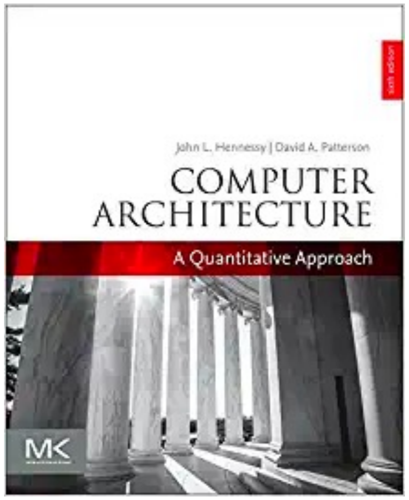


Chapter 4

Data-Level Parallelism (DLP) in Vector, SIMD, and GPU Architectures

Part 3: SIMD Extensions and GPU Intro

"We call these algorithms *data parallel* algorithms because their parallelism comes from simultaneous operations across large sets of data, rather than from multiple thread of control."
- W. Daniel Hillis and Guy L. Steele
"Data Parallel Algorithms," *Comm. ACM* (1986)

"If you were plowing a field, which would you rather use, two strong oxen or 1024 chickens?"
- Seymour Cray, Father of the Supercomputer
(arguing for two powerful vector processors
versus many simple processors)

Acknowledgements

- Thanks to many sources for slide material
  - © 1990 Morgan Kaufmann Publishers, © 2001-present Elsevier
    Computer Architecture: A Quantitative Approach by J. Hennessy & D. Patterson
  - © 1994 Morgan Kaufmann Publishers, © 2001-present Elsevier
    Computer Organization and Design by D. Patterson & J. Hennessy
  - © 2002 K. Asinovic & Arvind, MIT
  - © 2002 J. Kubiatowicz, University of California at Berkeley
  - © 2006, © 2010 No Starch Press for Inside the Machine by J. Stokes
  - © 2007 W.-M. Hwu & D. Kirk, University of Illinois & NVIDIA
  - © 2007-2010 J. Owens, University of California at Davis
  - © 2010 CRC Press for Introduction to Concurrency in Programming Languages by M. Sottile, T. Mattson, and C. Rasmussen
  - © 2017, IBM POWER9 Processor Architecture by Sadasivam et al., IBM
  - © 2016, © 2019 POWER9 Processor User’s Manual, IBM
  - © The OpenPOWER Foundation
  - © 2022, W. Feng, Virginia Tech

SIMD Extensions

- Media applications operate on data types narrower than the native word size
  - Examples: graphics and sound

- Major limitations (compared to vector instructions)
  - Number of data operands encoded into op code
    - Consequence?
  - No sophisticated addressing modes (strided, scatter-gather)
    - Consequence?
  - No mask registers
    - Consequence?

SIMD Instruction Set Extensions for Multimedia

SIMD Implementations

- Implementations
  - Intel MMX (1996)
    - Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Eight 16-bit integer ops
    - Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector Extensions (2010)
    - Four 64-bit integer/fp ops

  - Operands must be consecutive and aligned memory locations

SIMD Instruction Set Extensions for Multimedia

Example SIMD Code: DAXPY

```
   L.D         F0,a           ;load scalar a
   MOV         F1, F0         ;copy a into F1 for SIMD MUL
   MOV         F2, F0         ;copy a into F2 for SIMD MUL
   MOV         F3, F0         ;copy a into F3 for SIMD MUL
   DADDIU      R4,Rx,#512     ;last address to load
Loop:
   L.4D        F4,0[Rx]       ;load X[i], X[i+1], X[i+2],
   X[i+3]
   MUL.4D      F4,F4,F0
        ;a×X[i],a×X[i+1],a×X[i+2],a×X[i+3]
   L.4D        F8,0[Ry]       ;load Y[i], Y[i+1], Y[i+2],
   Y[i+3]
   ADD.4D      F8,F8,F4       ;a×X[i]+Y[i], ...,
   a×X[i+3]+Y[i+3]
   S.4D        0[Ry],F8       ;store into Y[i], ..., Y[i+3]
   DADDIU      Rx,Rx,#32      ;increment index to X
   DADDIU      Ry,Ry,#32      ;increment index to Y
   DSUBU       R20,R4,Rx      ;compute bound
   BNEZ        R20,Loop       ;check if done
```

