PROGRAMMING AND OPTIMIZATION FOR INTEL® ARCHITECTURE

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

Session 3

Colfax International — *colfaxresearch.com*

May 2017

© Colfax International, 2013–2017

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.

© Colfax International, 2013–2017

COURSE ROADMAP

- **Module I. Programming**
  - 01. Intel Architecture and Modern Code – May 15
  - 02. Xeon Phi, Coprocessors, Omni-Path – May 16
- **Module II. Expressing Parallelism**
  - 03. Automatic vectorization – May 17
  - 04. Multi-threading with OpenMP – May 18
  - 05. Distributed Computing, MPI – May 19
- **Module III. Optimization**
  - 06. Optimization Overview: N-body – May 22
  - 07. Scalar tuning, Vectorization – May 23
  - 08. Common Multi-threading Problems – May 24
  - 09. Multi-threading, Memory Aspect – May 25
  - 10. Access to Caches and Memory – May 26

May 2017

S	M	T	W	H	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

© Colfax International, 2013–2017

HOW SERIES ONLINE

Course page:
colfaxresearch.com/how-17-05

- Slides
- Code
- Video
- Chat

More workshops:
colfaxresearch.com/training

© Colfax International, 2013–2017

GET YOUR QUESTIONS ANSWERED: CHAT

colfaxresearch.com/how-17-05

© Colfax International, 2013–2017

GET YOUR QUESTIONS ANSWERED: FORUMS

READ | WATCH | LEARN | FORUMS | CONNECT | JOIN

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/discussion

© Colfax International, 2013–2017

HANDS-ON EXERCISES AND REMOTE ACCESS

- 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

© Colfax International, 2013–2017

§2. SIMD PARALLELISM AND VECTORIZATION

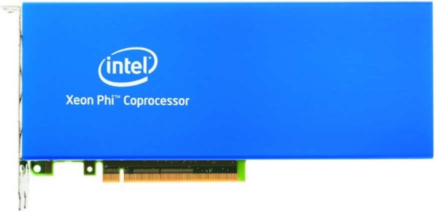
VECTOR INSTRUCTIONS IN INTEL ARCHITECTURE

COMPUTING PLATFORMS

Intel Xeon Processor

Current: Broadwell
Upcoming: Skylake

Intel Xeon Phi Coprocessor, 1st generation



Knights Corner (KNC)

Intel Xeon Phi Processor, 2nd generation*

* socket and coprocessor versions

Knights Landing (KNL)

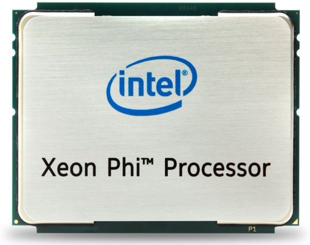
Multi-Core Architecture

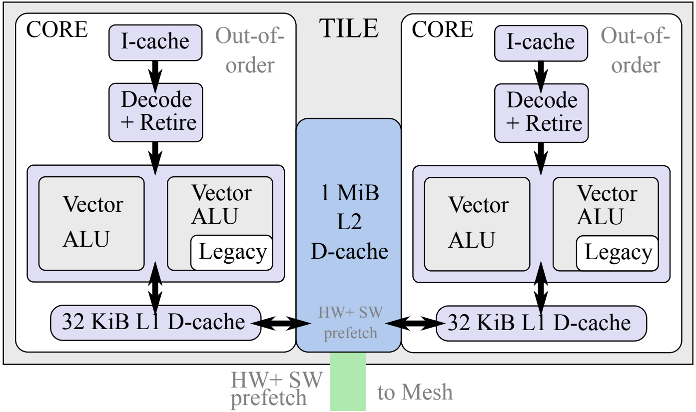
Intel Many Integrated Core (MIC) Architecture

