CS250P: Computer Systems Architecture
Explicit Parallelism

Sang-Woo Jun

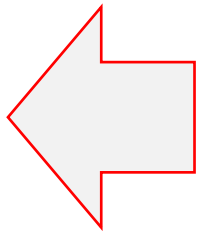
Fall 2023

UCI

Large amount of material adapted from MIT 6.004, “Computation Structures”, Morgan Kaufmann “Computer Organization and Design: The Hardware/Software Interface: RISC-V Edition”, and CS 152 Slides by Isaac Scherson

Modern Processor Topics - Performance

- Transparent Performance Improvements
  - Pipelining, Caches
  - Superscalar, Out-of-Order, Branch Prediction, Speculation, …
  - Covered in CS250A and others
- Explicit Performance Improvements
  - SIMD extensions, AES extensions, …
  - …



SIMD operations

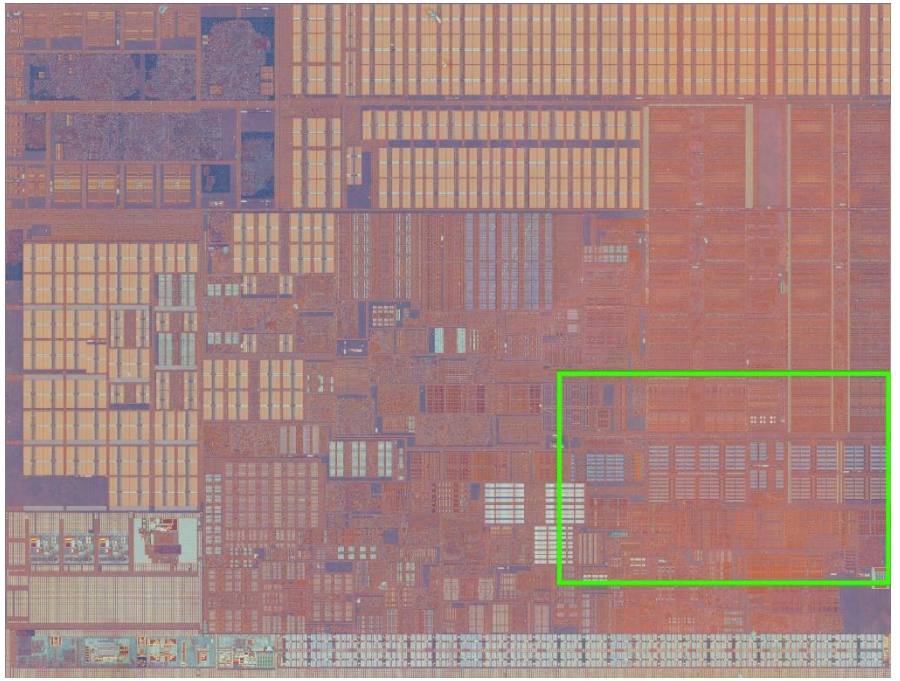
- Single ISA instruction performs same computation on multiple data
- Typically implemented with special, wider registers
- Example operation:
  - Load 32 bytes from memory to special register X
  - Load 32 bytes from memory to special register Y
  - Perform addition between each 4-byte value in X and each 4 byte value in Y
  - Store the four results in special register Z
  - Store Z to memory

For i in (0 to 7): Z[i] = X[i] + Y[i];

- RISC-V SIMD extensions (P) is still being worked on (as of 2021)

Example: Intel SIMD Extensions

- More transistors (Moore's law) but no faster clock, no more ILP…
  - More capabilities per processor has to be explicit!
- New instructions, new registers
  - Must be used explicitly by programmer or compiler!
- Introduced in phases/groups of functionality
  - SSE – SSE4 (1999 –2006)
    - 128 bit width operations
  - AVX, FMA, AVX2, AVX-512 (2008 – 2019)
    - 256 – 512 bit width operations
  - F16C, and more to come?



