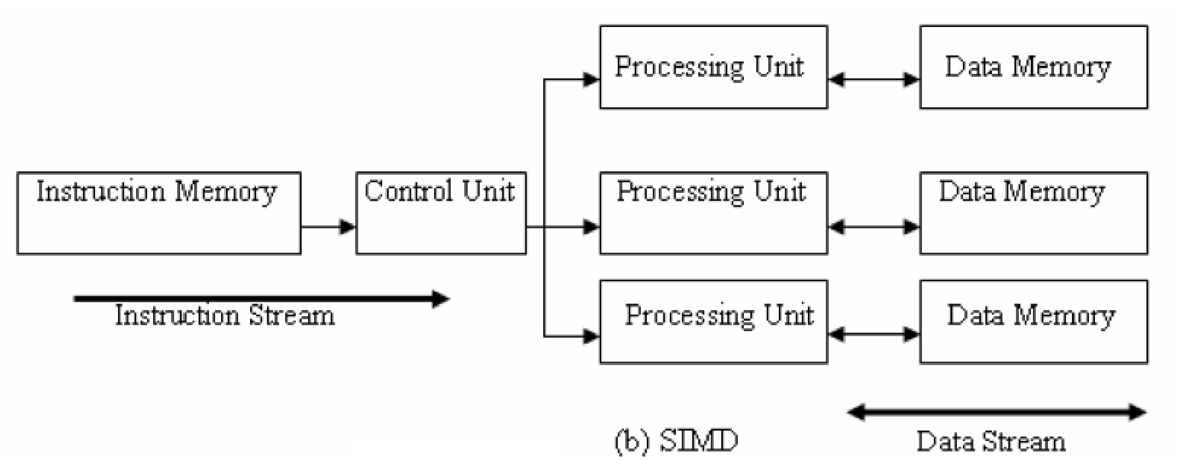
Streaming SIMD Extension (SSE)

SIMD architectures

- A data parallel architecture
- Applying the same instruction to many data
  - Save control logic
  - A related architecture is the vector architecture
  - SIMD and vector architectures offer high performance for vector operations.



Vector operations

- Vector addition Z = X + Y

  for (i=0; i<n; i++) z[i] = x[i] + y[i];

$$\begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \cdots\cdots \\ x_n + y_n \end{pmatrix}$$

- Vector scaling Y = a * X

  for(i=0; i<n; i++) y[i] = a*x[i];

$$a * \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} = \begin{pmatrix} a * x_1 \\ a * x_2 \\ \cdots\cdots \\ a * x_n \end{pmatrix}$$

