



Guided Code Vectorization with Intel[®] Advisor XE

The Hands-On Workshop (HOW) Series “Tools”

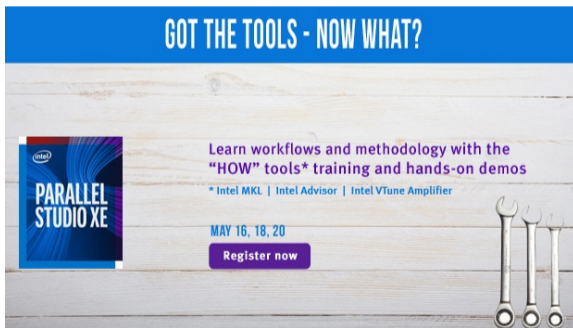
Ryo Asai — Colfax International
colfaxresearch.com

Disclaimer

While best efforts have been used in preparing this training, Colfax International makes no representations or warranties of any kind and assumes no liabilities of any kind with respect to the accuracy or completeness of the contents and specifically disclaims any implied warranties of merchantability or fitness of use for a particular purpose. The publisher shall not be held liable or responsible to any person or entity with respect to any loss or incidental or consequential damages caused, or alleged to have been caused, directly or indirectly, by the information or programs contained herein. No warranty may be created or extended by sales representatives or written sales materials.

About the Series

Hands-On Workshop (HOW “Tools” Series): webinars on efficient programming for the Intel architecture with the help of dedicated software development tools



GOT THE TOOLS - NOW WHAT?

Learn workflows and methodology with the “HOW” tools* training and hands-on demos

* Intel MKL | Intel Advisor | Intel VTune Amplifier

MAY 16, 18, 20

[Register now](#)

colfaxresearch.com/how-tools-16-05

Learn More



THE "HOW" SERIES

DEEP DIVE

WITH CODE MODERNIZATION EXPERTS

STARTS MAY 23

*10x 2-hour sessions | 24-hour 2-weeks remote access to a system | Filling up fast, register now!

Interested? Sign-up at:

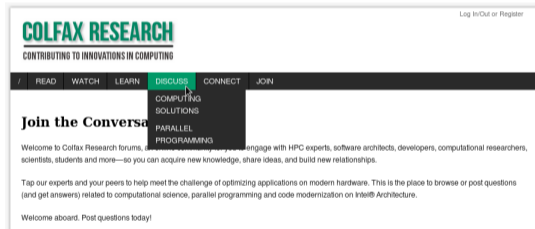
colfaxresearch.com/how-series

Get Your Questions Answered

Chat (for this course):
colfaxresearch.com/how-tools-16-05



Forums (general):
colfaxresearch.com/discussion



§2. Intel Architecture

Computing Platforms

Intel Xeon Processor



Current: Broadwell
Upcoming: Skylake

Multi-Core Architecture

Intel Xeon Phi Coprocessor, 1st generation



Current: Knights Corner (KNC)

Intel Xeon Phi Processor, 2nd generation*



* socket and coprocessor versions

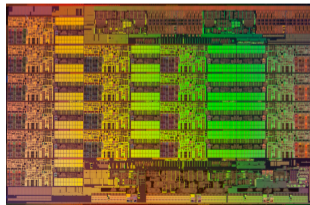
Upcoming: Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture

Intel Xeon CPU: Purpose and Specifications

General-purpose platform for demanding computing applications.

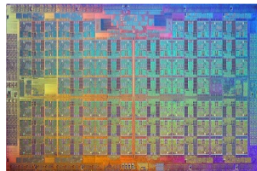
- Up to ~ 1 TFLOP/s in DP
- Up to ~ 2 TFLOP/s in SP
- Up to 768 GiB DDR4 RAM
- ~ 126 GB/s bandwidth
- Hardware-rich: forgiving of sub-optimal code



