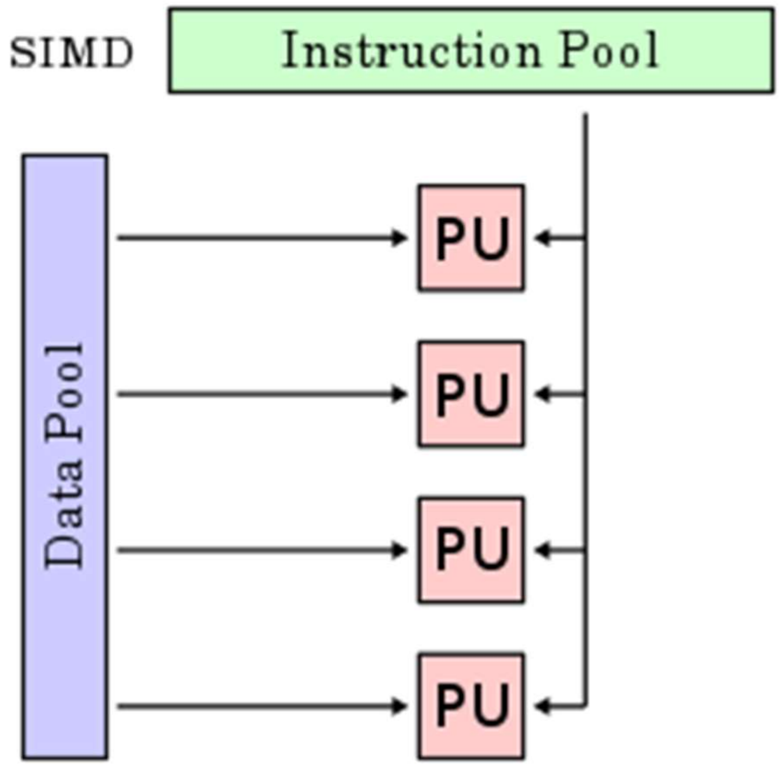
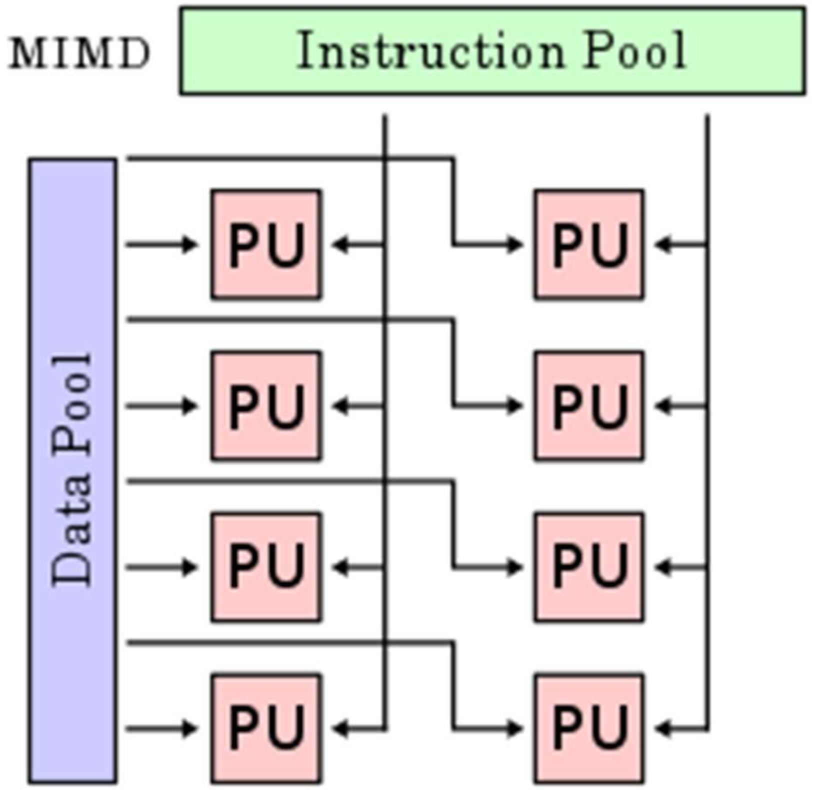
Alternative Kinds of Parallelism: Single Instruction/Multiple Data Stream



