LINMA2710 - Scientific Computing Single Instruction Multiple Data (SIMD)

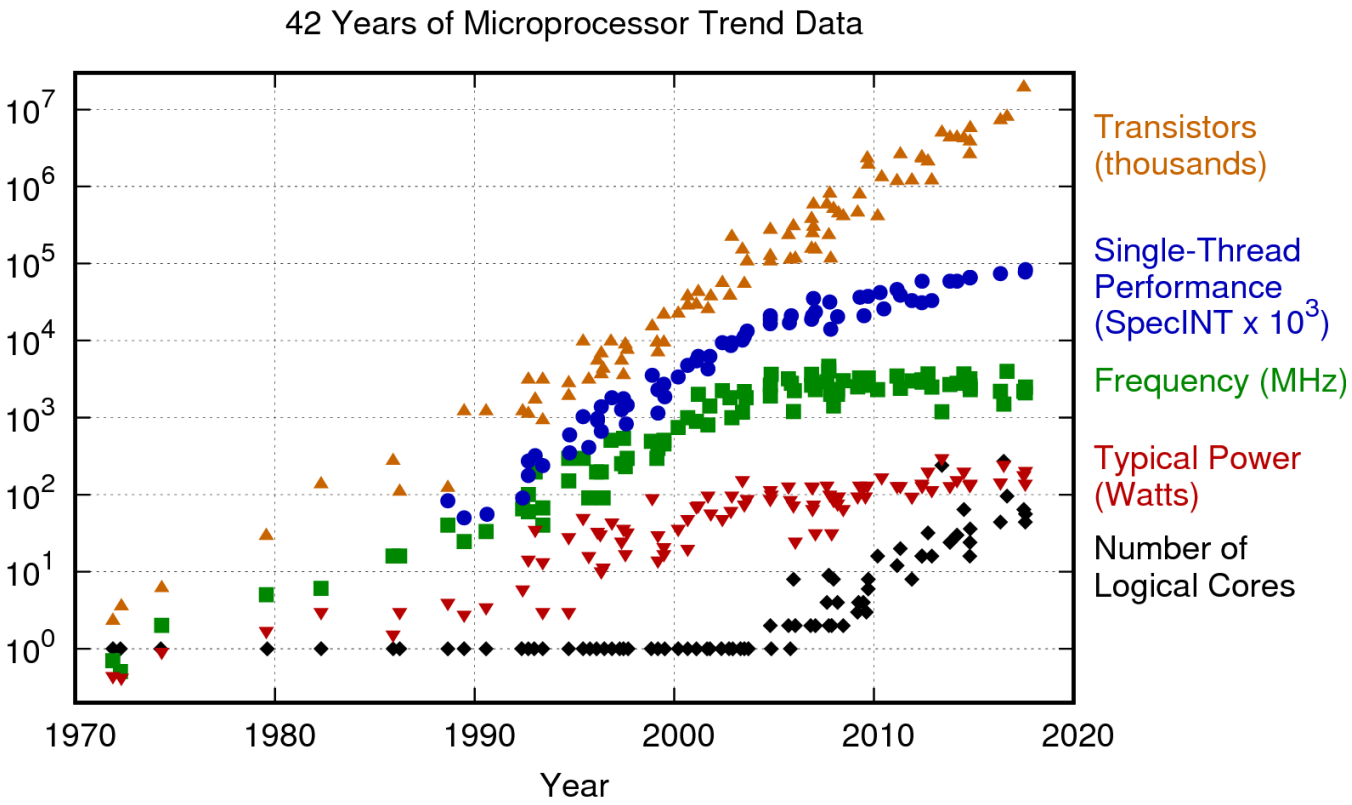
P.-A. Absil and B. Legat

☐ Full Width Mode ☐ Present Mode

Table of Contents

Motivation

The need for parallelism

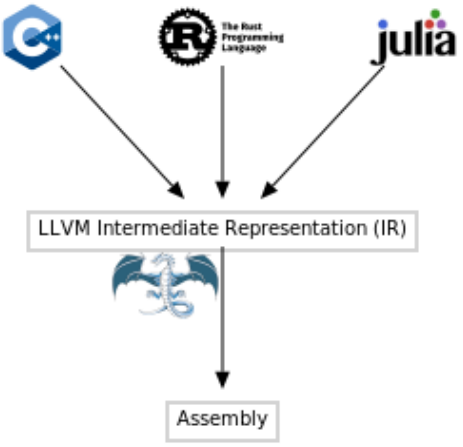


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Image source

A bit of historical context

- **1972** : C language created by Dennis Ritchie and Ken Thompson to ease development of Unix (previously developed in **assembly**)
- **1985** : C++ created by Bjarne Stroustrup
- **2003** : LLVM started at University of Illinois
- **2005** : Apple hires Chris Lattner from the university
- **2007** : He then creates the LLVM-based compiler Clang
- **2009** : Mozilla start developing an LLVM-based compiler for Rust
- **2009** : Development starts on Julia, with LLVM-based compiler