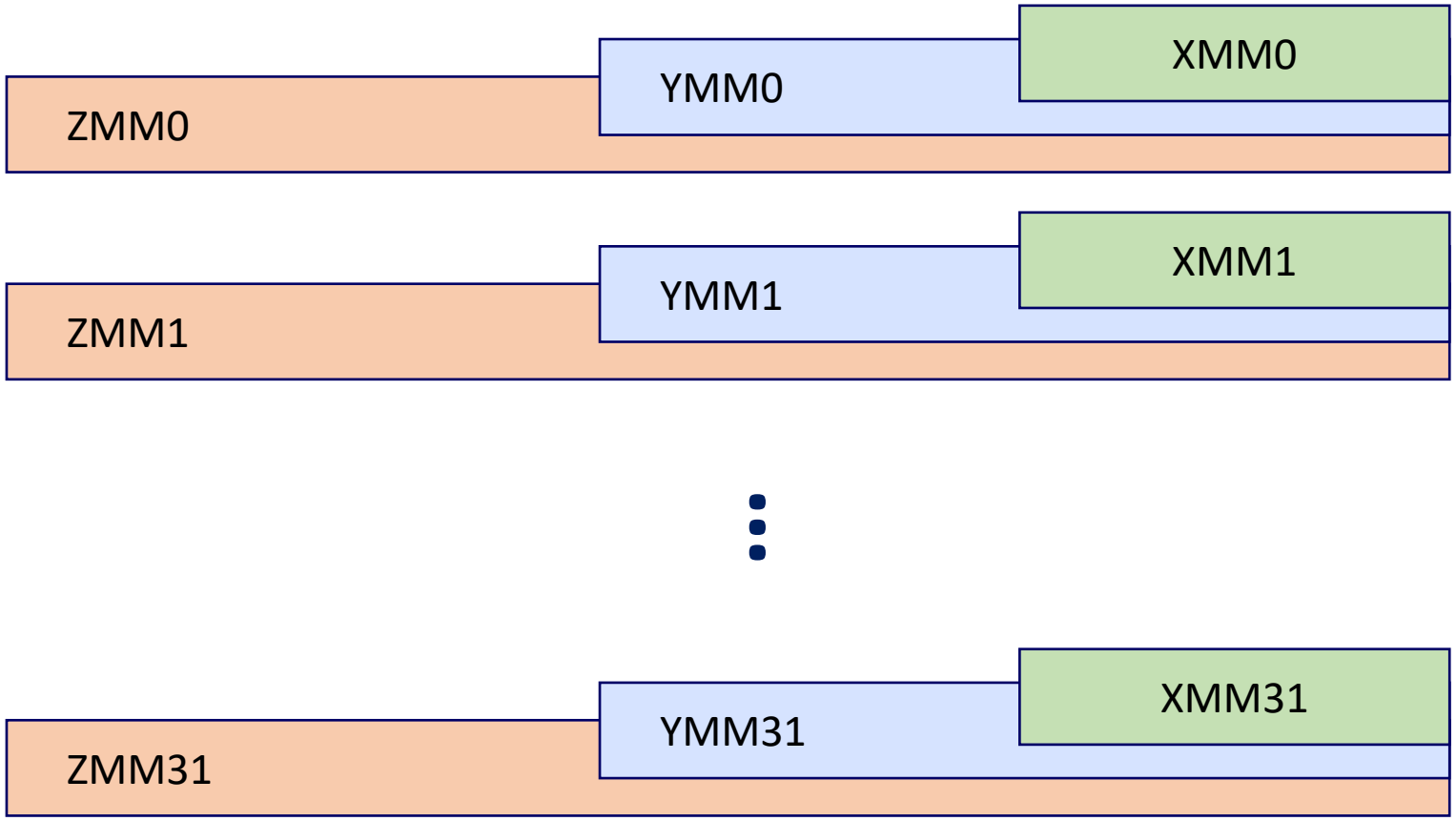
Performance matters, "AVX-512 Mask Registers, Again." 2020

Aside: Do I Have SIMD Capabilities?

