

**CS 61C: Great Ideas in Computer  
Architecture (Machine Structures)**  
*Lecture 19 – Data Parallelism II*  
*Flynn Taxonomy, SIMD*

Instructors:  
Michael Franklin  
Dan Garcia

<http://inst.eecs.Berkeley.edu/~cs61c/Fa11>

10/10/11

Fall 2011 -- Lecture #19

1

## Review

- Parallelism is one of the great ideas
- Request Level Parallelism
- Data Level Parallelism
  - At the disk/server level – scale out to solve bigger problems on more data
    - Map Reduce/Hadoop
  - At the memory level – today's topic

10/10/11

Fall 2011 -- Lecture #19

2


## New-School Machine Structures (It's a bit more complicated!)

*Software*


- Parallel Requests  
Assigned to computer  
e.g., Search "Katz"
- Parallel Threads  
Assigned to core  
e.g., Lookup, Ads
- Parallel Instructions  
>1 instruction @ one time  
e.g., 5 pipelined instructions
- **Parallel Data**  
>1 data item @ one time  
e.g., Add of 4 pairs of words
- Hardware descriptions  
All gates @ one time

*Hardware*

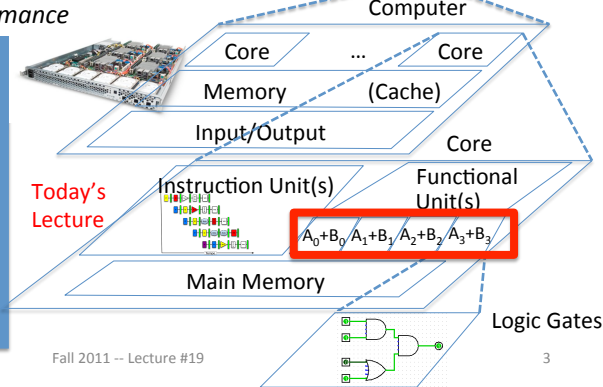
Warehouse Scale Computer



Smart Phone



*Harness Parallelism & Achieve High Performance*



Today's Lecture

10/10/11 Fall 2011 -- Lecture #19 3

## Agenda

- Flynn Taxonomy
- DLP and SIMD
- Intel SSE (part 1)

## Alternative Kinds of Parallelism: Hardware vs. Software

		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Xeon e5345 (Clovertown)	Windows Vista Operating System running on an Intel Xeon e5345 (Clovertown)

- Concurrent software can also run on serial hardware
- Sequential software can also run on parallel hardware
- Our focus today is on sequential or concurrent software running on parallel hardware

10/10/11

Fall 2011 -- Lecture #19

5

## Flynn Taxonomy

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345 (Clovertown)

- In 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 - next lecture)
- SIMD (aka hw-level *data parallelism*): specialized function units, for handling lock-step calculations involving arrays
  - Scientific computing, signal processing, multimedia (audio/video)

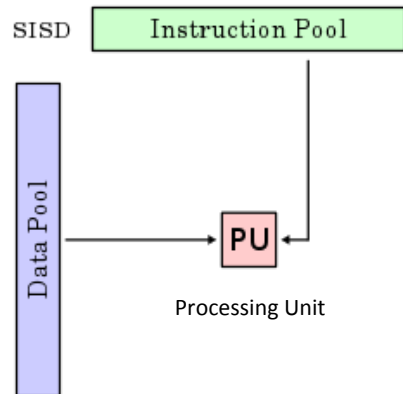
10/10/11

Fall 2011 -- Lecture #19

6

## Flynn Taxonomy: SISD

### Single Instruction/Single Data Stream



- Sequential computer that exploits no parallelism in either the instruction or data streams.
- Examples of SISD architecture are traditional uniprocessor machines

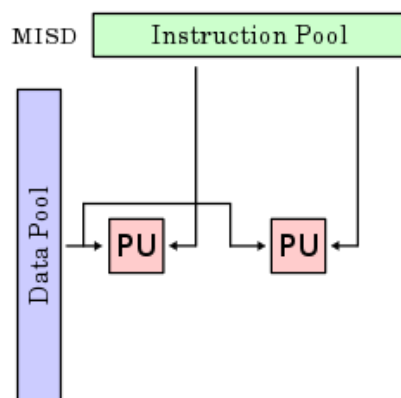
10/10/11

Fall 2011 -- Lecture #19

7

## Flynn Taxonomy: MISD

### Multiple Instruction/Single Data Stream



- Exploits multiple instruction streams against a single data stream for operations that can be naturally parallelized. E.g., certain kinds of array processors.
- No longer commonly encountered, mainly of historical interest only

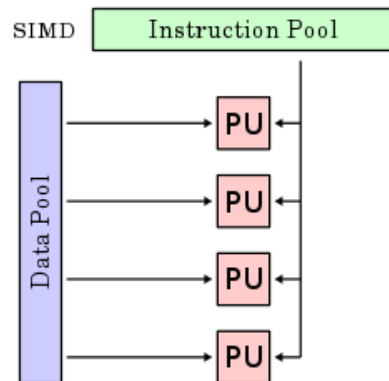
10/10/11

Fall 2011 -- Lecture #19

8

## Flynn Taxonomy: SIMD

### Single Instruction/Multiple Data Stream



- Computer that applies a single instruction stream to multiple data streams for operations that may be naturally parallelized
- e.g., SIMD instruction extensions or Graphics Processing Unit (GPU)

