



# PROGRAMMING AND OPTIMIZATION FOR INTEL<sup>®</sup> ARCHITECTURE

Hands-On Workshop (HOW) Series "Deep Dive"

Session 3

*Colfax International — [colfaxresearch.com](http://colfaxresearch.com)*

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.

# COURSE ROADMAP

- ▶ Module I. Programming Models
  - 01. Intel Architecture and Modern Code
  - 02. Xeon Phi, Coprocessors, Omni-Path
- ▶ Module II. Expressing Parallelism
  - 03. Automatic vectorization
  - 04. Multi-threading with OpenMP
  - 05. Distributed Computing, MPI
- ▶ Module III. Performance Optimization
  - 06. Optimization Overview: N-body
  - 07. Scalar tuning, Vectorization
  - 08. Common Multi-threading Problems
  - 09. Multi-threading, Memory Aspect
  - 10. Access to Caches and Memory

Course page:

[colfaxresearch.com/how-series](http://colfaxresearch.com/how-series)

- ▶ Slides
- ▶ Code
- ▶ Video
- ▶ Chat

More workshops:

[colfaxresearch.com/training](http://colfaxresearch.com/training)




# GET YOUR QUESTIONS ANSWERED: CHAT



[colfaxresearch.com/how-series](http://colfaxresearch.com/how-series)

# GET YOUR QUESTIONS ANSWERED: FORUMS

	<a href="#">READ</a>	<a href="#">WATCH</a>	<a href="#">LEARN</a>	<a href="#">FORUMS</a>	<a href="#">CONNECT</a>	<a href="#">JOIN</a>
---	----------------------	-----------------------	-----------------------	------------------------	-------------------------	----------------------

**Forum**

**Colfax Cluster**  
Discussion of Colfax Cluster usage policies, troubleshooting.

**Developer Training, HOW Series**  
Questions about any of the Colfax trainings? Usage of training servers, experience with specific exercises, inquiries on what's inside, suggestions for future trainings - post them here.

**Performance Optimization and Parallelism**  
Discuss with Colfax Research and colleagues any topics related to computational science, parallel programming, performance optimization and code modernization.

[colfaxresearch.com/forum](https://colfaxresearch.com/forum)

- ▶ All registrants receive an invitation from `cluster@colfaxresearch.com`
- ▶ Queue-based access to Intel Xeon E5, Intel Xeon Phi (KNC and KNL)
- ▶ Can access the cluster the entire 2 weeks of the workshop



## **§2. SIMD PARALLELISM AND VECTORIZATION**





# **VECTOR INSTRUCTIONS IN INTEL ARCHITECTURE**

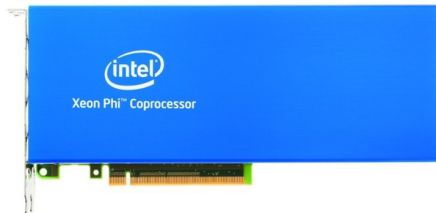
## Intel Xeon Processor



Current: Broadwell  
Upcoming: Skylake

Multi-Core Architecture

## Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

## Intel Xeon Phi Processor, 2nd generation\*



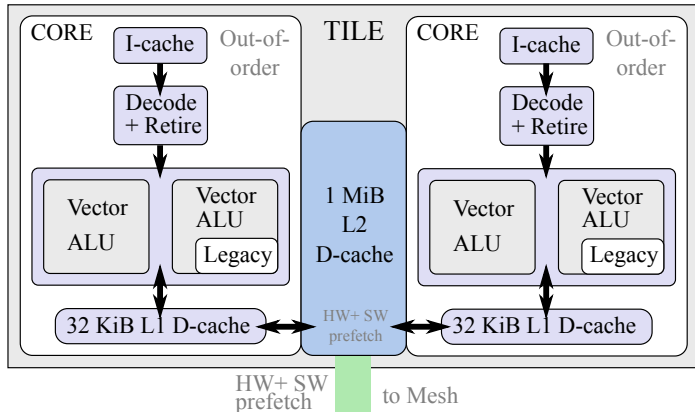
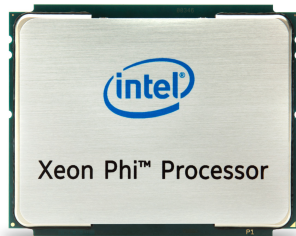
\* socket and coprocessor versions

