{p} \right\rfloor$
  - cyclic assignment of rows: process *i* is assigned rows *i*, *i+p*, ...
- Dynamic assignment
  - get a row index, work on the row, get a new row, ...
- What is the mechanism?
- Concurrency? Volume of Communication?

Shared Address Space Solver

- Single Program Multiple Data (SPMD)



- Assignment controlled by values of variables used as loop bounds

Parallel program in SPMD model

```
10.  procedure Solve(A)
11.       float **A;                          /*A is entire n+2-by-n+2 shared array,
                                               as in the sequential program*/
12.  begin
13.       int i,j, pid, done = 0;
14.       float temp, mydiff = 0;             /*private variables*/
14a.      int mymin = 1 + (pid * n/nprocs);   /*assume that n is exactly divisible by*/
14b.      int mymax = mymin + n/nprocs - 1    /*nprocs for simplicity here*/

15.       while (!done) do                    /*outer loop sweeps*/
16.          mydiff = diff = 0;               /*set global diff to 0 (okay for all to do it)*/
16a.         BARRIER(bar1, nprocs);           /*ensure all reach here before anyone modifies diff*/
17.          for i ← mymin to mymax do        /*for each of my rows*/
18.            for j ← 1 to n do              /*for all nonborder elements in that row*/
19.                 temp = A[i,j];
20.                 A[i,j] = 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
21.                              A[i,j+1] + A[i+1,j]);
22.                 mydiff += abs(A[i,j] - temp);
23.            endfor
24.          endfor
25a.         LOCK(diff_lock);                          /*update global diff if necessary*/
25b.           diff += mydiff;
25c.         UNLOCK(diff_lock);
25d.         BARRIER(bar1, nprocs);           /*ensure all reach here before checking if done*/
25e.         if (diff/(n*n) < TOL) then done = 1;  /*check convergence; all get
                                                     same answer*/
25f.         BARRIER(bar1, nprocs);
26.       endwhile
27.  end procedure
```

Kalray Multi-Purpose Processor Architecture

KALRAY

MPPA MANYCORE

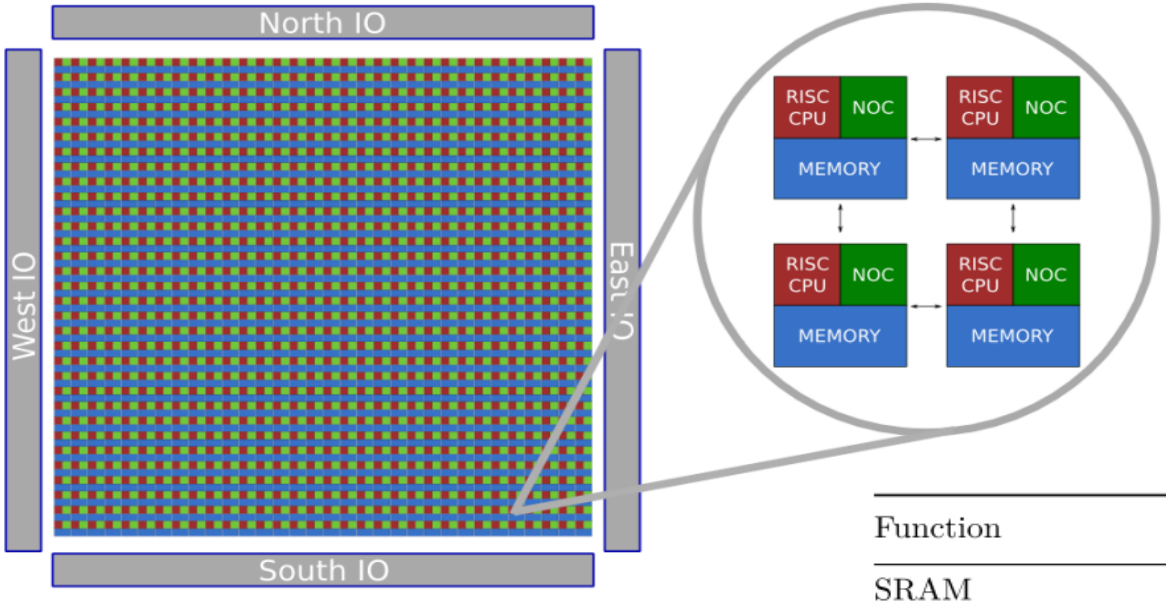
Kalray MPPA®-256 Processor with CMOS 28nm TSMC

256 VLIW processing engine cores + 32 VLIW resource management cores

Available since November 2012

- High processing performance
  700 GOPS – 230 GFLOPS SP
- Low power consumption
- High execution predictability
- High-level programming models
- PCI Gen3, Ethernet 10G, NoCX

Epiphany-V: A 1024 processor 64-bit RISC System-On-Chip



Function	Value (mm^2)	Share of Total Die Area
SRAM	62.4	53.3%
Register File	15.1	12.9%
FPU	11.8	10.1%
NOC	12.1	10.3%
IO Logic	6.5	5.6%
"Other" Core Stuff	5.1	4.4%
IO Pads	3.9	3.3%
Always on Logic	0.66	0.6%

Table 5: Epiphany-V Area Breakdown

Summary of Epiphany-V features:

- 1024 64-bit RISC processors
- 64-bit memory architecture
- 64/32-bit IEEE floating point support
- 64MB of distributed on-chip memory
- 1024 programmable I/O signals
- Three 136-bit wide 2D mesh NOCs
- 2052 Independent Power Domains
- Support for up to 1 billion shared memory processors
- Binary compatibility with Epiphany III/IV chips
- Custom ISA extensions for deep learning, communication, and cryptography

Computer Architecture & Design