© Colfax International, 2013–2017

KNL CORES

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

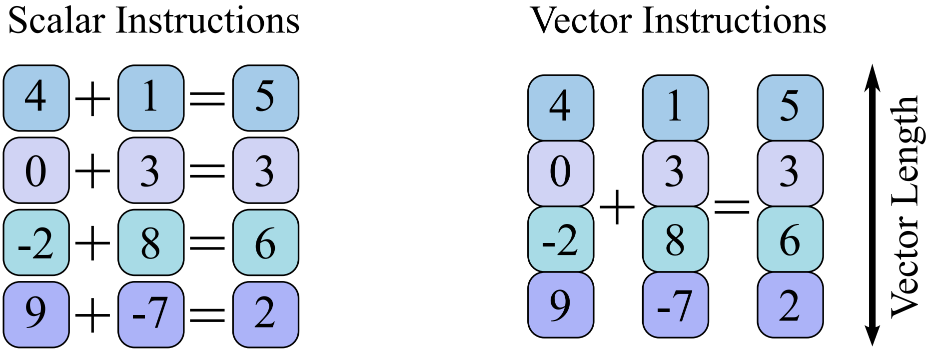




© Colfax International, 2013–2017

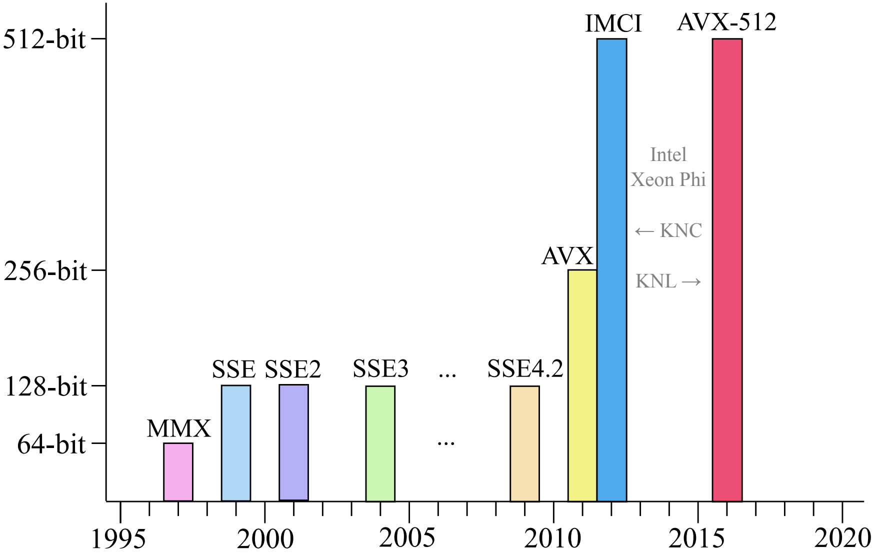
SHORT VECTOR SUPPORT

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



© Colfax International, 2013–2017

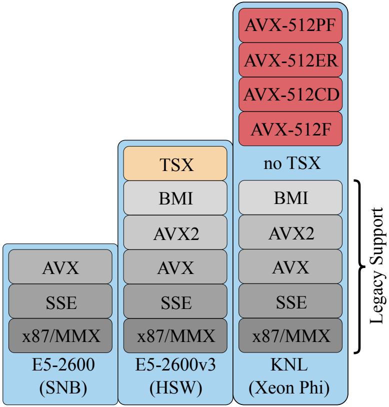
INSTRUCTION SETS IN INTEL ARCHITECTURE



© Colfax International, 2013–2017

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.
- ≤ Intel® AVX2
  - Legacy mode operation.
  - Binary compatibility with Xeon.
  - Does *not* include IMCI (from KNC).



© Colfax International, 2013–2017

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

© Colfax International, 2013–2017

VECTOR INTRINSICS

USING VECTOR INSTRUCTIONS: TWO APPROACHES

Automatic Vectorization →

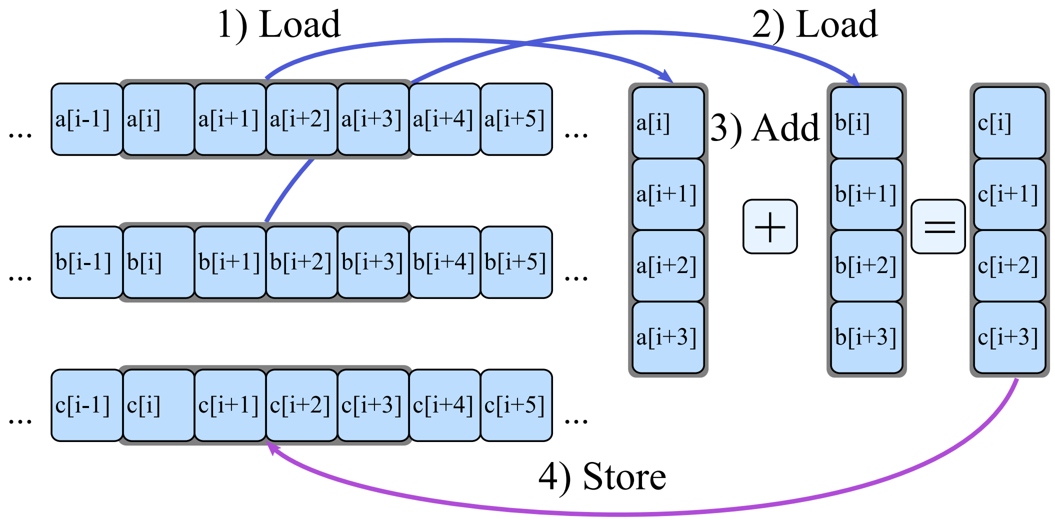
```
double A[vec_width], B[vec_width];
// ...
for(int i = 0; i < vec_width; i++)
  A[i]+=B[i];
```

```
double A[8], B[8];
__m512d A_v = _mm512_load_pd(A);
__m512d B_v = _mm512_load_pd(B);
A_v = _mm512_add_pd(A_v,B_v);
_mm512_store_pd(A, A_v);
```