Knights Landing (KNL)

Intel Many Integrated Core (MIC) Architecture

# KNL CORES

- ▶ Even more power in vector units
- ▶ Binary compatible with Xeon, but in legacy mode



# SHORT VECTOR SUPPORT

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

Scalar Instructions

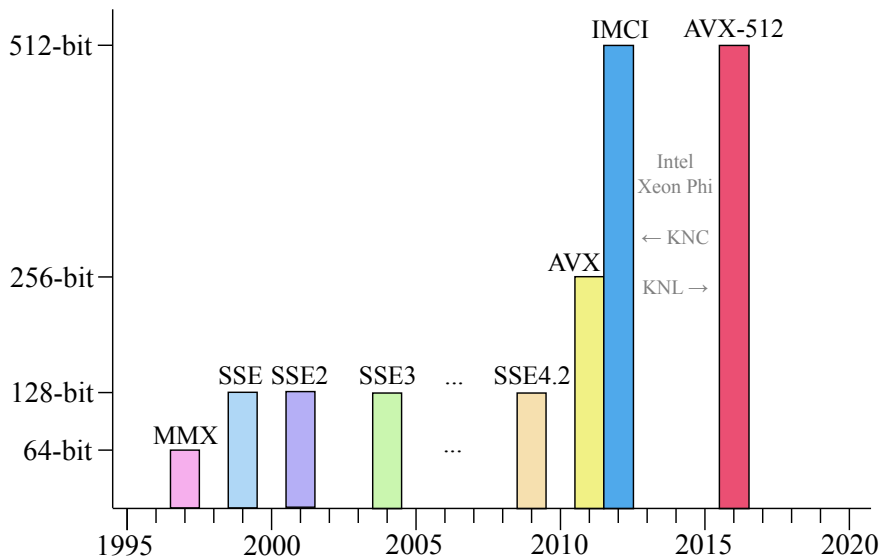
$$\begin{array}{r} 4 + 1 = 5 \\ 0 + 3 = 3 \\ -2 + 8 = 6 \\ 9 + -7 = 2 \end{array}$$

Vector Instructions

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

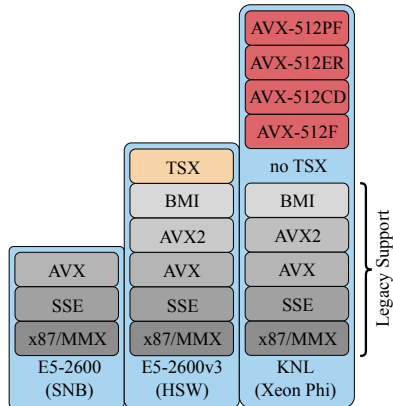
↑  
Vector Length  
↓

# INSTRUCTION SETS IN INTEL ARCHITECTURE



# AVX-512 IN INTEL XEON PHI PROCESSORS

- ▶ Intel® Advanced Vector Extensions 512 (AVX-512)
  - 512-bit vector registers.
  - Hardware gather/scatter, DP transcendental functions support and more.
  - Supported by non-Intel compilers like GCC.
- ▶  $\leq$  Intel® AVX2
  - Legacy mode operation.
  - Binary compatibility with Xeon.
  - Does *not* include IMCI (from KNC).



# AVX-512 MODULES

- ▷ AVX-512F (Fundamentals)
  - Extension of most AVX2 instructions to 512-bit vector registers.
- ▷ AVX-512CD (Conflict Detection)
  - Efficient conflict detection (application: binning).
- ▷ AVX-512ER (Exponential and Reciprocal)
  - Transcendental function (exp, rcp and rsqrt) support.
- ▷ AVX-512PF (Prefetch)
  - Prefetch for scatter and gather.