- less /proc/cpuinfo

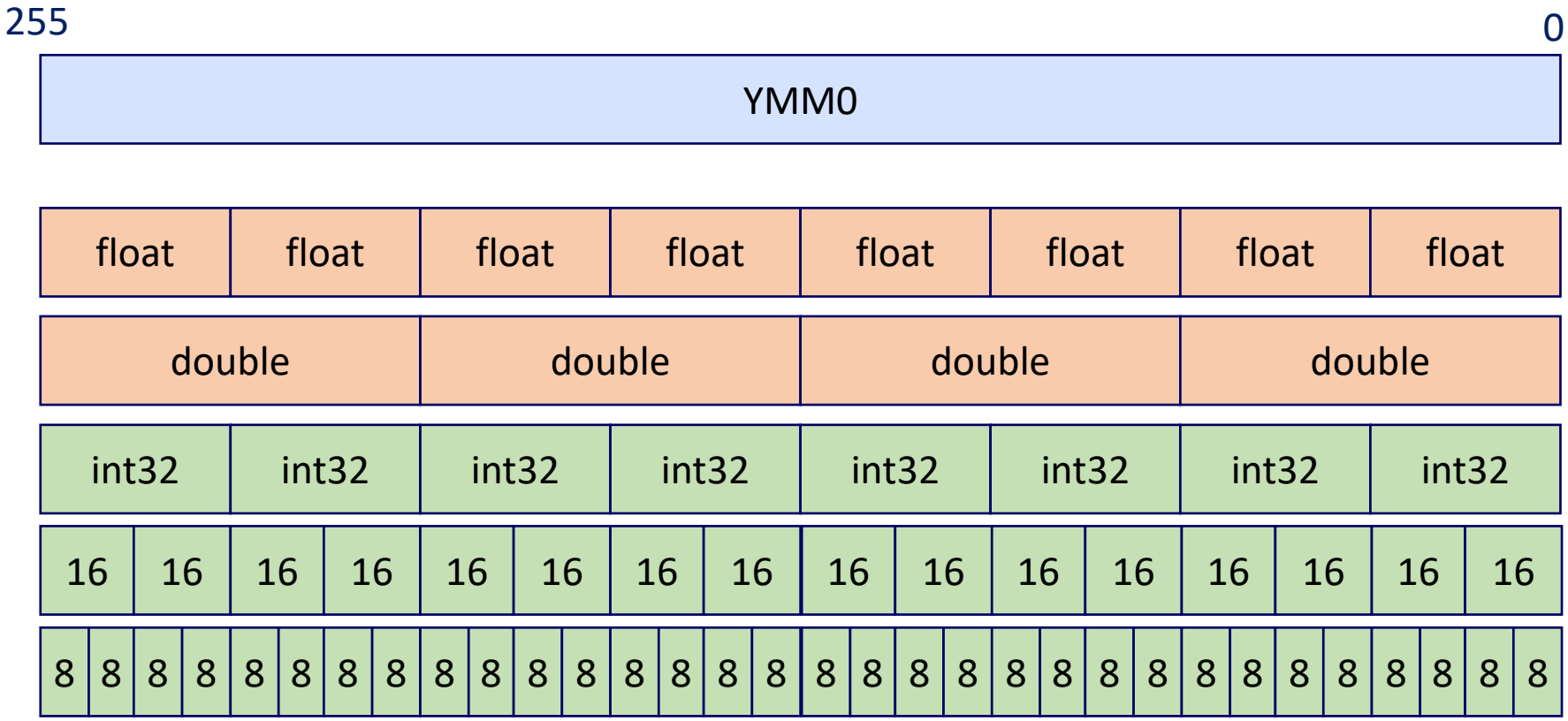
```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat p
se36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm con
stant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmp
erf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx1
6 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f
16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_single pti ssbd ibrs ibp
b stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2
 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves
dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp flush_l1d
```

Intel SIMD Registers (AVX-512)



- XMM0 – XMM15
  - 128-bit registers
  - SSE
- YMM0 – YMM15
  - 256-bit registers
  - AVX, AVX2
- ZMM0 – ZMM31
  - 512-bit registers
  - AVX-512

SSE/AVX Data Types

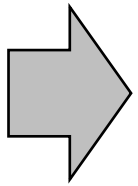


Operation on
32 8-bit values
in one instruction!

Compiler Automatic Vectorization

- In gcc, flags “-O3 -mavx -mavx2” attempts automatic vectorization
- Works pretty well for simple loops

```
int a[256], b[256], c[256];
void foo () {
    for (int i=0; i<256; i++) a[i] = b[i] * c[i];
}
```



```
.L2:
        vmovdqa xmm1, XMMWORD PTR b[rax]
        add     rax, 16
        vpmulld xmm0, xmm1, XMMWORD PTR c[rax-16]
        vmovaps XMMWORD PTR a[rax-16], xmm0
        cmp     rax, 1024
        jne     .L2
```

Generated using GCC explorer: https://gcc.godbolt.org/

- But not for anything complex
  - E.g., naïve bubblesort code not parallelized at all

Intel SIMD Intrinsics

- Use C functions instead of inline assembly to call AVX instructions
- Compiler manages registers, etc
- Intel Intrinsics Guide
  - https://software.intel.com/sites/landingpage/IntrinsicsGuide
  - One of my most-visited pages…

e.g.,

```
__m256 a, b, c;
__m256 d = _mm256_fmadd_ps(a, b, c); // d[i] = a[i]*b[i]+c[i] for i = 0 …7
```