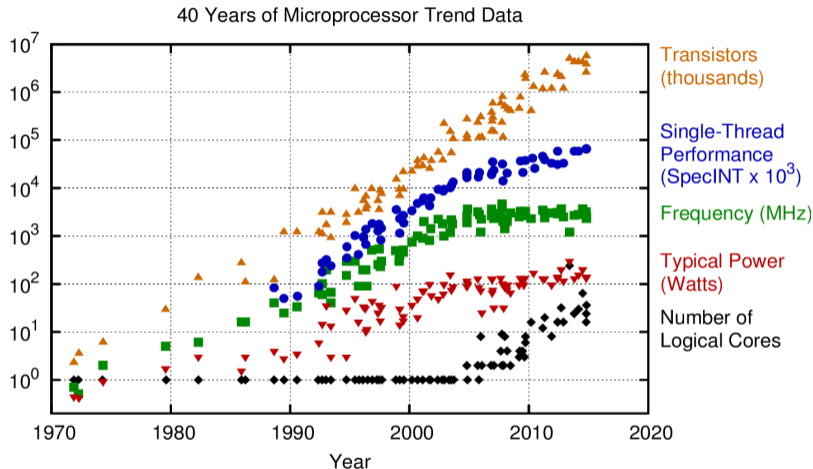
Intel Xeon Phi Processors (2nd Gen)

Specialized platform for demanding computing applications.

- Socket version or coprocessor
- 3+ TFLOP/s in DP
- 6+ TFLOP/s in SP
- Up to 16 GiB MCDRAM
- ~ 400 GB/s MCDRAM bandwidth
- Up to 384 GiB DDR4 RAM
- ~ 90 GB/s DDR4 bandwidth
- Supports common OS
- **Public disclosures**



Trend Towards of Parallelism

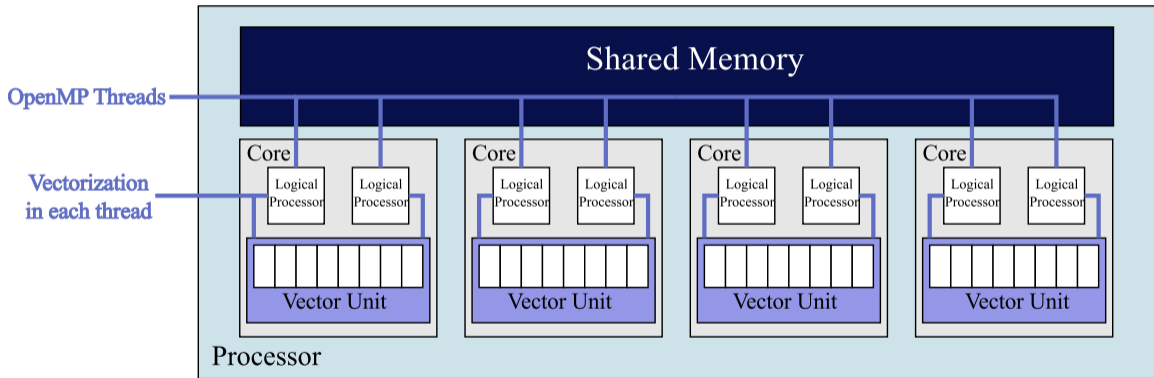


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
 New plot and data collected for 2010-2015 by K. Rupp

Source: <https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/>

Hardware Details of Parallelism

- **Data Parallelism** - Single Instruction Multiple Data (SIMD)
- **Task Parallelism** - Multiple tasks across multiple processor cores



Intel[®] Advisor[™] covers both: today we focus on Data Parallelism

§3. Background: Vectorization

Short Vector Support

Vector instructions – one of the implementations of SIMD (Single Instruction Multiple Data) parallelism.