Learn more: [colfaxresearch.com/knl-avx512](https://colfaxresearch.com/knl-avx512)



## VECTOR INTRINSICS



# USING VECTOR INSTRUCTIONS: TWO APPROACHES

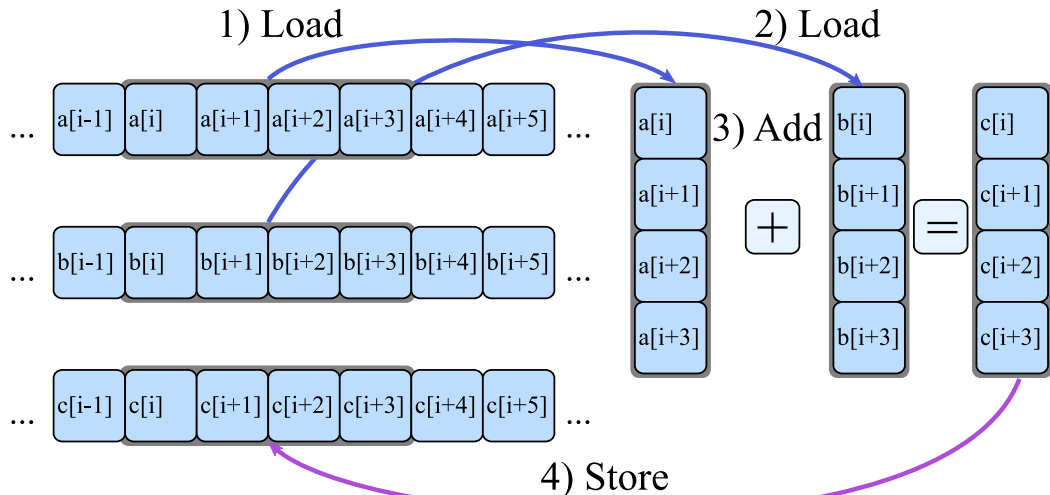
Automatic Vectorization →

```
1 double A[vec_width], B[vec_width];  
2 // ...  
3 for(int i = 0; i < vec_width; i++)  
4   A[i]+=B[i];
```

```
1 double A[8], B[8];  
2 __m512d A_v = _mm512_load_pd(A);  
3 __m512d B_v = _mm512_load_pd(B);  
4 A_v = _mm512_add_pd(A_v,B_v);  
5 _mm512_store_pd(A, A_v);
```

← Explicit Vectorization

# WORKFLOW OF VECTOR COMPUTATION



# DETECTING AVAILABLE INSTRUCTIONS

In the OS:

```
[student@cdt ~]% cat /proc/cpuinfo
...
fpu_exception      : yes
cpuid level        : 11
wp                 : yes
flags               : fpu vme de pse tsc msr pae mce
cx8 apic mtrr pge mca cmov pat pse36 clflush mmx
fxsr sse sse2 ss ht syscall nx lm constant_tsc
unfair_spinlock pni ssse3 cx16 sse4_1 sse4_2
x2apic popcnt aes hypervisor lahf_lm fsgsbase
bogomips           : 5985.17
clflush size       : 64
cache_alignment: 64
address sizes      : 46 bits physical, 48 bits virtual
...
```

In code (see also):

```
1 // Intel compiler
2 // preprocessor macros:
3
4 #ifdef __SSE__
5 // ...SSE code path
6 #endif
7
8 #ifdef __SSE4_2__
9 // ...SSE code path
10 #endif
11
12 #ifdef __AVX__
13 // ...AVX code path
14 #endif
```

<https://software.intel.com/sites/landingpage/IntrinsicsGuide>

The Intel Intrinsic Guide is an interactive reference tool for Intel intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, AVX-512, and more - without the need to write assembly code.

MMX  
 SSE  
 SSE2  
 SSE3  
 SSSE3  
 SSE4.1  
 SSE4.2  
 AVX  
 AVX2  
 FMA  
 AVX-512  
 KNC  
 SVML  
 Other

Application-Targeted  
 Arithmetic  
 Bit Manipulation  
 Cast  
 Compare  
 Convert  
 Cryptography  
 Elementary Math  
 Functions  
 General Support

```

__m128i_mm_add_epi16 (__m128i a, __m128i b)      paddw
__m128i_mm_add_epi32 (__m128i a, __m128i b)      paddq
__m128i_mm_add_epi64 (__m128i a, __m128i b)      paddq
__m128i_mm_add_epi8  (__m128i a, __m128i b)      paddb
__m128d_mm_add_pd    (__m128d a, __m128d b)      addpd
    
```

**Synopsis**

```

__m128d_mm_add_pd (__m128d a, __m128d b)
#include "emmintrin.h"
Instruction: addpd xmm, xmm
CPUID Flags: SSE2
    
```

**Description**

Add packed double-precision (64-bit) floating-point elements in a and b, and store the results in dst.

