Lecture 9

Vector Instructions
SSE Instructions
Instruction Level Parallelism

• Last class
  – Introduction to computer instructions
  – Instruction scheduling to avoid problems
  – Axpy loop unrolling

• This class
  – Further loops
  – Vector instructions
  – SIMD and SSE
Loop splitting

• Summing a vector
  \[ \text{sum} = 0; \]
  \[ \text{for } i = 1:n \]
  \[ \quad \text{sum} = \text{sum} + x(i) \]
  \[ \text{end} \]
• Now consider the first two iterations of the loop.
  – \( \text{sum}(2) = \text{sum}(1) + x(1); \)
  – \( \text{sum}(3) = \text{sum}(2) + x(2); \)
• Can only compute \( \text{sum}(2) \) after \( \text{sum}(1) \) has been computed.
  – But that computation requires three cycles
  – can only compute \( \text{sum}(3) \) after \( \text{sum}(2) \) causing another 3 cycle delay
• pipeline is never filled.
• Need to rearrange the loop using loop splitting.
• Explain splitting for the case that the dimension \( n \) of \( x \) is \( 2^m \).
Partitioned sums

- Split sum: divide vector $x$ into two $2^{m-1}$ vectors $x'$ and $x''$
- Sum them $y = x' + x''$
- Calculate (using the same procedure) the sum of $y$ by splitting it
- Repeat the same procedure, until we are left with a scalar
- This is the desired sum.
- Illustration for $m=3$

\[
\begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4
\end{pmatrix} + 
\begin{pmatrix}
  x_5 \\
  x_6 \\
  x_7 \\
  x_8
\end{pmatrix} = 
\begin{pmatrix}
  x_1 + x_5 \\
  x_2 + x_6 \\
  x_3 + x_7 \\
  x_4 + x_8
\end{pmatrix}
\]

\[
\begin{pmatrix}
  x_1 + x_5 \\
  x_2 + x_6 
\end{pmatrix} + 
\begin{pmatrix}
  x_3 + x_7 \\
  x_4 + x_8
\end{pmatrix} = 
\begin{pmatrix}
  x_1 + x_5 + x_3 + x_7 \\
  x_2 + x_6 + x_4 + x_8
\end{pmatrix}
\]

\[
(x_1 + x_5 + x_3 + x_7) + (x_2 + x_6 + x_4 + x_8) = x_1 + x_5 + x_3 + x_7 + x_2 + x_6 + x_4 + x_8,
\]
Properties

• Since quantities are independent, no pipeline issues
  – Loop can now be unrolled

• Memory locality is good, as contiguous elements are added in different parts of the loop

• Number of operations are the same (n additions)

• However, turns out numerical properties may also be good
  – Addition of like sized floats, leading to better preservation of significant bits

• Also can be used on parallel machines
Vector processors

- Also called **array processor**
- Instruction set includes mathematical operations on multiple data elements simultaneously.
- In contrast regular chips which handles one element at a time are called scalar processors
- Vector processors were common in the supercomputers of the 1980s and into the 1990s, especially Crays
- General increases in performance and processor design saw the near-disappearance of the vector processor as a general-purpose CPU.
- Today, most commodity CPU designs include single instructions for some vector processing on multiple (vectorised) data sets, typically known as SIMD (**Single Instruction, Multiple Data)**,
- Common examples in current architectures include **Streaming SIMD Extensions (SSE)** and **Advanced Vector Extensions (AVX)** instructions in current x86 processors
Vector processors

- Operations on vectors can lead to poor performance in the floating-point pipeline of a scalar processor
  - Vector operations must be implemented as sequences of scalar operations
  - When operations are performed in their natural order, they do not pipeline well
  - Hard job for compiler or programmer to reorder operations
- Solution: create processor with instructions and registers suitable for vectors
- Feasible because number of commonly used vector operations are small.
- Vector register = number of scalar registers for vector components
  - may be single precision, double-precision, or integer.
  - number of components in a vector register ranges from 64 to as many 4096.
  - number of vector registers ranges from 8 to 256.
- Operations: addition, componentwise multiplication and division, and logicals.
- Efficiency via combination of two techniques
  - Pipelining
  - SIMD Parallelism
- Pipelining means that vector operations have a latency or start up time.
- Parallelism means that once the pipeline is full, results can be generated at more than one component per clock cycle.
AXPY on a vector processor

• In implementing the AXPY we will assume that a is in F0 and the starting addresses of x and y are in R2 and R3.

1. LV V1, 0(R2) ; load x
2. MULVS.D V2, V1, F0 ; a*x
3. LV V3, 0(R3) ; get y
4. ADDV.D V4, V2, V3 ; a*x + y
5. SV 0(R3), V4 ; store y

• LV and SV load and store vectors.
• MULVS.D multiplies a vector by a scalar,
• ADDV.D adds two vectors.
• Loop performance
  – Assume all operations produce results at the rate of one component per cycle
  – After common startup time of $\lambda$ cycles.
  – There are five vector instructions.
  – For sequential execution total time is $5(\lambda + 64) = 5\lambda + 320$. 
Improving vector AXPY via overlaps and chained execution

- Exploit overlaps in computation.
  - no data dependencies between the MULVS.D and the second LV,
  - once first LV has completed, we can run the MULVS.D and the second LV in parallel
  - time is reduced to $4(\lambda + 64) = 4\lambda + 256$.

- Further reduction in time by directing output of one instruction directly into another.
  - called chaining.
  - Output from first LV can be fed directly into the MULVS.D.
  - The combined instructions finish at time $2\lambda + 64$.

- AXPY can be extensively chained
- tick separates startup from calculation.
- There are two chains in AXPY
  - first LV -> MULVS:D and the second LV -> ADDV:D -> SV.
  - Second chain cannot start until the first LV has finished if only one load-store unit.
- Last instruction completes at $4\lambda + 128$.
- An improvement over original $5\lambda + 320$.

Figure 3.4: Chaining AXPY
Streaming SIMD Extensions

- Streaming SIMD defines a new architecture for floating point operations
- Introduced in Pentium III in March 1999
  - Pentium III includes floating point, MMX technology, and XMM registers
- Use eight new 128-bit wide general-purpose registers (XMM0 - XMM7)
- Operate on IEEE-754 single-precision 32-bit real numbers
- Support packed and scalar operations on the new packed single precision floating point data types
- SIX iterations and extensions
- Can significantly speed up code