SIMD Instruction Set Extensions for Multimedia

SSE (Streaming SIMD)

- Intel 4-wide floating point
- Pentium III: 2 fully-pipelined, SIMD, single-precision FP units
- Eight 128-bit registers added to ISA
- In HW, (historically) broke 4-wide instructions into two 2-wide instructions
  - Interrupts are a problem
- MMX + SSE: added 10% to PIII die

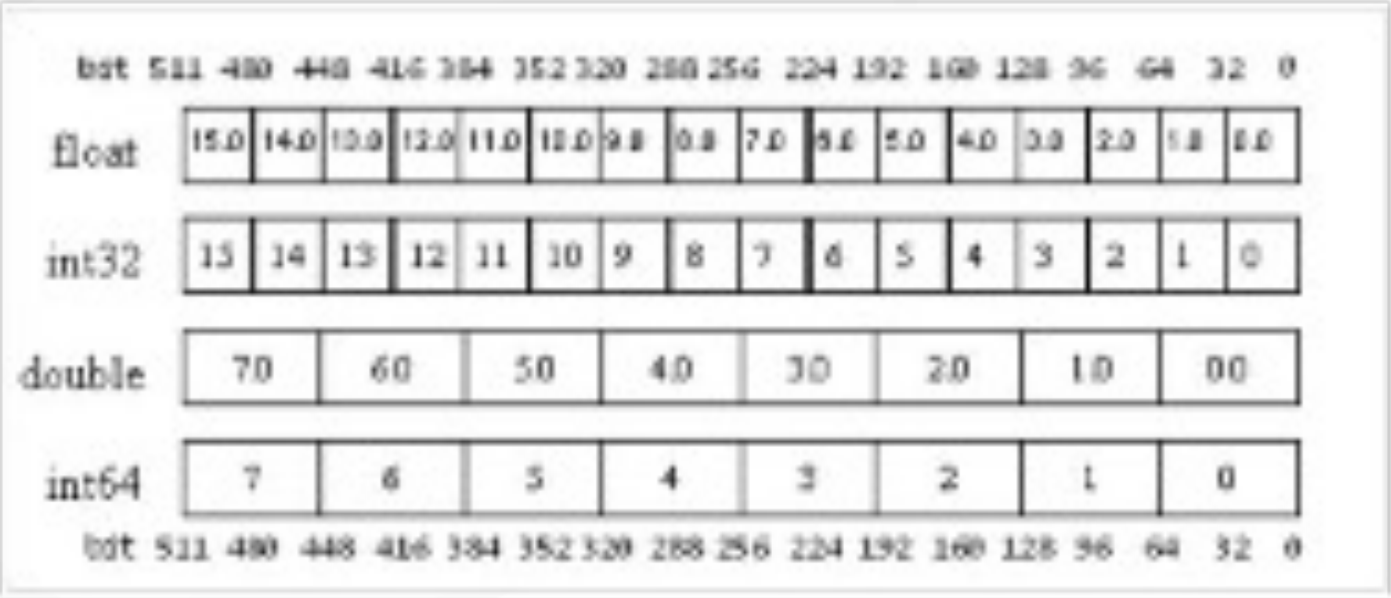
SIMD Instruction Set Extensions for Multimedia

Multimedia Extensions

- Very short vectors added to existing ISAs for micros
- Usually 64-bit registers split into 2x32b or 4x16b or 8x8b
  - Newer designs have 128-bit registers (Altivec, SSE2)
- Limited instruction set:
  - no vector length control
  - no strided load/store or scatter/gather
  - unit-stride loads must be aligned to 64/128-bit boundary
- Limited vector register length:
  - Requires superscalar dispatch to keep multiply/add/load units busy
  - Requires loop unrolling to hide latencies but increases register pressure
  - Trend towards fuller vector support in microprocessors

Larrabee Vectors
(Intel's Initial "GPU" Offering)

- 32 512b-wide new registers
- 8 16b-wide vector mask registers
- Register ops generally ternary (a=b op c)
- Extensive use of masks