Scalar Instructions

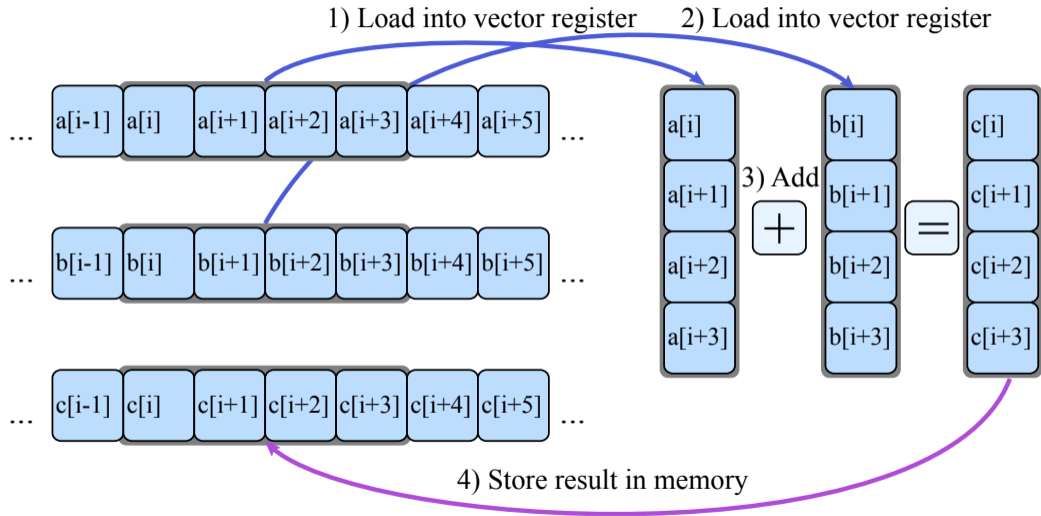
$$\begin{array}{r} 4 + 1 = 5 \\ 0 + 3 = 3 \\ -2 + 8 = 6 \\ 9 + -7 = 2 \end{array}$$

Vector Instructions

$$\begin{array}{r} 4 \\ 0 \\ -2 \\ 9 \end{array} + \begin{array}{r} 1 \\ 3 \\ 8 \\ -7 \end{array} = \begin{array}{r} 5 \\ 3 \\ 6 \\ 2 \end{array}$$

Vector Length

Workflow of Vector Computation



Two Approaches To Vectorization

Automatic Vectorization:

- Vectorization w/ compiler.
- Portable: just recompile.
- Tuning with directives.

```

1 double A[vec_width], B[vec_width];
2 // ...
3
4
5 // This loop will be auto-vectorized
6 for(int i = 0; i < vec_width; i++)
7     A[i]+=B[i];

```

```

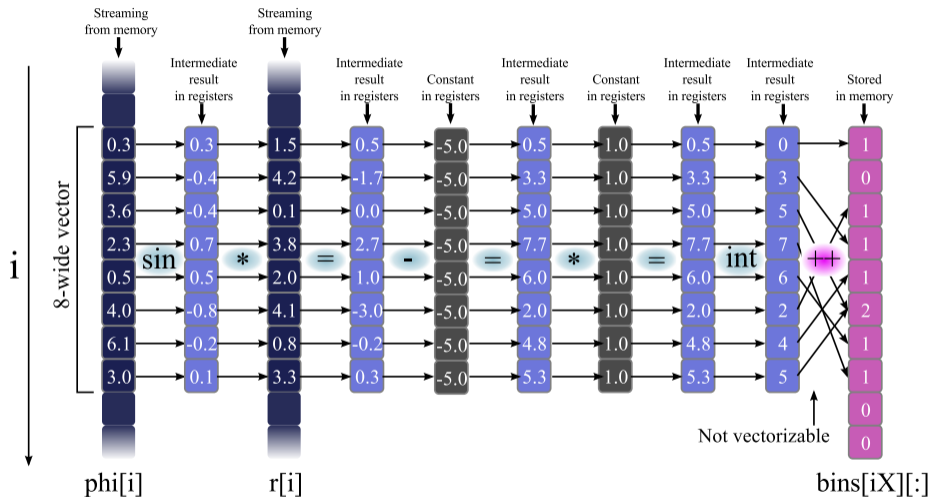
1 double A[vec_width], B[vec_width];
2 // ...
3 // This is explicitly vectorized
4 __m512d A_vec = _mm512_load_pd(A);
5 __m512d B_vec = _mm512_load_pd(B);
6 A_vec = _mm512_add_pd(A_vec, B_vec);
7 _mm512_store_pd(A, A_vec);

```

Explicit Vectorization:

- Vectorization w/ intrinsics
- Full control over instructions
- Limited portability

Auto-Vectorized Loops May Be Complex



See [this paper](#) for more details

Limits of Automatic Vectorization

The compiler will do the best it can. But there are issues it can't resolve.

