ispc: A SPMD Compiler for High-Performance CPU Programming

Matt Pharr and William R. Mark
Intel
14 May 2012

http://ispc.github.com
Motivation: 3 Modern Parallel Architectures

CPU: 2-10x

MIC: 50+x

GPU: 2-32x
Filling the Machine (CPU and GPU)

- Task parallelism across cores: run different programs (if wanted) on different cores

- Data-parallelism across SIMD lanes in a single core: run the same program on different input values
ispc: Key Features

- “SPMD on SIMD” on modern CPUs (coupled with task parallelism)
- Ease of adoption and integration
- C syntax and feature set, single coherent address space
- Performance transparency
- Scalability (cores * SIMD width)
SPMD 101

- Run the same program concurrently with different inputs
- Inputs = array/matrix elements, particles, pixels, ...

```c
float func(float a, float b) {
    if (a < 0.) a = 0.;
    return a + b;
}
```

- The contract:
  Programmer guarantees independence across program instances;
  Compiler is free to run those instances in parallel
SPMD On A GPU SIMD Unit

\[ a = b + c; \]
\[ \text{if } (a < 0) \]
\[ \quad ++b; \]
\[ \text{else} \]
\[ \quad ++c; \]

~PTX

\[ \text{fadd} \]
\[ \text{cmp, jge } l_a \]
\[ \text{fadd, jmp } l_b \]
\[ l_a: \]
\[ \text{fadd} \]
\[ l_b: \]

(Based on http://bps10.idav.ucdavis.edu/talks/03-fatahalian_gpuArchTeraflop_BPS_SIGGRAPH2010.pdf)
SPMD On A GPU SIMD Unit

\[ a = b + c; \]
\[ \text{if } (a < 0) \]
\[ ++b; \]
\[ \text{else} \]
\[ ++c; \]

(Based on http://bps10.idav.ucdavis.edu/talks/03-fatahalian_gpuArchTeraflop_BPS_SIGGRAPH2010.pdf)
a = b + c;
if (a < 0)
    ++b;
else
    ++c;

(Based on http://bps10.idav.ucdavis.edu/talks/03-fatahalian_gpuArchTeraflop_BPS_SIGGRAPH2010.pdf)
SPMD on SIMD Execution

Transform control-flow to data-flow

if (test) {
    true stmts;
} else {
    false stmts;
}

old_mask = current_mask

if (test) {
    test_mask = evaluate test
    current_mask &= test_mask
    // emit true stmts, predicate with current_mask
    current_mask = old_mask & ~test_mask
    // emit false stmts, predicate with current_mask
    current_mask = old_mask

[Allen et al. 1983, Karrenberg and Hack 2011]
• Map *program instances* to individual lanes of the SIMD unit
• e.g. 8 instances on 8-wide AVX SIMD unit
• A *gang* of program instances runs concurrently
• One gang per hardware thread / execution context
void sqr4(float value) {
    for (int i = 0; i < 4; ++i)
        value *= value;
}

Scalar + Vector Computation

• “Uniform” variables have a single value over the set of SPMD program instances

• Stored in scalar registers

• Perf benefits: multi-issue, BW, control flow coherence

• Geomean 2.22x perf. benefit on example workloads

```c
void sqr4(float value) {
    for (uniform int i = 0; i < 4; ++i)
        value *= value;
}
```
Simple Example: Reduction

C++

```cpp
int count = .....;
int *a = new int[count];
// initialize a[...]
int sum = array_sum(a, count);
```

ispc

```ispc
export uniform int
array_sum(uniform int a[],
          uniform int count) {
    int partial = 0;
    for (uniform int i = 0; i < count;
         i += programCount)
        partial += a[i + programIndex];
    return reduce_add(sum);
}
```
const uniform int task_size = 4096;

export uniform int
array_sum(uniform int a[], uniform int count) {
    int n_tasks = count / task_size;

    uniform int sum = 0;
    launch [n_tasks] sum_task(a, count, sum);
    return sum;
}
const uniform int task_size = 4096;

export uniform int
array_sum(uniform int a[], uniform int count) {
    int n_tasks = count / task_size;

    uniform int sum = 0;
    launch [n_tasks] sum_task(a, count, sum);
    return sum;
}

task void
sum_task(uniform int a[], uniform int count, uniform int &sum) {
    uniform int start = task_size * taskIndex;
    uniform int end = min(task_size * (taskIndex + 1), count);

    int partial = 0;
    foreach (i = start ... end)
        partial += a[i];

    uniform int local_sum = reduce_add(partial);
    atomic_add_global(&sum, local_sum);
}
float value = ...;

uniform float tmp[programCount];
tmp[programIndex] = value;

value = tmp[(programIndex + 1) % programCount];