Intrinsic Naming Convention

- _mm<width>_[function]_[type]
  - E.g., _mm256_fmadd_ps :
    perform fmadd (floating point multiply-add) on
    256 bits of
    packed single-precision floating point values (8 of them)

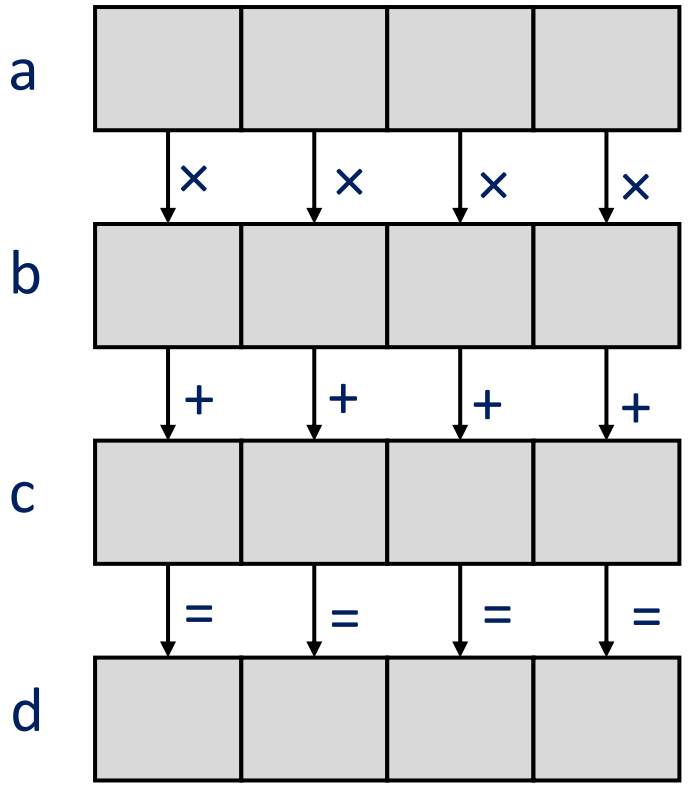
Width	Prefix
128	_mm_
256	_mm256_
512	_mm512_

Type	Postfix
Single precision	_ps
Double precision	_pd
Packed signed integer	_epiNNN (e.g., epi256)
Packed unsigned integer	_epuNNN (e.g., epu256)
Scalar integer	_siNNN (e.g., si256)

Not all permutations exist! Check guide

Example: Vertical Vector Instructions

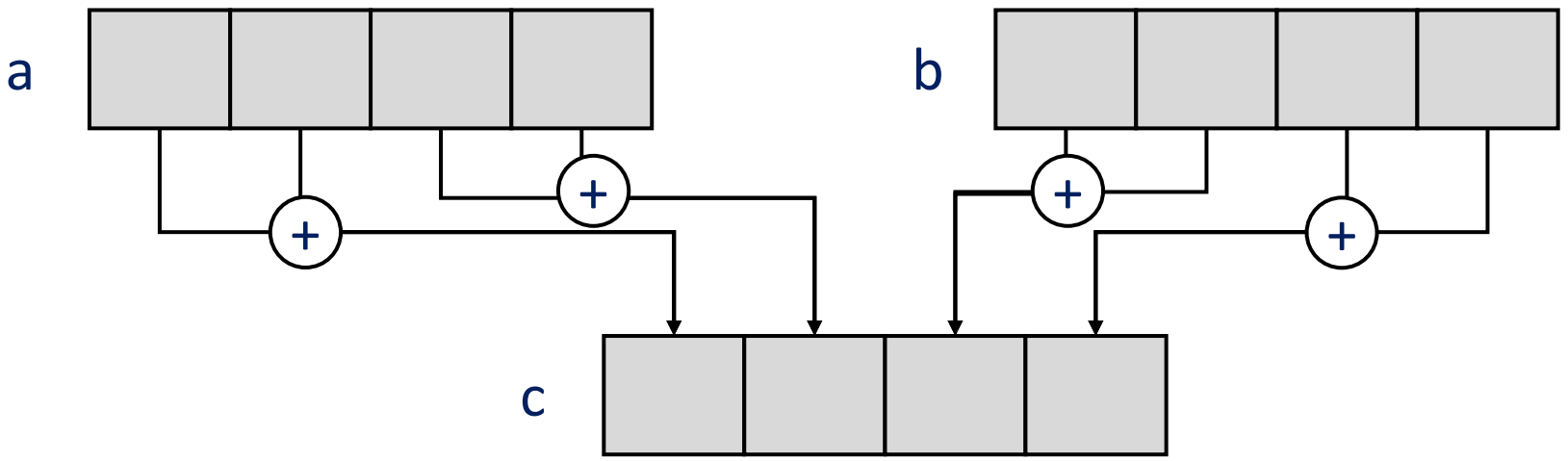
- Add/Subtract/Multiply
  - _mm256_add/sub/mul/div_ps/pd/epi
    - Mul only supported for epi32/epu32/ps/pd
    - Div only supported for ps/pd
    - Consult the guide!
- Max/Min/GreaterThan/Equals
- Sqrt, Reciprocal, Shift, etc…
- FMA (Fused Multiply-Add)
  - (a*b)+c, -(a*b)-c, -(a*b)+c, and other permutations!
  - Consult the guide!
- …



