Vector and SIMD Processors

Eric Welch & James Evans

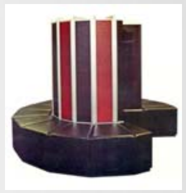
Multiple Processor Systems

Spring 2013

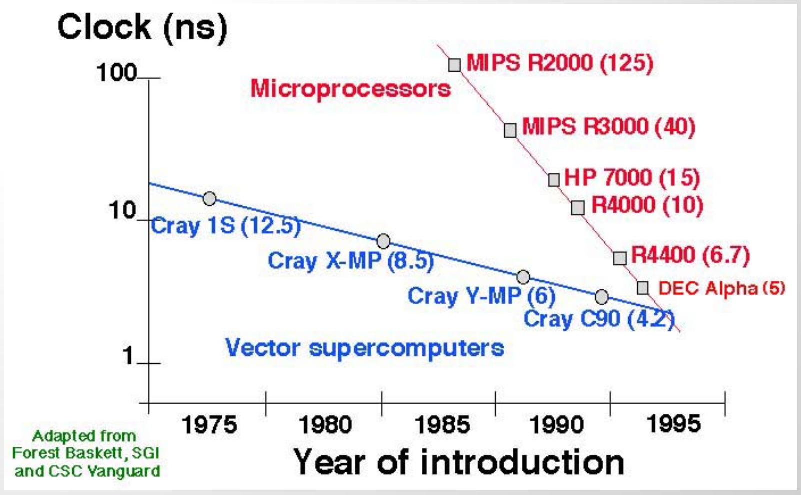
Outline

History of Vector Processors

- *Early Work*
  - o Development started in the early 1960s at Westinghouse
    - ▪ Goal of the *Solomon project* was to substantially increase arithmetic performance by using many simple co-processors under the control of a single *master* CPU
    - ▪ Allowed single algorithm to be applied to large data set

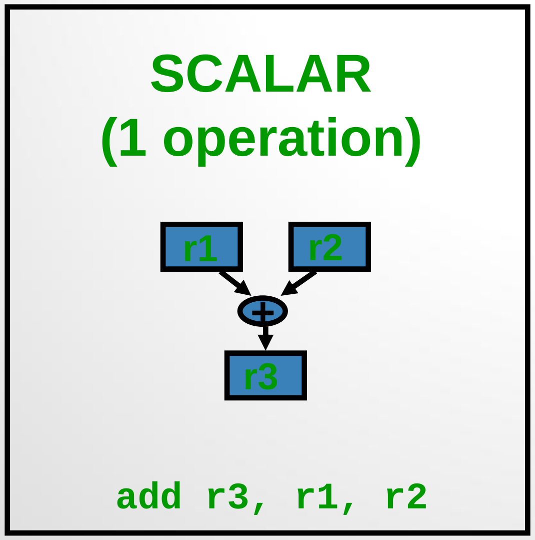


- *Supercomputers*
  - o Dominated supercomputer design through the 1970s into the 1990s
  - o Cray platforms were the most notable vector supercomputers
    - ▪ Cray -1: Introduced in 1976
    - ▪ Cray-2, Cray X-MP, Cray Y-MP
  - o Demise
    - ▪ In the late 1990s, the price-to-performance ratio drastically increased for conventional microprocessor designs

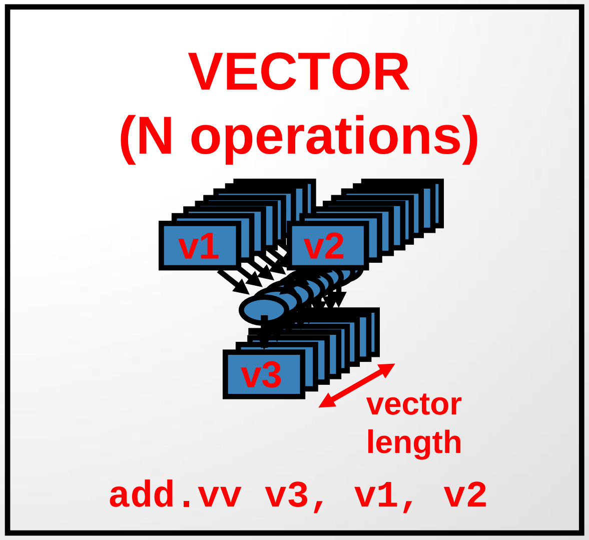


Description of Vector Processors

- CPU that implements an instruction set that operates on 1-D arrays, called *vectors*
- Vectors contain multiple data elements
- Number of data elements per vector is typically referred to as the *vector length*
- Both instructions and data are pipelined to reduce decoding time



```
add r3, r1, r2
```



