Programming with SIMD Instructions

Debrup Chakraborty

Computer Science Department, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional
México D.F., México.
email: debrup@cs.cinvestav.mx

November 13, 2014

<table>
<thead>
<tr>
<th>Single Data</th>
<th>Single Instruction</th>
<th>Multiple Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction</td>
<td>SISD</td>
<td>MISD</td>
</tr>
<tr>
<td>Multiple Data</td>
<td>SIMD</td>
<td>MIMD</td>
</tr>
</tbody>
</table>
### SISD

<table>
<thead>
<tr>
<th></th>
<th>Single Instruction</th>
<th>Multiple Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Data</td>
<td>SISD</td>
<td>MISD</td>
</tr>
<tr>
<td>Multiple Data</td>
<td>SIMD</td>
<td>MIMD</td>
</tr>
</tbody>
</table>

- One instruction operating on one data in the same time (traditional sequential processing).
- Flynn includes pipelined architectures also in this category.
- Intel processors $< 1996$ and AMD $< 1998$
**MISD**

<table>
<thead>
<tr>
<th>Single Data</th>
<th>Single Instruction</th>
<th>Multiple Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SISD</td>
<td>MISD</td>
<td></td>
</tr>
</tbody>
</table>

| Multiple Data     | SIMD               | MIMD                 |

- Executes different instructions on the same data at the same time.
- This is not common.
Execute the same instruction on multiple data at the same time.

First Intel processor: Intel Pentium MMX (1996), MMX instructions

First AMD processor: AMD K6-2 (1998), 3DNow! instructions
Executes asynchronously distinct instructions on distinct data.

Multiprocessor architectures, clusters etc.
A Brief History of Intel Processors

1971
Intel launched its first microprocessor, the 4004.

1972
The company announced the first 8-bit microprocessor - the 8008.

1974
The Intel 8080 microprocessor was introduced, considered by many to be the first true general-purpose microprocessor.

1976
Intel 8085 microprocessor was introduced, delivering a 5-volt power supply advantage.

1982
Launched the high-performance 16-bit 80286 microprocessor with 134,000 transistors.

1985
A 32-bit chip, the advanced Intel 386™ processor was launched.

1989
Introduced the Intel i860 processor, the first commercial microprocessor with more than 1M transistors.

1978
Introduced the 8086 16-bit microprocessor which became an industry standard.
A Brief History of Intel Processors

1992
Introduced first OverDrive processors, which allowed users of certain upgradable PCs to boost system performance.

1993
Intel Pentium Processors arrives, five times more powerful than the original Intel i486™ processor.

1995
The Intel Pentium Pro processor: high-performance chip for 32-bit workstations and servers.

1998
Intel rolls out the Intel Pentium II Xeon processor for workstation and server markets.

1998
Intel announced its first high-performance, low-power processors based on the Intel StrongARM technology, for handheld computing and communication devices.

1999
Intel Pentium III and Pentium III Xeon processors hits the market.

2000
The Intel Pentium 4 Processor heralds new performance with 42-million transistors.
A Brief History of Intel Processors

2012
- Intel 3rd-gen Core™ processor (IVY Bridge) launched.

2011
- ‘Visibly Smart’ 2nd Gen Intel Core™ Processors (Sandy Bridge) launched.

2008
- Energy-efficient computer chip, Intel Atom™ processor introduced to provide wireless capability to small mobile computing devices.

2007
- Launched Core™ 2 Quad processors. This year also sees breakthrough in 45nm process technology that allows more than 2 million Intel 45nm transistors to fit in a sentence period.

2006
- World’s first Quad-Core Processor for desktop & mainstream servers and more ... Intel Centrino® Duo Mobile Technology, Intel VIV™ technology and Intel Core™ 2 Duo processor is launched.

2003
- Introduced Intel Centrino Processor Technology: high performance, great battery life, and integrated wireless LAN capability to thinner laptop PCs.

2002
- Intel delivered its first chip 0.13 micron technology on 300mm (12-inch) wafers.

2001
- Server workhorses; Intel Itanium Processor and Intel Xeon Processors launched.
Intel SIMD Instruction Sets

- **MMX instructions**: Multimedia extensions. 8 registers of 64 bits.
- **SSE instructions**: Streaming SIMD Extensions. Includes 128 bit registers, and a variety of instructions for bit manipulations, arithmetic etc. Recently includes dedicated instructions for cryptography.
- **AVX instructions**: Advanced Vectorial Extension, includes 256 bit registers.
- More extensions on the way.
History of SSE

- **Intel® MMX™**
  - Jan 1997
  - Number of Instructions: 56
  - Size: 350 nm

- **Streaming SIMD Extensions (SSE)**
  - Feb 1999
  - Number of Instructions: 70
  - Size: 250 nm

- **Streaming SIMD Extensions 2 (SSE2)**
  - Dec 2000
  - Number of Instructions: 144
  - Size: 180 nm

- **Streaming SIMD Extensions 3 (SSE3)**
  - Feb 2004
  - Number of Instructions: 13
  - Size: 90 nm

- **Supplemental Streaming SIMD**
  - Jul 2006
  - Number of Instructions: 32
  - Size: 65 nm

- **Streaming SIMD Extensions 4 (SSE4)**
  - Feb 2008
  - Number of Instructions: 54
  - Size: 45 nm

- **AES Instruction Set**
  - 2010
  - Number of Instructions: 7
  - Size: 32 nm

- **AVX Instruction Set 256 - bits**
  - 2011
  - Number of Instructions: \( \ldots \)
  - Size: 32 nm
How SSE instructions work?

