Compiling for Scalable Computing Systems – the Merit of SIMD

Ayal Zaks
Intel Corporation
Acknowledgements: too many to list
Takeaways

1. SIMD is mainstream and ubiquitous in HW
2. Compiler support for SIMD is necessary, but it’s insufficient
3. Help SIMD become mainstream in SW
   – with Compiler, Programming Language and Tools support
More Cores, Threads, Wider SIMD

<table>
<thead>
<tr>
<th>Core(s)</th>
<th>Thread(s)</th>
<th>SIMD Width</th>
<th>Intel® Xeon® processor</th>
<th>Intel® Xeon® processor</th>
<th>Intel® Xeon® processor</th>
<th>Intel® Xeon® processor</th>
<th>Intel® Xeon® processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>128</td>
<td>64-bit SSE3</td>
<td>5100 series SSSE3</td>
<td>5500 series (Nehalem) SSE4.2</td>
<td>5600 series SSE4.2</td>
<td>Sandy Bridge EP AVX</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>128</td>
<td>5500 series (Nehalem) SSE4.2</td>
<td>5600 series SSE4.2</td>
<td>Sandy Bridge EP AVX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>128</td>
<td>5500 series (Nehalem) SSE4.2</td>
<td>5600 series SSE4.2</td>
<td>Sandy Bridge EP AVX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>128</td>
<td>5500 series (Nehalem) SSE4.2</td>
<td>5600 series SSE4.2</td>
<td>Sandy Bridge EP AVX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>128</td>
<td>5500 series (Nehalem) SSE4.2</td>
<td>5600 series SSE4.2</td>
<td>Sandy Bridge EP AVX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>256</td>
<td>Sandy Bridge EP AVX</td>
<td>Ivy Bridge EP AVX2</td>
<td>Ivy Bridge EP AVX2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>36</td>
<td>256</td>
<td>Sandy Bridge EP AVX</td>
<td>Ivy Bridge EP AVX2</td>
<td>Ivy Bridge EP AVX2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>&gt;61</td>
<td>512</td>
<td>Knights Corner</td>
<td>Knights Landing1 AVX-512</td>
<td>Knights Landing1 AVX-512</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Product specification for launched and shipped products are available on [ark.intel.com](http://ark.intel.com). 1. Not launched or in planning.

1. SIMD is mainstream and ubiquitous in HW
Single Instruction **Multiple Data** in a

```c
float a[N], b[N];
for (int i = 0; i < 8; ++i) {
    c[i] = a[i] + b[i];
}
```

**Compiler Vectorization**

16 x 256-bit registers
In each register, e.g.,
8 float or 4 double or
8 int or 4 long

```
VADDPS YMM0, YMM1, YMM2
```

32 x 512-bit registers
In each register, e.g.,
16 float or 8 double or
16 int or 8 long

```
VADDPS ZMM0, ZMM1, ZMM2
```
Wider Vectors + Richer Instructions

2. Compiler support for SIMD is necessary
Insufficient? Solved in the 70’s, no?

The CRAY-1’s Fortran compiler (cft) is designed to give the scientific user immediate access to the benefits of the CRAY-1’s vector processing architecture. An optimizing compiler, cft, “vectorizes” innermost DO loops. Compatible with the ANSI 1966 Fortran Standard and with many commonly supported Fortran extensions, cft does not require any source program modifications or the use of additional nonstandard Fortran statements to achieve vectorization. Thus the user’s investment of hundreds of man months of effort to develop Fortran programs for other contemporary computers is protected.

Solved innermost DO loop auto-vectorization
DO 1 k = 1,n
1   A(k) = B(k) + C(k)

K=1
Ld C(1)
Ld B(1)
Add
St A(1)

K=2
Ld C(2)
Ld B(2)
Add
St A(2)

K=1..2
Ld C(1)  Ld C(2)
Ld B(1)  Ld B(2)
Add       Add
St A(1)   St A(2)

Scalar code

GCC by Dorit Nuzman/IBM
LLVM by Nadav Rotem/Apple

Vector code

Vector code generation is straightforward
Emphasis on analysis and disambiguation
Vectorization Today

for(p=0; p<N; p++) {
    // Brown work
    if(...) {
        // Green work
    } else {
        // Red work
    }
    while(...) {
        // Gold work
        // Purple work
    }
    y = foo(x);
    // Pink work
}

Vector code generation has become a more difficult problem
Two main Divergence challenges for SIMD: 1. Control, 2. Data
Vectorization Today: Challenges*