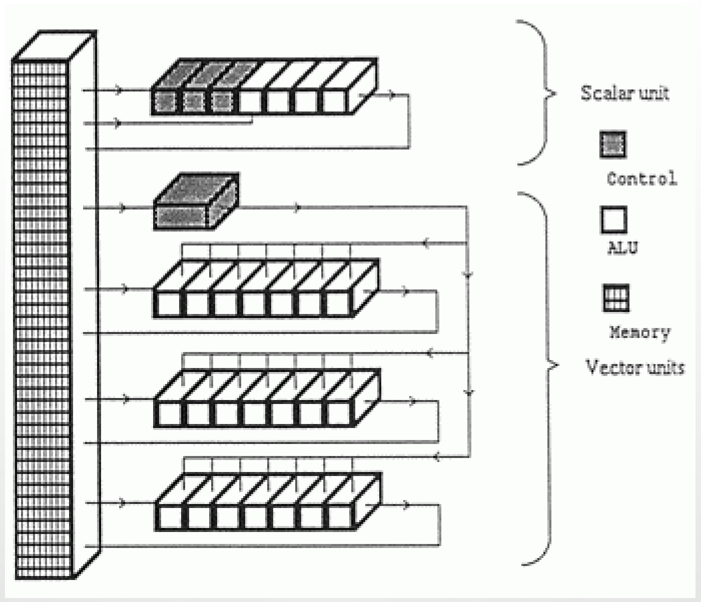
```
add.vv v3, v1, v2
```

Advantages of Vector Processors

- *Require Lower Instruction Bandwith*
  - Reduced by fewer fetches and decodes
- *Easier Addressing of Main Memory*
  - Load/Store units access memory with known patterns
- *Elimination of Memory Wastage*
  - Unlike cache access, every data element that is requested by the processor is actually used – no cache misses
  - Latency only occurs once per vector during pipelined loading
- *Simplification of Control Hazards*
  - Loop-related control hazards from the loop are eliminated
- *Scalable Platform*
  - Increase performance by using more hardware resources
- *Reduced Code Size*
  - Short, single instruction can describe N operations

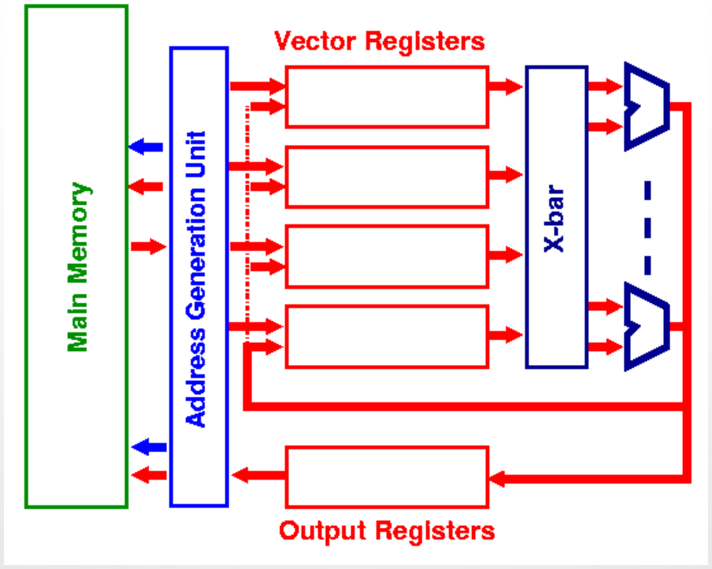
Vector Processor Architectures

- *Memory-to-Memory Architecture* (Traditional)
  - For all vector operation, operands are fetched directly from main memory, then routed to the functional unit
  - Results are written back to main memory
  - Includes early vector machines through mid 1980s:
    - Advanced Scientific Computer (TI), Cyber 200 & ETA-10
  - Major reason for demise was due to *large startup time*



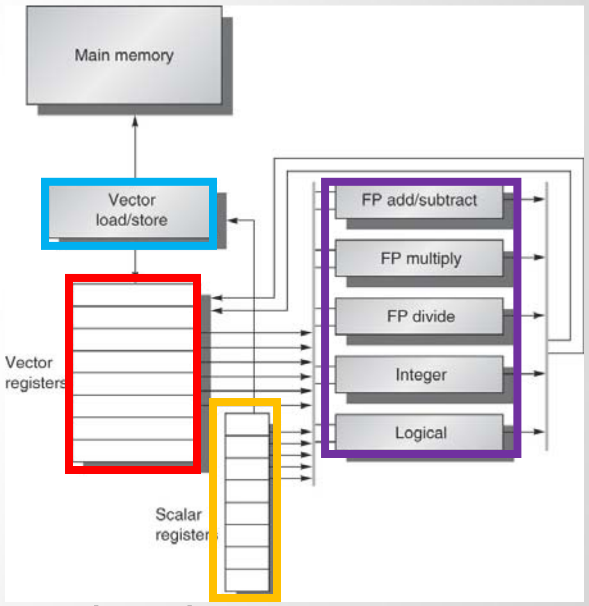
Vector Processor Architectures (cont)

- *Register-to-Register Architecture* (Modern)
  - All vector operations occur between vector registers
  - If necessary, operands are fetched from main memory into a set of vector registers (load-store unit)
  - Includes all vector machines since the late 1980s:
    - Convex, Cray, Fujitsu, Hitachi, NEC
  - SIMD processors are based on this architecture



Components of Vector Processors

- *Vector Registers*
  - Typically 8-32 vector registers with 64 - 128 64-bit elements
  - Each contains a vector of double-precision numbers
  - Register size determines the maximum vector length
  - Each includes at least 2 read and 1 write ports
- *Vector Functional Units (FUs)*
  - Fully pipelined, new operation every cycle
  - Performs arithmetic and logic operations
  - Typically 4-8 different units
- *Vector Load-Store Units (LSUs)*
  - Moves vectors between memory and registers
- *Scalar Registers*
  - Single elements for interconnecting FUs, LSUs, and registers

