SPMD Programming in C++

Making SIMD easy in the language we know and love

About me

Name: Nicolas Guillemot

Background: Video Games Tech

Past: InLight, Electronic Arts, Intel

Current: Graphics @ University of Victoria

Overview

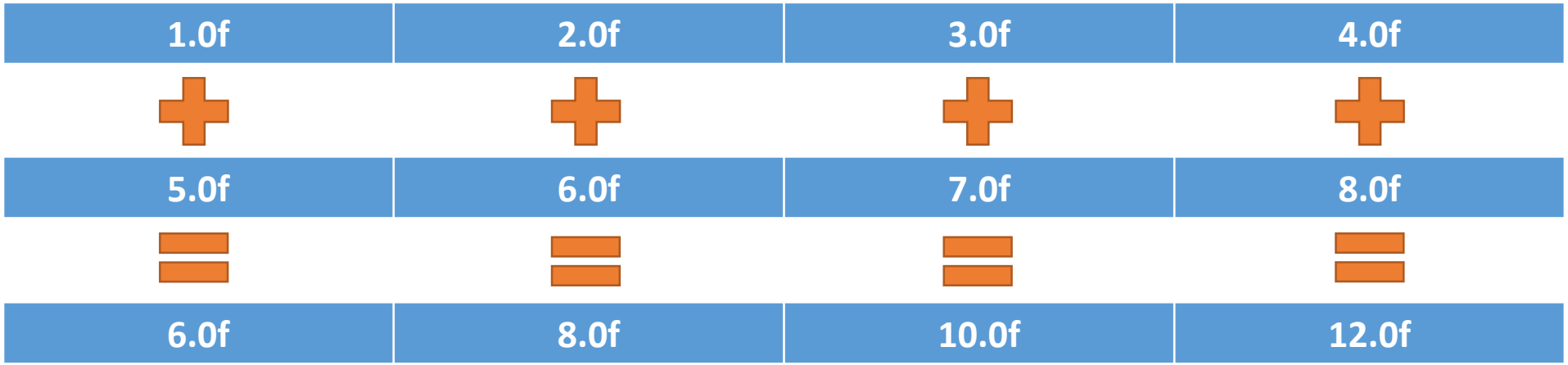
- SIMD Building Blocks
- SPMD Intro & Case Studies
  - ISPC Case Study
  - AMD GCN Shaders Case Study
- SPMD in C++: CppSPMD

SIMD Building Blocks

Using assembly intrinsics

SIMD Operators

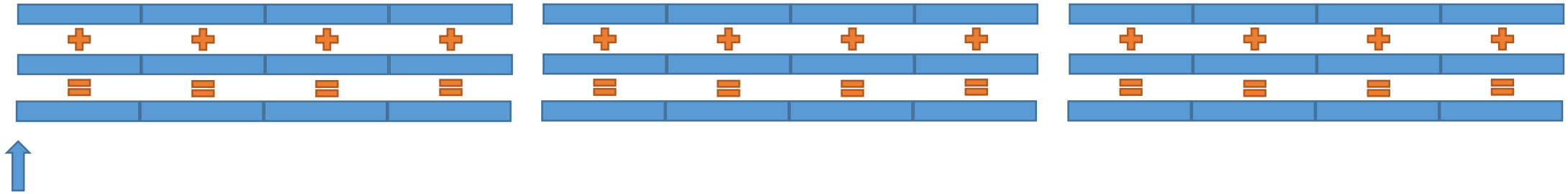
Operators executed in parallel




```
__m128 a = _mm_set_ps(1.0f, 2.0f, 3.0f, 4.0f);
__m128 b = _mm_set_ps(5.0f, 6.0f, 7.0f, 8.0f);
__m128 c = _mm_add_ps(a, b);
```

SIMD Loops

Iterate in steps of SIMD width



```
assert(N % 4 == 0);
for (int i = 0; i < N; i += 4) {
    __m128 a = _mm_load_ps(&in1[i]);
    __m128 b = _mm_load_ps(&in2[i]);
    __m128 c = _mm_add_ps(a, b);
    _mm_store_ps(&out[i], c);
}
```

SIMD Conditions

```
float f(float a) {
    if (a < 0)
        a = 0;
    else
        a += 1;
    return a;
}
```

SIMD Conditions

```
__m128 f(__m128 a) {
  __m128 mask = _mm_cmplt_ps(a, 0);
  a = _mm_blendv_ps(a, 0, mask);
  mask = _mm_not_ps(mask);
  a = _mm_blendv_ps(a, a + 1, mask);
  return a;
}
```

Other SIMD Control Flow?