**Operation**

```

FOR j := 0 to 1
    i := j*64
    dst[i+63:i] := a[i+63:i] + b[i+63:i]
ENDFOR
    
```

**Performance**

Architecture	Latency	Throughput
Haswell	3	0.8
Ivy Bridge	3	1

# EXAMPLE: NUMERICAL INTEGRATION

$$I(a, b) = \int_0^a \frac{1}{\sqrt{x}} dx$$

Rectangle method:

$$\Delta x = \frac{a}{n}$$

$$x_i = (i+1)\Delta x,$$

$$I(a, b) = \sum_{i=0}^{n-1} \frac{1}{\sqrt{x_i}} \Delta x + O(\Delta x).$$

```

1 float Integrate(const float a,
2                 const int N) {
3     const float dx = a/float(n);
4     float S = 0.0f;
5     for (int i = 0; i < n; i++) {
6         const float xi = dx*float(i+1);
7         S += 1.0f/sqrtf(xi) * dx;
8     }
9     return S;
10 }
```

# IMPLEMENTATION WITH SSE4.2

```

1 float Integrate(const float a, const int n) {
2     __m128 dx = _mm_set1_ps(a/float(n));
3     __m128 S  = _mm_set1_ps(0.0f);
4     for (int i = 0; i < n; i += 4) {
5         __m128i ip1 =
6             _mm_set_epi32(i+4, i+3, i+2, i+1);
7         __m128 ip1f = _mm_cvtepi32_ps(ip1);
8         __m128 xi = _mm_mul_ps(dx, ip1f);
9         __m128 fi = _mm_rsqrt_ps(xi);
10        __m128 dS = _mm_mul_ps(fi, dx);
11        S = _mm_add_ps(S, dS);
12    }
13    ConverterType c;
14    c.v = S;
15    return c.f[0] + c.f[1] + c.f[2] + c.f[3];
16 }

```

That is fine, *but...*

- ▶ Assuming  $n$  is a multiple of 4
- ▶ Only for SSE4.2 (circa 2011)
- ▶ No memory access. If we had some, peeling may be needed



## **§3. AUTOMATIC VECTORIZATION**



# LOOPS



# AUTOMATIC VECTORIZATION OF LOOPS

```

1  #include <stdio>
2
3  int main(){
4      const int n=1024;
5      int A[n] __attribute__((aligned(64)));
6      int B[n] __attribute__((aligned(64)));
7
8      for (int i = 0; i < n; i++)
9          A[i] = B[i] = i;
10
11     // This loop will be auto-vectorized
12     for (int i = 0; i < n; i++)
13         A[i] = A[i] + B[i];
14
15     for (int i = 0; i < n; i++)
16         printf("%2d %2d %2d\n",i,A[i],B[i]);
17 }

```

```

vega@lyra% icpc autovec.cc -qopt-report
vega@lyra% cat autovec.optrpt
...
LOOP BEGIN at autovec.cc(12,3)
remark #15399: vectorization support:
unroll factor set to 2 [autovec.cc(12,3)]
remark #15300: LOOP WAS VECTORIZED
[autovec.cc(12,3)]
LOOP END
...
vega@lyra% ./a.out
0 0 0
1 2 1
2 4 2
3 6 3
4 8 4
...

```

## TARGETING A SPECIFIC INSTRUCTION SET

- x [code] to target specific processor architecture
- ax [code] for multi-architecture dispatch

code	Target architecture
MIC-AVX512	Intel Xeon Phi processors (KNL)
CORE-AVX512	Future Intel Xeon processors
CORE-AVX2	Intel Xeon processor E3/E5/E7 v3, v4 family
AVX	Intel Xeon processor E3/E5 and E3/E5/E7 v2 family
SSE4.2	Intel Xeon processor 55XX, 56XX, 75XX and E7 family
host	architecture on which the code is compiled

# GCC SUPPORT FOR AVX-512

GCC  $\geq$  4.9.1 supports AVX-512 instruction set.

```
user@knl% g++ -v
gcc version 4.9.2 (GCC)
user@knl% g++ foo.cc -mavx512f -mavx512er -mavx512cd -mavx512pf
```

Basic automatic vectorization support: add -O3.