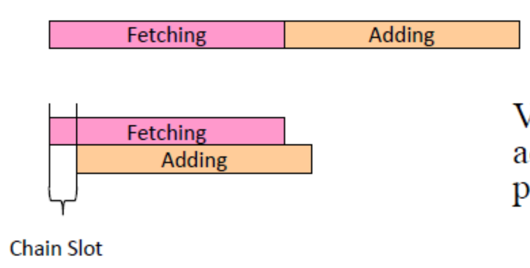


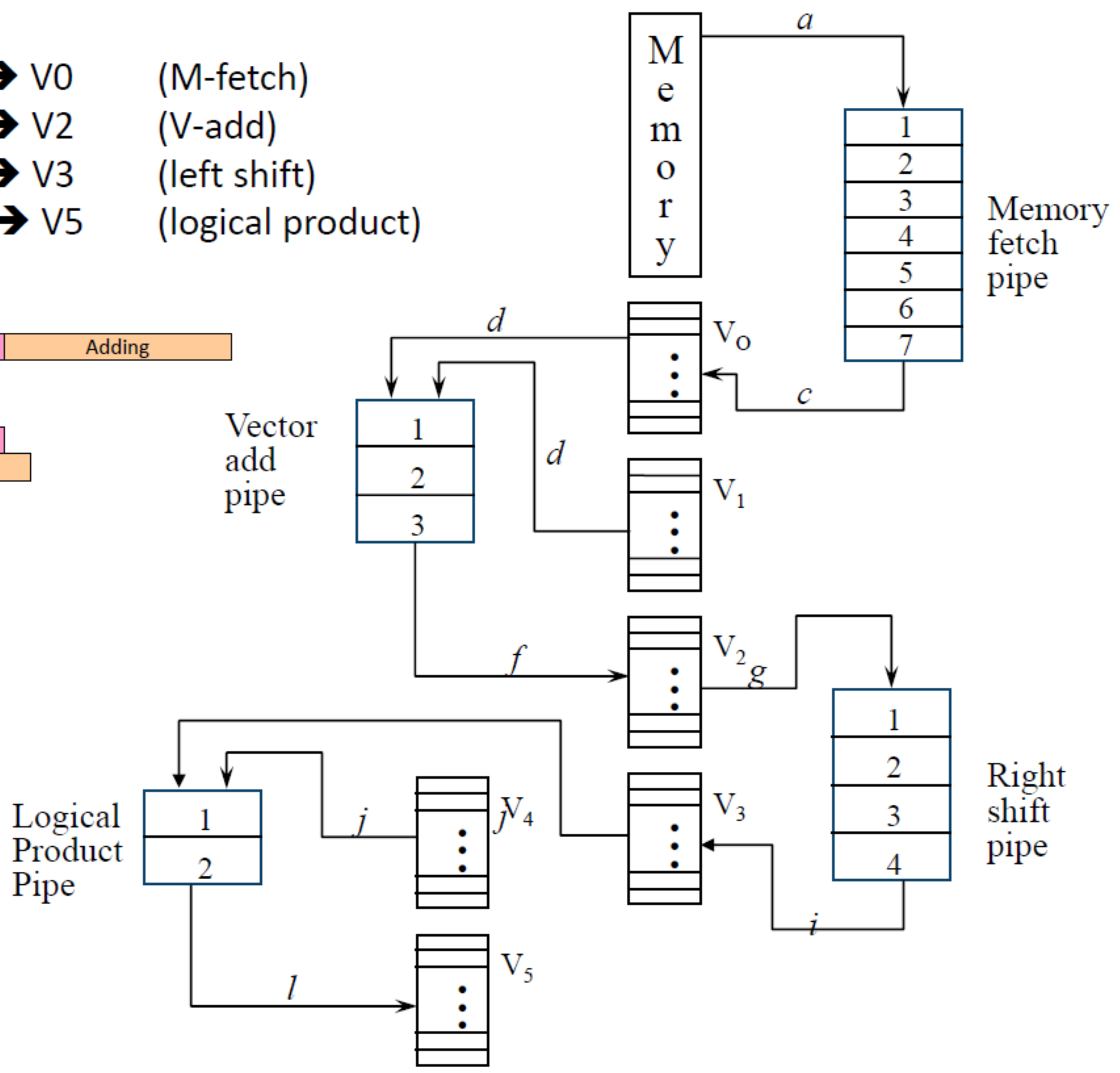
Performance Optimizations

- *Increase Memory Bandwidth*
  - Memory banks are used to reduce load/store latency
  - Allow multiple simultaneous outstanding memory requests
- *Strip Mining*
  - Generates code to allow vector operands whose size is less than or greater than size of vector registers
- *Vector Chaining*
  - Equivalent to data forwarding in vector processors
  - Results of one pipeline are fed into operand registers of another pipeline
- *Scatter and Gather*
  - Retrieves data elements scattered thorughout memory and packs them into sequential vectors in vector registers
  - Promotes data locality and reduces data pollution
- *Multiple Parallel Lanes, or Pipes*
  - Allows vector operation to be performed in parallel on multiple elements of the vector