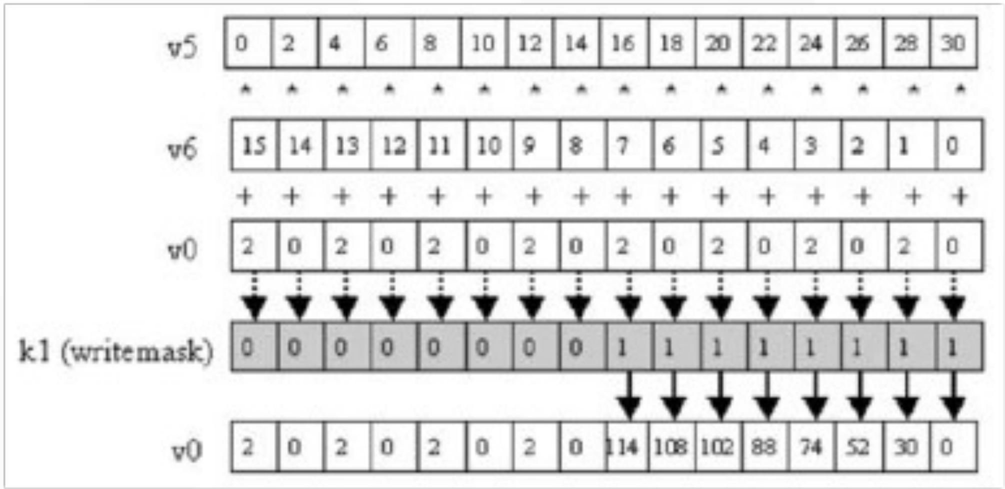
bit 511 480 448 416 384 352 320 288 256 224 192 160 128 96 64 32 0

float	15.0	14.0	13.0	12.0	11.0	10.0	9.0	8.0	7.0	6.0	5.0	4.0	3.0	2.0	1.0	0.0
int32	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
double	7.0		6.0		5.0		4.0		3.0		2.0		1.0		0.0	
int64	7		6		5		4		3		2		1		0	

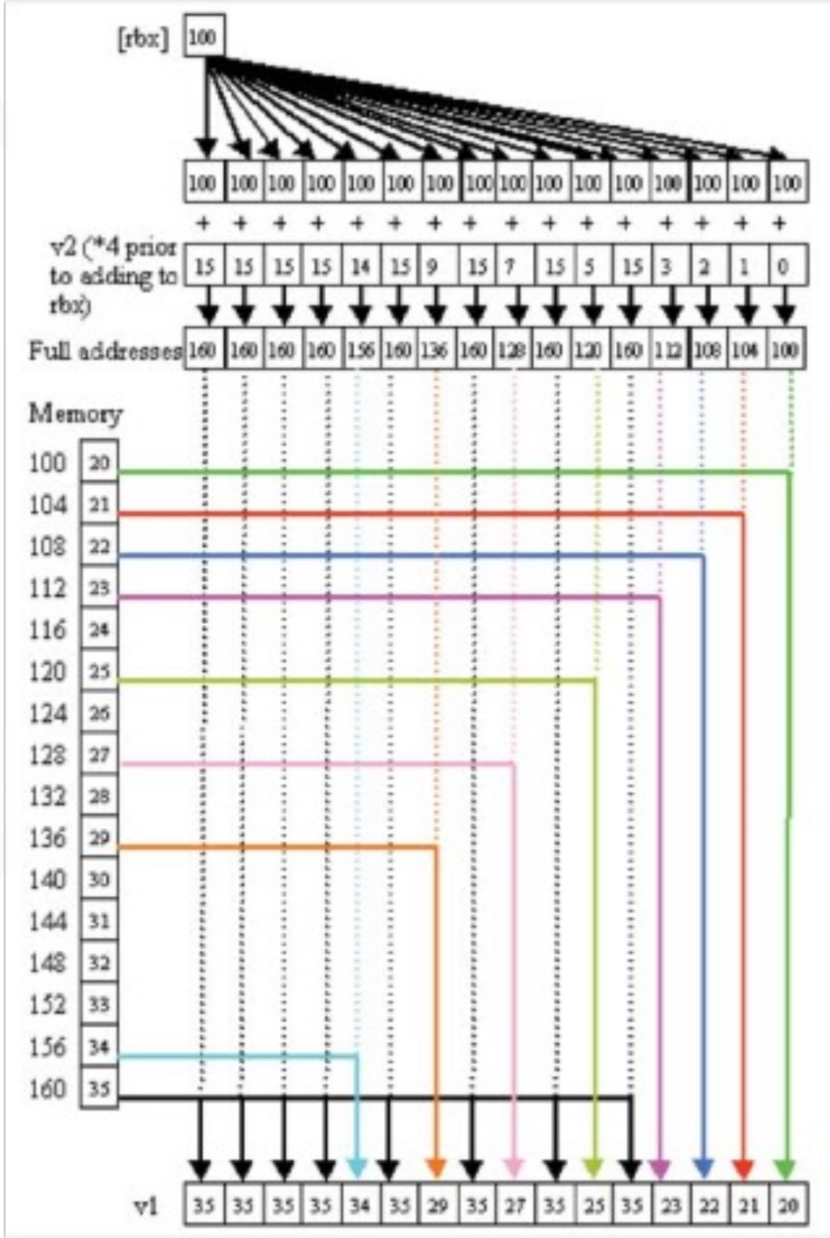
bit 511 480 448 416 384 352 320 288 256 224 192 160 128 96 64 32 0