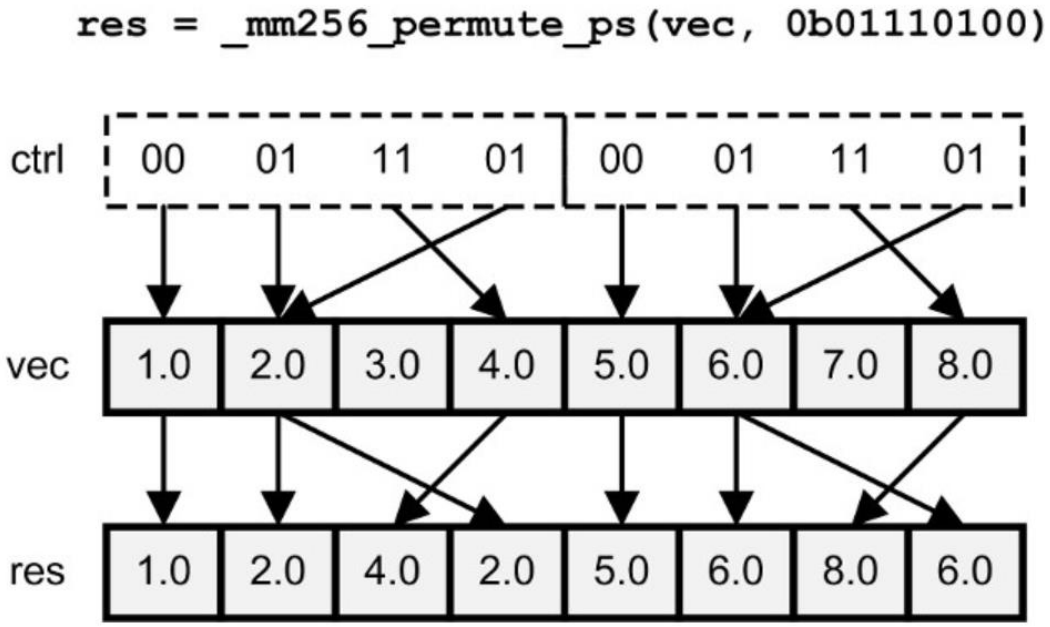

```
__m256 a, b, c;
__m256 d = _mm256_fmadd_pd(a, b, c);
```

Horizontal Vector Instructions

- Horizontal add/subtraction
  - Adds adjacent pairs of values
  - E.g., __m256d _mm256_hadd_pd (__m256d a, __m256d b)