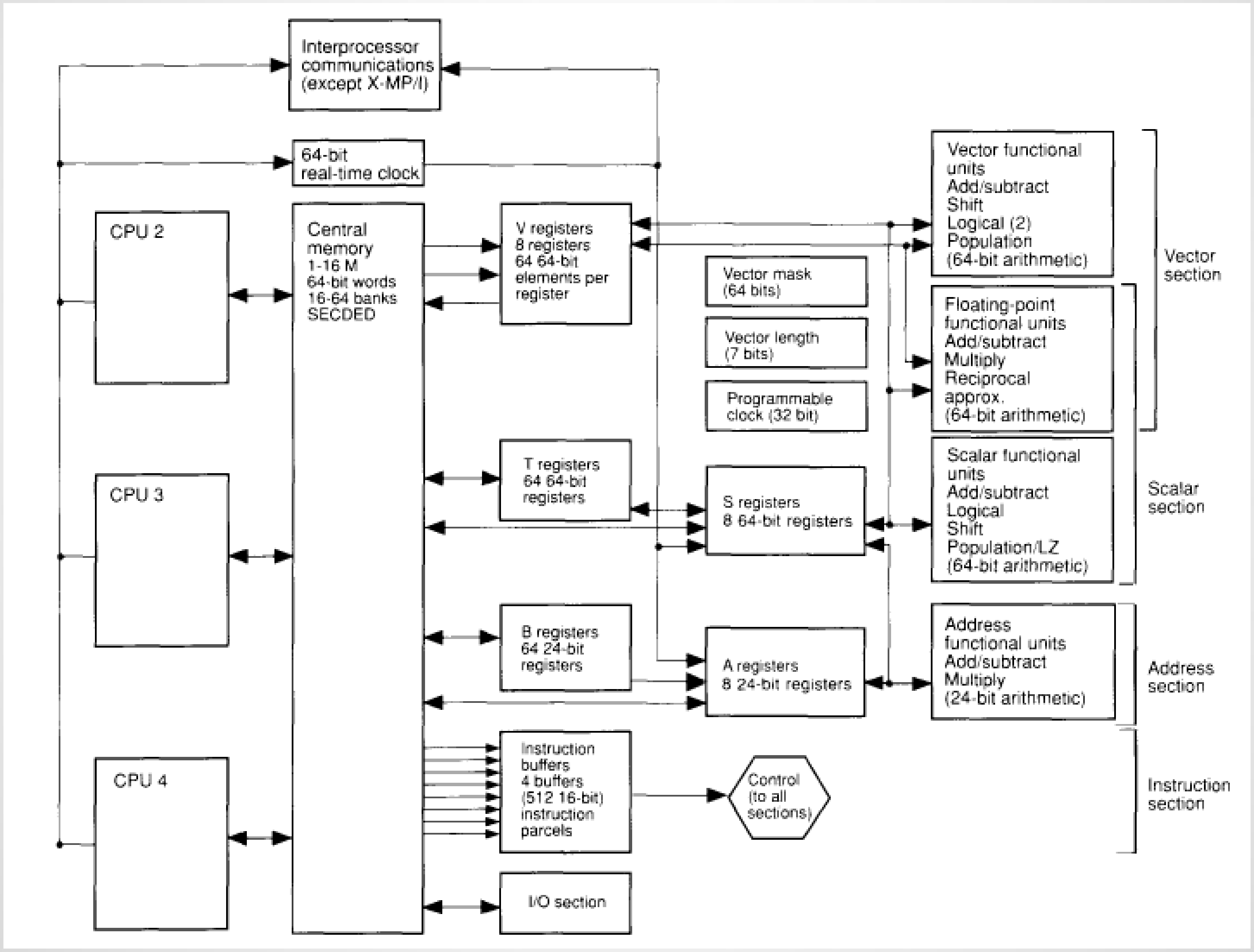
Vector Chaining Example

Mem ➔ V0	(M-fetch)
V0 + V1 ➔ V2	(V-add)
V2 < A3 ➔ V3	(left shift)
V3 ^ V4 ➔ V5	(logical product)