Unvectorizable loops

```
vega@lyra% cat worker.optrpt
...
LOOP BEGIN at worker.cc(11,5)
  remark: loop was not vectorized: ...
  remark: vector dependence: ...
  remark: vector dependence: ...
LOOP END
```

Inefficient loops

```
vega@lyra% cat worker.optrpt
...
LOOP BEGIN at worker.cc(12,3)
  remark: LOOP WAS VECTORIZED
  remark: masked strided loads: 4
  remark: type converts: 2
LOOP END
```

Advisor guides developers in resolving these issues.

§4. Advisor Basics

Intel® Parallel Studio XE

Software	Composer	Professional	Cluster
Intel C, C++ and Fortran compilers	x	x	x
Intel Math Kernel Library (MKL)	x	x	x
Intel Threading Building Blocks (TBB)	x	x	x
Intel Performance Primitives (IPP)	x	x	x
Intel VTune Amplifier XE		x	x
Intel Inspector XE		x	x
Intel Advisor XE		x	x
Intel MPI Library			x
Intel Trace Analyzer and Collector (ITAC)			x

Setting Up

Starting Advisor GUI:

```
vega@lyra% source /opt/intel/advisor_xe/advixe-vars.sh
vega@lyra% advixe-gui
```

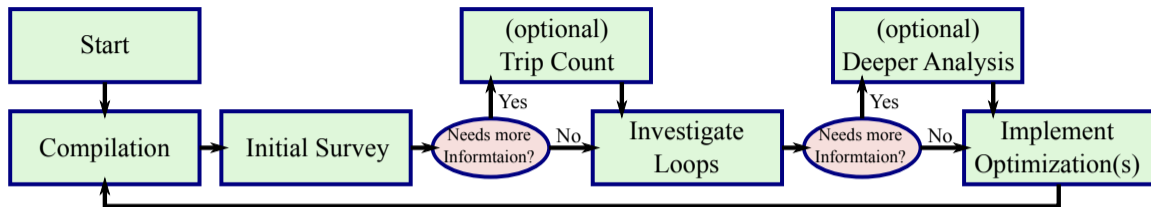
Compile your code:

```
vega@lyra% icpc -g -O3 -xCORE-AVX2 myapp.cc -o run-app
vega@lyra% #alternatively, use -xhost instead of -xCORE-AVX2
```

Create project:

The screenshot shows the Intel Advisor GUI with the 'Survey Launch Application' configuration window open. The window has a tabbed interface with 'Analysis Target', 'Binary/Symbol Search', and 'Source Search' tabs. The 'Survey Launch Application' tab is active, displaying a dropdown menu with 'Survey Launch Application' selected. Below the dropdown, there is a text box with the instruction: 'Specify and configure the application executable (target) to analyze. Press F1 for more details.' A yellow error message box is visible, stating: 'Cannot find application file "/path/to/run-app"'. Below the error message, there is a text input field for 'Application:' containing '/path/to/run-app' and a 'Browse...' button. Below that, there is a text input field for 'Application parameters:' and a 'Modifv...' button.