- Divergence Analysis
- Predication and Masking Optimizations
- Outer-loop Vectorization
- Less-than-full-vector Vectorization
- Gather/Scatter Optimizations
- Sophisticated Idioms Vectorization
- Partial Vectorization
- Function Vectorization

* nonexclusive list

Insufficient: Increasing need for user guidance for correctness and profitability
Vector Program Development

Applicability
Development cost
Performance

Intrinsics

Maintainability
Scalability

Explicit Vectorization

Automatically using the compiler

MS128 t1, t2;
  t1 = _mm_mulps(&a[i], &b[i]);
  t2 = _mm_addps(t1, t1);

#pragma omp simd
for (i=0; i<n; i++)
  c[i] = a[i] + b[i];
Explicit Vectorization – Example

```c
#pragma omp parallel for
for (int y = 0; y < ImageHeight; ++y) {
    #pragma omp simd
    for (int x = 0; x < ImageWidth; ++x) {
        count[y][x] = mandelbrot(in_vals[y][x], max_iter);
    }
}
```

```c
#pragma omp declare simd uniform(limit)
int mandelbrot(fcomplex c, int limit) {
    fcomplex z = c; int iters = 0;
    while ((cabsf(z) < 2.0f) && (iters < limit)) {
        z = z * z + c; iters++;
    }
    return iters;
}
```

Graphs showing Mandelbrot Normalized Speedup with different thread counts.
(Explicit) Whole Function Vectorization

• WFV: Ralf Karrenberg, Saarland U.
  – Expands vectorization across function boundary
  – Originally for implicit data-parallel languages

• OpenMP 4.0:
  – Employs WFV with new, explicit SIMD annotations
  – Recent examples published include SIMD examples
    http://openmp.org/mp-documents/openmp-examples-4.0.2.pdf
Some Recent References re:WFV*

• Books:
  – Automatic SIMD Vectorization of SSA-based Control Flow Graphs, Ralf Karrenberg, July 2015
  – High Performance Parallelism Pearls volume 2, James Reinders and Jim Jeffers, August 2015

• Conference Papers:
  – Locality-centric thread scheduling for bulk-synchronous programming models on CPU architectures, Hee-Seok Kim et al., CGO 2015. Effectively maximizes WFV factors where possible and profitable on CPUs
  – Exploring the Design Space of SPMD Divergence Management on Data-Parallel Architectures, Yunsup Lee et al., Micro-47. Compares SW and HW techniques for supporting control divergence on GPUs
  – The Impact of the SIMD Width on Control-Flow and Memory Divergence, Thomas Schaub et al., HiPEAC 2015. Examines scalability upto 1024 *element* SIMD.
  – Optimizing Overlapped Memory Accesses in User-directed Vectorization, Diego Caballero et al., to appear in ICS 2015. Technique and OpenMP proposal to support memory divergence, on Xeon Phi

• Workshop papers
  – Predicate Vectors If You Must, Shahar Timnat et al., WPMVP@PPoPP 2014
  – Streamlining Whole Function Vectorization in C using Higher Order Vector Semantics, Gil Rapaport et al., PLC@IPDPS 2015

* nonexclusive list
Feedback from static optimizing compilation improving
Intel® Advisor XE – Vectorization Advisor

Integrates Compiler diagnostics + Performance Data + SIMD efficiency stats
Guidance: detect penalties and recommend improvements using OpenMP 4.0
Deep dive: dependence checkers, memory access pattern analysis
Takeaways

1. SIMD is mainstream and ubiquitous in HW
2. Compiler support is necessary, fun, but insufficient
3. Help SIMD be mainstream in SW by Explicit Vectorization
   – Need user guidance for today’s applications
   – Similar to what OpenMP/Cilk/TBB did for parallelization
   – Maps threaded execution to SIMD hardware
   – Has advantages in development cost, applicability, performance, maintenance, scalability
4. Tools are improving
The 4th Compiler, Architecture and Tools Conference

Mark Your Calendars and Stay Tuned!

November 23rd, 2015

@ Intel IDC Haifa

Organized by: Gadi Haber, Ayal Zaks, Michal Nir and Leeor Peled from Intel SSGi and PEG Dorit Nuzman from IBM, Erez Petrank from the Technion, Yosi Ben-Asher from Haifa U.
Legal Disclaimer & Optimization Notice

• INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

• Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

• Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers 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 SSE2, SSE3, and 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. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
THE END