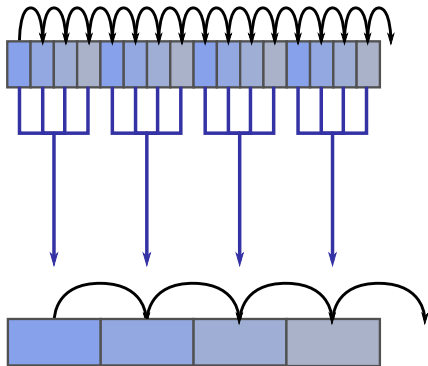
```
1 // ... foo.cc ... //
2 for(int i = 0; i < n; i++)
3   B[i] = A[i] + B[i];
```

```
user@knl% g++ -s foo.cc -mavx512f -O3
user@knl% cat foo.s
...
vmovapd -16432(%rbp,%rax), %zmm0
vaddpd -8240(%rbp,%rax), %zmm0, %zmm0
vmovapd %zmm0, -8240(%rbp,%rax)
```

# LIMITATIONS ON AUTOMATIC VECTORIZATION

- ▶ Innermost loops\*
- ▶ Known number of iterations
- ▶ No vector dependence
- ▶ Functions must be SIMD-enabled

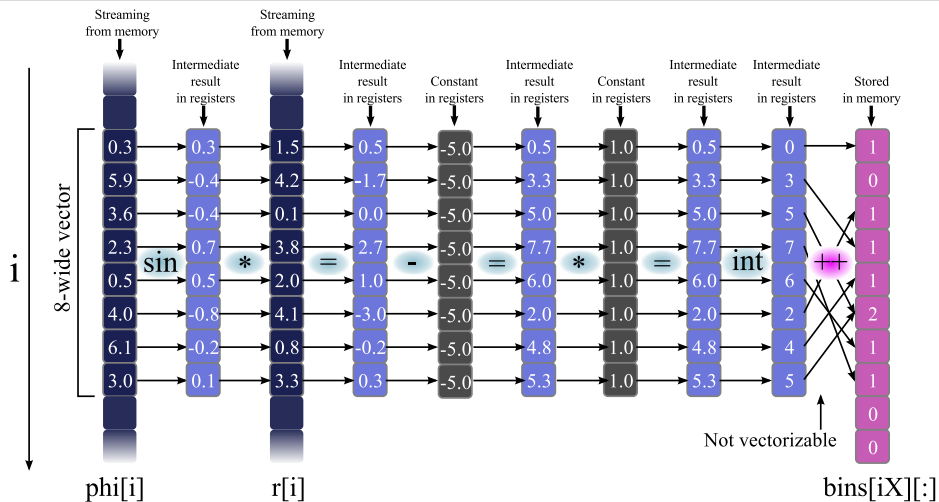
\* `#pragma omp simd` to override



# AUTO-VECTORIZED LOOPS MAY BE COMPLEX

```
1  for (int i = ii; i < ii + tileSize; i++) { // Auto-vectorized
2
3  // Newton's law of universal gravity
4  const float dx = particle.x[j] - particle.x[i]; // x[j] is a const
5  const float dy = particle.y[j] - particle.y[i]; // x[i] -> vector
6  const float dz = particle.z[j] - particle.z[i];
7  const float rr = 1.0f/sqrtf(dx*dx + dy*dy + dz*dz + softening);
8  const float drPowerN32 = rr*rr*rr;
9
10 // Calculate the net force
11 Fx[i-ii] += dx * drPowerN32;
12 Fy[i-ii] += dy * drPowerN32;
13 Fz[i-ii] += dz * drPowerN32;
14 }
```

# AUTO-VECTORIZED LOOPS MAY BE COMPLEX (EXAMPLE 2)



See [this paper](#) for more details



**PRAGMA SIMD**

## VECTORIZE MORE LOOPS: `#pragma omp simd`

Used to “enforce vectorization of loops”, which includes:

- ▶ Loops with SIMD-enabled functions
- ▶ Second innermost loops
- ▶ Failed vectorization due to compiler decision
- ▶ Where guidance is required (vector length, reduction, etc.)