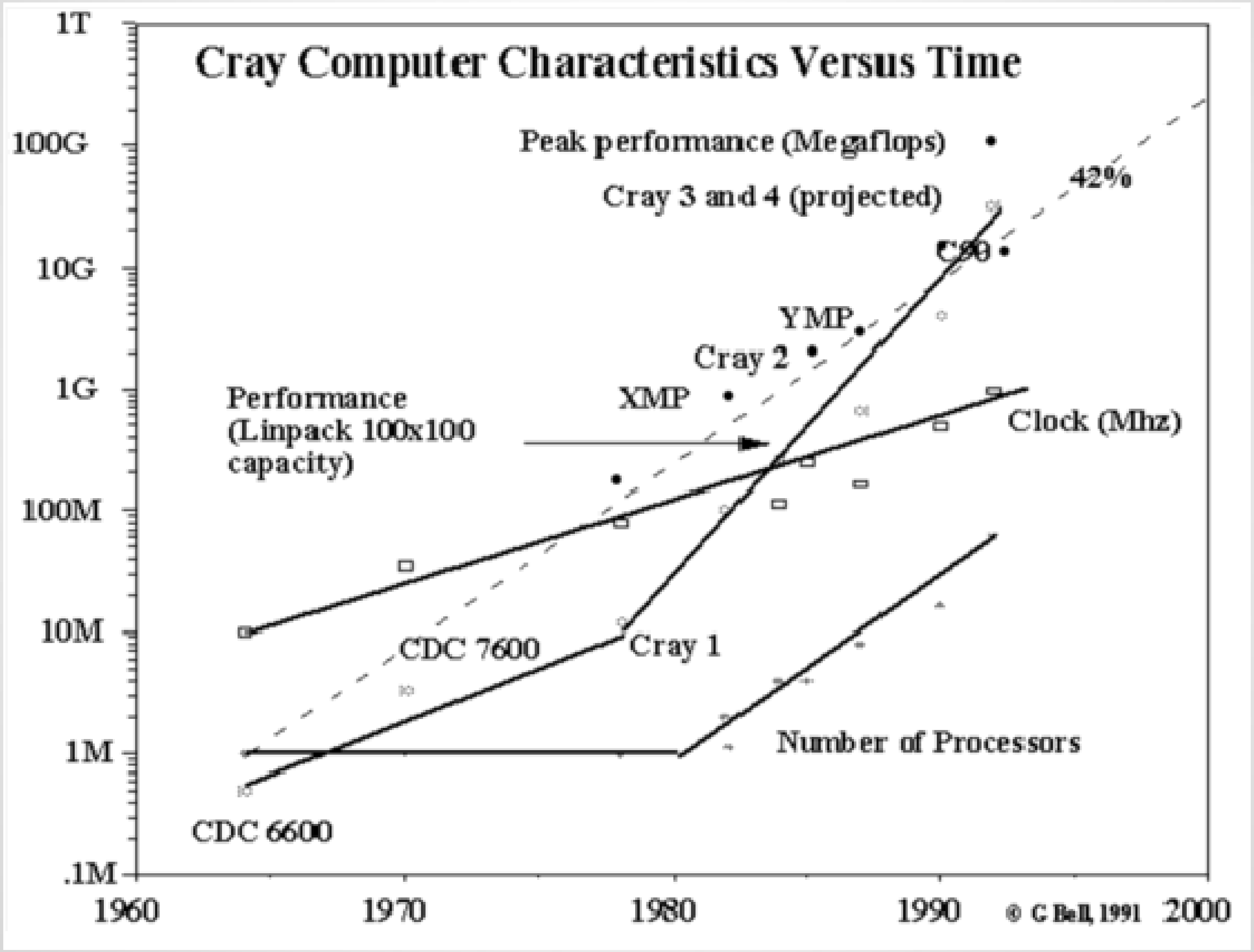




Organization of Cray Supercomputer


Interprocessor communications (except X-MP/I)
64-bit real-time clock
CPU 2
CPU 3
CPU 4
Central memory 1-16 M 64-bit words 16-64 banks SECDED
V registers 8 registers 64 64-bit elements per register
Vector mask (64 bits)
Vector length (7 bits)
Programmable clock (32 bit)
T registers 64 64-bit registers
S registers 8 64-bit registers
B registers 64 24-bit registers
A registers 8 24-bit registers
Instruction buffers 4 buffers (512 16-bit) instruction parcels
Control (to all sections)
I/O section
Vector functional units Add/subtract Shift Logical (2) Population (64-bit arithmetic)
Floating-point functional units Add/subtract Multiply Reciprocal approx. (64-bit arithmetic)
Scalar functional units Add/subtract Logical Shift Population/LZ (64-bit arithmetic)
Address functional units Add/subtract Multiply (24-bit arithmetic)
Vector section
Scalar section
Address section
Instruction section

Performance of Cray Supercomputers



Modern SIMD Introduction

- *Single Instruction Multiple Data* is part of Flynn's taxonomy (not MIMD as discussed in class)
- Performs same instruction on multiple data points concurrently
- Takes advantage of data level parallelism within an algorithm
- Commonly used in image and signal processing applications
  - Large number of samples or pixels calculated with the same instruction
- Disadvantages:
  - Larger registers and functional units use more chip area and power
  - Difficult to parallelize some algorithms (Amdahl's Law)
  - Parallelization requires explicit instructions from the programmer