- For?
- While?
- Do-While?
- Break?
- Continue?
- Switch?
- Early return?
- Indirect function call?

… Exercise for the reader.

Hint: Masks. Masks everywhere.

Assembly Intrinsics: Pros and Cons?

- Assembly-level optimizations possible ☺
- Algorithm-level optimizations tedious ☹

- Code is not portable ☹
- Complicated duplicated code for different CPUs ☹

How can we fix these problems?

Proposed Solution: SPMD-on-SIMD

“Single Program Multiple Data”

- Program appears serial, is SIMD data-parallel.

“Maximal Convergence” guarantee:

- Lock-step SIMD: No “threads”, no “barriers”, no black magic.
- “Synchronization” at sequence points, same as everyday C++.

Getting a feel for SPMD

Two Case Studies:

- ISPC (CPU language)
- AMD GCN Shaders (GPU language)

Goal: Get familiar with SPMD.

Case Study #1: The ISPC Compiler

"ispc: A SPMD Compiler for High-Performance CPU Programming"

Matt Pharr, William R. Mark. InPar 2012.

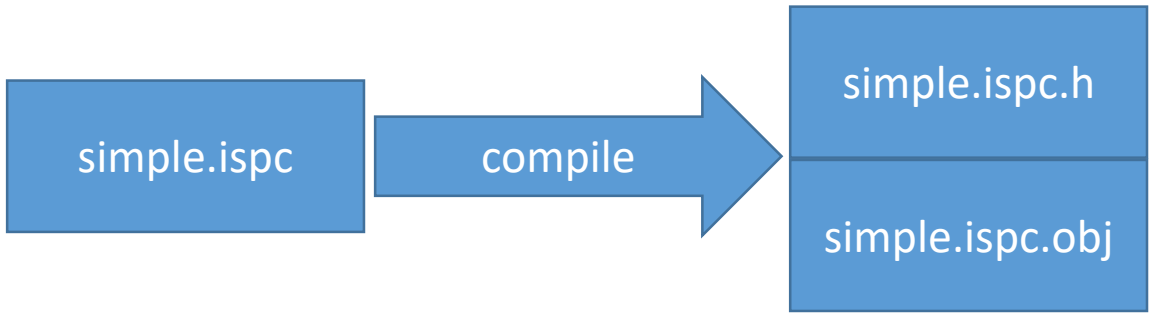
http://ispc.github.io/

What is ISPC?

- C-like language for SPMD-on-SIMD.
- “Shaders” for CPU
- Open Source! Supports x86/64, ARM, Xeon Phi, Sony PS4…

ISPC Example

```
export void simple(uniform float vin[], uniform float vout[],
                   uniform int n)
{
    foreach (index = 0 ... n)
    {
(Optional->) varying float v = vin[index];
        if (v < 3.0f)
            v = v * v;
        else
            v = sqrt(v);
        vout[index] = v;
    }
}
```



Case Study #2: AMD GCN Shaders

AMD “Graphics Core Next” GPU, Launched 2011

http://developer.amd.com/wordpress/media/2012/12/AMD_Southern_Islands_Instruction_Set_Architecture.pdf

What is AMD GCN?

- AMD’s current GPU architecture and ISA.
- Designed for GPGPU (General Purpose GPU) computation.
- Used in current video game consoles (PS4, Xbox One).