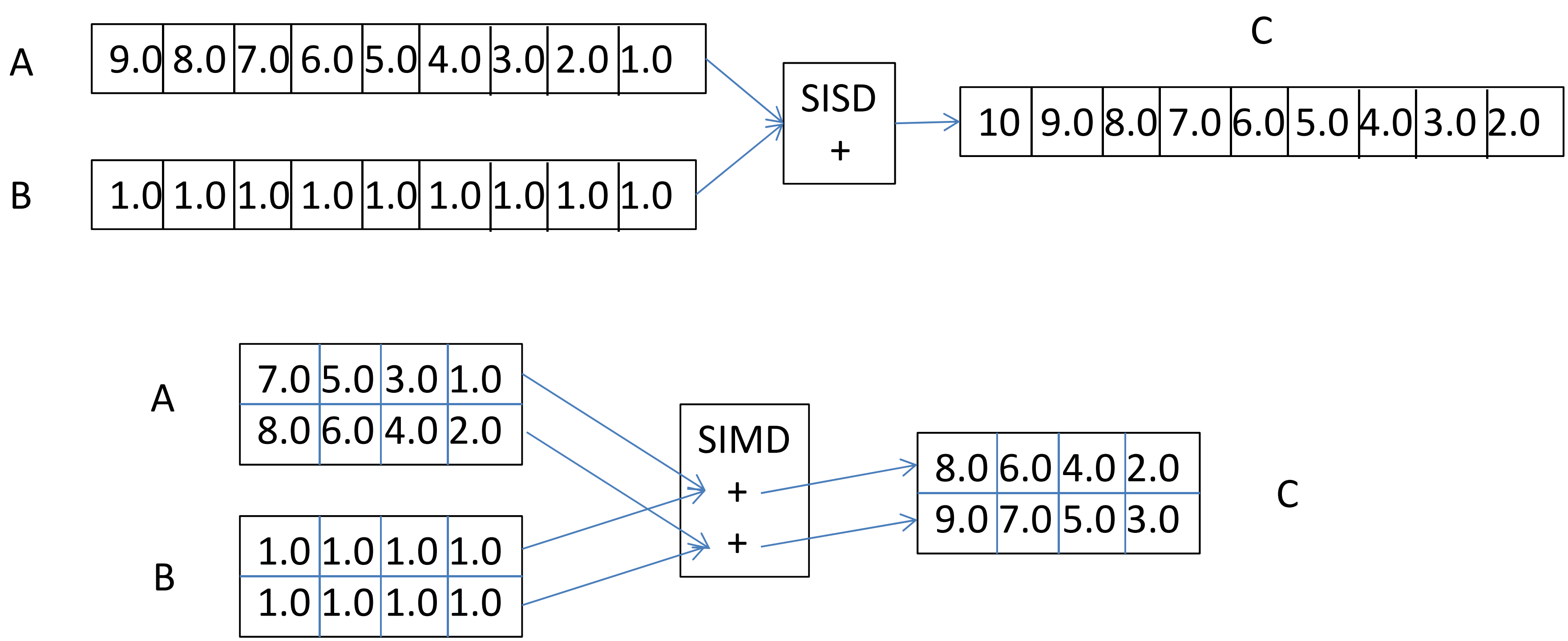
- Dot product

  for(i=0; i<n; i++) r += x[i]*y[i];

$$\begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} \bullet \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix} = x_1 * y_1 + x_2 * y_2 + \cdots\cdots + x_n * y_n$$

SISD and SIMD vector operations

- C = A + B
  - For (i=0;i<n; i++) c[i] = a[i] + b[i]



x86 architecture SIMD support

- Both current AMD and Intel's x86 processors have ISA and microarchitecture support SIMD operations.
- ISA SIMD support
  - MMX, 3DNow!, SSE, SSE2, SSE3, SSE4, AVX
    - See the flag field in /proc/cpuinfo
  - SSE (Streaming SIMD extensions): a SIMD instruction set extension to the x86 architecture
    - Instructions for operating on multiple data simultaneously (vector operations).
- Micro architecture support
  - Many functional units
  - 8 128-bit vector registers, XMM0, XMM1, …, XMM7

SSE programming

- Vector registers support three data types:
  - Integer (16 bytes, 8 shorts, 4 int, 2 long long int, 1 dqword)
  - single precision floating point (4 floats)
  - double precision float point (2 doubles).

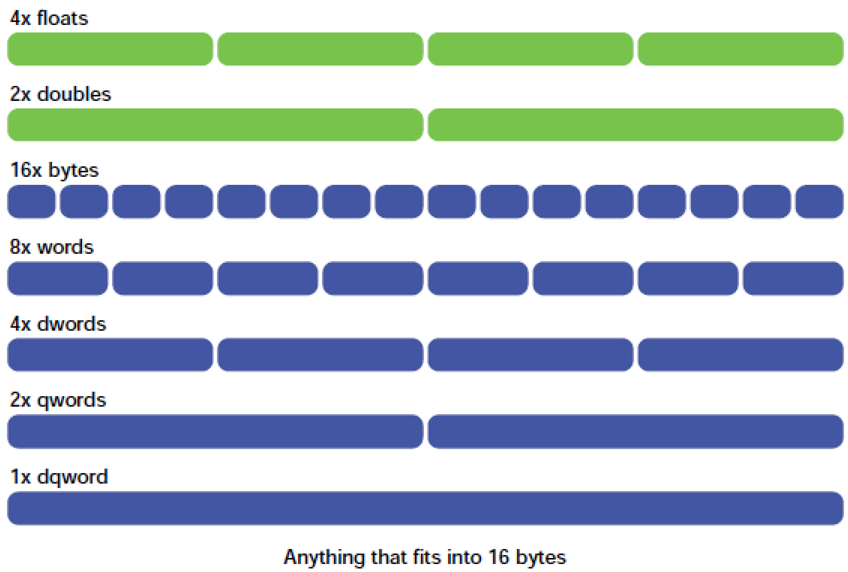


Figure 1. SSE/SSE2 data types

[Klimovitski 2001]

SSE instructions

- Assembly instructions
  - Data movement instructions
    - moving data in and out of vector registers
  - Arithmetic instructions
    - Arithmetic operation on multiple data (2 doubles, 4 floats, 16 bytes, etc)
  - Logical instructions
    - Logical operation on multiple data
  - Comparison instructions
    - Comparing multiple data
  - Shuffle instructions
    - move data around SIMD registers
  - Miscellaneous
    - Data conversion: between x86 and SIMD registers
    - Cache control: vector may pollute the caches
    - State management:

SSE instructions

- Data Movement Instructions:

  MOVUPS - Move 128bits of data to an SIMD register from memory or SIMD register. Unaligned.
  MOVAPS - Move 128bits of data to an SIMD register from memory or SIMD register. Aligned.
  MOVHPS - Move 64bits to upper bits of an SIMD register (high).
  MOVLPS - Move 64bits to lower bits of an SIMD register (low).
  MOVHLPS - Move upper 64bits of source register to the lower 64bits of destination register.
  MOVLHPS - Move lower 64bits of source register to the upper 64bits of destination register.
  MOVMSKPS = Move sign bits of each of the 4 packed scalars to an x86 integer register.
  MOVSS - Move 32bits to an SIMD register from memory or SIMD register.