SIMD Processor Performance Trends

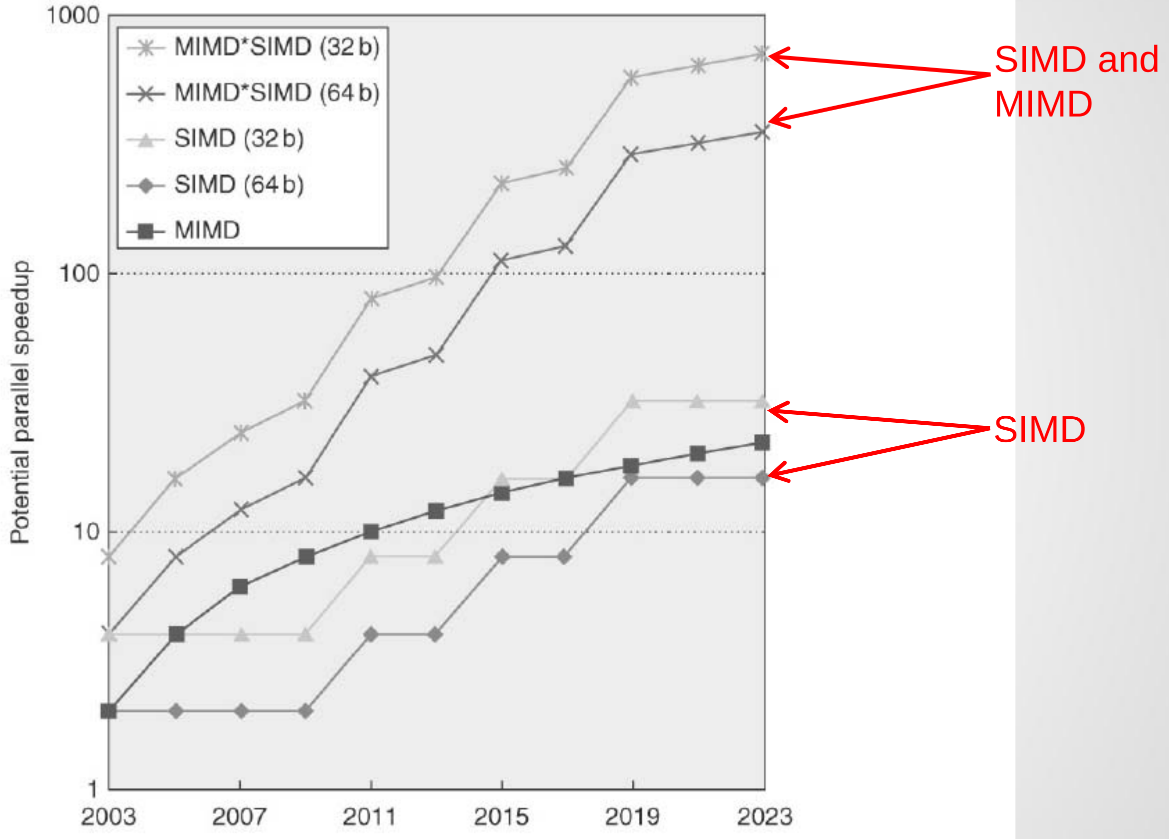


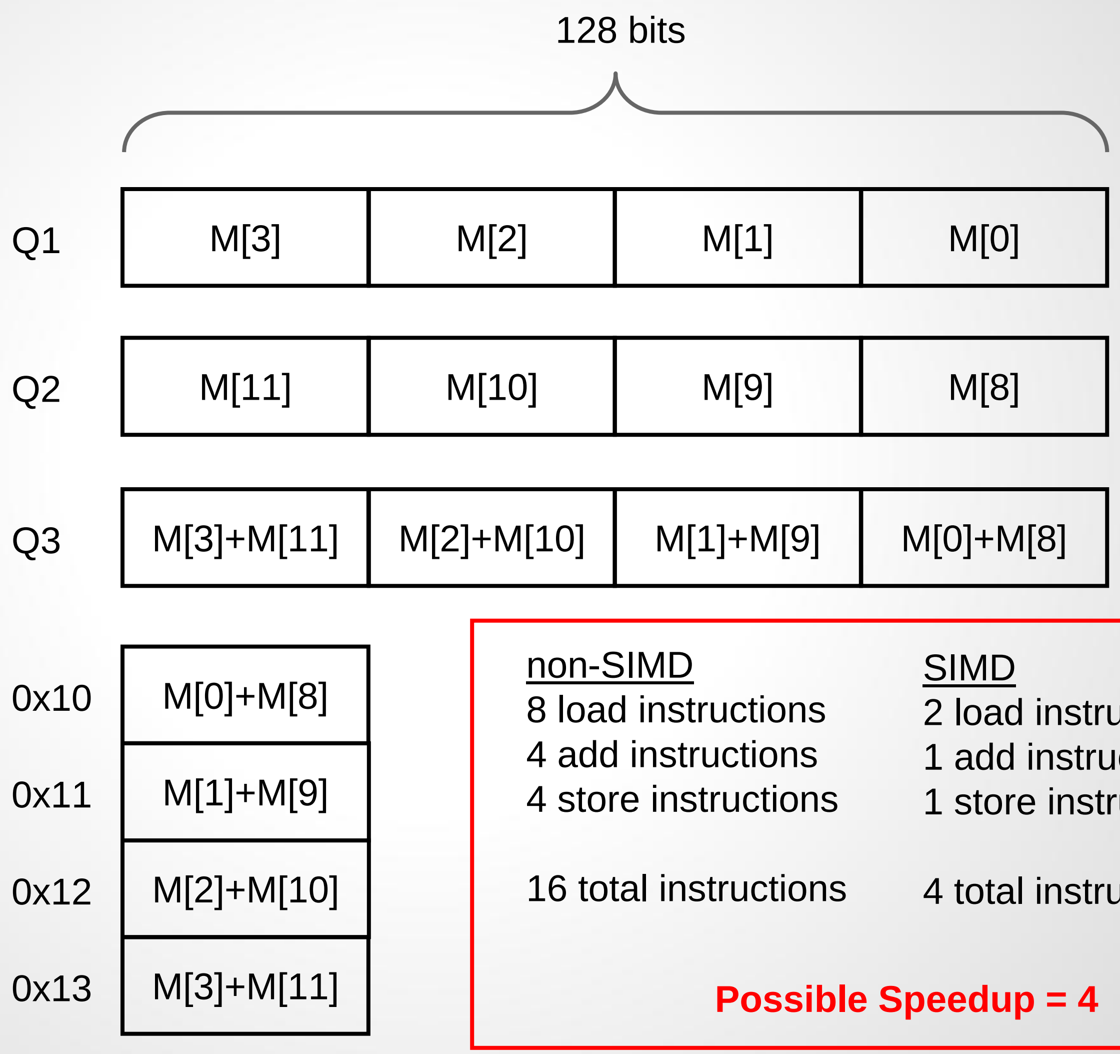
Figure 4.1 Potential speedup via parallelism from MIMD, SIMD, and both MIMD and SIMD over time for x86 computers. This figure assumes that two cores per chip for MIMD will be added every two years and the number of operations for SIMD will double every four years.

Modern SIMD Processors

- Most modern CPUs have SIMD architectures
  - Intel SSE and MMX, ARM NEON, MIPS MDMX
- These architectures include instruction set extensions which allow both sequential and parallel instructions to be executed
- Some architectures include separate SIMD coprocessors for handling these instructions
- ARM NEON
  - Included in Cortex-A8 and Cortex-A9 processors
- Intel SSE
  - Introduced in 1999 in the Pentium III processor
  - SSE4 currently used in Core series

SIMD Processor Introduction

```
VLD.32 Q1, 0x0
VLD.32 Q2, 0x8
VADD.U32 Q3, Q2, Q1
VST.32 Q3, 0x10
```