Shuffling/Permutation

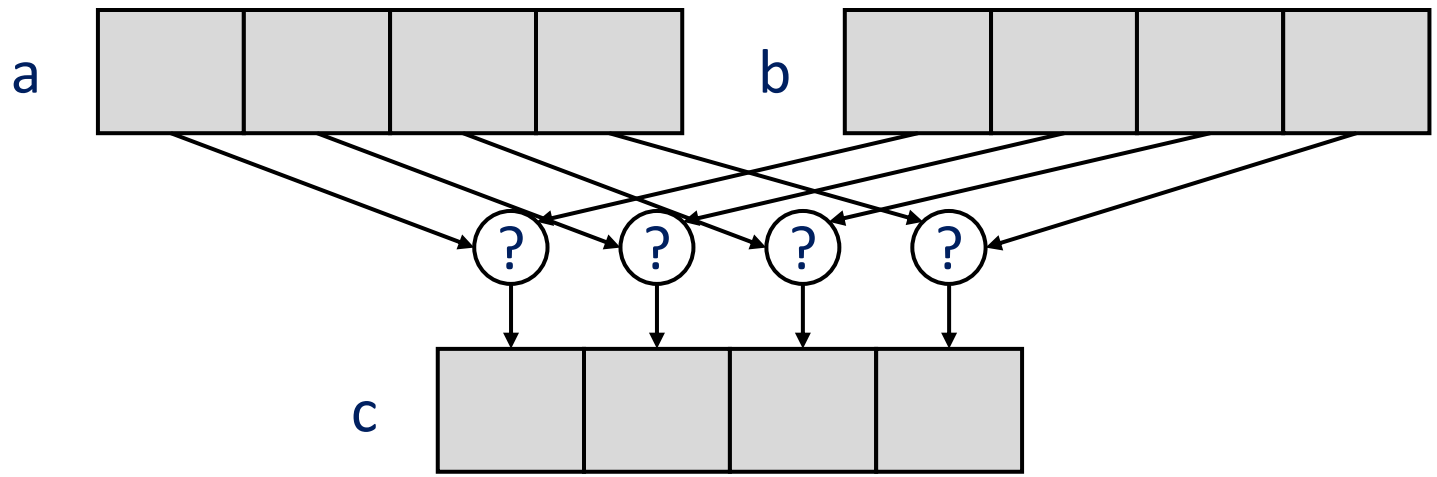


Matt Scarpino, "Crunching Numbers with AVX and AVX2," 2016

- Within 128-bit lanes
  - _mm256_shuffle_ps/pd/… (a,b, imm8)
  - _mm256_permute_ps/pd
  - _mm256_permutevar_ps/…
- Across 128-bit lanes
  - _mm256_permute2x128/4x64 : Uses 8 bit control
  - _mm256_permutevar8x32/… : Uses 256 bit control
- Not all type permutations exist for each type, but variables can be cast back and forth between types

Blend

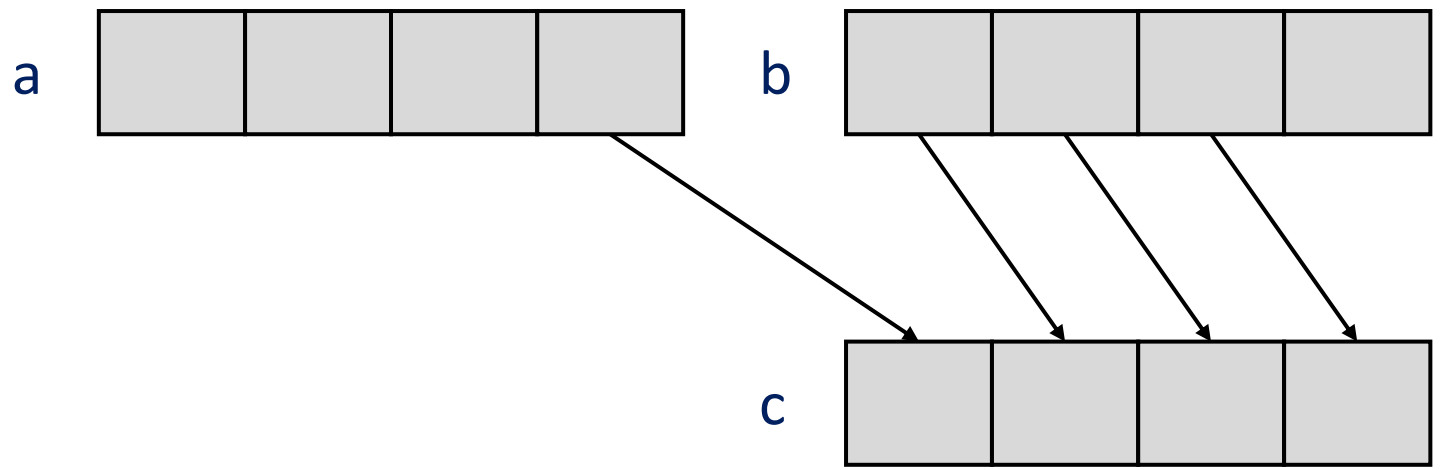
- Merges two vectors using a control
  - _mm256_blend_... : Uses 8 bit control
    - e.g., _mm256_blend_epi32
  - _mm256_blendv_... : Uses 256 bit control
    - e.g., _mm256_blendv_epi8



Alignr

- Right-shifts concatenated value of two registers, by byte
  - Often used to implement circular shift by using two same register inputs
  - _mm256_alignr_epi8 (a, b, count)