Advisor Workflow



- **Compilation** – Compile the application with the appropriate flags
- **Initial Survey** – General overview analysis for finding hotspots
- **Trip Count** – Additional information on loop/function call count
- **Investigate Loops** – View analysis details from the Advisor
- **Deeper Analysis** – Gather any additional information requested
- **Implement Optimization** – One optimization per iteration

CLI

For running on remote systems without GUI, use the `advixe-cl` command to run from command line.

```
vega@lyra% # run Survey Analysis
vega@lyra% advixe-cl -c survey ./app-CPU
vega@lyra% # find LoopID to mark
vega@lyra% advixe-cl -report survey -project-dir [proj]
vega@lyra% # run Deeper Analysis with marked loops
vega@lyra% advixe-cl -c map -mark-up-list=[LoopID],[LoopID]
```

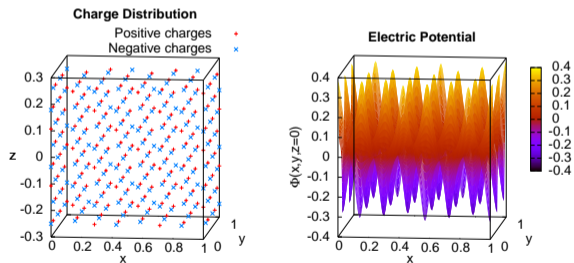
[Advisor documentaion page on CLI](#)

§5. Example Optimization

Example: Coulomb's Law Application

$$\Phi(\vec{R}_j) = - \sum_{i=1}^m \frac{q_i}{|\vec{r}_i - \vec{R}_j|}, \quad (1)$$

$$|\vec{r}_i - \vec{R}| = \sqrt{(r_{i,x} - R_x)^2 + (r_{i,y} - R_y)^2 + (r_{i,z} - R_z)^2}. \quad (2)$$



Paper: <http://xeonphi.com/papers/autovec>

Initial Implementation

```
1 struct Charge {  
2     float x, y, z, q;  
3 } chgs[m]; // Coordinates and value of this charge
```

```
1 for (int i=0; i<m; i++) { // Coulomb's law  
2     // Non-unit stride: (&chg[i+1].x - &chg[i].x) != sizeof(float)  
3     const float dx=chg[i].x - Rx;  
4     const float dy=chg[i].y - Ry;  
5     const float dz=chg[i].z - Rz;  
6     phi -= 8.98755178737e9 * chg[i].q / sqrt(dx*dx+dy*dy+dz*dz);  
7 }
```

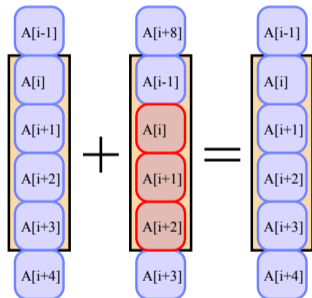
Vector Dependence

True Vector Dependence

```

1 for(int i = 1; i < n; i++)
2   A[i] = A[i] + A[i-1];

```



Vector Dependence:

A memory location that is written to is also read from in the same vector operation.

$A[i]$ is written into in the first SIMD lane but it is also read from in the second SIMD lane.

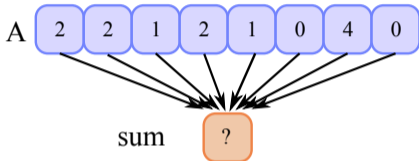
→ Can not be done in data parallel manner

Unsafe to Vectorize: Compiler will not vectorize this

Reduction: Form of Vector Dependence

```

1 for(int i = 0; i < n; i++)
2   sum = A[i];
  
```



Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

Issue: Assumed dependency present

The compiler assumed there is an anti-dependency (Write after read - WAR) or true dependency (Read after write - RAW) in the

Recommendation: Remove dependency

The Dependencies analysis shows there is a real dependency in the loop. To fix: Do one of the following:

- Rewrite the code to remove the dependency.
- If there is an anti-dependency, enable vectorization using the directive `#pragma simd vector_length(k)`, where `k`

Read More:

- [simd](#)
- [Getting Started with Intel Compiler Pragmas and Directives](#) and [Vectorization Resources for Intel® Advisor Users](#)

One way to resolve it is to use the OpenMP SIMD reduction.

```

1 #pragma omp simd reduction({op}:{list})
  
```

- `op` – supported reduction operator (e.g. +, *)
- `list` – reduced scalar variables (e.g. phi, a)