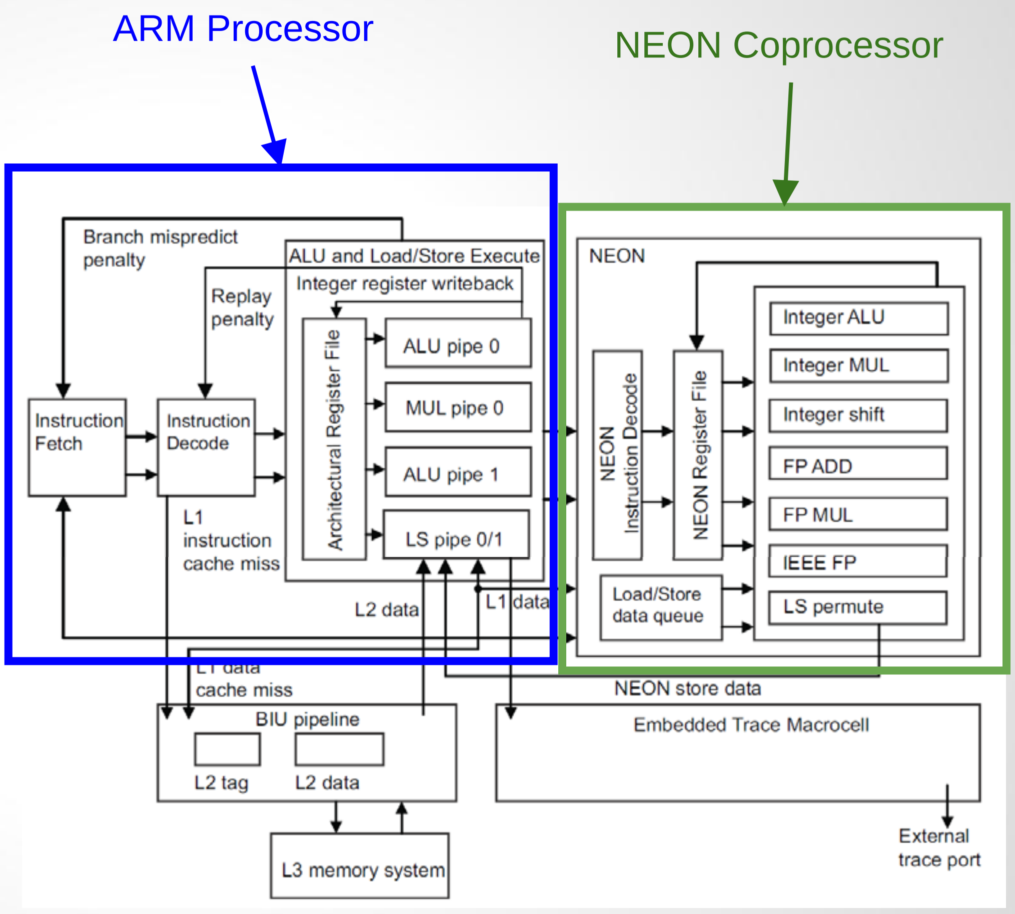


non-SIMD	SIMD
8 load instructions	2 load instructions
4 add instructions	1 add instruction
4 store instructions	1 store instruction
16 total instructions	4 total instructions

Possible Speedup = 4

ARM NEON SIMD Architecture

- 16 128-bit SIMD registers
- Separate sequential and SIMD processors
- Both have access to same L2 cache but separate L1 caches
- Instructions fetched in ARM processor and sent to NEON coprocessor

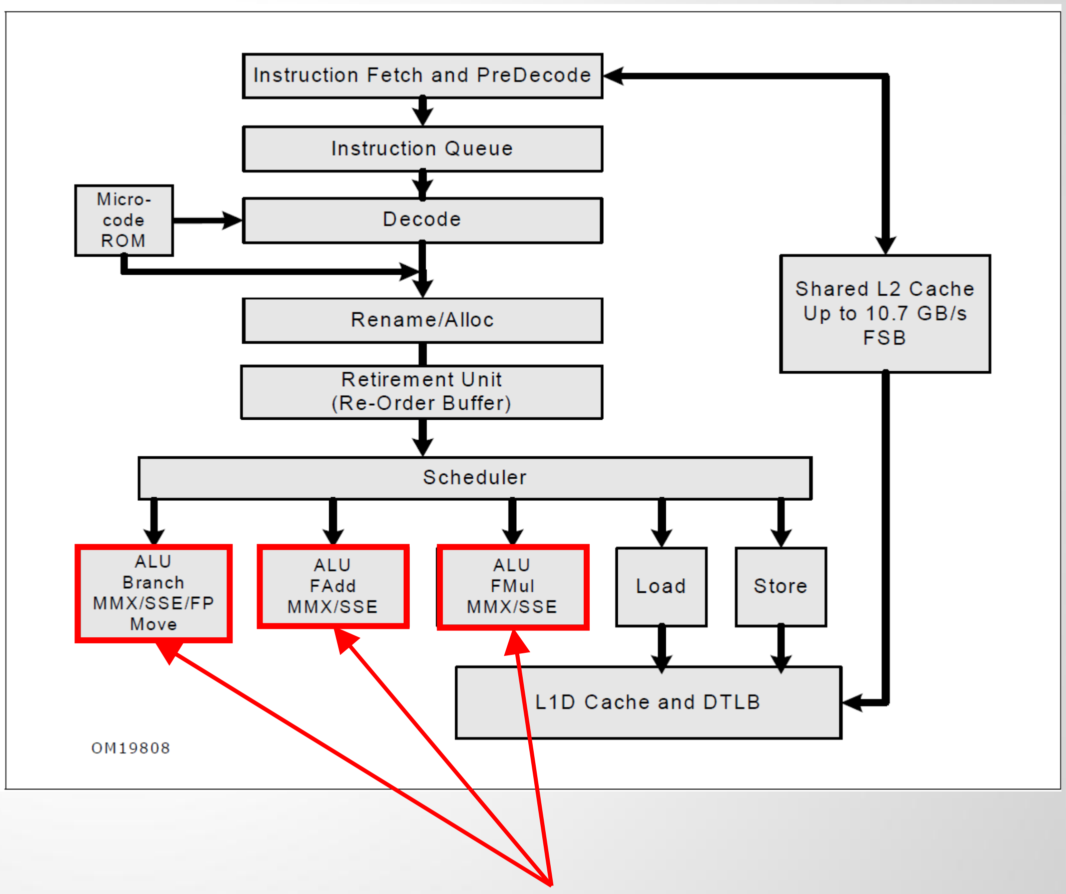


ARM Cortex-A8 Processor and NEON SIMD coprocessor

Intel SSE SIMD Architecture

- *Streaming SIMD Extensions*
- 16 128-bit registers
- SIMD instructions executed along with sequential instructions
- Adds floating point operations to Intel's MMX SIMD