A sum function in C and Julia

```c
float sum(float *vec, int length) {
    float total = 0;
    for (int i = 0; i < length; i++) {
        total += vec[i];
    }
    return total;
}
```

```
c_sum(x::Vector{Cfloat}) = ccall(("sum", sum_float_lib), Cfloat, (Ptr{Cfloat}, Cint), x, length(x));
```

```
julia_sum (generic function with 1 method)
```

```
function julia_sum(v::Vector{T}) where {T}
    total = zero(T)
    for i in eachindex(v)
        total += v[i]
    end
    return total
end
```

Let's make a small benchmark

vec_float =

▸ [0.25398, 0.748738, 0.240008, 0.030562, 0.307776, 0.719664, 0.253093, 0.50683, 0.723948, 0

```
1 vec_float = rand(Float32, 2^16)
```

32742.973f0

```
1 @btime c_sum($vec_float)
```

```
242.509 μs (0 allocations: 0 bytes)
```

32742.973f0

```
1 @btime julia_sum($vec_float)
```

```
60.632 μs (0 allocations: 0 bytes)
```

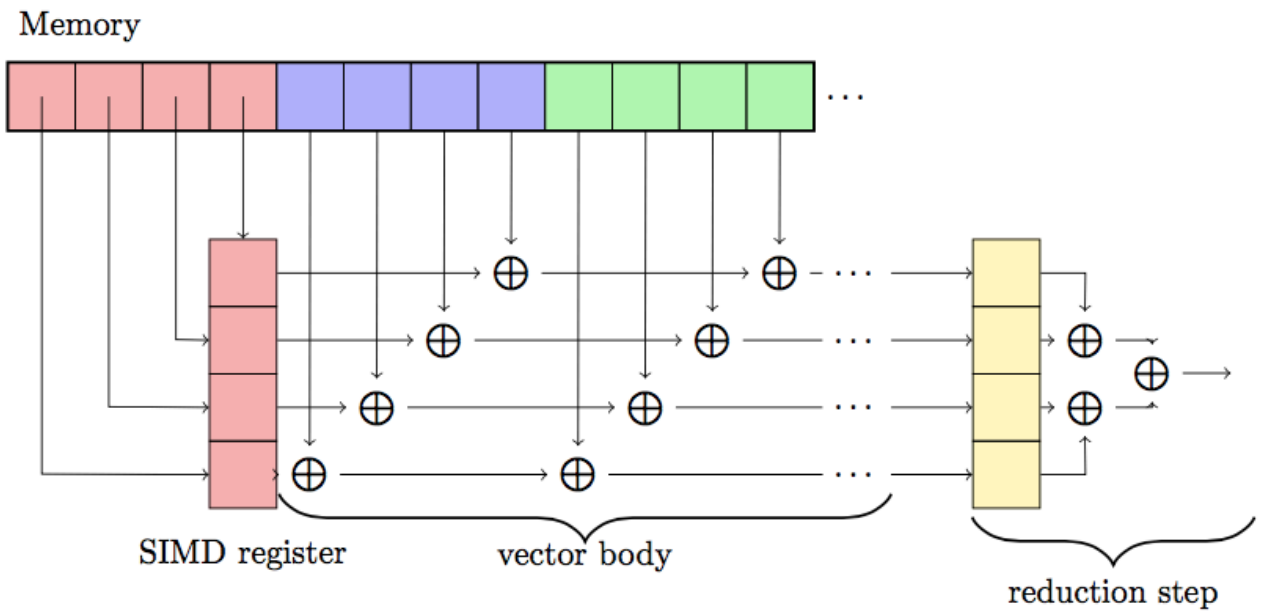
▸ **How to speed up the C code ?**

Tip

As accessing global variables is slow in Julia, it is important to add `$` in front of them when using `btime`. This is less critical in Pluto though as it handles global variables differently. To see why, try removing the `$`, you should see `1` allocations instead of zero.

```c
float sum(float *vec, int length) {
    float total = 0;
    for (int i = 0; i < length; i++) {
        total += vec[i];
    }
    return total;
}
```