See OpenMP reference for syntax; `#pragma simd`



# EXAMPLE FOR #pragma omp simd

```
1  const int N=128, T=4;
2  float A[N*N], B[N*N], C[T*T];
3
4  for (int jj = 0; jj < N; jj+=T) // Tile in j
5      for (int ii = 0; ii < N; ii+=T) // and tile in i
6          #pragma omp simd // Vectorize outer loop
7              for (int k = 0; k < N; ++k) // long loop, vectorize it
8                  for (int i = 0; i < T; i++) { // Loop between ii and ii+T
9                      // Instead of a loop between jj and jj+T, unrolling that loop:
10                     C[0*T + i] += A[(jj+0)*N + k]*B[(ii+i)*N + k];
11                     C[1*T + i] += A[(jj+1)*N + k]*B[(ii+i)*N + k];
12                     C[2*T + i] += A[(jj+2)*N + k]*B[(ii+i)*N + k];
13                     C[3*T + i] += A[(jj+3)*N + k]*B[(ii+i)*N + k];
14                 }
```



## ARRAY NOTATION

# EXTENSIONS FOR ARRAY NOTATION

Array notation is a method for specifying

- ▶ slices of arrays (begin, length)

```
1 A[0:16] += B[32:16]; // B[32]...B[47] added to A[0]...A[15]
```

- ▶ a stride (begin, length, stride)

```
1 A[0:16:2] += B[32:16:4]; // B[32],B[36]...B[92] added A[0],A[2]...A[30]
```

- ▶ Multi-dimensional arrays

```
1 A[:, :] += B[:, :]; // Add B to A; arrays are of the same shape
```

Better than strided loops (e.g., [this paper](#)).

# EXPRESSIONS WITH ARRAY NOTATION MAY BE COMPLEX

Example from <https://colfaxresearch.com/efft>

```

1 evenrek[:] = evens[kk :kTILE:2];
2 evenimk[:] = evens[kk+1:kTILE:2];
3 oddrek [:] = odds [kk :kTILE:2];
4 oddimk [:] = odds [kk+1:kTILE:2];
5
6 evens[kk :kTILE:2] = evenrek[:] + coslist[:] * oddrek[:] - sinlist[:] * oddimk[:];
7 evens[kk+1:kTILE:2] = evenimk[:] + sinlist[:] * oddrek[:] + coslist[:] * oddimk[:];
8
9 oddmirrek[:] = odds[size-kk :kTILE:-2];
10 oddmirimk[:] = odds[size-kk+1:kTILE:-2];
11
12 odds[size-kk :kTILE:-2] =
13     evenrek[:] - coslist[:] * oddrek[:] + sinlist[:] * oddimk[:];
14 odds[size-kk+1:kTILE:-2] =
15     -evenimk[:] + sinlist[:] * oddrek[:] + coslist[:] * oddimk[:];
16 // ...

```



## **SIMD-ENABLED FUNCTIONS**

# SIMD-ENABLED FUNCTIONS

Define function in one file (e.g., library), use in another

```
1 // Compiler will produce 3 versions:
2 #pragma omp declare simd
3 float my_simple_add(float x1, float x2){
4     return x1 + x2;
5 }
```

```
1 // May be in a separate file
2 #pragma omp simd
3 for (int i = 0; i < N, ++i) {
4     output[i] = my_simple_add(inputa[i], inputb[i]);
5 }
```

# SIMD-ENABLED FUNCTIONS MAY BE COMPLEX

```
1  #pragma omp declare simd
2  float MyErfElemental(const float inx){
3      const float x = fabsf(inx); // Absolute value (in each vector lane)
4      const float p = 0.3275911f; // Constant parameter across vector lanes
5      const float t = 1.0f/(1.0f+p*x); // Expression in each vector lanes
6      const float l2e = 1.442695040f; // log2f(expf(1.0f))
7      const float e = exp2f(-x*x*l2e); // Transcendental in each vector lane
8      float res = -1.453152027f + 1.061405429f*t; // Computing a polynomial
9      res = 1.421413741f + t*res; // in each vector lane
10     res = -0.284496736f + t*res;
11     res = 0.254829592f + t*res;
12     res *= e;
13     res = 1.0f - t*res; // Analytic approximation in each vector lane
14     return copysignf(res, inx); // Copy sign in each vector lane
15 }
```



