Expressing vectorization

Milind Girkar
Intel Corp.
The Goals of Intel® Architecture are to deliver:

- Industry leading performance/watt for serial & highly parallel workloads
- Optimized Efficiency for a Heterogeneous Solution in combination with Intel® Xeon® processors
- Complete set of software tools to deploy scalable solutions efficiently
Overview

Multiple Levels of Parallelism

SIMD

Node: Threading, and Threading Help

Mixing CPU and Intel® Xeon Phi™

Cluster Analysis
Hotspot analysis results

Intel VTune Amplifier XE 2011

Hotspots - Hotspots

Analysis Target: /Function /Call Stack

- initialize_2D_buffer
  - setup_2D_buffer ← render_one_pixel ← draw_trace ← thread
- grid_intersect
- sphere_intersect
- video::main_loop

Selected 1 row(s):

CPU Time:

- 5.202s
- 5.202s
- 0.674s
- 0.507s
- 0.204s

CPU time:

- 1 stack(s) selected. Viewing 1 of 1
- Current stack is 100.0% of selection
- 100.0% (5.202s of 5.202s)

Threads:

- WinMainCRTStartup...
- Thread (0x5f5c)
- thread_video (0x988)

CPU Usage:

- Threads
- CPU Time

No filters are applied.

Module: [All] 
Thread: [All] 
Call Stack Modes: Only user functions

Copyright © 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
CPU HW Sampling results
### Source View

**tachyon_vtune_amp_xe - Microsoft Visual Studio**

![Image of Visual Studio interface showing Intel VTune Amplifier XE 2011](source_view.png)

#### Hotspots - Top-down Tree

<table>
<thead>
<tr>
<th>Address</th>
<th>Line</th>
<th>Assembly</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x103a</td>
<td>82</td>
<td>jnl 0x1058 &lt;Block 8&gt;</td>
<td>0.606s</td>
</tr>
<tr>
<td>0x103c</td>
<td>Block 7:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x103f</td>
<td>84</td>
<td>mov edx, dword ptr [ebp+0x0]</td>
<td>0.130s</td>
</tr>
<tr>
<td>0x1041</td>
<td>84</td>
<td>mov eax, dword ptr [edx]</td>
<td>0.079s</td>
</tr>
<tr>
<td>0x1044</td>
<td>84</td>
<td>add eax, 0x2</td>
<td>1.140s</td>
</tr>
<tr>
<td>0x1047</td>
<td>84</td>
<td>mov ecx, dword ptr [ebp-0x8]</td>
<td>0.424s</td>
</tr>
<tr>
<td>0x104d</td>
<td>84</td>
<td>imul ecx, ecx, 0xb4</td>
<td>0.066s</td>
</tr>
<tr>
<td>0x1050</td>
<td>84</td>
<td>mov edx, dword ptr [ebp-0x4]</td>
<td>0.043s</td>
</tr>
<tr>
<td>0x1053</td>
<td>84</td>
<td>mov dword ptr [edx+ecx*4],</td>
<td>0.536s</td>
</tr>
<tr>
<td>0x1056</td>
<td>85</td>
<td>jmp 0x102a &lt;Block 5&gt;</td>
<td>1.073s</td>
</tr>
<tr>
<td>0x1058</td>
<td>Block 8:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1058</td>
<td>86</td>
<td>jmp 0x100f &lt;Block 2&gt;</td>
<td>2.463s</td>
</tr>
</tbody>
</table>

---

*Other brands and names are the property of their respective owners.*
Intel® Inspector XE 2011
Memory and Thread Analysis

Memory Analysis finds
• Memory leaks and memory corruption
• Memory allocation & de-allocation API mismatches
• Inconsistent memory API usage

Thread Analysis finds
• Data races and Deadlocks
• Thread and sync APIs used
• Latent bugs within increasing complex parallel programs
• Stack memory accesses by another thread
Locks analysis type - results

Sorted list of synch objects causing the most thread wait time

Clicking on a synch object displays the source code for the acquisition of that object
Intel® Advanced Vector Extensions

Since 2001:
- 128-bit Vectors

2010
2011
2012
2013

Roadmap illustration - subject to change

Goal: 8X peak FLOPs over 4 generations

AVX 1.0: 2X flops: 256-bit wide floating-point vectors

Ivybridge (22nm Tick)

Half-float support, Random Numbers

Haswell (22 nm Tock)

AVX2: FMA (2x peak flops)
256-bit integer SIMD. “Gather” Instructions.