Type Conversion

Consistency of Precision: Constants

```
1 // Bad: 2 is "int"
2 long b=a*2;
3
4 // Bad: overflow
5 long n=100000*100000;
6
7 // Bad: excessive
8 float p=6.283185307179586;
9
10 // Bad: 2 is "int"
11 float q=2*p;
12
13 // Bad: 1e9 is "double"
14 float r=1e9*p;
15
16 // Bad: 1 is "int"
17 double t=s+1;
```

```
1 // Good: 2L is "long"
2 long b=a*2L;
3
4 // Good: correct
5 long n=100000L*100000L;
6
7 // Good: accurate
8 float p=6.283185f;
9
10 // Good: 2.0f is "float"
11 float q=2.0f*p;
12
13 // Good: 1e9f is "float"
14 float r=1e9f*p;
15
16 // Good: 1.0 is "double"
17 double t=s+1.0;
```

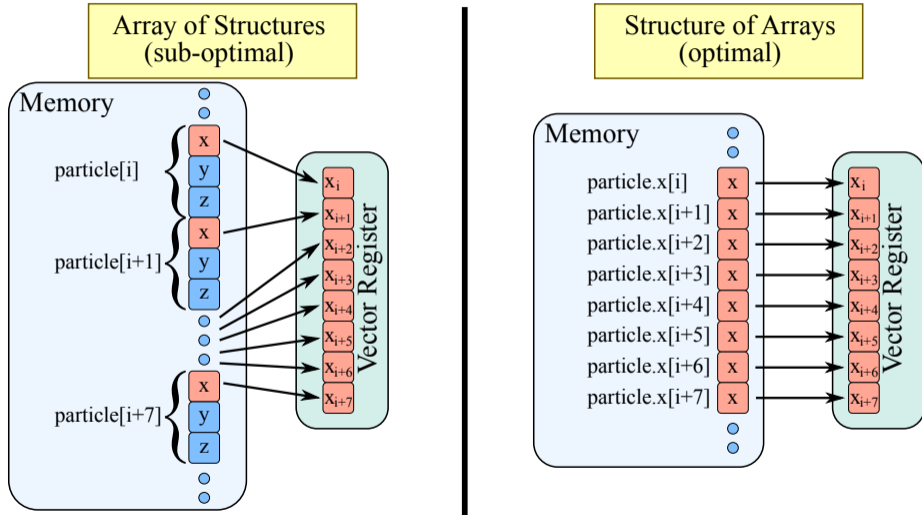
Consistency of Precision: Functions

```
1 // Bad: 3.14 is a double
2 float x = 3.14;
3
4 // Bad: sin() is a
5 // double precision function
6 float s = sin(x)
7
8 // Bad: round() takes double
9 // and returns double
10 long v = round(x);
11
12 // Bad: abs() is not from IML
13 // it takes int and returns int
14 int v = abs(x);
```

```
1 // Good: 3.14f is a float
2 float x = 3.14f;
3
4 // Good: sin() is a
5 // single precision function
6 float s = sinf(x)
7
8 // Good: lroundf() takes float
9 // and returns long
10 long v = lroundf(x);
11
12 // Good: fabsf() is from IML
13 // It takes and returns a float
14 float v = fabsf(x);
```

Data Structure

Arrays of Structures versus Structures of Arrays



Optimized Solution: Structure of Arrays, Unit-Stride Access

```
1 struct Charge_Distribution {  
2     // Data layout permits effective vectorization of Coulomb's law application  
3     float *x, *y, *z, *q; // Arrays of x-, y- and z-coordinates of charges  
4 };
```

```
1 // This version vectorizes better thanks to unit-stride data access  
2 float phi_p = 0.0f;  
3 #pragma omp simd reduction(+: phi_p)  
4 for (int i=0; i<m; i++) {  
5     // Unit stride: (&chg.x[i+1] - &chg.x[i]) == sizeof(float)  
6     const float dx=chg.x[i] - Rx;  
7     const float dy=chg.y[i] - Ry;  
8     const float dz=chg.z[i] - Rz;  
9     phi_p -= chg.q[i] / sqrtf(dx*dx+dy*dy+dz*dz);  
10 }  
11 phi = phi_p;
```

§6. Closing Words

Are We Done?

