Leveraging SIMD

SIMD

SIMD (Single Instruction, Multiple Data) is a parallel computing architecture where a single instruction operates simultaneously on multiple data elements. It's used to exploit data-level parallelism, particularly in vector processing.

Key points:

- Common in CPU vector instructions (e.g., SSE, AVX on x86) and GPU shaders.
- Ideal for operations like matrix/vector math, image processing, or signal processing.
- Contrasts with **SISD (Single Instruction, Single Data)** and **MIMD (Multiple Instruction, Multiple Data)** in Flynn's taxonomy.

SIMD instructions

Scalar Operation

$$A_x + B_x = C_x$$

$$A_y + B_y = C_y$$

$$A_z + B_z = C_z$$

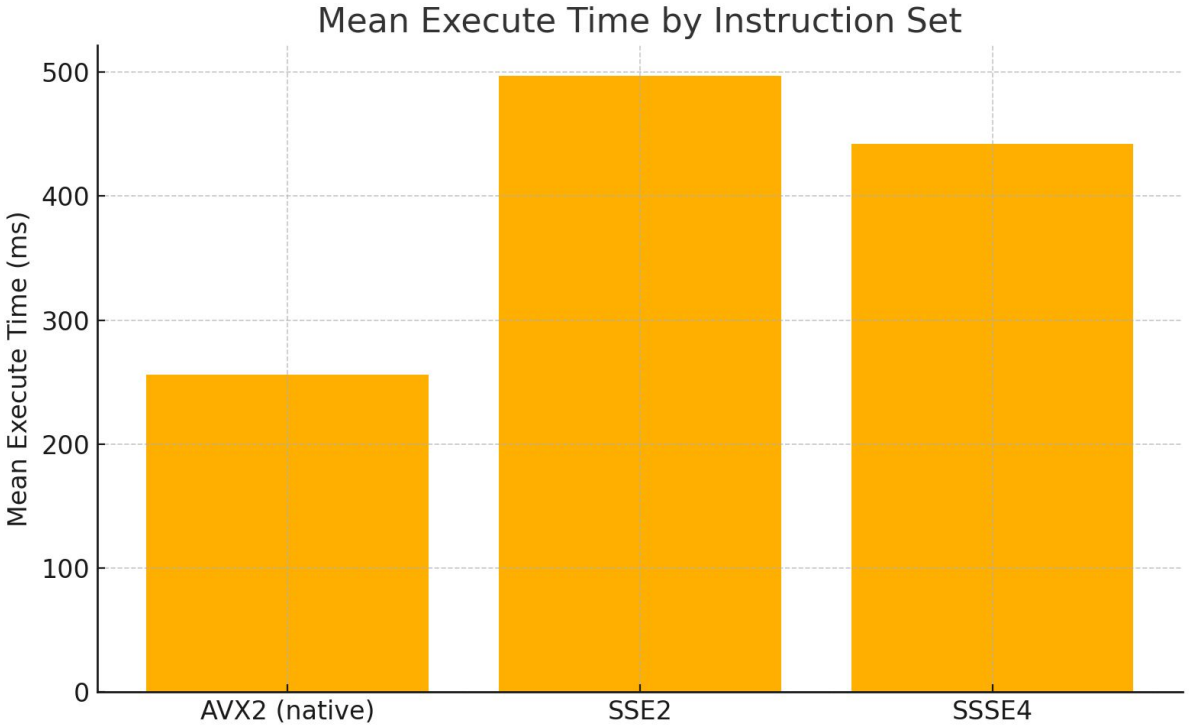
$$A_w + B_w = C_w$$

SIMD Operation of Vector Length 4

$$\begin{matrix} A_x \\ A_y \\ A_z \\ A_w \end{matrix} + \begin{matrix} B_x \\ B_y \\ B_z \\ B_w \end{matrix} = \begin{matrix} C_x \\ C_z \\ C_z \\ C_w \end{matrix}$$

Do we care?

FInufft: `perftest --N1 64 --N2 64 --N3 64 --t hreads 1 --tol 1e-7 --upsampfact 2.00 --prec d`



Conclusions

Let’s compile everything with AVX2!

Conclusions

Let’s compile everything with AVX2

There are limitations!!!

Limitations

- SIMD is not always faster than scalar
- SIMD instructions do not have the same performance on different CPUs
- SIMD instructions are not portable

Solutions

- Measure performance! Preferably on multiple CPUs and multiple compilers (example)
- Dynamic dispatch to select the fastest instruction set

There are many instruction sets:

```
- lscpu
- Architecture:             x86_64
-  CPU op-mode(s):         32-bit, 64-bit
-  Address sizes:          46 bits physical, 48 bits virtual
-  Byte Order:             Little Endian
- CPU(s):                   22
-  On-line CPU(s) list:    0-21
- Vendor ID:                GenuineIntel
-  Model name:             Intel(R) Core(TM) Ultra 7 155H
-    CPU family:           6
-    Model:                170
-    Thread(s) per core:   2
-    Core(s) per socket:   16
-    Socket(s):            1
-    Stepping:             4
-    CPU(s) scaling MHz:   23%
-    CPU max MHz:          4800.0000
-    CPU min MHz:          400.0000
-    BogoMIPS:             5990.40
-    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pb
-                          e syscall nx pdpelgb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfm
-                          perf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4
-                          _2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb intel_pp
-                          in ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2
-                          erms invpcid rdseed adx smap clflushopt clwb intel_pt sha_ni xsaveopt xsavec xgetbv1 xsaves split_lock_detect user_shs
-                          tk avx_vnni dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp hwp_pkg_req hfi vnmi umip pku ospke waitpkg
-                          gfni vaes vpclmulqdq rdpid bus_lock_detect movdiri movdir64b fsrm md_clear serialize arch_lbr ibt flush_l1d arch_capab
-                          ilities
```

X86-64 psABI

The **x86-64 Processor Supplement to the System V Application Binary Interface (psABI)** defines the standard conventions for binary interface compatibility on the x86-64 architecture. source

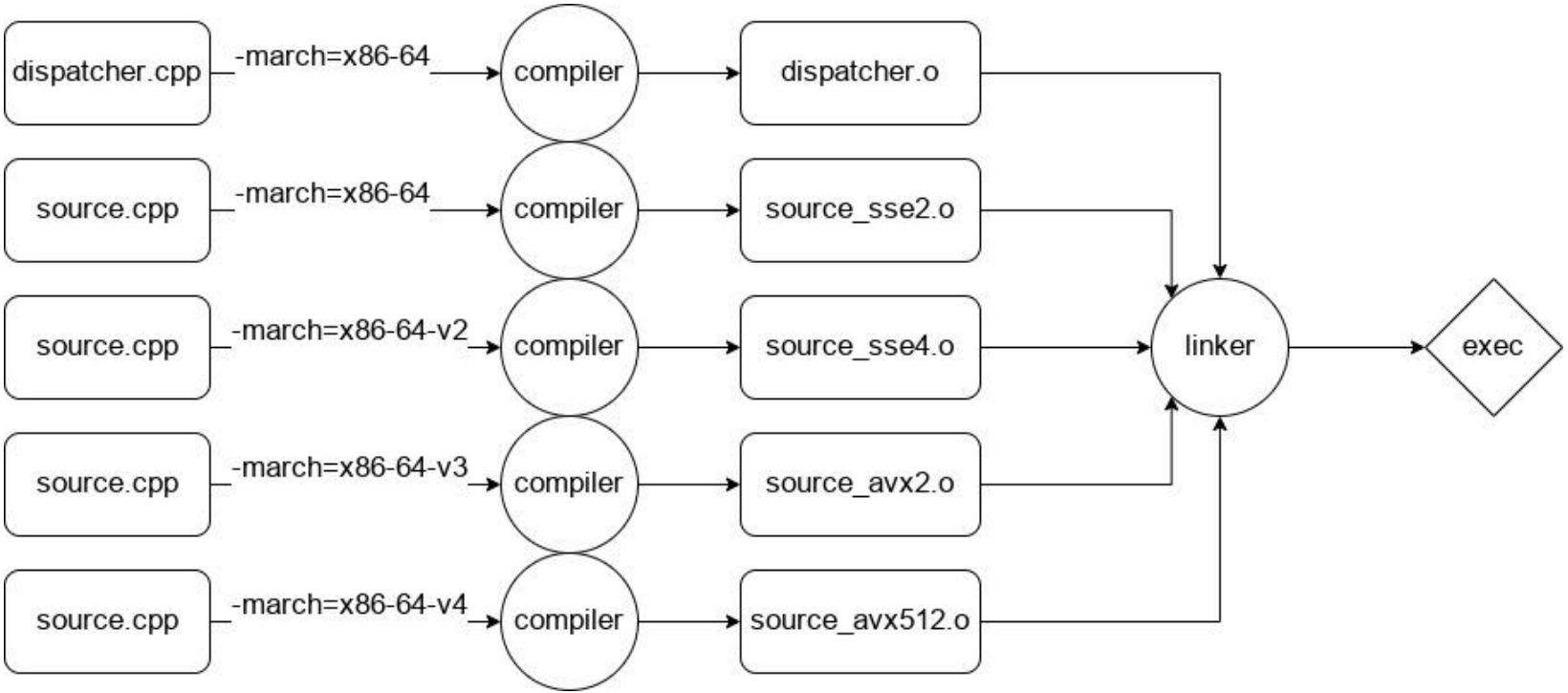
Table 3.1: Micro-Architecture Levels

Level Name	CPU Feature	Example instruction
(baseline)	CMOV	`cmov`
	CX8	`cmpxchg8b`
	FPU	`fld`
	FXSR	`fxsave`
	MMX	`emms`
	OSFXSR	`fxsave`
	SCE	`syscall`
	SSE	`cvtss2si`
	SSE2	`cvtpi2pd`
`x86-64-v2`	CMPXCHG16B	`cmpxchg16b`
	LAHF-SAHF	`lahf`
	POPCNT	`popcnt`
	SSE3	`addsubpd`
	SSE4_1	`blendpd`
	SSE4_2	`pcmpestri`
	SSSE3	`phaddd`
`x86-64-v3`	AVX	`vzeroall`
	AVX2	`vpermd`
	BMI1	`andn`
	BMI2	`bzhi`
	F16C	`vcvtph2ps`
	FMA	`vfmadd132pd`
	LZCNT	`lzcnt`
	MOVBE	`movbe`
	OSXSAVE	`xgetbv`
`x86-64-v4`	AVX512F	`kmovw`
	AVX512BW	`vdbpsadbw`
	AVX512CD	`vplzcntd`
	AVX512DQ	`vpmullq`
	AVX512VL	n/a

Compiler support

- **GCC 11**, released in **April 2021**, added support for specifying x86-64 microarchitecture levels using the `-march=` flag. This enhancement allows developers to target specific CPU feature sets more effectively. gcc
- **Clang 12**, released in **April 2021**, introduced similar support for x86-64 microarchitecture levels. llvm
- **MSVC?** No support.

Runtime dispatch (simplified)



In practice

https://github.com/flatironinstitute/finufft/blob/master/cmake/CheckAVX.cpp

https://github.com/DiamonDinoia/random

https://xsimd.readthedocs.io/en/latest/api/dispatching.html

LRU cache

Python functools.cache https://docs.python.org/3/library/functools.html