Summing with SIMD



Faster Julia code

▶ **How to get the same speed up from the Julia code ?**

```
julia_sum_fast (generic function with 1 method)
```

```
function julia_sum_fast(v::Vector{T}) where {T}
    total = zero(T)
    for i in eachindex(v)
        @fastmath total += @inbounds v[i]
    end
    return total
end
```

```
32743.055f0
```

```
@btime julia_sum_fast($vec_float)
```

```
  2.546 μs (0 allocations: 0 bytes)
```

julia_sum_simd (generic function with 1 method)

```
function julia_sum_simd(v::Vector{T}) where {T}
    total = zero(T)
    @simd for i in eachindex(v)
        total += v[i]
    end
    return total
end
```

32743.055f0

```
@btime julia_sum_simd($vec_float)
```

```
  2.545 μs (0 allocations: 0 bytes)
```

Careful with fast math

▶ **Why are the three elements in the center of the vector ignored in this example ?**

test_kahan = ▶ [1.0, 2.98023f-8, 2.98023f-8, 2.98023f-8, 0.000119209]

```
test_kahan = Cfloat[1.0, eps(Cfloat)/4, eps(Cfloat)/4, eps(Cfloat)/4,
1000eps(Cfloat)]
```

1.000119298696518

```
sum(Float64.(test_kahan))
```

1.0001192f0

```
c_sum(test_kahan[[1, 5]])
```

1.0001192f0

```
c_sum(test_kahan)
```

To improve the accuracy this, we consider the Kahan summation algorithm.

1.0001193f0

```
c_sum_kahan(test_kahan)
```

Optimization level : -O0 ▾ Enable -ffast-math ? ☐

▶ **What happens when `-ffast-math` is enabled ?**

For further details, see this blog post.

Tip

`eps` gives the difference between `1` and the number closest to `1`. See also `prevfloat` and `nextfloat`.

```
float sum_kahan(float* vec, int length) {
    float total, c, t, y;
    int i;
    total = c = 0.0f;
    for (i = 0; i < length; i++) {
      y = vec[i] - c;
      t = total + y;
      c = (t - total) - y;
      total = t;
   }
   return total;
}
```

SIMD inspection

Instruction sets

The data is **packed** on a single SIMD unit whose width and register depends on the instruction set family. The single instruction is then run in parallel on all elements of this small **vector** stored in the SIMD unit. These give the prefix `vp` to the instruction names that stands from *Vectorized Packed*.

Instruction Set Family	Width of SIMD unit	Register
Streaming SIMD Extension (SSE)	128-bit	%xmm
Advanced Vector Extensions (AVX)	256-bit	%ymm
AVX-512	512-bit	%zmm

```
▸ ProcessChain([Process(`lscpu`, ProcessExited(0)), Process(`grep Flag`, ProcessExited(0))
1 run(pipeline(`lscpu`, `grep Flag`))
```

```
Flags:                           fpu vme de pse tsc msr pae mce cx8
apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx m
mxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl tsc_reliable nonst
op_tsc cpuid extd_apicid aperfmperf tsc_known_freq pni pclmulqdq ssse3 fma cx1
6 pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm
cmp_legacy svm cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext vmm
call fsgsbase bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb
sha_ni xsaveopt xsavec xgetbv1 xsaves user_shstk clzero xsaveerptr rdpru arat
npt nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthr
eshold v_vmsave_vmload umip vaes vpclmulqdq rdpid fsrm
```

Tip

To determine which instruction set is supported for your computer, look at the `Flags` list in the output of `lscpu`. We can check in the Intel® Intrinsics Guide that `avx`, `avx2` and `avx_vnni` are in the AVX family.

SIMD at LLVM level

How can you check that SIMD is enabled? Let's check at the level of LLVM IR.

```
f (generic function with 1 method)
```

```
function f(x1, x2, x3, x4, y1, y2, y3, y4)
    z1 = x1 + y1
    z2 = x2 + y2
    z3 = x3 + y3
    z4 = x4 + y4
    return z1, z2, z3, z4
end
```

```
@code_llvm debuginfo=:none f(1, 2, 3, 4, 5, 6, 7, 8)
```

```
; Function Signature: f(Int64, Int64, Int64, Int64, Int64, Int64, Int64, In
t64)
define void @julia_f_34215(ptr noalias nocapture noundef nonnull sret([4 x i6
4]) align 8 dereferenceable(32) %sret_return, i64 signext %"x1::Int64", i64 si
gnext %"x2::Int64", i64 signext %"x3::Int64", i64 signext %"x4::Int64", i64 si
gnext %"y1::Int64", i64 signext %"y2::Int64", i64 signext %"y3::Int64", i64 si
gnext %"y4::Int64") #0 {
top:
  %0 = add i64 %"y1::Int64", %"x1::Int64"
  %1 = add i64 %"y2::Int64", %"x2::Int64"
  %2 = add i64 %"y3::Int64", %"x3::Int64"
  %3 = add i64 %"y4::Int64", %"x4::Int64"
  store i64 %0, ptr %sret_return, align 8
  %"new::Tuple.sroa.2.0.sret_return.sroa_idx" = getelementptr inbounds i8, ptr
%sret_return, i64 8
  store i64 %1, ptr %"new::Tuple.sroa.2.0.sret_return.sroa_idx", align 8
  %"new::Tuple.sroa.3.0.sret_return.sroa_idx" = getelementptr inbounds i8, ptr
%sret_return, i64 16
  store i64 %2, ptr %"new::Tuple.sroa.3.0.sret_return.sroa_idx", align 8
  %"new::Tuple.sroa.4.0.sret_return.sroa_idx" = getelementptr inbounds i8, ptr
%sret_return, i64 24
  store i64 %3, ptr %"new::Tuple.sroa.4.0.sret_return.sroa_idx", align 8
  ret void
}
```

Tip

If we see `add i64`, it means that each `Int64` is added independently

Packing the data to enable SIMD

```
f_broadcast (generic function with 1 method)
```

```
function f_broadcast(x, y)
    z = x .+ y
    return z
end
```

```
@code_llvm debuginfo=:none f_broadcast((1, 2, 3, 4), (1, 2, 3, 4))
```

```
; Function Signature: f_broadcast(NTuple{4, Int64}, NTuple{4, Int64})
define void @julia_f_broadcast_32914(ptr noalias nocapture noundef nonnull sret([4 x i64]) align 8 dereferenceable(32) %sret_return, ptr nocapture noundef nonnull readonly align 8 dereferenceable(32) %"x::Tuple", ptr nocapture noundef nonnull readonly align 8 dereferenceable(32) %"y::Tuple") #0 {
top:
  %0 = load <4 x i64>, ptr %"x::Tuple", align 8
  %1 = load <4 x i64>, ptr %"y::Tuple", align 8
  %2 = add <4 x i64> %1, %0
  store <4 x i64> %2, ptr %sret_return, align 8
  ret void
}
```

Tip

`load <4 x i64>` means that 4 `Int64` are loaded into a 256-bit wide SIMD unit.

SIMD at assembly level

```
1 @code_native debuginfo=:none f_broadcast((1, 2, 3, 4), (1, 2, 3, 4))
```

```
    .text
    .file   "f_broadcast"
    .section    .ltext,"axl",@progbits
    .globl  julia_f_broadcast_33214         # -- Begin function julia_f_broadc
ast_33214
    .p2align    4, 0x90
    .type   julia_f_broadcast_33214,@function
julia_f_broadcast_33214:                # @julia_f_broadcast_33214
; Function Signature: f_broadcast(NTuple{4, Int64}, NTuple{4, Int64})
# %bb.0:                                # %top
    #DEBUG_VALUE: f_broadcast:x <- [$rsi+0]
    #DEBUG_VALUE: f_broadcast:y <- [$rdx+0]
    push    rbp
    vmovdqu ymm0, ymmword ptr [rdx]
    mov rbp, rsp
    mov rax, rdi
    vpaddq  ymm0, ymm0, ymmword ptr [rsi]
    vmovdqu ymmword ptr [rdi], ymm0
    pop rbp
    vzeroupper
    ret
.Lfunc_end0:
    .size   julia_f_broadcast_33214, .Lfunc_end0-julia_f_broadcast_33214
                                        # -- End function
    .type   ".L+Core.Tuple#33216",@object   # @"+Core.Tuple#33216"
    .section    .lrodata,"al",@progbits
    .p2align    3, 0x0
".L+Core.Tuple#33216":
    .quad   "_L+Core.Tuple#33216.jit"
```

Tip

The suffix `v` in front of the instruction stands for `vectorized`. It means it is using a SIMD unit.

Tuples implementing the array interface

N = 2

```
let
    T = Float64
    A = rand(SMatrix{N,N,T})
    x = rand(SVector{N,T})
    @code_llvm debuginfo=:none A * x
end
```

```
; Function Signature: *(StaticArraysCore.SArray{Tuple{2, 2}, Float64, 2,
4}, StaticArraysCore.SArray{Tuple{2}, Float64, 1, 2})
define void @"julia_*_34161"(ptr noalias nocapture noundef nonnull sret([1 x
[2 x double]]) align 8 dereferenceable(16) %sret_return, ptr nocapture noundef
nonnull readonly align 8 dereferenceable(32) %"A::SArray", ptr nocapture nound
ef nonnull readonly align 8 dereferenceable(16) %"B::SArray") #0 {
top:
  %"A::SArray[3]_ptr" = getelementptr inbounds i8, ptr %"A::SArray", i64 16
  %0 = load <2 x double>, ptr %"B::SArray", align 8
  %1 = load <2 x double>, ptr %"A::SArray", align 8
  %2 = shufflevector <2 x double> %0, <2 x double> poison, <2 x i32> zeroiniti
alizer
  %3 = fmul <2 x double> %1, %2
  %4 = load <2 x double>, ptr %"A::SArray[3]_ptr", align 8
  %5 = shufflevector <2 x double> %0, <2 x double> poison, <2 x i32> <i32 1, i
32 1>
  %6 = fmul contract <2 x double> %4, %5
  %7 = fadd contract <2 x double> %3, %6
  store <2 x double> %7, ptr %sret_return, align 8
  ret void
}
```

Tip

Small arrays that are allocated on the stack like tuples and implemented in `StaticArrays.jl`. Operating on them leverages SIMD.

Auto-Vectorization

LLVM Loop Vectorizer for a C array

```
1 emit_llvm(c_sum_code_for_llvm, cflags = [sum_opt; sum_flags]);
```

```
; ModuleID = '/tmp/jl_qpGpKQ/main.c'
source_filename = "/tmp/jl_qpGpKQ/main.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-i128:128-f8
0:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @sum(ptr noundef %0, i32 noundef %1) #0 {
  %3 = alloca ptr, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store ptr %0, ptr %3, align 8
  store i32 %1, ptr %4, align 4
  store i32 0, ptr %5, align 4
  store i32 0, ptr %6, align 4
  br label %7

7:                                                ; preds = %19, %2
  %8 = load i32, ptr %6, align 4
  %9 = load i32, ptr %4, align 4
  %10 = icmp slt i32 %8, %9
  br i1 %10, label %11, label %22

11:                                               ; preds = %7
  %12 = load ptr, ptr %3, align 8
  %13 = load i32, ptr %6, align 4
  %14 = sext i32 %13 to i64
  %15 = getelementptr inbounds i32, ptr %12, i64 %14
```

```
int sum(int *vec, int length) {
    int total = 0;
    for (int i = 0; i < length; i++) {
        total += vec[i];
    }
    return total;
}
```

No pragma

No pragma

No pragma

Element type : int
Optimization level : -O0

- [ ] -msse3
- [ ] -mavx2
- [ ] -mavx512f
- [ ] -ffast-math
- [ ] -march=native

LLVM Loop Vectorizer for a C++ vector

```
; ModuleID = '/tmp/jl_n70vzh/main.c'
source_filename = "/tmp/jl_n70vzh/main.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-i128:128-f8
0:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @sum(ptr noundef %0, i32 noundef %1) #0 {
  %3 = alloca ptr, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store ptr %0, ptr %3, align 8
  store i32 %1, ptr %4, align 4
  store i32 0, ptr %5, align 4
  store i32 0, ptr %6, align 4
  br label %7

7:                                                ; preds = %19, %2
  %8 = load i32, ptr %6, align 4
  %9 = load i32, ptr %4, align 4
  %10 = icmp slt i32 %8, %9
  br i1 %10, label %11, label %22

11:                                               ; preds = %7
  %12 = load ptr, ptr %3, align 8
  %13 = load i32, ptr %6, align 4
  %14 = sext i32 %13 to i64
  %15 = getelementptr inbounds i32, ptr %12, i64 %14
```

32742.973f0

```
1 @btime cpp_sum($vec_float)
```

```
  279.128 μs (0 allocations: 0 bytes)
```

```
cpp_sum(x::Vector{Cfloat}) = ccall(("c_sum", cpp_sum_float_lib), Cfloat,
(Ptr{Cfloat}, Cint), x, length(x));
```

```
#include <vector>

int my_sum(std::vector<int> vec) {
  int total = 0;
  for (int i = 0; i < vec.size(); i++) {
    total += vec[i];
  }
  return total;
}

extern "C" {
int c_sum(int *array, int length) {
  std::vector<int> v;
  v.assign(array, array + length);
  return my_sum(v);
}}
```

No pragma
No pragma
No pragma
Element type : int
Optimization level : -O0

- [ ] -msse3
- [ ] -mavx2
- [ ] -mavx512f
- [ ] -ffast-math
- [ ] -march=native

Tip

Easily call C++ code from Julia or Python by adding a C interface like the `c_sum` in this example.

LLVM Superword-Level Parallelism (SLP) Vectorizer

```
f (generic function with 2 methods)
```

```
1 f(a, b) = (a[1] + b[1], a[2] + b[2], a[3] + b[3], a[4] + b[4])
```

```
1 @code_llvm debuginfo=:none f((1, 2, 3, 4), (5, 6, 7, 8))
```

```
; Function Signature: f(NTuple{4, Int64}, NTuple{4, Int64})
define void @julia_f_34271(ptr noalias nocapture noundef nonnull sret([4 x i64]) align 8 dereferenceable(32) %sret_return, ptr nocapture noundef nonnull readonly align 8 dereferenceable(32) %"a::Tuple", ptr nocapture noundef nonnull readonly align 8 dereferenceable(32) %"b::Tuple") #0 {
top:
  %0 = load <4 x i64>, ptr %"a::Tuple", align 8
  %1 = load <4 x i64>, ptr %"b::Tuple", align 8
  %2 = add <4 x i64> %1, %0
  store <4 x i64> %2, ptr %sret_return, align 8
  ret void
}
```

Inspection with godbolt Compiler Explorer

Source Editor: C source #1

```c
void foo(int a1, int a2, int b1, int b2, int *A) {
  A[0] = a1 * (a1 + b1);
  A[1] = a2 * (a2 + b2);
  A[2] = a1 * (a1 + b1);
  A[3] = a2 * (a2 + b2);
}
```

Compiler Output: x86-64 clang 19.1.0 (Editor #1)

Flags: `-O3 -mavx2`

```
foo:
        add     edx, edi
        imul    edx, edi
        mov     dword ptr [r8], edx
        add     ecx, esi
        imul    ecx, esi
```

Edit on Com

Example source

Interleaving

```
; ModuleID = '/tmp/jl_EpGMdq/main.c'
source_filename = "/tmp/jl_EpGMdq/main.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-i128:128-f8
0:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind optnone uwtable
define dso_local void @add(ptr noundef %0, i32 noundef %1) #0 {
  %3 = alloca ptr, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store ptr %0, ptr %3, align 8
  store i32 %1, ptr %4, align 4
  store i32 0, ptr %5, align 4
  store i32 0, ptr %6, align 4
  br label %7

7:                                                ; preds = %22, %2
  %8 = load i32, ptr %6, align 4
  %9 = load i32, ptr %4, align 4
  %10 = icmp slt i32 %8, %9
  br i1 %10, label %11, label %25

11:                                               ; preds = %7
  %12 = load ptr, ptr %3, align 8
  %13 = load i32, ptr %6, align 4
  %14 = sext i32 %13 to i64
  %15 = getelementptr inbounds i32, ptr %12, i64 %14
```

With `interleave_count(2)` and

- `-O1`, we see `!{!"llvm.loop.interleave.count", i32 2}` at the end but it hasn't ben applied
- `-O2`, we see the interleaving is applied

```
void add(int *a, int length) {
    int total = 0;
    for (int i = 0; i < length; i++) {
        a[i] = a[i] + 1;
    }
    return;
}
```

No pragma

No pragma
No pragma
Element type : int
Optimization level : -O0
- [ ] -msse3
- [ ] -mavx2
- [ ] -mavx512f
- [ ] -ffast-math
- [ ] -march=native

Modern CPU architectures

Reorder buffer:

```
a = b + c;
d = a + c; // Needs to wait for a
e = b - c; // Independent from a
           // Can be executed before
```

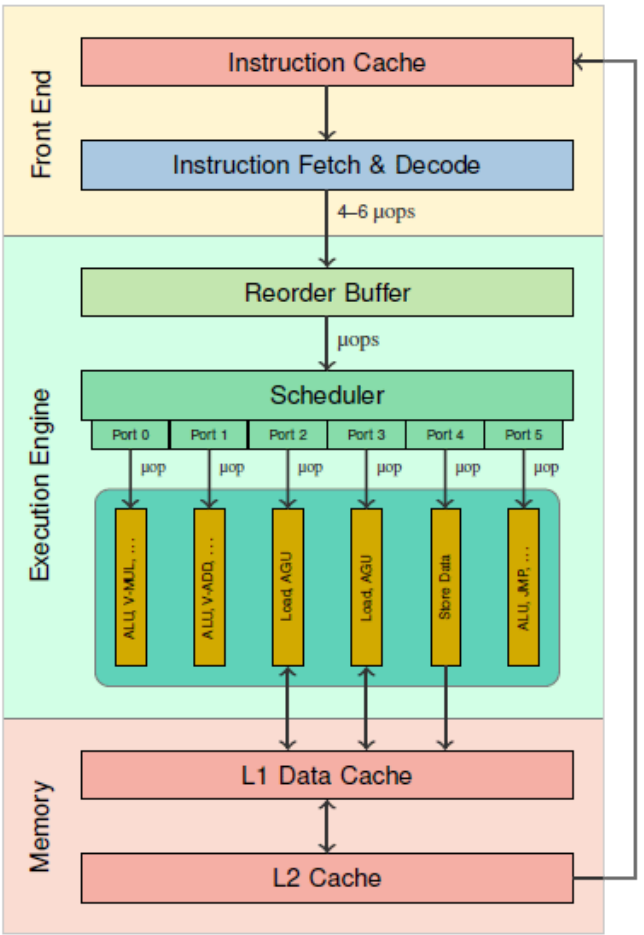
Execution ports

- Arithmetic Logic Unit (ALU)

```
a + b
```

- Address Generation Unit (AGU)
  - Load AGU in port 2 & 3
  - Store AGU in port 4

```
a[i] -> a + i * sizeof(...)
```