Vector Efficiency: one metric for determining if you are done.

The screenshot displays the Loop Analytics interface for a specific loop. At the top, a table lists function call sites and loops, with columns for Vector Issues, Self Time, Total Time, Type, Why No Vectorization?, Vectorized Loops, and Instruction Set Analysis. The selected loop is 'Loop in CalculateElectricPotential at w...', which is vectorized (Body) and uses AVX2 instructions. It shows a self time of 152.823s and a total time of 152.823s. The vectorization gain is 6.81x, and the efficiency is approximately 85%. The instruction set analysis shows FMA and Square Roots instructions.

Below the table, the Loop Analytics tab is active, showing a summary of the loop's performance. A callout box indicates that this is an early version of the tab, which will be enhanced over time.

152.823s
Vectorized (Body) Total time

AVX; FMA 152.823s
Instruction Set Self time

Memory 17% (3)
Compute 78% (14)
Other 6% (1)

Instruction Mix Summary

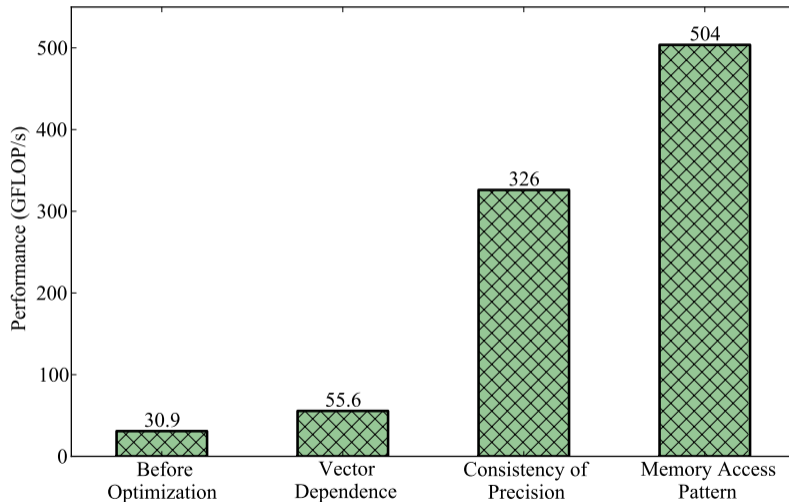
6.81x
Vectorization Gain

~85%
Vectorization Efficiency

Traits
 FMA
 Square Roots

Instruction Mix
 Memory: 3 Compute: 14 Other: 1

Final Performance Data



§7. Resources

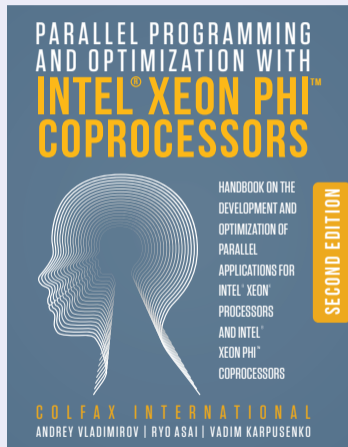
Supplementary Materials: Textbook

ISBN: 978-0-9885234-0-1 (2nd edition, 508 pages, Electronic or Print)

Parallel Programming and Optimization with Intel® Xeon Phi™ Coproprocessors

Handbook on the Development and
Optimization of Parallel Applications
for Intel® Xeon® Processors
and Intel® Xeon Phi™ Coprocessors

© Colfax International, 2015



<http://xeonphi.com/book>

Colfax Research

<http://colfaxresearch.com/>

Learn More



THE "HOW" SERIES

DEEP DIVE

WITH CODE MODERNIZATION EXPERTS

STARTS MAY 23

*10x 2-hour sessions | 24-hour 2-weeks remote access to a system | Filling up fast, register now!

Interested? Sign-up at:

colfaxresearch.com/how-series

Slides, Code, Video

You can download slides, code and watch the video recording of this webinar here (requires registration for a free Colfax Research account):

colfaxresearch.com/how-tools-16-05