- Utilize dedicated registers.

![Diagram showing MMX and SSE registers](image)
How SSE instructions work?

- Multiple data can be packed in a single register

![Diagram showing different ways of packing data in a single register with SSE instructions.](image-url)
**Task:** For each $f$ in array compute

$$f = \sqrt{f}.$$ 

**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
}

Debrup (Computer Science Department, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, México D.F., México. Email: debrup@cs.cinvestav.mx)
**SIMD:**

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
}
## Summary of SSE registers

<table>
<thead>
<tr>
<th></th>
<th>Number of registers</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX</td>
<td>8</td>
<td>64-bits</td>
</tr>
<tr>
<td>SSE</td>
<td>8</td>
<td>128-bits</td>
</tr>
<tr>
<td>SSE2</td>
<td>16</td>
<td>128-bits</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>AVX</td>
<td>16</td>
<td>256-bits</td>
</tr>
</tbody>
</table>
Sample SSE Instructions

- **movss xmm, m32**
  Load a single-precision (32-bit) floating-point element from memory into the lower of xmm, and zero the upper 3 elements. Memory address does not need to be aligned on any particular boundary.

- **movaps xmm, m128**
  Load 128-bits (composed of 4 packed single-precision (32-bit) floating-point elements) from memory into destination. Memory address must be aligned on a 16-byte boundary.

- **movdqa xmm1, m128,**
  Load 128-bits of integer data from memory into destination. Memory address must be aligned on a 16-byte boundary.

(Other usages possible)
Sample SSE Instructions

movlps xmm0, [eax]

movlhp xmm1, xmm0

movaps [eax], xmm0

Debrup (Computer Science Department, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional México D.F., México.

email: debrup@cs.cinvestav.mx

Programming with SIMD Instructions

November 13, 2014
Sample SSE Instructions

- Scalar operations (ss Single scalar)
- Packed (ps Parallel scalar)
Initially it was done by "Inline Assembly":

```asm
__asm {
    MOV EAX Op_A
    MOV EBX, Op_B
    MOVUPS XMM0, [EAX]
    MOVUPS XMM1, [EBX]
    ADDPS XMM0, XMM1
    MOVUPS [Op_C], XMM0
}
```

Complicated, not very readable, programmer needs to take care of low level details like register allocation etc.
A better alternative is to use Intel *intrinsics*...

```
__m128 _mm_add_ps(__m128 a , __m128 b );
```

- They are functions coded in assembly in appropriate header files.
- The syntax is much intuitive, and the programmer need not take care of low level details.
- Most compilers (say GCC, ICC) has a good understanding of the intrinsics and can generate optimized codes with them.
What do we need?

- A processor which supports the instructions that we want to use.
- An appropriate compiler, which understand intrinsics (GCC or ICC, in general)
- The headers (.h) which corresponds to the instructions.
- Compile with appropriate flags to enable the instruction sets.
- Know the syntax of the instructions.
<table>
<thead>
<tr>
<th>Instructions</th>
<th>Headers</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>-MMX</td>
<td>mmintrin.h</td>
<td>-mmsx</td>
</tr>
<tr>
<td>-SSE</td>
<td>xmmmintrin.h</td>
<td>-msse</td>
</tr>
<tr>
<td>-SSE2</td>
<td>emmintrin.h</td>
<td>-msse2</td>
</tr>
<tr>
<td>-SSE3</td>
<td>pmmintrin.h</td>
<td>-msse3</td>
</tr>
<tr>
<td>-SSSE3</td>
<td>tmmintrin.h</td>
<td>-mssse3</td>
</tr>
<tr>
<td>-SSE4.1 et SSE4.2</td>
<td>smmintrin.h</td>
<td>-msse4.1 -msse4.2</td>
</tr>
<tr>
<td>-AES et PCLMUL</td>
<td>wmmmintrin.h</td>
<td>-maes -mpclmul</td>
</tr>
</tbody>
</table>
In intrinsics we can use some nonstandard data types: __m128d, __m128i.

General syntax for function names: _mm_<name>_<type>

- The prefix _mm_ is always present.
- The second part is <name>, generally it is same as the assembly mnemonic, but not always.
- The final part <type> indicates the packing information.
Intrinsics

Examples:

```c
__m128i _mm_add_epi8(__m128i a, __m128i b)
__m128i _mm_add_epi32(__m128i a, __m128i b)
__m128i _mm_add_epi64(__m128i a, __m128i b)
__m128i _mm_and_si128(__m128i a, __m128i b)
__m128i _mm_xor_si128(__m128i a, __m128i b)
__m128i _mm_or_si128(__m128i a, __m128i b)
```

_piX: vector MM (64-bits) packed with X-bit words
_epiX: vector XMM (128-bits) packed with X-bit words
_si64: vector MM (64-bits) of a single 64-bit word
_si128: vector XMM (128-bits) of a single 128-bit word

see instruction list at: