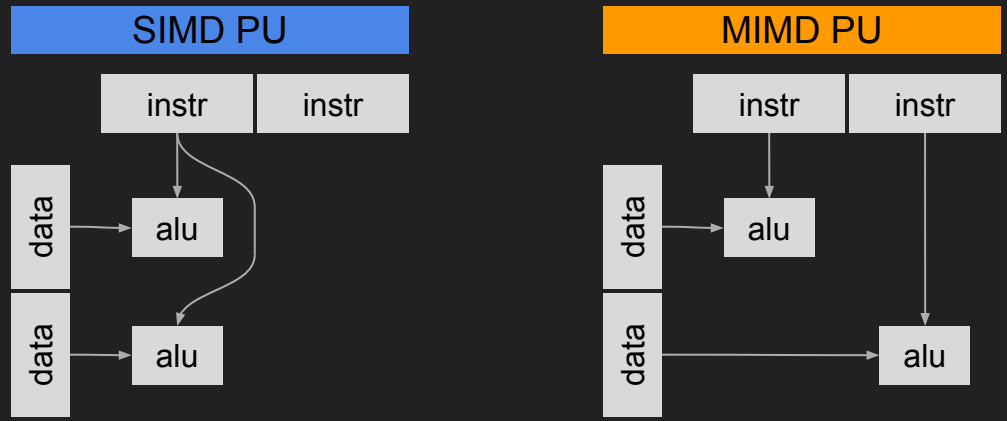
# Get Your DLP at Incredibly Low SIMD Prices!

Martin Krastev
Chaos Group

# Intro

The two most established types of parallelism this side of multi-threading (TLP)

- ILP – instruction-level parallelism – superscalar CPUs
- DLP – data-level parallelism – vector CPUs (Cray-1), GPUs, SIMD, MIMD




Flynn's Taxonomy, multiple-data only

# History

SIMD, consumer-CPU-side

- Intel MMX – i8-i64 – 1997
- AMD 3DNow! – f32 – 1998
- AIM (Apple/IBM/Motorola) PowerPC VMX/AltiVec – f32, i8-i32 – 1998
- Intel SSE – f32 – 1999
- Intel SSE2 – f32-f64, i8-i64 – supersedes MMX, baseline amd64
- ARM VFP (chained SIMD) – f32-f64; armv7 NEON (true SIMD) – f32, i8-i64; armv8 NEON – f16-f64, i8-i64; SVE – f16-f32, i8-i64, 128-2048-bit
- Intel SSE3/SSSE3/SSE4.1/SSE4.2/AVX/AVX2/AVX512 (and counting)
- MIPS MSA – f32-f64, i8-i64

# How to benefit?

SIMD in today’s CPUs utilized from C/C++ as

- Assembly language (inline asm)
- Autovectorizing compilers
- Intrinsics
- Generic vectors

# How to benefit? cont’d

Things we won’t discuss today but which deserve your attention

- Intel SPMD compiler ISPC (amd64, arm64 (expt)) https://github.com/ispc/ispc
- Impala DSL (amd64, arm64, GPUs) https://anydsl.github.io/
- SIMDe translator from SSE/AVX https://github.com/nemequ/simde/ https://godbolt.org/z/g5FABQ

# Cost of utilisation from C/C++

In terms of effectiveness/effort

- Assembly language (inline asm) – high effect, high effort
- Autovectorizing compilers – low-to-mid effect, low effort
- Intrinsics – mid-to-high effect, mid-to-high effort
- Generic vectors – mid effect, low effort

# Cost of utilisation from C/C++

In terms of effectiveness/effort

- Assembly language (inline asm) – high effect, high effort
- Autovectorizing compilers – low-to-mid effect, **low effort**
- Intrinsics – mid-to-high effect, mid-to-high effort
- Generic vectors – mid effect, **low effort**

# Examples: autovectorization

- trivial
  https://godbolt.org/z/MPNkHZ
  https://godbolt.org/z/3u56Uj
- less trivial
  https://godbolt.org/z/loNdc2
- complex
  https://godbolt.org/z/rYcwpH

# Takeaways: autovectorization

- Works best with homogeneous array operations and continuous indexation
- Breaks down if compiler is unable to prove absence of aliasing among callee input/output args
    - Resolution: inlining the callee where args' original declarations are visible

# Examples: generic vectors

- trivial
  https://godbolt.org/z/Z6zHHd
  https://godbolt.org/z/m4HCma
- less trivial
  https://godbolt.org/z/NHnARU
- complex
  https://godbolt.org/z/RSYK28

# Takeaways: generic vectors

- Do not suffer from (sub-vec) aliasing
- Once vectorization is outlined by the developer, optimal mapping to underlying SIMD facilities can be achieved
- Treating vectors as arrays can be convenient but counterproductive under some circumstances

# Mind the gap

- Generic vectors are a GCC extension (vector_size) – GCC and Clang support it, MSVC does not (and they refuse to support it until it gets in the standard)
- Unequal levels of support between GCC and Clang
  - Clang supports arbitrary-length vectors; GCC – only powers of two
  - Clang supports ‘extended vectors’ – OpenCL-compliant generic vectors (ext_vector_type); GCC does not
    https://gcc.gnu.org/onlinedocs/gcc/Vector-Extensions.html
    https://clang.llvm.org/docs/LanguageExtensions.html#vectors-and-extended-vectors
  - Codegen also can differ significantly, but that is par for the course with modern optimizing compilers

# GLSL/OCL/CUDA coding style

Even without Clang's ext_vector_type extension 'plain' generic vectors allow for writing compute-kernel-style code in a manner similar to compute APIs
https://github.com/blu/gemm/blob/master/genvec_unittest00.cpp

The best part – good utilisation of SIMD facilities!

# Interoperability

- We can treat generic vectors as arrays ([] operator), but mind the occasional suboptimal/broken codegen – developer's discretion advised!
- We can mix intrinsics and generic vectors! Bulk of the vector code can be written in generic vectors, with small parts left to arch-specific intrinsics https://github.com/blu/gemm/blob/master/sgemm.cpp

# Interoperability example

A SIMD abstraction in the style of GLSL/OCL/CUDA vector types
for GCC, Clang, MSVC – amd64, arm64

Single-header implementation – generic vectors (~~MSVC~~) + intrinsics
https://github.com/ChaosGroup/cg2_2014_demo/blob/master/common/vectnative.hpp

Transcendentals – intrinsics
https://github.com/ChaosGroup/cg2_2014_demo/blob/master/common/cephes_sse.h
https://github.com/ChaosGroup/cg2_2014_demo/blob/master/common/cephes_avx.h
https://github.com/ChaosGroup/cg2_2014_demo/blob/master/common/cephes_neon.h

# Questions?

martin.krastev@chaosgroup.com