GCN ASM Basics

- Vector Registers: r0, r1, r2…

```
v_add_f32    r2, r0, r1
```

- Scalar Registers: s0, s1, s2…

```
s_and_b32    s2, s0, s1
```

- Vector Condition Code: vcc

```
v_cmp_lt_f32    r0, r1
```

- Execution mask: exec

```
s_and_b64    exec, vcc, exec
```

GCN Shader Example

GLSL Shader Code

```
float fn0(float a, float b)
{
    if (a > b)
        return a * a;
    else
        return a - b;
}
```

GCN Assembly

```
    v_cmp_gt_f32        r0, r1          // a > b
    s_mov_b64           s0, exec        // Save current exec mask
    s_and_b64           exec, vcc, exec // Mask exec (for “if”)
    s_cbranch_vccz      fn0_else        // Branch if all lanes fail
    v_mul_f32           r2, r0, r0      // result = a * a
fn0_else:
    s_not_b64           exec, exec      // Flip exec (for “else”)
    s_and_b64           exec, s0, exec  // Respect initial exec
    s_cbranch_execz     fn0_end         // Branch if all lanes fail
    v_sub_f32           r2, r0, r1      // result = a - b
fn0_end:
    s_mov_b64    exec, s0               // Restore exec mask
```

Case Studies Summary

- SPMD maps C languages to SIMD
  - Can write high-level code
  - *And* can still understand performance

- Useful on both CPU and GPU – versatility
- High-performance CPU/GPU SPMD languages exist *today*.

- Now, how to implement it in C++?

SPMD in C++: CppSPMD

Translating SPMD-on-SIMD into C++

About: CppSPMD

- Header-only C++ library.
- Subset of ISPC in plain C++.
- Implemented with intrinsics, but cross-platform interface.
- Lambdas and masks implement SPMD control flow.
- **Proof of Concept Only** – Not for production.

ISPC vs CppSPMD: Data Types

ISPC		**C++**
• uniform int	→	• int
• uniform float	→	• float
• varying int	→	• vint
• varying float	→	• vfloat

Implementing “SPMD if” (simplified)

```
spmd_if(v < 3.0f, [&] {
    store(v, v * v);
});
```

```
void spmd_if(vbool cond, auto ifBody) {
    // save old execution mask
    exec_t old_exec = exec;
    // apply "if" mask
    exec = exec & cond;

    // "all off" optimization
    if (exec != 0) {
        ifBody();
    }
    // restore execution mask
    exec = old_exec;
}
```

Implementing “Varying” Variables

```
spmd_if(v < 3.0f, [&] {
    store(v, v * v);
});
```

```
struct vfloat {
    __m128 _v;
};

vfloat operator*(vfloat a, vfloat b) {
    return vfloat{ _mm_mul_ps(a._v, b._v) };
}

void store(vfloat& dst, vfloat src) {
    dst._v = _mm_blendv_ps(dst._v, src._v, exec);
}
```

Sample Program

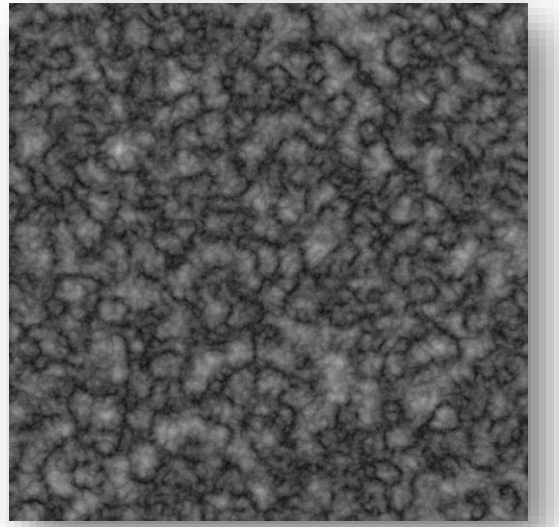
```
struct simple : spmd_kernel {
  void _call(float vin[], float vout[], int n) {
    spmd_foreach(0, n, [&](lint index)
    {
      vfloat v = load(index[vin]);
      spmd_ifelse(v < 3.0f,
        [&] { store(v, v * v); },
      /* else */
        [&] { store(v, sqrt(v)); });
      store(index[vout], v);
    });
  }
};
```

```
float vin[16] = { ... };
float vout[16];
int main() {
    spmd_call<simple>(vin, vout, 16);
}
```

Performance?