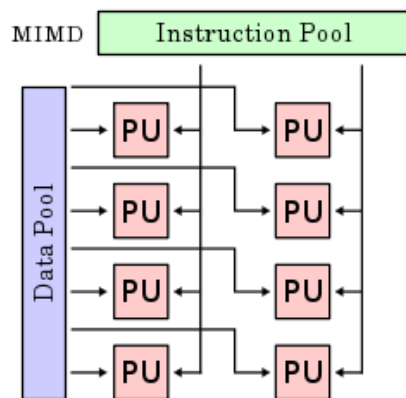
10/10/11

Fall 2011 -- Lecture #19

9

## Flynn Taxonomy: MIMD

### Multiple Instruction/Multiple Data Streams



- Multiple autonomous processors simultaneously executing different instructions on different data.
- MIMD architectures include multicore and Warehouse Scale Computers
- *(Discuss in subsequent lectures)*

10/10/11

Fall 2011 -- Lecture #19

10

## SIMD Architectures

- *Data parallelism*: executing one operation on multiple data streams
- Example to provide context:
  - Multiplying a coefficient vector by a data vector (e.g., in filtering)
$$y[i] := c[i] \times x[i], 0 \leq i < n$$
- Sources of performance improvement:
  - One instruction is fetched & decoded for entire operation
  - Multiplications are known to be independent
  - Pipelining/concurrency in memory access as well

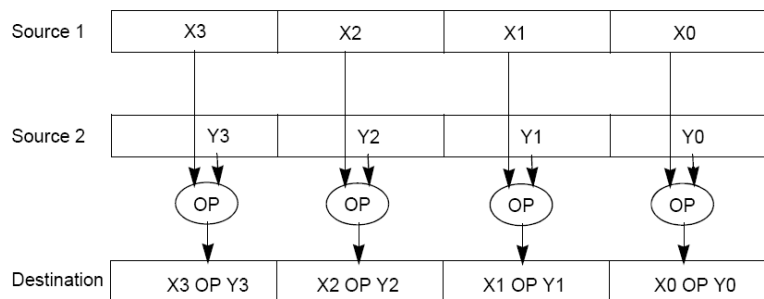
10/10/11

Fall 2011 -- Lecture #19

Slide 11

## “Advanced Digital Media Boost”

- To improve performance, Intel’s SIMD instructions
  - Fetch one instruction, do the work of multiple instructions
  - MMX (MultiMedia eXtension, Pentium II processor family)
  - SSE (*Streaming SIMD Extension, Pentium III and beyond*)



10/10/11

Fall 2011 -- Lecture #19

12

## Example: SIMD Array Processing

```

for each f in array
    f = sqrt(f)

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

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

```

10/10/11

Fall 2011 -- Lecture #19

13

## SSE Instruction Categories for Multimedia Support

Instruction category	Operands
Unsigned add/subtract	Eight 8-bit or Four 16-bit
Saturating add/subtract	Eight 8-bit or Four 16-bit
Max/min/minimum	Eight 8-bit or Four 16-bit
Average	Eight 8-bit or Four 16-bit
Shift right/left	Eight 8-bit or Four 16-bit

- SSE2+ supports wider data types to allow 16 x 8-bit and 8 x 16-bit operands

10/10/11

Fall 2011 -- Lecture #19

15





## SSE/SSE2 Floating Point Instructions

Data transfer	Arithmetic	Compare
MOV {A/U} {SS/PS/SD/PD} xmm, mem/xmm	ADD {SS/PS/SD/PD} xmm, mem/xmm	CMP {SS/PS/SD/PD}
	SUB {SS/PS/SD/PD} xmm, mem/xmm	
MOV {H/L} {PS/PD} xmm, mem/xmm	MUL {SS/PS/SD/PD} xmm, mem/xmm	
	DIV {SS/PS/SD/PD} xmm, mem/xmm	
	SQRT {SS/PS/SD/PD} mem/xmm	
	MAX {SS/PS/SD/PD} mem/xmm	
	MIN {SS/PS/SD/PD} mem/xmm	

xmm: one operand is a 128-bit SSE2 register  
 mem/xmm: other operand is in memory or an SSE2 register  
 {SS} Scalar Single precision FP: one 32-bit operand in a 128-bit register  
 {PS} Packed Single precision FP: four 32-bit operands in a 128-bit register  
 {SD} Scalar Double precision FP: one 64-bit operand in a 128-bit register  
 {PD} Packed Double precision FP, or two 64-bit operands in a 128-bit register  
 {A} 128-bit operand is aligned in memory  
 {U} means the 128-bit operand is unaligned in memory  
 {H} means move the high half of the 128-bit operand  
 {L} means move the low half of the 128-bit operand