- Single Instruction, Multiple Data streams (SIMD)
  - Computer that exploits multiple data streams against a single instruction stream to operations that may be naturally parallelized, e.g., SIMD instruction extensions or Graphics Processing Unit (GPU)

Alternative Kinds of Parallelism: Multiple Instruction/Multiple Data Streams



- Multiple Instruction, Multiple Data streams (MIMD)
  - Multiple autonomous processors simultaneously executing different instructions on different data.
  - MIMD architectures include multicore and Warehouse Scale Computers (datacenters)

Flynn Taxonomy of parallel computers

		Data streams	
		Single	Parallel
Instruction Streams	Single	**SISD**: Intel Pentium 4	**SIMD**: SSE x86, ARM neon, GPGPUs, …
	Multiple	**MISD**: No examples today	**MIMD**: SMP (Intel, ARM, …)

- **From 2011, SIMD and MIMD most common parallel computers**
- **Most common parallel processing programming style: Single Program Multiple Data ("SPMD")**
  - Single program that runs on all processors of an MIMD
  - Cross-processor execution coordination through conditional expressions (thread parallelism)
- **SIMD (aka hw-level *data parallelism*): specialized function units, for handling lock-step calculations involving arrays**
  - Scientific computing, signal processing, multimedia (audio/video processing)

SIMD Architectures

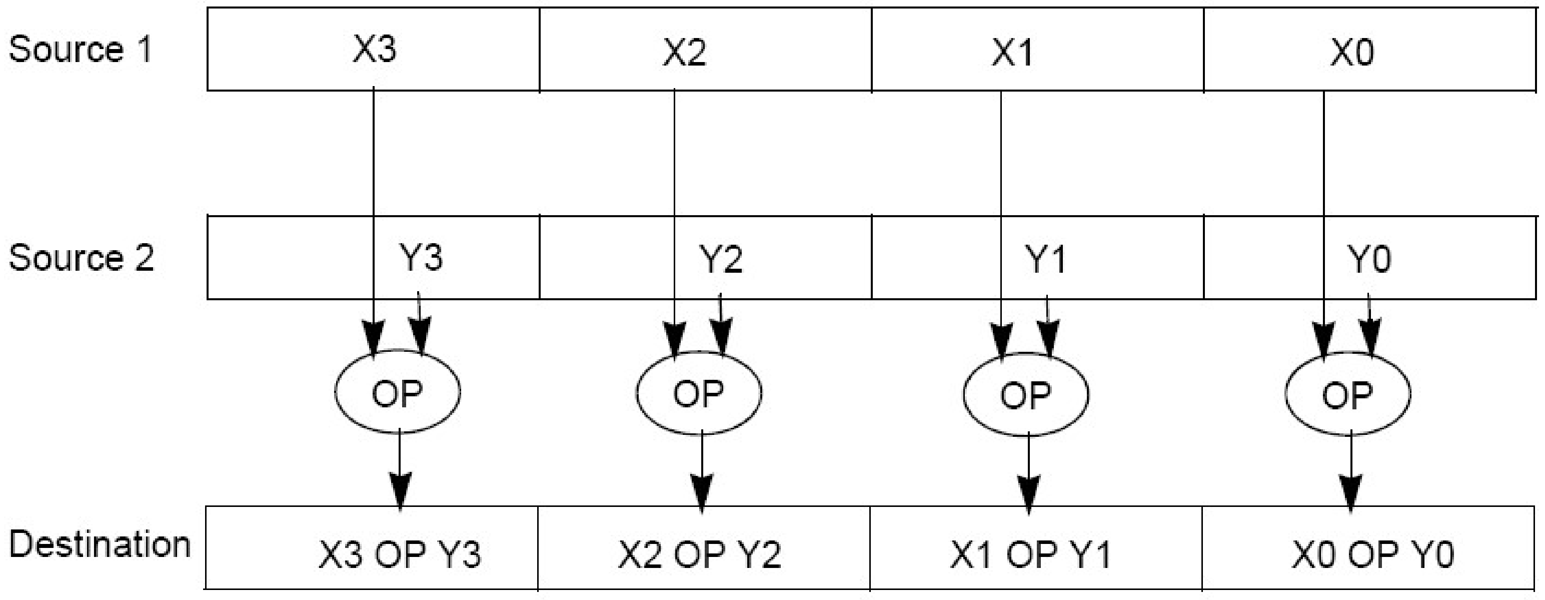
- *Data parallelism*: executing **one operation on multiple data streams**
  - Single control unit
  - Multiple datapaths (processing elements – PEs) running in parallel
    - *PEs are interconnected and exchange/share data as directed by the control unit*
    - *Each PE performs the same operation on its own local data*