Larrabee Example

```
vmadd231ps v0 {k1}, v5, v6
```



v5	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
v6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
v0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0
k1 (writemask)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
v0	2	0	2	0	2	0	2	0	114	108	102	88	74	52	30	0



[rbx]	100

	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
v2 (*4 prior to adding to rbx)	15	15	15	15	14	15	9	15	7	15	5	15	3	2	1	0
Full addresses	160	160	160	160	156	160	136	160	128	160	120	160	112	108	104	100

Memory

100	20
104	21
108	22
112	23
116	24
120	25
124	26
128	27
132	28
136	29
140	30
144	31
148	32
152	33
156	34
160	35

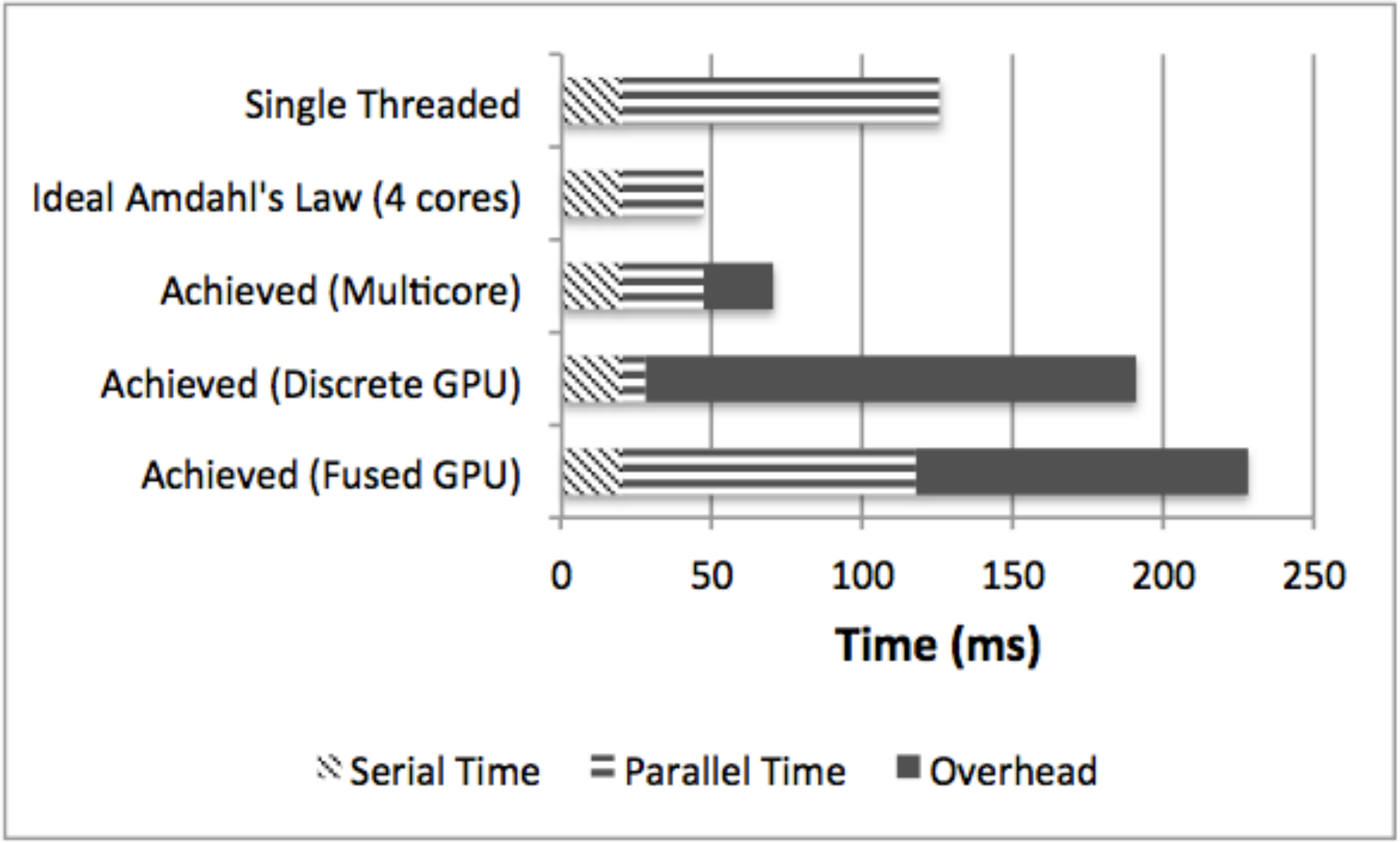
v1	35	35	35	35	34	35	29	35	27	35	25	35	23	22	21	20