10/10/11

Fall 2011 -- Lecture #19

18

## Example: Add Two Single Precision FP Vectors

Computation to be performed:

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

mov a ps : **move** from mem to XMM register, memory aligned, **packed single precision**

add ps : **add** from mem to XMM register, **packed single precision**

SSE Instruction Sequence:

mov a ps : **move** from XMM register to mem, memory aligned, **packed single precision**

```
movaps address-of-v1, %xmm0
    // v1.w | v1.z | v1.y | v1.x -> xmm0
addps address-of-v2, %xmm0
    // v1.w+v2.w | v1.z+v2.z | v1.y+v2.y | v1.x+v2.x -> xmm0
movaps %xmm0, address-of-vec_res
```

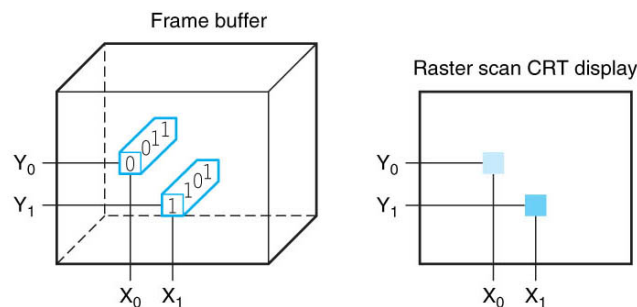
10/10/11

Fall 2011 -- Lecture #19

19

## Displays and Pixels

- Each coordinate in frame buffer on left determines shade of corresponding coordinate for the raster scan CRT display on right. Pixel  $(X_0, Y_0)$  contains bit pattern 0011, a lighter shade on the screen than the bit pattern 1101 in pixel  $(X_1, Y_1)$



10/10/11

Fall 2011 -- Lecture #19

21

## Example: Image Converter

- Converts BMP (bitmap) image to a YUV (color space) image format:
  - Read individual pixels from the BMP image, convert pixels into YUV format
  - Can pack the pixels and operate on a set of pixels with a single instruction
- E.g., bitmap image consists of 8 bit monochrome pixels
  - Pack these pixel values in a 128 bit register (8 bit \* 16 pixels), can operate on 16 values at a time
  - Significant performance boost

10/10/11

Fall 2011 -- Lecture #19

22

## Example: Image Converter

- FMADDPS – Multiply and add packed single precision floating point instruction
- One of the typical operations computed in transformations (e.g., DFT or FFT)

$$P = \sum_{n=1}^N f(n) \times x(n)$$

10/10/11

Fall 2011 -- Lecture #19

23

## Example: Image Converter

Floating point numbers  $f(n)$  and  $x(n)$  in `src1` and `src2`; `p` in `dest`;

C implementation for  $N = 4$  (128 bits):

```
for (int i =0; i< 4; i++)
    p = p + src1[i] * src2[i];
```

Regular x86 instructions for the inner loop:

```
//src1 is on the top of the stack; src1 * src2 -> src1
    fmul DWORD PTR _src2$[%esp+148]
//p = ST(1), src1 = ST(0); ST(0)+ST(1) -> ST(1); ST-Stack Top
    faddp %ST(0), %ST(1)
```

(Note: Destination on the right in x86 assembly)

Number regular x86 Fl. Pt. instructions executed:  $4 * 2 = 8$

10/10/11

Fall 2011 -- Lecture #19

24

## Example: Image Converter

Floating point numbers  $f(n)$  and  $x(n)$  in `src1` and `src2`; `p` in `dest`;  
C implementation for  $N = 4$  (128 bits):

- ```
for (int i =0; i< 4; i++)
    p = p + src1[i] * src2[i];
```
- SSE2 instructions for the inner loop:
 

```
//xmm0 = p, xmm1 = src1[i], xmm2 = src2[i]
    mulps %xmm1, %xmm2 // xmm2 * xmm1 -> xmm2
    addps %xmm2, %xmm0 // xmm0 + xmm2 -> xmm0
```
  - Number regular instructions executed: 2 SSE2 instructions vs. 8 x86
  - SSE5 instruction accomplishes same in one instruction:
 

```
fmaddps %xmm0, %xmm1, %xmm2, %xmm0
    // xmm2 * xmm1 + xmm0 -> xmm0
    // multiply xmm1 x xmm2 paired single,
    // then add product paired single to sum in xmm0
```
  - Number regular instructions executed: 1 SSE5 instruction vs. 8 x86

10/10/11

Fall 2011 -- Lecture #19

25

## So, in conclusion...

- Flynn Taxonomy of Parallel Architectures
  - SIMD: Single Instruction Multiple Data
  - MIMD: Multiple Instruction Multiple Data
  - SISD: Single Instruction Single Data (unused)
  - MISD: Multiple Instruction Single Data
- Intel SSE SIMD Instructions
  - One instruction fetch that operates on multiple operands simultaneously
  - 128/64 bit XMM registers

10/10/11

Fall 2011 -- Lecture #19

26