## **MULTIVERSIONING, POINTER DISAMBIGUATION**



# TRUE VECTOR DEPENDENCE

- ▶ True vector dependence – vectorization impossible:

```
1 for (int i = 1; i < n; i++)  
2   a[i] += a[i-1]; // dependence on the previous element
```

- ▶ Safe to vectorize:

```
1 for (int i = 0; i < n-1; i++)  
2   a[i] += a[i+1]; // no dependence on the previous element
```

- ▶ May be safe to vectorize:

```
1 for (int i = 16; i < n; i++)  
2   a[i] += a[i-16]; // no dependence if vector length <=16
```

# ASSUMED VECTOR DEPENDENCE

Not enough information to confirm or rule out vector dependence:

```
1 void AmbiguousFunction(int n, int *a, int *b) {  
2     for (int i = 0; i < n; i++)  
3         a[i] = b[i];  
4 }
```

- ▶ If a, b are not aliased or  $b > a$ , then safe to vectorize
- ▶ If a, b are aliased (e.g.,  $b == a - 1$ ), requires scalar computation

# MULTIVERSIONING

```
user@host% icpc -c code.cc -qopt-report
user@host% cat code.optrpt
...
LOOP BEGIN at code.cc(4,1)
<Multiversiomed v1>
    remark #25228: LOOP WAS VECTORIZED
LOOP END
...
LOOP BEGIN at code.cc(4,1)
<Multiversiomed v2>
    remark #15304: loop was not vectorized: non-vectorizable loop instance ....
LOOP END
```

Pointers checked for aliasing *runtime* to choose code path.

# POINTER DISAMBIGUATION

Prevent multiversioning or allow vectorization with a directive:

```
1 #pragma ivdep
2   for (int i = 0; i < n; i++)
3     // ...
```

```
user@host% icpc -c code.cc -qopt-report -qopt-report-phase:vec
user@host% cat vdep.optrpt
...
LOOP BEGIN at code.cc(4,1)
  remark #25228: LOOP WAS VECTORIZED
LOOP END
...
```

Alternative: keyword `restrict` – more fine-grained, weaker.



## **ADDITIONAL CONTROLS**

# VECTORIZATION DIRECTIVES

- ▷ `#pragma omp simd`
- ▷ `#pragma vector always`
- ▷ `#pragma vector aligned | unaligned`
- ▷ `__assume_aligned` keyword
- ▷ `#pragma vector nontemporal | temporal`
- ▷ `#pragma novector`
- ▷ `#pragma ivdep`
- ▷ `restrict` qualifier and `-restrict` command-line argument
- ▷ `#pragma loop count`

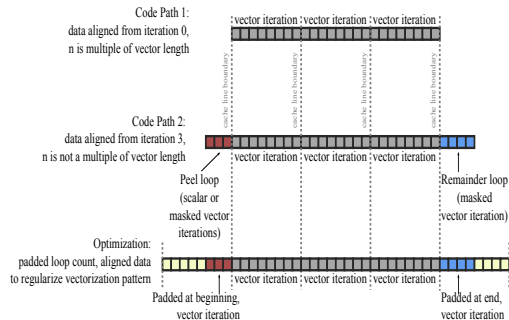


**LOOP WAS VECTORIZED, WHAT NOW?**

# LOOP WAS VECTORIZED, NOW WHAT?

1. Unit-stride access
2. Data alignment
3. Container padding
4. Eliminate peel loops
5. Eliminate multiversioning
6. **Optimize data re-use in caches**

```
for (i = 0; i < n; i++) A[i] = ...
```





# LOOP WAS VECTORIZED, NOW WHAT?

1. Unit-stride access
2. Data alignment
3. Container padding
4. Eliminate peel loops
5. Eliminate multiversioning
6. **Optimize data re-use in caches**

## Vector Arithmetics is Cheap, Memory Access is Expensive

If you don't optimize cache usage, vectorization will not matter.

