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

- **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

---

**Agenda**

- Flynn Taxonomy
- DLP and SIMD
- Intel SSE (part 1)
**Alternative Kinds of Parallelism: Hardware vs. Software**

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Sequential</th>
<th>Concurrent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>Matrix Multiply written in MatLab running on an Intel Pentium 4</td>
<td>Windows Vista Operating System running on an Intel Pentium 4</td>
</tr>
<tr>
<td>Parallel</td>
<td>Matrix Multiply written in MATLAB running on an Intel Xeon e5345 (Clovertown)</td>
<td>Windows Vista Operating System running on an Intel Xeon e5345 (Clovertown)</td>
</tr>
</tbody>
</table>

- 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

**Flynn Taxonomy**

<table>
<thead>
<tr>
<th>Data Streams</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Streams</td>
<td>SISD: Intel Pentium 4</td>
<td>SIMD: SSE instructions of x86</td>
</tr>
<tr>
<td></td>
<td>MISD: No examples today</td>
<td>MIMD: Intel Xeon e5345 (Clovertown)</td>
</tr>
</tbody>
</table>

- 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)
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.

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.
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)

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

• 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

“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)
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
\}

SSE Instruction Categories
for Multimedia Support

<table>
<thead>
<tr>
<th>Instruction category</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned add/subtract</td>
<td>Eight 8-bit or Four 16-bit</td>
</tr>
<tr>
<td>Saturating add/subtract</td>
<td>Eight 8-bit or Four 16-bit</td>
</tr>
<tr>
<td>Max/min/minimum</td>
<td>Eight 8-bit or Four 16-bit</td>
</tr>
<tr>
<td>Average</td>
<td>Eight 8-bit or Four 16-bit</td>
</tr>
<tr>
<td>Shift right/left</td>
<td>Eight 8-bit or Four 16-bit</td>
</tr>
</tbody>
</table>

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

128-Bit SIMD Data Types

- **Note:** In Intel Architecture (unlike MIPS) a word is 16 bits
  - Single precision FP: Double word (32 bits)
  - Double precision FP: Quad word (64 bits)

---

**XMM Registers**

- Architecture extended with eight 128-bit data registers: XMM registers
  - IA 64-bit address architecture: available as 16 64-bit registers (XMM8 – XMM15)
  - E.g., 128-bit packed single-precision floating-point data type (doublewords), allows four single-precision operations to be performed simultaneously
SSE/SSE2 Floating Point Instructions

<table>
<thead>
<tr>
<th>Data transfer</th>
<th>Arithmetic</th>
<th>Compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A/U</td>
<td>SS/PS/SD/ PD</td>
<td>xmm, mem/xmm</td>
</tr>
<tr>
<td>SUB</td>
<td>SS/PS/SD/ PD</td>
<td>xmm, mem/xmm</td>
</tr>
<tr>
<td>MOV (H/L)</td>
<td>(PS/PD)</td>
<td>xmm, mem/xmm</td>
</tr>
<tr>
<td>DIV</td>
<td>SS/PS/SD/ PD</td>
<td>xmm, mem/xmm</td>
</tr>
<tr>
<td>SQRT</td>
<td>SS/PS/SD/ PD</td>
<td>mem/xmm</td>
</tr>
<tr>
<td>MAX</td>
<td>SS/PS/SD/ PD</td>
<td>mem/xmm</td>
</tr>
<tr>
<td>MIN</td>
<td>SS/PS/SD/ PD</td>
<td>mem/xmm</td>
</tr>
</tbody>
</table>

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

Example: Add Two Single Precision FP Vectors

Computation to be performed:

```plaintext
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;
```

SSE Instruction Sequence:

```plaintext
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
```

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

- Each coordinate in frame buffer on left determines shade of corresponding coordinate for the raster scan CRT display on right. Pixel (X0, Y0) contains bit pattern 0011, a lighter shade on the screen than the bit pattern 1101 in pixel (X1, Y1)

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
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)

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

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):

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

Regular x86 instructions for the inner loop:

```c
// 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: \( 4 \times 2 = 8 \)
Example: Image Converter

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

\[
\text{for (int } i = 0; i < 4; i++) \quad p = p + \text{src1}[i] * \text{src2}[i];
\]

- SSE2 instructions for the inner loop:
  \[
  \text{//xmm0 = p, xmm1 = src1[i], xmm2 = src2[i]}
  \text{mulps %xmm1, %xmm2} \quad // \text{xmm2 * xmm1 -> xmm2}
  \text{addps %xmm2, %xmm0} \quad // \text{xmm0 + xmm2 -> xmm0}
  \]
- Number regular instructions executed: 2 SSE2 instructions vs. 8 x86
- SSE5 instruction accomplishes same in one instruction:
  \[
  \text{fmaddps %xmm0, %xmm1, %xmm2, %xmm0}
  \text{// xmm2 * xmm1 + xmm0 -> xmm0}
  \text{// multiply xmm1 x xmm2 paired single,}
  \text{// then add product paired single to sum in xmm0}
  \]
- Number regular instructions executed: 1 SSE5 instruction vs. 8 x86

So, in conclusion...

- Flynn Taxonomy of Parallel Architectures
  - \textit{SIMD: Single Instruction Multiple Data}
  - \textit{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