Example of 64-bit values being shifted by 8

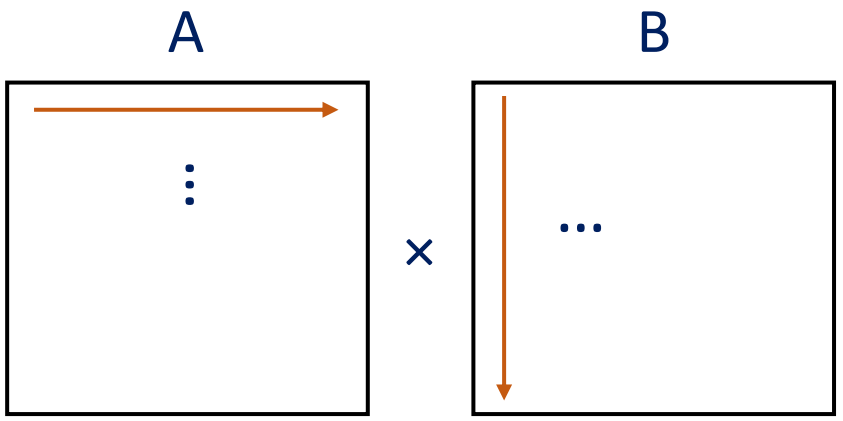


Helper Instructions

- Cast
  - __mm256i <-> __mm256, etc…
  - Syntactic sugar -- does not spend cycles
- Convert
  - 4 floats <-> 4 doubles, etc…
- Movemask
  - __mm256 mask to -> int imm8
- And many more…

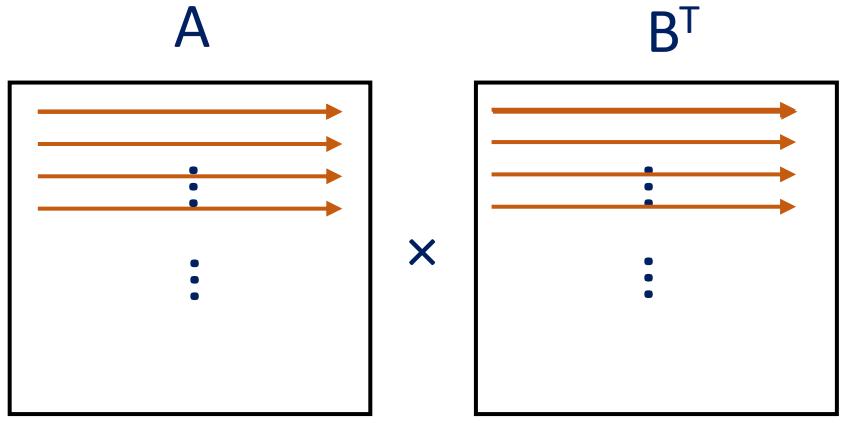
Case Study: Matrix Multiplication

- Remember simply transposing matrix B brought 6x performance
  - At that point, we are bottlenecked by single-thread processing performance
  - Adding SIMD gets us more!
  - After this we are again bottlenecked by memory, but that is for another time



63.19 seconds

VS



1.26 seconds

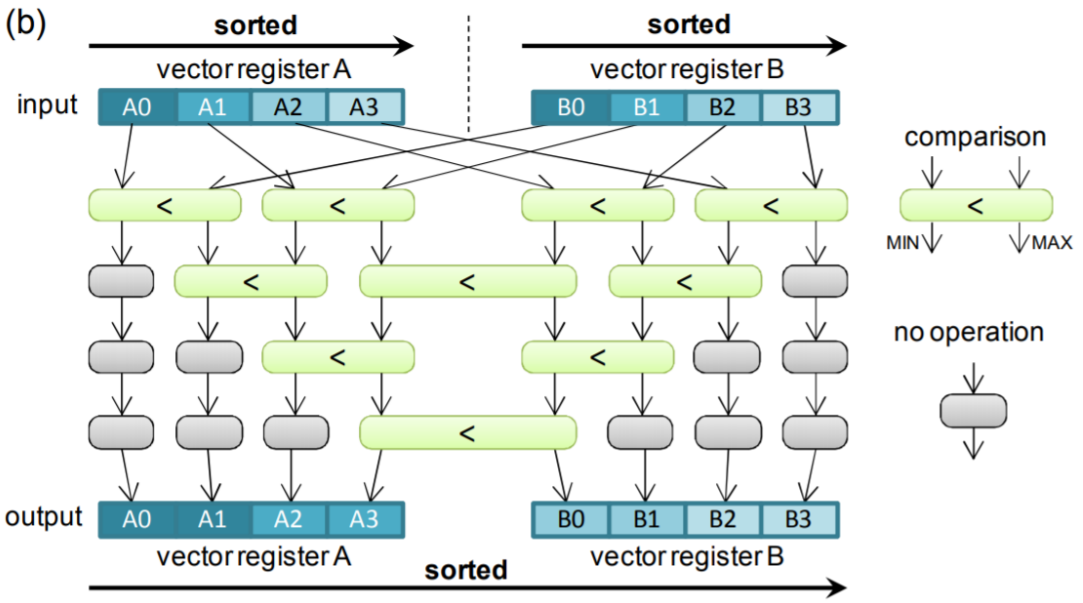
(50x performance!)