Perlin Noise (768x768, 100 runs) – Intel Core i7-5960X

Mode	Speed Multiplier	Runtime
Plain C++ (VC++)	1.0x	45.5s
CppSPMD (VC++)	4.4x	10.3s
ISPC (LLVM, AVX2)	7.3x	6.2s
CppSPMD (VC++, hand-opts)	7.8x	5.8s
CppSPMD (ICC, hand-opts, PGO*)	8.9x	5.1s



Intel Core i7-6650U (relative to 5960x)

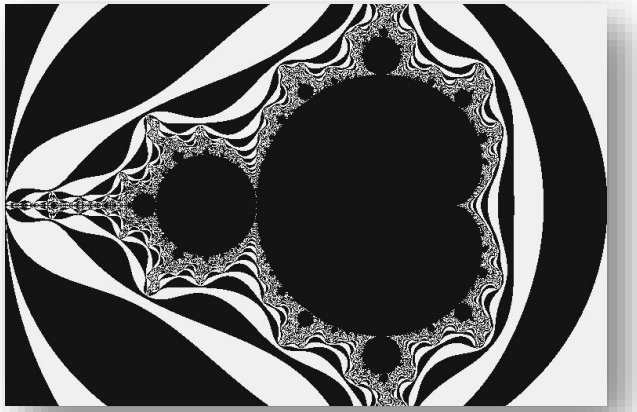
CppSPMD (ICC, hand-opts, PGO)	**11x**	**4.1s**

*PGO: Profile-Guided Optimization

Performance?

Mandelbrot Set (768x512, 1000 runs) – Intel Core i7-5960X

Mode	Speed Multiplier	Runtime
Plain C++ (VC++)	1.0x	97s
CppSPMD (VC++)	1.8x	53s
CppSPMD (ICC, PGO*)	3.2x	30s
ISPC (LLVM, AVX2)	6.0x	16s
CppSPMD (ICC, PGO, hand-opts)	6.5x	15s

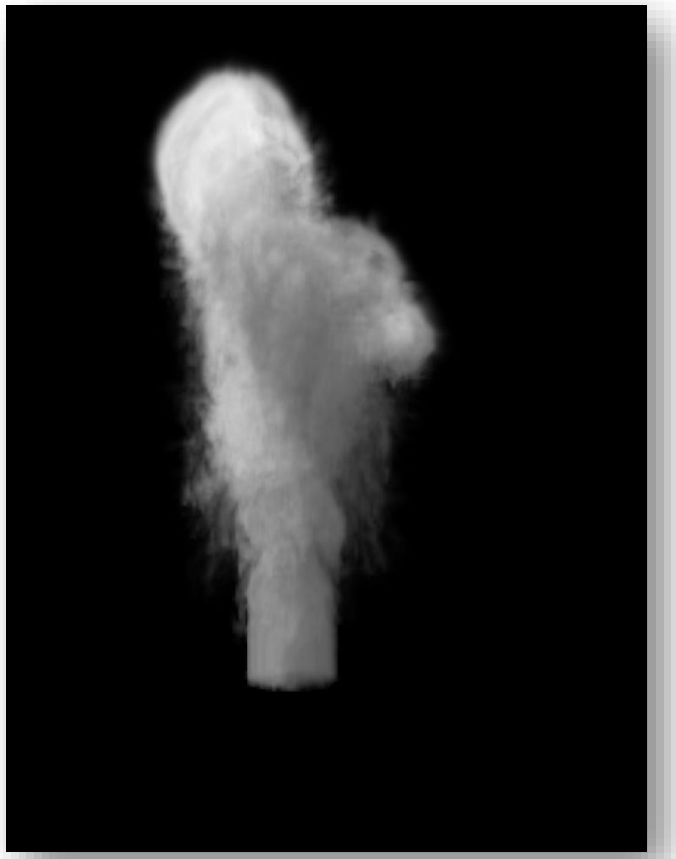


*PGO: Profile-Guided Optimization

Performance?

