Stencil Optimizations for Intel® Xeon® Scalable processors via YASK – Yet Another Stencil Kernel

2018 Rice Oil & Gas HPC Conference
March 12-13, 2018

Chuck Yount, Principal Engineer, Intel Corporation, AZ, USA
chuck.yount@intel.com

https://github.com/intel/yask
Introduction and motivation
- Target Domain: HPC stencils
- Intel® Xeon® Scalable Processors
- Challenge: automate code modernization for stencils

YASK
- Scope and goals
- Block diagram
- Example features: DSL, auto-tuning, vector-folding
- Performance of Iso3DFD kernels on Intel® Xeon® Gold Processor 6148

Future Work and Summary
YASK Scope: HPC Stencil Computation

- Iterative kernels that update elements in one or more N-dimensional grids using a fixed pattern of computation on neighboring elements
- Fundamental algorithm in many scientific simulations, commonly used for solving differential equations using finite-difference methods (FDM)

Images from https://commons.wikimedia.org
Intel® Xeon® Scalable Processors
(previously code-named Skylake—“SKL” or Skylake Xeon—“SKX”)

• Above graphic shows maximum sockets. Two-socket platforms are common in HPC installations.
• Also available: Gold (5000 Series), Silver (4000 Series), and Bronze (3000 Series)
SIMD Instruction Sets

**AVX-512**: Foundation
- 512-bit FP/Integer Vectors
- 32 SIMD registers
- 8 mask registers
- Vector gather/scatter

**Conflict Detection** for vectorizing histogram-type algorithms

**PreFetch** gather/scatter

**Exponential and Reciprocal** instructions

**Byte and Word** integer SIMD elements

**Double- and Quad-word** int SIMD

**Vector-Length** orthogonality (128 and 256-bit operations)

1. Previous code-names of Intel® Xeon® processors

March 13, 2018

Intel Corporation
Modernizing Stencil Code for Xeon CPUs

Challenges

- Dozens of optimizations to try
  - Some optimizations, like block size, may require tedious tuning
  - Implementing each optimization can be complex and error-prone
  - Many may provide no or negative benefit
  - Tuning changes with each stencil + CPU architecture (or SKU) combination

- Optimal overall tuning requires trading off multiple (sometimes conflicting) optimizations, each with multiple parameters

- Traditional optimization techniques are intermixed with the original code that describes the functional description of the stencil
  - Optimization risks breaking the functionality, adding a testing burden
  - Difficult to change stencil after optimizations are applied
  - Domain experts may reject code that obfuscates the underlying math!
YASK Objectives

Primary Deliverable

- An open-source software framework to rapidly implement high-performance stencil kernels for Intel® Xeon® and Intel® Xeon Phi™ processors

Supporting Strategies

- Enable straightforward specification of stencil equations in a simple and intuitive domain-specific language (DSL)
- Generate high-performing kernel code from the DSL input
  - Separate the functional description from the optimization techniques
  - Expose optimization trade-off options without requiring code changes
  - Automate searching through the optimization design space
- Provide a simple command-line application to test and tune stencil performance
- Provide integration of generated code into larger applications via shared libraries with C++ and Python APIs

Reduce the “ninja gap”
YASK High-Level Flow

Stencil specification in YASK DSL → Stencil compiler → Optimized stencil calculation and prefetch code

Stencil compiler → Loop compiler → Nested loops with OpenMP, prefetch code, etc.

Loop compiler

Optimized stencil kernel library with C++ and Python APIs → and/or

Optimized stencil perf-eval binary

Performance results

Usage models

Application results

Customer application

Build process

Primary input

Primary output

Intel C++ compiler

Other C++ code
Sample YASK Features and Optimizations

Discussed in following slides with isotropic 3D finite-difference example

- Domain-specific language (DSL) for stencils
  - YASK compiler converts simple C++-based functional description to complex vector intrinsics
- Cache-blocking
  - Cache-block sizes are automatically tuned at runtime to increase performance
- Vector folding
  - Multi-dimensional SIMD data layout reduces memory bandwidth demands