- Example to provide context:
  - Multiplying a coefficient vector by a data vector (e.g., in filtering)

$$y[i] := c[i] \times x[i],\ 0 \leq i < n$$

“Advanced Digital Media Boost”

- To improve performance, SIMD instructions
  - Fetch one instruction, do the work of multiple instructions



Example: SIMD Array Processing

```
for each f in array
    f = sqrt(f)
```

SISD

```
for each f in array
{
    load f to the floating-point register
    calculate the square root
    write the result from the register to memory
}
```

SIMD

```
for each 4 members in array
{
    load 4 members to the SIMD register
    calculate 4 square roots in one operation
    write the result from the register to memory
}
```

Data Level Parallelism and SIMD

- SIMD wants adjacent values in memory that can be operated in parallel
- Usually specified in programs as loops

```
for(i=1000; i>0; i=i-1)
    x[i] = x[i] + s;
```

- How can reveal more data level parallelism than available in a single iteration of a loop?
- *Unroll loop* and adjust iteration rate

Loop Unrolling

Loop Unrolling can be implemented from C code

```
for(i=1000; i>0; i=i-1)
 x[i] = x[i] + s;
```

into

```
for(i=1000; i>0; i=i-4)
{
  x[ i ] = x[ i ] + s;
   x[i-1] = x[i-1] + s;
   x[i-2] = x[i-2] + s;
   x[i-3] = x[i-3] + s;
}
```

Loop Unrolling (MIPS)

Assumptions:

- R1 is initially the address of the element in the array with the highest address
- F2 contains the scalar value s
- 8(R2) is the address of the last element to operate on.

```
for(i=1000; i>0; i=i-1)
  x[i] = x[i] + s;
```

```
Loop:
1. l.d        F0, 0(R1)     ; F0=array element
2. add.d      F4,F0,F2      ; add s to F0
3. s.d        F4,0(R1)      ; store result
4. addui      R1,R1,#-8     ; decrement pointer 8 byte
5. bne        R1,R2,Loop    ; repeat loop if R1 != R2
```

Loop Unrolled

```
Loop:   l.d     F0,0(R1)
        add.d   F4,F0,F2
        s.d     F4,0(R1)
        l.d     F6,-8(R1)
        add.d   F8,F6,F2
        s.d     F8,-8(R1)
        l.d     F10,-16(R1)
        add.d   F12,F10,F2
        s.d     F12,-16(R1)
        l.d     F14,-24(R1)
        add.d   F16,F14,F2
        s.d     F16,-24(R1)
        addui   R1,R1,#-32
        bne     R1,R2,Loop
```

NOTE:

1. Different Registers eliminate stalls
2. Only 1 Loop Overhead every 4 iterations
3. This unrolling works if loop_limit(mod 4) = 0

Loop Unrolled Scheduled

```
Loop:   l.d     F0,0(R1)
        l.d     F6,-8(R1)
        l.d     F10,-16(R1)
        l.d     F14,-24(R1)
        add.d   F4,F0,F2
        add.d   F8,F6,F2
        add.d   F12,F10,F2
        add.d   F16,F14,F2
        s.d     F4,0(R1)
        s.d     F8,-8(R1)
        s.d     F12,-16(R1)
        s.d     F16,-24(R1)
        addui   R1,R1,#-32
        bne     R1,R2,Loop
```

4 Loads side-by-side: Could replace with 4 wide SIMD Load

4 Adds side-by-side: Could replace with 4 wide SIMD Add

4 Stores side-by-side: Could replace with 4 wide SIMD Store

Generalizing Loop Unrolling

- A loop of **n iterations**
- **k copies** of the body of the loop

Then we will run the loop with 1 copy of the body **n(mod k)** times and

with k copies of the body **floor(n/k)** times