Why reordering ?

```
load  r1 ← a
load  r2 ← b
load  r3 ← c
load  r4 ← d
add   r5 ← r1 + r2
add   r6 ← r3 + r4
add   r7 ← r6 + 1
```

Assume

- L1 latency is 4 cycles
- L2 latency is 12 cycles
- `a` is missing from L1 cache but is in L2 cache

Without reordering

Cycle	ALU	ALU	Load	Load
1			a	b
2			c	d
3-5			L2 ⌛	L1 ⌛
6-12			L2 ⌛	
13	r1 + r2	r3 + r4		
14	r6 + 1			

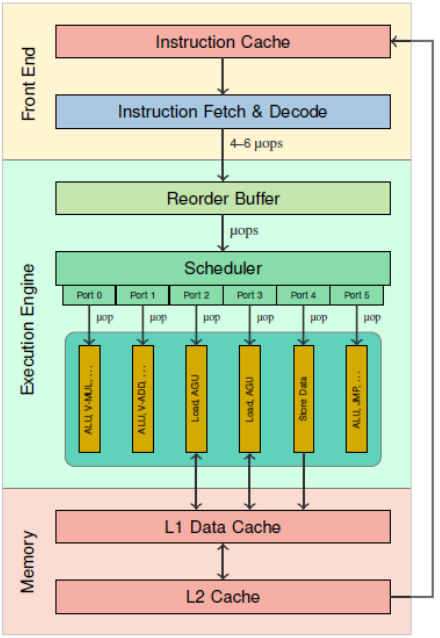
With reordering

Cycle	ALU	ALU	Load	Load
1			a	b
2			c	d
3-5			L2 ⌛	L1 ⌛
6		r3 + r4	L2 ⌛	
7		r6 + 1	L2 ⌛	
8-12			L2 ⌛	
13	r1 + r2			

Without unrolling

```
loop:
  load  r1 ← a[i]
  load  r2 ← b[i]
  add   r3 ← r1 + r2
  store c[i] ← r3
  add   i ← i + 1
  cmp   i, N
  jl  #loop ; Jump if Less
```

Cycle	ALU	ALU	Load	Load	Store	Jump
1			a[i]	b[i]		
2-4			L1 ⌛	L1 ⌛		
5	r1 + r2					
6	i + 1				c[i]	
7	i <? N					
8						#loop



Interleaving allowed by unrolling

```
loop2:
  load  r1 ← a[i]
  load  r2 ← b[i]
  load  r4 ← a[i+1]
  load  r5 ← b[i+1]
  add   r3 ← r1 + r2
  add   r6 ← r4 + r5
  store c[i] ← r3
  store c[i+1] ← r6
  add   i ← i + 2
  cmp   i, N-1
  jl  #loop2
  cmp   i, N
  jl  #loop ; tail case, no unrolling
```

Cycle	ALU	ALU	Load	Load	Store	Jump
1			a[i]	b[i]		
2			a[i+1]	b[i+1]		
3-4			L1 ⌛	L1 ⌛		
5	r1 + r2					
6		r4 + r5			c[i]	
7	i + 2				c[i+1]	
8	i <? N-1					
9						#loop2

Further notes

- Can be controlled with `#pragma clang loop interleave_count(4)` but the compiler usually already does a good job choosing the `interleave_count`
- This count depends on whether it is *compute bound* or *memory bound*, see next lecture where we discuss these.
- The above examples assumes the value of `a[i]` etc... are in L1 cache. If not (aka "L1 cache miss"), the latency for loading `a[i]`, will be much longer.