Case Study: Sorting

- Important, fundamental application!
- Can be parallelized via divide-and-conquer
- How can SIMD help?

The Two Register Merge

- Sort units of two pre-sorted registers, K elements
  - minv = A, maxv = B
  - // Repeat K times
    - minv = min(minv,maxv)
    - maxv = max(minv,maxv)
    - // circular shift one value down
    - minv = alignr(minv, minv, sizeof(int))



Inoue et.al., "SIMD- and Cache-Friendly Algorithm for Sorting an Array of Structures," VLDB 2015

SIMD And Merge Sort

- Hierarchically merged sorted subsections
- Using the SIMD merger for sorting
  - vector_merge is the two-register sorter from before

```
aPos = bPos = outPos = 0;
vMin = va[aPos++];
vMax = vb[bPos++];
while (aPos < aEnd && bPos < bEnd) {
  /*  merge vMin and vMax */
  vector_merge(vMin, vMax);

  /*  store the smaller vector as output*/
  vMergedArray[outPos++] = vMin;

  /*  load next vector and advance pointer   */
  /*  a[aPos*4] is first element of va[aPos] */
  /*  and b[bPos*4] is that of vb[bPos]      */
  if (a[aPos*4] < b[bPos*4])
      vMin = va[aPos++];
  else
      vMin = vb[bPos++];
}
```

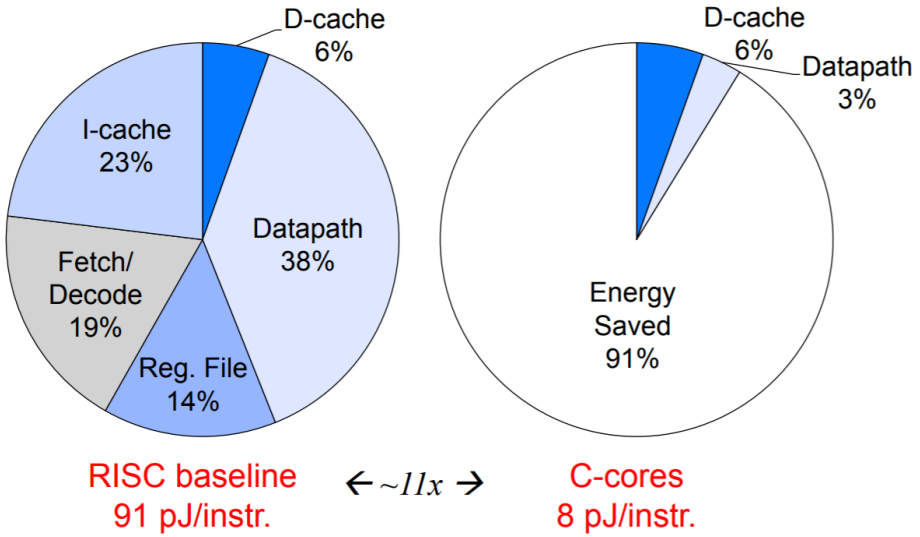
Inoue et.al., “SIMD- and Cache-Friendly Algorithm for Sorting an Array of Structures,” VLDB 2015

Topic Under Active Research!

- Papers being written about…
  - Architecture-optimized matrix transposition
  - Register-level sorting algorithm
  - Merge-sort
  - … and more!
- Good find can accelerate your application kernel Nx

Processor Microarchitectural Effects on Power Efficiency

- The majority of power consumption of a CPU is not from the ALU
  - Cache management, data movement, decoding, and other infrastructure
  - Adding a few more ALUs should not impact power consumption
- Indeed, 4X performance via AVX does not add 4X power consumption
  - From i7 4770K measurements:
  - Idle: 40 W
  - Under load : 117 W
  - Under AVX load : 128 W



Michael Taylor, "Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse," 2012