Knights Landing /Future Xeon

512- bit Vectors
32 registers
Masking, Broadcast

Sandy Bridge (32 nm Tock)

Since 2001: 128-bit Vectors

Performance / core

Copyright© 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Intel® AVX Technology

256b AVX1 → 256b AVX2 → 512b AVX-512

<table>
<thead>
<tr>
<th>AVX</th>
<th>AVX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>256-bit basic FP</td>
<td>Float16 (IVB 2012)</td>
</tr>
<tr>
<td>16 registers</td>
<td>256-bit FP FMA</td>
</tr>
<tr>
<td>NDS (and AVX128)</td>
<td>256-bit integer</td>
</tr>
<tr>
<td>Improved blend</td>
<td>PERMD</td>
</tr>
<tr>
<td>MASKMOV</td>
<td>Gather</td>
</tr>
<tr>
<td>Implicit unaligned</td>
<td></td>
</tr>
</tbody>
</table>

SNB 2011   HSW 2013

AVX-512
- 512-bit FP/Integer
- 32 registers
- 8 mask registers
- Gather/Scatter
- Embedded rounding
- Embedded broadcast
- Scalar/SSE/AVX “promotions”
- HPC additions
- Transcendental support
- Gather/Scatter

Future Processors in planning, subject to change

*Other brands and names are the property of their respective owners.

Copyright © 2013, Intel Corporation. All rights reserved.
Vectorization

Conversion of serial code into SIMD instructions that simultaneously operate on multiple data elements.

Vector code, four elements per iteration

Scalar code, one element per iteration

float a[N], b[N]; int i;
for (i = 0; i < N; i++)
a[i] = a[i] + b[i];

loop:
  movaps xmm0, _a[eax]
  addps xmm0, _b[eax]
  movaps _a[eax], xmm0
  add eax, 16
  cmp eax, ecx
  jl loop

Copyright © 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Vectorization Directives

Data dependence hints
- "restrict" keyword
- #pragma ivdep

Loop count hints
- #pragma loop count (<int>)
- #pragma loop [min | max | avg] count (<int>)

Heuristic-related hints
- #pragma novector
- #pragma vector always

Data alignment directive
- C/C++
  - Windows: __declspec(align(16)) float A[1000];
  - Linux/MacOS:
    float A[1000] __attribute__((aligned (16)));
- Fortran
  - !DIR$ ATTRIBUTES ALIGN: 16:: A

Data alignment assertion (16B example)
- C/C++: __assume_aligned(p,16);
- Fortran: !DIR$ ASSUME_ALIGNED A(1):16

Aligned loop assertion
- C/C++: #pragma vector aligned
- Fortran: !DIR$ VECTOR ALIGNED

Aligned malloc
- _aligned_malloc()
- _mm_malloc()
Motivation for Explicit Vectorization

- Most users do not rely on automatic threading/vectorization
  - Subject to limits of compiler analysis capabilities…
  - Most frequent reason: Dependence issues
  - Many other potential reasons
    - Function calls in loop block
    - Complex control flow / conditional branches
    - Loop not “countable”
    - Not inner loop
    - Loop body too complex
    - Vectorization seems inefficient

- For threading, most users have moved to explicit expression
  - OpenMP*, Intel® Cilk™, Intel® Threading Building Blocks, others
  - So, Intel started the Intel® Cilk™ Plus effort
  - Some parts now standardized into OpenMP 4.0

*OpenMP is a trademark of the OpenMP Architecture Review Board.
Explicit vectorization

```c
#pragma simd

for (int k=0; k<foo(); k++) {
    int t; // private
    t = a[k] + b[k];
    c[k] = t;
    while () { ... }
}

c[0:n] = a[0:n] + b[0:n];
```

- Treat it like a countable loop
- Local variables considered private.
- Reductions etc.
- Other structured control flow constructs
- Fortran array sections in C/C++
Intel Cilk™ Plus Language extensions

- SIMD functions
- Array Notation
- SIMD loops
- Cilk keywords
SIMD functions

//callee.c
__declspec(vector) // CilkPlus
#pragma omp declare simd // OpenMP 4.0
float myfunc(float x, float y)
{
    return x*x + y*y;
}