Graphical Processing Units

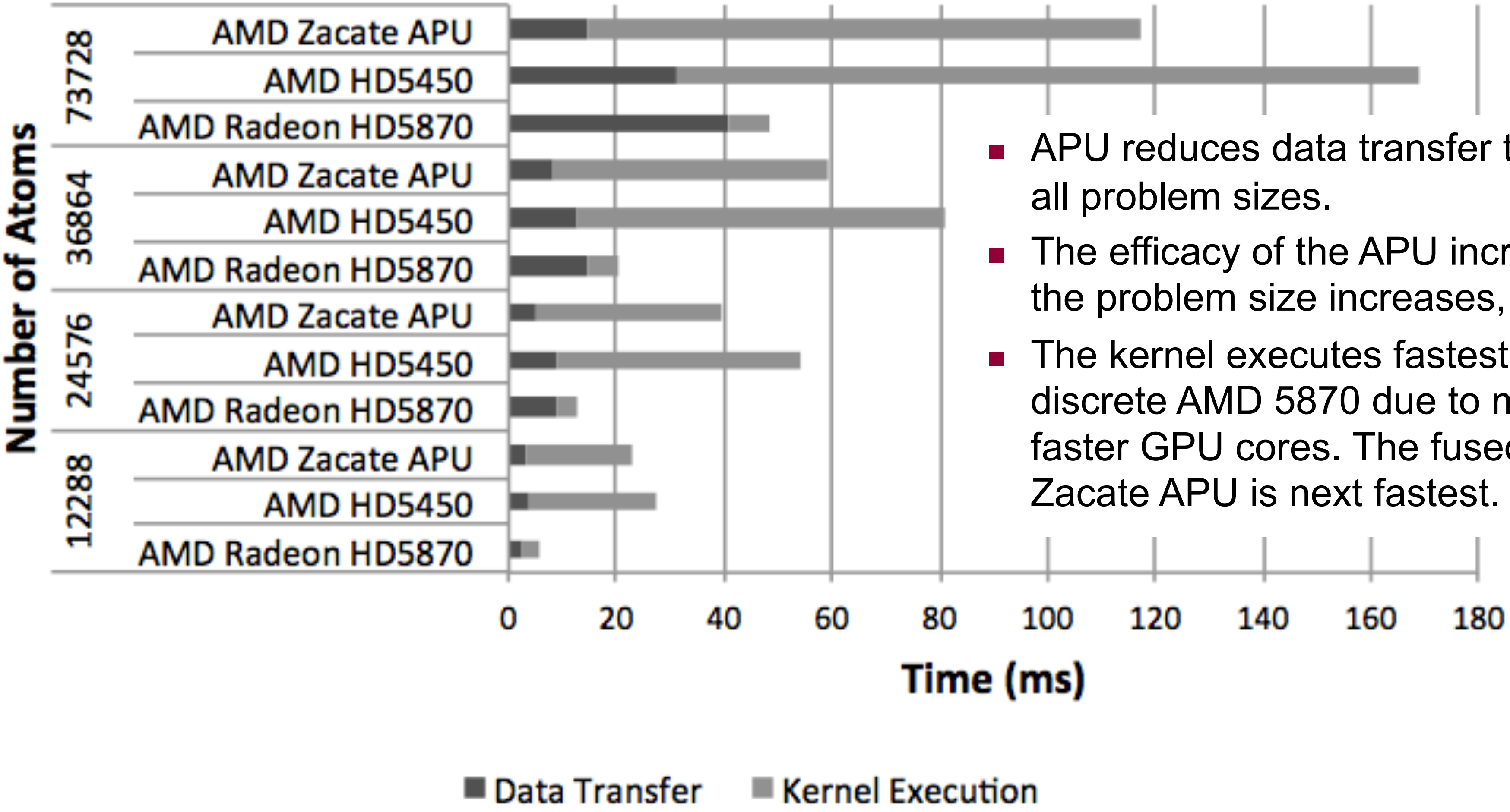
- Given the hardware invested to do graphics well, how can be supplement it to improve performance of a wider range of applications?