Volume Rendering (896x1184, 1 run) – Intel Core i7-5960X

Mode	Speed Multiplier	Runtime
Plain C++ (VC++)	1.0x	37.5s
CppSPMD (VC++)	2.9x	13s
CppSPMD (ICC)	5.4x	6.9s
ISPC (LLVM, AVX2)	6.6x	5.6s



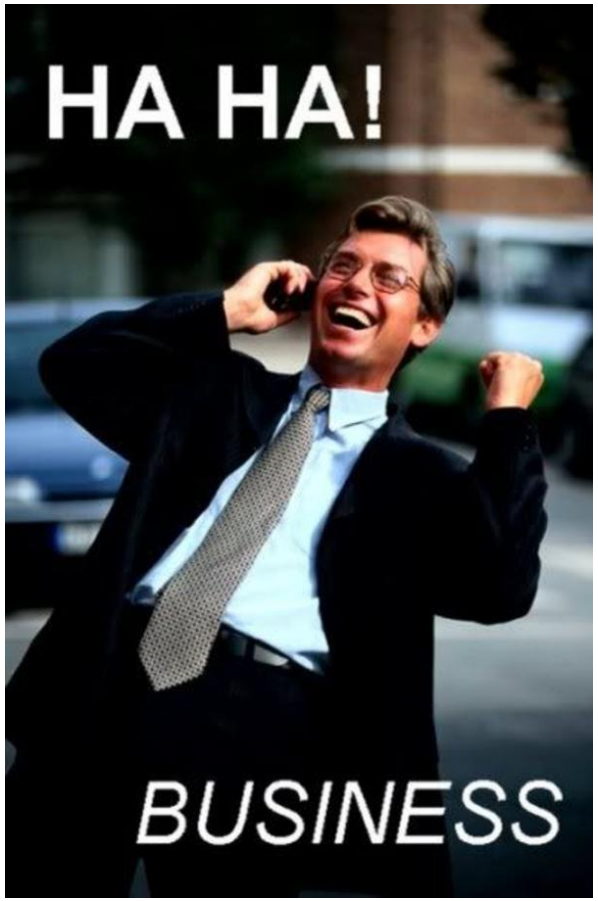
Performance?

Binomial Options (128k options, 100 runs) – Intel Core i7-5960X

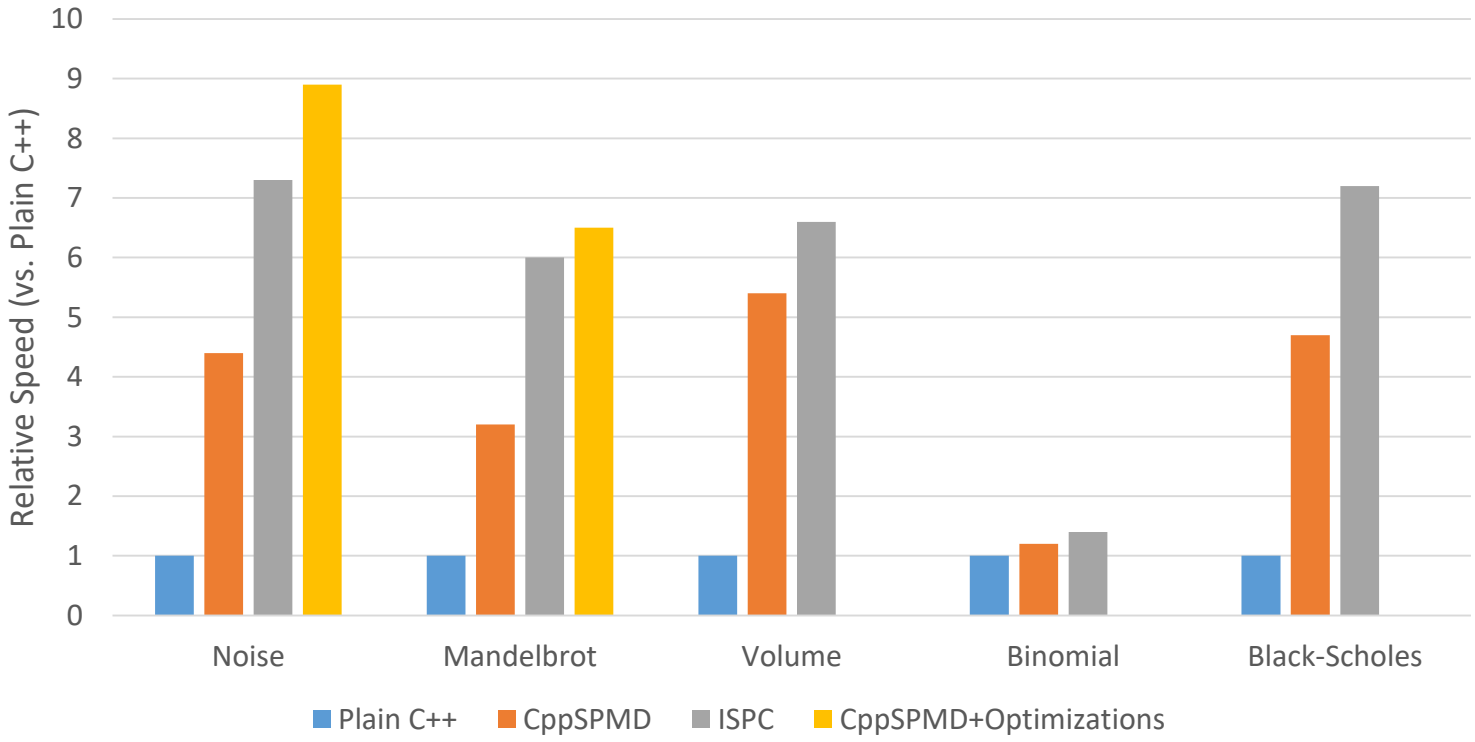
Mode	Speed Multiplier	Runtime
Plain C++ (VC++)	1.0x	22s
CppSPMD (VC++)	1.0x	22s
CppSPMD (ICC)	1.2x	18s
ISPC (LLVM, AVX2)	1.4x	16s