You will be bottlenecked by memory access.

## REVIEW AND WHAT'S NEXT

Discussed today:

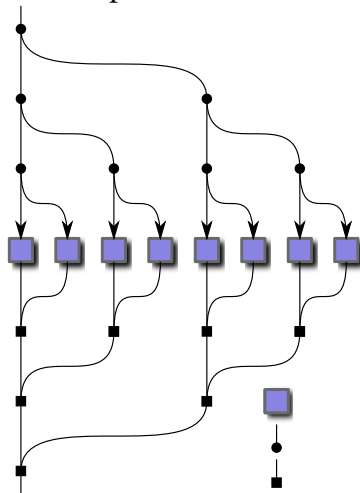
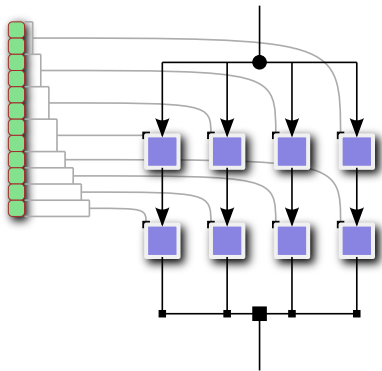
- ▶ Vectorization – support for data parallelism in each core
- ▶ Automatic vectorization enabled at default optimization level
- ▶ Loops, SIMD-enabled functions, array notation
- ▶ Argument `-qopt-report` produces a report on vectorization success

Later in the course – tuning automatic vectorization:

- ▶ alignment
- ▶ unit-stride data structures
- ▶ vectorization pattern regularization
- ▶ programming techniques for exposing automatic vectorization
- ▶ compiler hints.

# WHAT'S NEXT

Next session: expressing thread parallelism with OpenMP.



**COLFAX RESEARCH**  
CONTRIBUTING TO INNOVATIONS IN COMPUTING

Log In/Out or Register

READ WATCH LEARN CONNECT JOIN

To search, type and hit enter

**Popular**

**The Hands-On Tutorials (HOT) webinars: details on efficient programming for Intel architecture**

**The Hands-On Workshop (HOW) Series**

**Introduction to Intel DAAL, Part 1: Polynomial Regression with Batch Mode Computation**

**Parallel Programming Book**

**Research and Educational Publications**

**Introduction to Intel DAAL, Part 1: Polynomial Regression with Batch Mode Computation**

**Optimization Techniques for the Intel MIC Architecture, Part 3 of 3: False Sharing and Padding**

**Software Developer's Introduction to the HGST Ultrastar Archive H800 SMR Drives**

**Optimization Techniques for the Intel MIC Architecture, Part 2 of 3: Strip-Mining for Vectorization**

**Optimization Techniques for the Intel MIC Architecture, Part 1 of 3: Multi-Threading and Parallel Reduction**

**Performance to Power and Performance to Cost Ratios with Intel Xeon Phi Coprocessors (and why ix Acceleration May Be Enough)**

**Featured Video**

See Research material on vectorization in a streaming mode

**Events**

**Presentations**

**Cardview**

**Consulting**

Colfax offers consulting services for enterprises, research help you:

- Optimize your existing application to take advantage of parallelism, from vectors to cores to clusters and
- Future-proof your application for upcoming innovations.
- Accelerate your application using coprocessor tech.
- Investigate the potential system configurations that satisfy your cost, power, performance requirements.
- Take a clean slate to develop a novel approach to reduce your computing pro...

All Video Content - CPU VSI - Chapter 1 - Episode 1.1

**Episode 2.1 - Purpose of the MIC architecture**

**Software Developer's Introduction to the HGST Ultrastar Archive H800 SMR Drives**

**Fluid Dynamics with Fortran on Intel Xeon Phi coprocessors**

**Configuration and Benchmarks of Peer-to-Peer Communication over Gigabit Ethernet and InfiniBand in a Cluster with Intel Xeon Phi Coprocessors**

**Interview with James Reinders: future of Intel MIC architecture, parallel programming, education**

<https://colfaxresearch.com/>