SSE instructions

- Arithmetic instructions
  - pd: two doubles, ps: 4 floats, ss: scalar
  - ADD, SUB, MUL, DIV, SQRT, MAX, MIN, RCP, etc
    - ADDPS – add four floats, ADDSS: scalar add
- Logical instructions
  - AND, OR, XOR, ANDN, etc
    - ANDPS – bitwise AND of operands
    - ANDNPS – bitwise AND NOT of operands
- Comparison instruction:
  - CMPPS, CMPSS – compare operands and return all 1's or 0's

SSE instructions

- Shuffle instructions
  - SHUFPS: shuffle number from one operand to another
  - UNPCKHPS - Unpack high order numbers to an SIMD register. Unpckhps [x4,x3,x2,x1][y4,y3,y2,y1] = [y4, x4, y3, x3]
  - UNPCKLPS
- Other
  - Data conversion: CVTPS2PI mm,xmm/mem64
  - Cache control
    - MOVNTPS stores data from a SIMD floating-point register to memory, bypass cache.
  - State management: LDMXCSR load MXCSR status register.

SEE programming in C/C++

- Map to *intrinsics*
  - An *intrinsic* is a function known by the compiler that directly maps to a sequence of one or more assembly language instructions. Intrinsic functions are inherently more efficient than called functions because no calling linkage is required.
  - Intrinsics provides a C/C++ interface to use processor-specific enhancements
  - Supported by major compilers such as gcc

SSE intrinsics

- Header files to access SEE intrinsics
  - #include <mmintrin.h>   // MMX
  - #include <xmmintrin.h> // SSE
  - #include <emmintrin.h> //SSE2
  - #include <pmmintrin.h> //SSE3
  - #include <tmmintrin.h> //SSSE3
  - #include <smmintrin.h> // SSE4
- MMX/SSE/SSE2 are mostly supported
- SSE4 are not well supported.
- When compile, use –msse, -mmmx, -msse2 (machine dependent code)
  - Some are default for gcc.
- A side note:
  - Gcc default include path can be seen by 'cpp –v'
  - On linprog, the SSE header files are in
    - /usr/local/lib/gcc/x86_64-unknown-linux-gnu/4.3.2/include/

SSE intrinsics

- Data types (mapped to an xmm register)
  - __m128: float
  - __m128d: double
  - __m128i: integer
- Data movement and initialization
  - _mm_load_ps, _mm_loadu_ps, _mm_load_pd, _mm_loadu_pd, etc
  - _mm_store_ps, …
  - _mm_setzero_ps

SSE intrinsics

- Data types (mapped to an xmm register)
  - __m128: float
  - __m128d: double
  - __m128i: integer
- Data movement and initialization
  - _mm_load_ps, _mm_loadu_ps, _mm_load_pd, _mm_loadu_pd, etc
  - _mm_store_ps, …
  - _mm_setzero_ps
  - _mm_loadl_pd, _mm_loadh_pd
  - _mm_storel_pd, _mm_storeh_pd

SSE intrinsics

- Arithemetic intrinsics:
  - _mm_add_ss, _mm_add_ps, …
  - _mm_add_pd, _mm_mul_pd
- More details in the MSDN library at
  http://msdn.microsoft.com/en-us/library/y0dh78ez(v=VS.80).aspx
- See ex1.c, and sapxy.c

SSE intrinsics

- Data alignment issue
  - Some intrinsics may require memory to be aligned to 16 bytes.
    - May not work when memory is not aligned.
  - See sapxy1.c
- Writing more generic SSE routine
  - Check memory alignment
  - Slow path may not have any performance benefit with SSE
  - See sapxy2.c

Summary

- Contemporary CPUs have SIMD support for vector operations
  - SSE is its programming interface
- SSE can be accessed at high level languages through intrinsic functions.
- SSE Programming needs to be very careful about memory alignments
  - Both for correctness and for performance.

References

- Intel® 64 and IA-32 Architectures Software Developer's Manuals (volumes 2A and 2B). http://www.intel.com/products/processor/manuals/
- SSE Performance Programming, http://developer.apple.com/hardwaredrivers/ve/sse.html
- Alex Klimovitski, "Using SSE and SSE2: Misconcepts and Reality." Intel Developer update magazine, March 2001.
- Intel SSE Tutorial : *An Introduction to the SSE Instruction Set, http://neilkemp.us/src/sse_tutorial/sse_tutorial.html#D*
- SSE intrinsics tutorial, http://www.formboss.net/blog/2010/10/sse-intrinsics-tutorial/
- MSDN library, MMX, SSE, and SSE2 intrinsics: http://msdn.microsoft.com/en-us/library/y0dh78ez(v=VS.80).aspx