Black-Scholes (128k options, 1000 runs) – Intel Core i7-5960X

Mode	Speed Multiplier	Runtime
Plain C++ (VC++)	1.0x	8s
CppSPMD (VC++)	4.0x	2s
CppSPMD (ICC)	4.7x	1.7s
ISPC (LLVM, AVX2)	7.2x	1.1s



Performance Conclusions



- Want amazing results with less effort? ISPC.
- Want C++ and willing to hand-optimize more? CppSPMD.

Quirk #1: load(), store()

- Need exec in "`vfloat::operator=(vfloat)`" for masked store.
- Need exec in "`vfloat_ref::operator vfloat()`" for masked load.
- Cannot use exec from `vfloat` overloads. ☹
  - C++ defect?

My dream:

```
struct spmd_kernel {
    __m128 exec;
    struct vfloat { __m128 _v; };

    vfloat& operator=(vfloat& v, vfloat v2) {
        v._v = _mm_blendv_ps(v._v, v2._v, exec);
    }
};
```

Quirk #2: index[ptr]

- Want to overload `float*::operator[](vint)`
  - Cannot. ☹
  - C++ defect?

<u>Recall:</u>

```
ptr[i] == *(ptr + i) == *(i + ptr) == i[ptr]
```

Can overload `vint::operator[](float*)`! ☺

“`data[i]`” => “`i[data]`”

Quirk #3: spmd_call

- Need implicit pass-by-value of `exec` in function calls.
  - Cannot. ☹
  - C++ defect?

Solution: `spmd_call` does it.

```
struct simple : spmd_kernel {
    void _call() {
        spmd_call<other>(1,2,3);
    }
};

spmd_call<simple>();
```

Quirk #4: [&] { lambdas } everywhere

Example:

```
spmd_if(v < 3.0f, [&] {
    v = v * v;
});
```

- Usually inlined, so not a perf problem. Just syntax.
- Future: Native C++ language support?
- Macro magic?

Quirk #5: Inheriting spmd_kernel

- Inherits exec mask & related logic.

- Could be used for configuration:

```
struct simple : spmd_kernel_avx2 { ... };

struct simple : spmd_kernel<Width = 16> { ... };
```

In Conclusion

- SPMD is effective, established, portable.
- Can be implemented in simple C++ code.

- Best bet today? Use ISPC!
- Tomorrow, C++ maybe?
  - Language support?
  - Compiler optimization support?
  - Let’s close the gap!

Thanks!

Questions? Comments?

Sample implementation & tests:
https://github.com/nlguillemot/CppSPMD/releases/tag/v1.0
My Twitter:
https://twitter.com/nlguillemot