• Program execution is *maximally converged*

• Program instances can communicate, without explicit synchronization, at program sequence *points*

• See user’s guide for details
struct Point {
    float x, y, z;
}

uniform Point a[...];
int index = { 0, 1, 2, ... };
float x = a[index].x;
Data Layout: SOA

```c
struct Point4 {
    float x[4], y[4], z[4];
};

uniform Point4 a[...];
int index = { 0, 1, 2, ... };
float x = a[index / 4].x[index & 3];
```

| x0 | x1 | x2 | x3 | y0 | y1 | y2 | y3 | z0 | z1 | z2 | z3 | x4 | x5 | ...
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---

float x = a[index / 4].x[index & 3]
Data Layout: SOA

```c
struct Point {
    float x, y, z;
};

soa<4> Point a[...];
int index = { 0, 1, 2, ... };
float x = a[index].x;
```

![Data Layout Diagram]

```c
float x = a[index].x;
```
## Performance vs. Serial C++

<table>
<thead>
<tr>
<th>Test Case</th>
<th>1 core / 1 thread x 8-wide AVX</th>
<th>4 cores / 8 threads x 8-wide AVX</th>
</tr>
</thead>
<tbody>
<tr>
<td>AO Bench</td>
<td>6.19x</td>
<td>28.06x</td>
</tr>
<tr>
<td>Binomial</td>
<td>7.94x</td>
<td>33.43x</td>
</tr>
<tr>
<td>Black-Scholes</td>
<td>8.45x</td>
<td>32.48x</td>
</tr>
<tr>
<td>Deferred Shading</td>
<td>5.02x</td>
<td>23.06x</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>6.21x</td>
<td>20.28x</td>
</tr>
<tr>
<td>Perlin Noise</td>
<td>5.37x</td>
<td>-</td>
</tr>
<tr>
<td>Ray Tracer</td>
<td>4.31x</td>
<td>20.29x</td>
</tr>
<tr>
<td>Stencil</td>
<td>4.05x</td>
<td>15.53x</td>
</tr>
<tr>
<td>Volume Rendering</td>
<td>3.60x</td>
<td>17.53x</td>
</tr>
</tbody>
</table>
## Performance vs. Serial C++

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Performance x</th>
</tr>
</thead>
<tbody>
<tr>
<td>AO Bench</td>
<td>182.36x</td>
</tr>
<tr>
<td>Binomial</td>
<td>63.85x</td>
</tr>
<tr>
<td>Black-Scholes</td>
<td>83.97x</td>
</tr>
<tr>
<td>Ray Tracer</td>
<td>195.67x</td>
</tr>
<tr>
<td>Volume Rendering</td>
<td>243.18x</td>
</tr>
</tbody>
</table>

40 cores / 80 threads x 4-wide SSE
Related Work

- C* (Thinking Machines), MPL (MasPar), CUDA, OpenCL
- RenderMan shading language
- VecIMP, IVL
- Task parallel systems: Cilk, OpenMP, TBB, GCD, ConcRT, ...
ispc is Open Source

- Released June 2011—thousands of downloads since then
- BSD license
- Built on top of LLVM
- \{OS X, Linux, Windows\} x \{32, 64 bit\} x \{SSE2, SSE4, AVX, AVX2\}

http://ispc.github.com
Summary

• Provide highly-optimizing, programmer-controlled transformations

• SPMD on SIMD, soa qualifier

• Ease of (incremental) adoption, integration

• Share application data structures, no driver/data copying, lightweight function-call boundary, C-based syntax

• See paper for discussion of key optimizations performed by compiler...
Acknowledgements

- Tim Foley, Geoff Berry
- The LLVM developers
- Geoff Lowney, Jim Hurley, Elliot Garbus

http://ispc.github.com
Optimization Notice

Intel compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.