To explore beyond the scope of this presentation

- Nested OpenMP
  - Two levels of OpenMP parallelism increase temporal locality of L1 and L2 caches shared between threads and/or cores (especially effective on Xeon Phi tiles)
- Temporal wave-front blocking
  - Allows efficient use of large shared cache (e.g., Xeon Phi MCDRAM in cache mode)
- Software prefetching
  - Automatically generated by stencil compiler for L1 and/or L2
- MPI halo exchange
  - Allows domain decomposition across NUMA domains in a node and across nodes in a cluster

Optimizations applied automatically from DSL input
Iso3DFD Specification in YASK DSL

- 51-point stencil, 61 FP ops
- 16th order accurate in space, 2nd order accurate in time

$$p(t+1, i, j, k) \leftarrow 2p(t, i, j, k) - p(t-1, i, j, k) + v(i, j, k)\left(c_0p(t, i, j, k) + \sum_{r=1}^{8} c_r \left[p(t, i-r, j, k) + p(t, i+r, j, k) + p(t, i, j-r, k) + p(t, i, j+r, k) + p(t, i, j, k-r) + p(t, i, j, k+r)\right]\right)$$

Clear and concise functional definition
Iso3DFD Specification in YASK DSL

- 51-point stencil, 61 FP ops
- 16th order accurate in space, 2nd order accurate in time

\[ p(t + 1, i, j, k) \leftarrow 2p(t, i, j, k) - p(t - 1, i, j, k) + \]
\[ v(i, j, k) \left( c_0 p(t, i, j, k) + \]
\[ \sum_{r=1}^{8} c_r \left[ p(t, i - r, j, k) + p(t, i + r, j, k) + \right. \]
\[ p(t, i, j - r, k) + p(t, i, j + r, k) + \]
\[ p(t, i, j, k - r) + p(t, i, j, k + r) \] \]

```cpp
#include "Soln.hpp"
class Iso3dfdStencil : public StencilRadiusBase {
public:
  MAKE_STEP_INDEX(t);
  MAKE_DOMAIN_INDEX(x);
  MAKE_DOMAIN_INDEX(y);
  MAKE_DOMAIN_INDEX(z);
  MAKE_MISC_INDEX(r); // to index the coeff.
  MAKE_GRID(p, t, x, y, z); // varying 3D grid.
  MAKE_GRID(v, x, y, z); // constant 3D grid.
  MAKE_ARRAY(coeff, r); // FD coefficients.

  Iso3dfdStencil(StencilList& stencils, int radius=8) :
    StencilRadiusBase("iso3dfd", stencils, radius) {};

  virtual void define() {
    GridValue next_p = p(t, x, y, z) * coeff(0);
    for (int r = 1; r <= _radius; r++)
      next_p += (p(t, x-r, y, z) + p(t, x+r, y, z) +
                 p(t, x, y-r, z) + p(t, x, y+r, z) +
                 p(t, x, y, z-r) + p(t, x, y, z+r)) *
                 coeff(r);
    next_p = (2.0 * p(t, x, y, z)) -
              p(t-1, x, y, z) + (next_p * v(x, y, z)) ;
    p(t+1, x, y, z) EQUALS next_p;
  }
}; // class Iso3dfdStencil.
```

Clear and concise functional definition
Applying Internal Block Auto-Tuner to Iso3DFD

- Search ran 162 trials
- Best size combination found at trial 147: 136×32×132 points
- 7.64 to 9.98 G Pts/sec: **1.3x speedup**
- Integrated auto-tuner alleviates need to tune and re-tune for each stencil + architecture combination

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. For more complete information visit www.intel.com/benchmarks. Measurements collected by Intel as of Dec, 2017 on Xeon 6148, Red Hat Enterprise Linux 7.3. See complete configuration details on "Configuration" slide. *Other names and brands may be claimed as the property of others. Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.
8 new vectors must be read for \( k \pm r \) points
(4 for \( k+r \) and 4 for \( k-r \) for \( r=1..4 \))

8 new vectors must be read for \( j \pm r \) points due to overlap along x axis

Only 1 new vector must be read for \( i \pm r \) points

Total memory BW cost for traditional “in-line” vectors = 17 new vector inputs for each vector of output

Inner 3D loop iterates in x direction, i.e., same dimension as vectorization
**2D Vector-Folding**