//SIMD version created by the compiler
// __m128 == float4
__m128 myfunc$vec(__m128 vx, __m128 vy)
{
    return vx*vx + vy*vy;
}
Using SIMD functions

SIMD function prototype

// caller.c
// caller knows that vector version of myfunc exists
__declspec(vector)
float myfunc(float x, float y);
extern float c[N], a[N], b[N];

void caller1(int n)
{
    c[0:n] = myfunc(a[0:n], b[0:n]);
}

void caller2(int n)
{
    #pragma simd
    for (int k=0; k<n; k++)
        c[k] = myfunc(a[k], b[k]);
}

void caller3(int n)
{
    _Cilk_for (int k = 0; k < n ; k++) {
        c[k] = myfunc(a[k], b[k]);
    }
}
Supporting Xeon Phi as coprocessor

• Intel pragmas for offload supported in Intel Parallel Studio XE 2013 for Linux

• Intel Compiler 14.0 supports the recently standardized OpenMP* 4.0 device constructs

• OpenMP 4.0, Intel offload pragmas are broadly equivalent

*The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board.
<table>
<thead>
<tr>
<th>Topic</th>
<th>OpenMP 4.0 Target</th>
<th>Intel Pragma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data placement</td>
<td>data clauses</td>
<td>map(to,from,tofrom,alloc)</td>
</tr>
<tr>
<td>Code placement</td>
<td>data construct</td>
<td>target data</td>
</tr>
<tr>
<td>Offloading</td>
<td>update construct</td>
<td>target update</td>
</tr>
<tr>
<td>Declare target call construct</td>
<td>declare target directive</td>
<td>declare_target</td>
</tr>
<tr>
<td>Device clause</td>
<td>device clause</td>
<td>device clause, ICV</td>
</tr>
<tr>
<td>API routines</td>
<td></td>
<td>get/set dev num</td>
</tr>
<tr>
<td>Asynchronous/Synchronous control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Re-use tasking and add task dependency(in</td>
<td>out</td>
<td>inout)</td>
</tr>
<tr>
<td>Array sections</td>
<td>array sections</td>
<td>array sections</td>
</tr>
</tbody>
</table>
Example – Mandelbrot

```c
#pragma omp declare target
#pragma omp declare simd simdlen(16)

uint32_t mandel(fcomplex c)
{
    // Computes number of iterations (count variable) that it takes
    // for parameter c to be known to be outside mandelbrot set
    uint32_t count = 1; fcomplex z = c;
    for (int32_t i = 0; i < max_iter; i += 1) {
        z = z * z + c;
        int t = cabsf(z) < 2.0f;
        count += t;
        if (!t) { break;}
    }
    return count;
}

#pragma omp target device(0) map(to:in_vals) map(from:count)
#pragma omp parallel for schedule(guided)
for (int32_t y = 0; y < ImageHeight; ++y) {
    #pragma omp simd
    for(int32_t x = 0; x < ImageWidth; ++x) {
        count[y][x] = mandel(in_vals[y][x]);
    }
}
```
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. See slide 22 for configurations and benchmark results disclaimer.
Results have been measured by Intel based on software, benchmark or other data of third parties and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance. Intel does not control or audit the design or implementation of third party data referenced in this document. Intel encourages all of its customers to visit the websites of the referenced third parties or other sources to confirm whether the referenced data is accurate and reflects performance of systems available for purchase.

Configuration: Intel® Core™ i7 CPU X980 system (6 cores with Hyper-Threading On), running at 3.33GHz, with 4.0GB RAM, 12M smart cache, 64-bit Windows Server 2008 R2 Enterprise SP1. For more information go to http://www.intel.com/performance
Working with the Standards Community

CilkPlus Extensions

gcc, llvm in the works

Open sourced the cilk runtime

Proposed to C++, WG formed

ISO/IEC JTC1/SC22/WG21; INCITS/PL22.16 (C++)

INCITS/PL22.3 (Fortran)

OpenMP 4.0
Summary

• Efficient performance through vectorization/parallelism is beyond what can be done automatically

• Explicit vectorization achieves predictable vectorization
  • Similar to what OpenMP does for parallelization

• Working towards standardization and supporting standardized parallelism/vectorization/offload constructs
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product or der.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725 begin_of_the_skype_highlighting 1-800-548-4725 FREE end_of_the_skype_highlighting, or go to: http://www.intel.com/design/literature.htm

Copyright© 2013, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk 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