Intel Core Architecture

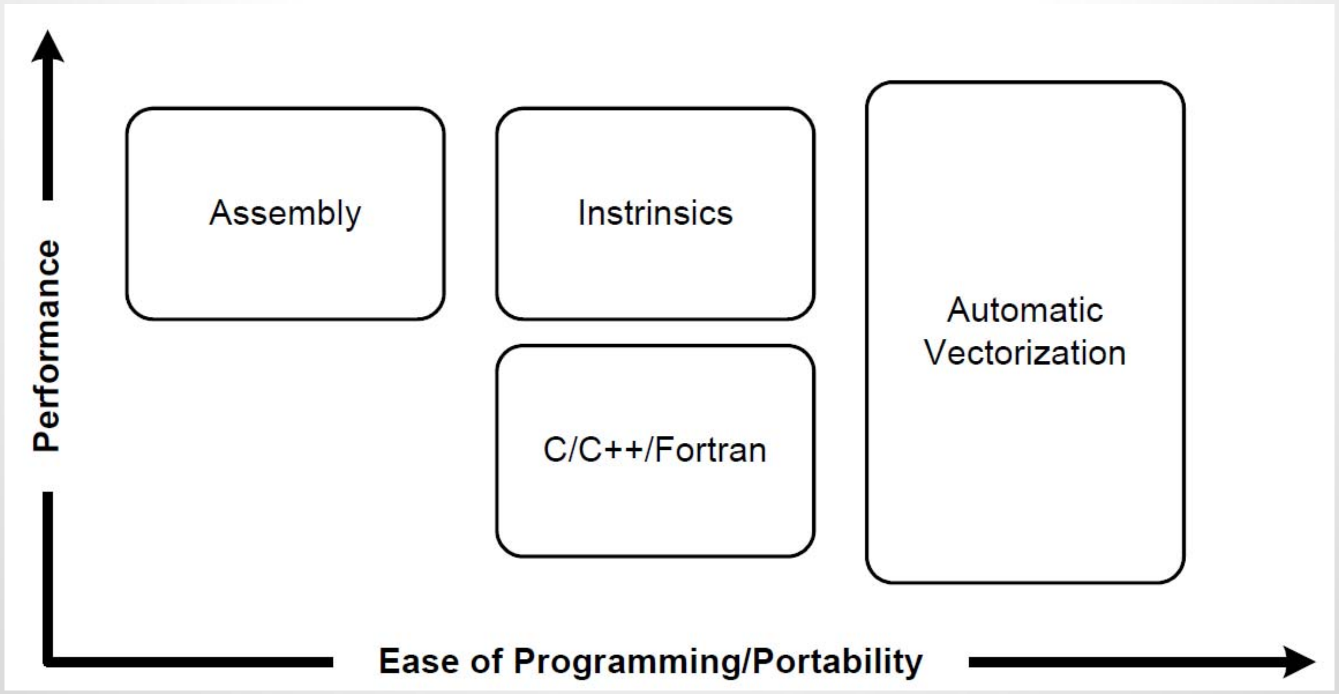


Instruction Fetch and PreDecode
Instruction Queue
Micro-code ROM
Decode
Rename/Alloc
Retirement Unit (Re-Order Buffer)
Shared L2 Cache Up to 10.7 GB/s FSB
Scheduler
ALU Branch MMX/SSE/FP Move
ALU FAdd MMX/SSE
ALU FMul MMX/SSE
Load
Store
L1D Cache and DTLB
OM19808

Combined SSE Functional Units

Software Programming

- Intel and ARM both have vectorizing compilers which will compile code using SIMD instructions
- Many audio/video SIMD libraries available
- To achieve best performance, custom coding at the assembly level should be used



Specialized Instructions

- NEON SIMD
  - VZIP – Interleaves two vectors
  - VMLA – Multiply and accumulate
  - VRECPE – Reciprocal estimate
  - VRSQRTE – Reciprocal square root estimate
- Intel SSE4
  - PAVG – Vector average
  - DPPS, DPPD – Dot product
  - PREFETCHT0 – Prefetch data into all cache levels
  - MONITOR, MWAIT – Used to synchronize across threads

Performance Impact of SIMD

- NEON on Cortex-A8 with gcc compiler
- Applied to an image warping algorithm
  - Mapping a pixel from a source to destination image by an offset
  - Calculated on four pixels in parallel (max speedup = 4)
- Two vectorization methods
  - Intrinsics - Using intrinsic functions to vectorize
  - Manual - Vectorizing using instructions at the assembly level

	Original	Intrinsics	Manual
Execution time (s)	10.01	4.56	3.24
Speedup	1.000	2.195	3.090

Comparison of different SIMD programming methods

Performance Impact of SIMD

- SSE on Intel i7 and AltiVec on IBM Power 7 processors
- SIMD applied to Media Bench II which contains multimedia applications for encoding/decoding media files (JPEG, H263, MPEG2)
- Tested three compilers with three methods:
  - Auto Vectorization - No changes to code
  - Transformations - Code changes to help compiler vectorize
  - Intrinsics - Functions which compile to SIMD

Method	XLC	ICC	GCC
Auto Vectorization	1.66 (52.94%)	1.84 (71.77%)	1.58 (44.71%)
Transformations	2.97	2.38	-
Intrinsics	3.15	2.45	-

Average speedup and parallelizable loops for Media Bench II

Conclusions

- Vector processors provided the early foundation for processing large amounts of data in parallel
- Vector processing techniques can still be found in video game consoles and graphics accelerators
- SIMD extensions are a decendant of vector processors and included in most modern processors
- Challenging programming and Amdahl’s Law are the main factors limiting the performance of SIMD

Questions?