- Basic Idea
  - Heterogeneous execution model
    - CPU is the *host*, GPU is the *device*
    - What makes this model challenging?
  - Develop a C/C++-like programming language for GPU
  - Unify all forms of GPU parallelism as *CUDA thread* (NVIDIA)
    - OpenCL: Emerging vendor-independent language for multiple platforms
  - Programming model is "Single Instruction Multiple Thread"

Amdahl's Law:
Different Parallel Devices

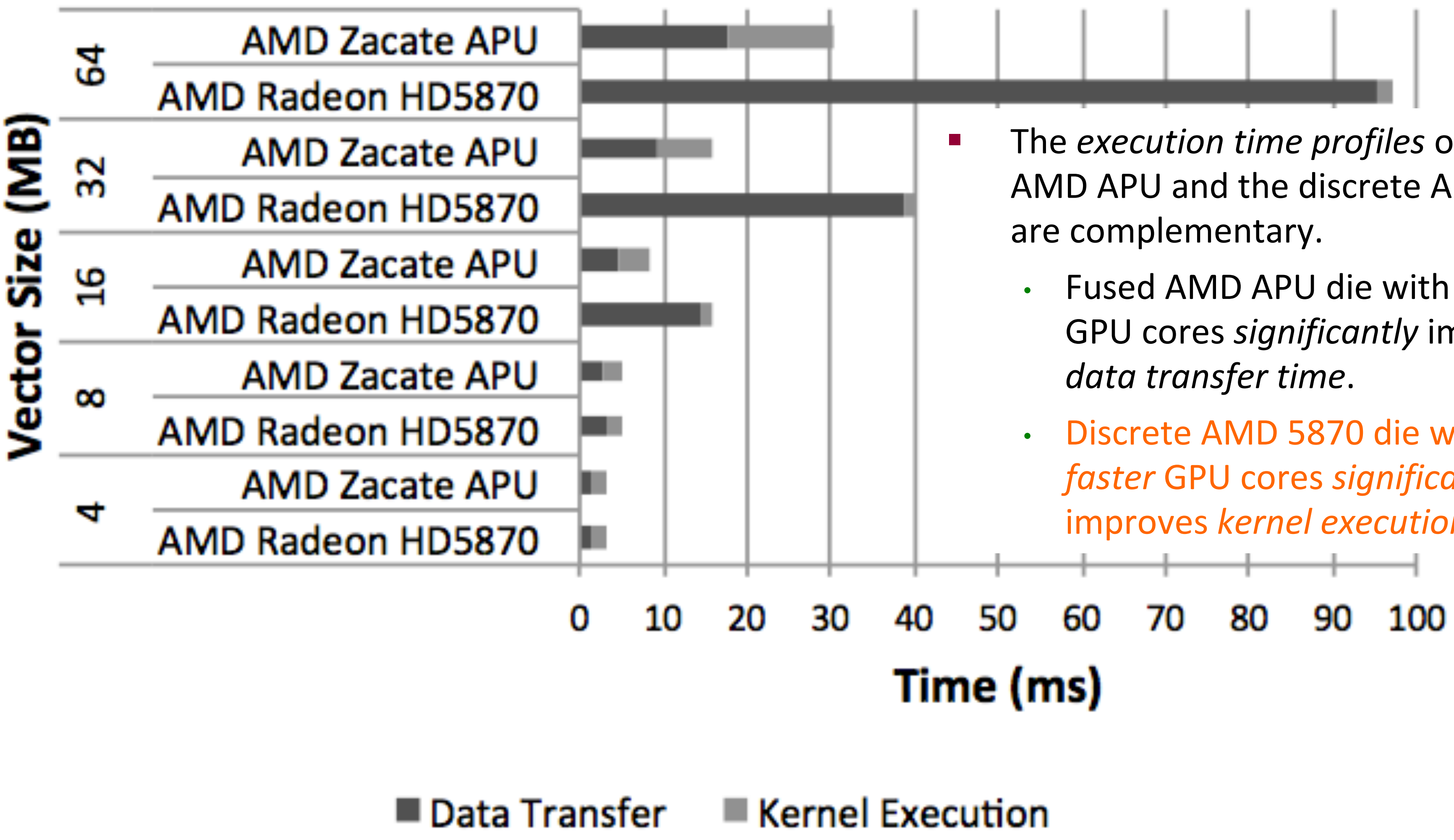


Performance:
Molecular Modeling (N-Body)



- APU reduces data transfer times for all problem sizes.