Additional practical limits include:

Limit	Typical value	Applies when

Load ports	2/cycle	Always — hard throughput ceiling
Load buffer (MOB)	~70-100 entries	All loads, hit or miss
Line fill buffers	~12 entries	L1 cache misses only
Reorder buffer	~200 µops	Window used for reordering

Fused instructions

Fused Multiply-Add (FMA)

```
; ModuleID = '/tmp/jl_qPF5iD/main.c'
source_filename = "/tmp/jl_qPF5iD/main.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-i128:128-f8
0:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

; Function Attrs: noinline nounwind optnone uwtable
define dso_local void @elementwise_muladd(ptr noundef %0, ptr noundef %1, ptr
noundef %2, i32 noundef %3) #0 {
  %5 = alloca ptr, align 8
  %6 = alloca ptr, align 8
  %7 = alloca ptr, align 8
  %8 = alloca i32, align 4
  %9 = alloca i32, align 4
  %10 = alloca i32, align 4
  store ptr %0, ptr %5, align 8
  store ptr %1, ptr %6, align 8
  store ptr %2, ptr %7, align 8
  store i32 %3, ptr %8, align 4
  store i32 0, ptr %9, align 4
  store i32 0, ptr %10, align 4
  br label %11

11:                                               ; preds = %33, %4
  %12 = load i32, ptr %10, align 4
  %13 = load i32, ptr %8, align 4
  %14 = icmp slt i32 %12, %13
  br i1 %14, label %15, label %36
```

```
void elementwise_muladd(int *c, int *a, int *b, int length) {
    int total = 0;
    for (int i = 0; i < length; i++) {
        c[i] += a[i] * b[i];
    }
    return;
}
```

No pragma
No pragma
No pragma
Element type : int

Optimization level : -O0

- [ ] -msse3
- [ ] -mavx2
- [ ] -mavx512f
- [ ] -ffast-math
- [ ] -march=native

Explicit FMA: [ ]

Further readings

Slides inspired from:

- SIMD in Julia
- Demystifying Auto-vectorization in Julia
- Auto-Vectorization in LLVM
- Information on processor architecture
- Advanced books on optimization

The End

```
using SimpleClang, PlutoUI, PlutoUI.ExperimentalLayout, HypertextLiteral; Luxor,@htl_
StaticArrays, BenchmarkTools, PlutoTeachingTools
```

```
img (generic function with 3 methods)
```

```
begin
struct Path
    path::String
end

function imgpath(path::Path)
    file = path.path
    if !('.' in file)
        file = file * ".png"
    end
    return joinpath(joinpath(@__DIR__, "images", file))
end

function img(path::Path, args...; kws...)
    return PlutoUI.LocalResource(imgpath(path), args...)
end

struct URL
    url::String
end

function save_image(url::URL, html_attributes...; name = split(url.url, '/')[end], kws...)
    path = joinpath("cache", name)
    return PlutoTeachingTools.RobustLocalResource(url.url, path, html_attributes...), path
end

function img(url::URL, args...; kws...)
    r, _ = save_image(url, args...; kws...)
    return @htl("<a href=$(url.url)>$r</a>")
end

function img(file::String, args...; kws...)
    if startswith(file, "http")
        img(URL(file), args...; kws...)
    else
        img(Path(file), args...; kws...)
    end
end
end
```

```
qa (generic function with 2 methods)
```

```
begin
function qa(question, answer)
    return @htl("<details><summary>$question</summary>$answer</details>")
end
function _inline_html(m::Markdown.Paragraph)
    return sprint(Markdown.htmlinline, m.content)
end
function qa(question::Markdown.MD, answer)
    # 'html(question)' will create '<p>' if 'question.content[]' is 'Markdown.Paragraph'
    # This will print the question on a new line and we don't want that:
    h = HTML(_inline_html(question.content[]))
    return qa(h, answer)
end
end
```

```
function Luxor.placeimage(url::URL, pos; scale = 1.0, centered = true, kws...)
    r, path = save_image(url; kws...)
    if r.mime isa MIME"image/svg+xml"
        img = Luxor.readsvg(path)
    else
        img = Luxor.readpng(path)
    end
    Luxor.gsave()
    Luxor.scale(scale)
    Luxor.placeimage(img, pos / scale; centered)
    Luxor.grestore() # undo 'Luxor.scale'
    return
end
```

```
CenteredBoundedBox (generic function with 1 method)
```

```
function CenteredBoundedBox(str)
    xbearing, ybearing, width, height, xadvance, yadvance =
        Luxor.textextents(str)
    lcorner = Luxor.Point(xbearing - width/2, ybearing + height/2)
    ocorner = Luxor.Point(lcorner.x + width, lcorner.y + height)
    return Luxor.BoundingBox(lcorner, ocorner)
end
```

```
boxed (generic function with 1 method)
```

```
function boxed(str::AbstractString, p; hue = "lightgrey")
    Luxor.gsave()
    Luxor.translate(p)
    Luxor.sethue(hue)
    Luxor.poly(CenteredBoundedBox(str) + 5, action = :stroke, close=true)
    Luxor.sethue("black")
    Luxor.text(str, Luxor.Point(0, 0); halign = :center, valign = :middle)
    #settext("<span font='26'>$str</span>", halign="center", markup=true)
    Luxor.origin()
    Luxor.grestore() # strokecolor
end
```