4 new 4x2x1 vectors must be read for $j \pm r$ points

Only 1 new vector must be read for $k \pm r$ points

2 new vectors must be read for $i \pm r$ points

Inner 3D loop iterates in $z$ direction, i.e., *perpendicular* to 2D vector

Total memory BW cost for 4x2x1 vector with z-axis loop = 7 new vector inputs for each vector of output (*2.4x lower* than in-line)
2D “4x2” vector folding

Logical indices in 2D with 8-element SIMD in x and y dimensions

<table>
<thead>
<tr>
<th></th>
<th>1,1</th>
<th>1,2</th>
<th>1,3</th>
<th>1,4</th>
<th>2,1</th>
<th>2,2</th>
<th>2,3</th>
<th>2,4</th>
<th>1,5</th>
<th>1,6</th>
<th>1,7</th>
<th>1,8</th>
<th>2,5</th>
<th>2,6</th>
<th>2,7</th>
<th>2,8</th>
<th>1,9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5,1</td>
<td>5,2</td>
<td>5,3</td>
<td>5,4</td>
<td>5,5</td>
<td>5,6</td>
<td>5,7</td>
<td>5,8</td>
<td>5,9</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4,1</td>
<td>4,2</td>
<td>4,3</td>
<td>4,4</td>
<td>4,5</td>
<td>4,6</td>
<td>4,7</td>
<td>4,8</td>
<td>4,9</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3,1</td>
<td>3,2</td>
<td>3,3</td>
<td>3,4</td>
<td>3,5</td>
<td>3,6</td>
<td>3,7</td>
<td>3,8</td>
<td>3,9</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2,1</td>
<td>2,2</td>
<td>2,3</td>
<td>2,4</td>
<td>2,5</td>
<td>2,6</td>
<td>2,7</td>
<td>2,8</td>
<td>2,9</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Access to elements in custom memory layout encapsulated behind C++ & Python APIs

Load and permute instructions generated automatically by YASK stencil compiler

• 2D vector folding layout (8×1)
• Two aligned vectors are colored
• Unaligned read shown with bold borders done by loading aligned vectors and then shuffling the requisite elements via an AVX-512 *permute* instruction
Applying Vector Folding to Iso3DFD

- AVX-512 zmm registers hold 16 SP FP values
- 15 configurations tested
  - Traditional vectorization is 1x1x16 points (‘z’ is inner dimension)
  - Default fold for 3D problems is 4x4x1 points
  - Best fold for this stencil is 2x8x1 points
- 6.31 to 9.46 G Pts/sec: 1.5x speedup
- YASK stencil compiler enables rapid exploration of complex optimization options

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. For more complete information visit www.intel.com/benchmarks. See complete configuration details on “Configuration” slide. *Other names and brands may be claimed as the property of others. Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.
Integration with Devito from Imperial College in London

- Devito: tool for performing optimized Finite Difference (FD) computation from high-level symbolic problem definitions
- Performs automated code generation and Just-In-time (JIT) compilation based on symbolic equations defined in SymPy
- http://www.opesci.org/devito
- Using YASK code generator and kernel libraries as a high-performance backend

Improving performance of

- Applying stencils specified in sub-domains, e.g., for absorbing boundary conditions or layers
- MPI exchanges, e.g., by overlapping communications with computation
- Temporal blocking

More example code
- Please contribute!
Summary

YASK: turn-key solution for high-performance stencil-code production

- Simple definition of stencil in a C++-based domain-specific language
- Automatic generation of many state-of-the-art optimizations
- Automatic tuning to adjust blocking to stencil and CPU
- C++ and Python APIs to facilitate integration into applications

Open source code available

- MIT license
- Contains examples for more complex stencils, e.g., full staggered grid (FSG)
- [https://github.com/intel/yask](https://github.com/intel/yask)

Please contact the author with questions or integration assistance
Additional Legal Disclaimers

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Statements in this document that refer to Intel’s plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel’s results and plans is included in Intel’s SEC filings, including the annual report on Form 10-K.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, Xeon, Xeon Phi, the Intel logo and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

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.

© 2016 Intel Corporation