- The efficacy of the APU increases as the problem size increases, but …
- The kernel executes fastest on discrete AMD 5870 due to more and faster GPU cores. The fused AMD Zacate APU is next fastest.

Performance:
Reduction (Dense Linear Algebra)

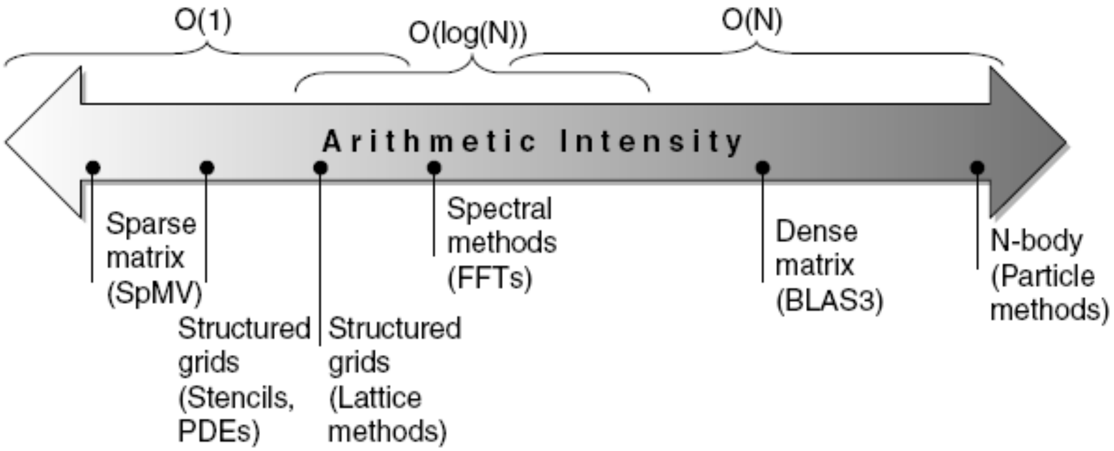


- The *execution time profiles* of the fused AMD APU and the discrete AMD 5870 are complementary.
  - Fused AMD APU die with only 80 GPU cores *significantly* improves *data transfer time*.
  - Discrete AMD 5870 die with 1600 *faster* GPU cores *significantly* improves *kernel execution time.*

Roofline Performance Model

[Williams et al., 2009]

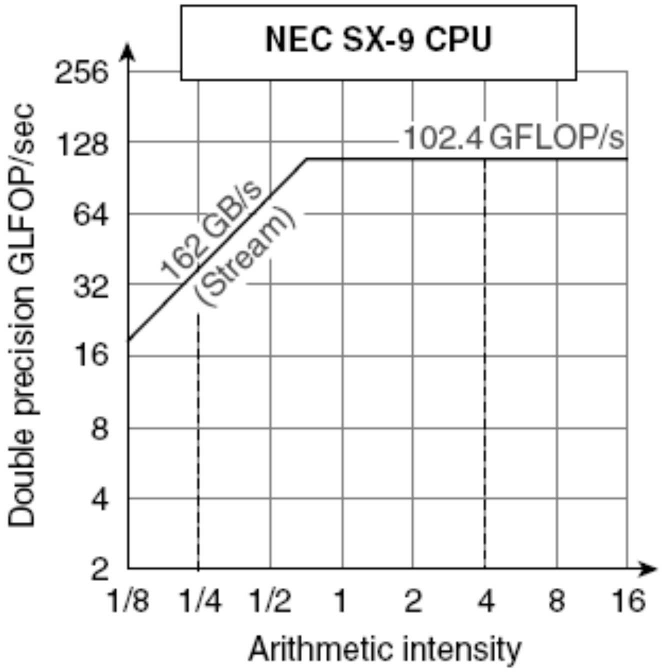
- Basic Idea
  - Plot peak floating-point throughput as a function of arithmetic intensity
  - Ties together floating-point performance, memory performance for a target machine, and arithmetic intensity
- Arithmetic Intensity
  - Floating-point operations per byte read

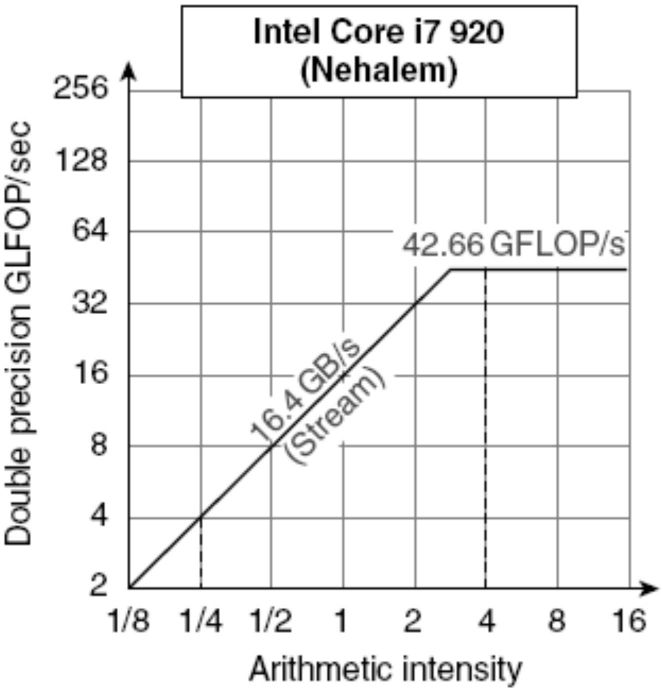


SIMD Instruction Set Extensions for Multimedia

Examples

- Attainable GFLOPs/sec = min (Peak Memory BW × Arithmetic Intensity, Peak Floating Point Perf.)


NEC SX-9 CPU
Double precision GLFOP/sec
102.4 GFLOP/s
162 GB/s
(Stream)
1/8
1/4
1/2
Arithmetic intensity


Intel Core i7 920
(Nehalem)
Double precision GLFOP/sec
42.66 GFLOP/s
16.4 GB/s
(Stream)
1/8
1/4
1/2
Arithmetic intensity