← Explicit Vectorization

© Colfax International, 2013–2017

WORKFLOW OF VECTOR COMPUTATION



© Colfax International, 2013–2017

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
...
```

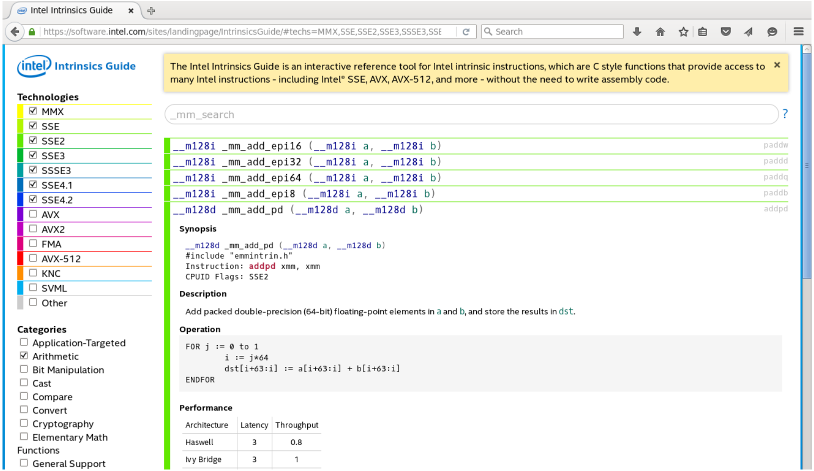
In code (see also):

```
// Intel compiler
// preprocessor macros:

#ifdef __SSE__
// ...SSE code path
#endif

#ifdef __SSE4_2__
// ...SSE code path
#endif

#ifdef __AVX__
// ...AVX code path
#endif
```


© Colfax International, 2013–2017

INTEL INTRINSICS GUIDE

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



© Colfax International, 2013–2017

EXAMPLE: NUMERICAL INTEGRATION

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

Rectangle method:

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

```
float Integrate(const float a,
                const int N) {
  const float dx = a/float(n);
  float S = 0.0f;
  for (int i = 0; i < n; i++) {
    const float xi = dx*float(i+1);
    S += 1.0f/sqrtf(xi) * dx;
  }
  return S;
}
```


© Colfax International, 2013–2017

IMPLEMENTATION WITH SSE4.2

```
float Integrate(const float a, const int n) {
  __m128 dx = _mm_set1_ps(a/float(n));
  __m128 S  = _mm_set1_ps(0.0f);
  for (int i = 0; i < n; i += 4) {
    __m128i ip1  =
                 _mm_set_epi32(i+4, i+3, i+2, i+1);
    __m128  ip1f = _mm_cvtepi32_ps(ip1);
    __m128  xi = _mm_mul_ps(dx, ip1f);
    __m128  fi = _mm_rsqrt_ps(xi);
    __m128  dS = _mm_mul_ps(fi, dx);
    S  = _mm_add_ps(S, dS);
  }
  ConverterType c;
  c.v = S;
  return c.f[0] + c.f[1] + c.f[2] + c.f[3];
}
```

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

© Colfax International, 2013–2017

§3. AUTOMATIC VECTORIZATION

LOOPS

AUTOMATIC VECTORIZATION OF LOOPS

```
#include <cstdio>

int main(){
  const int n=1024;
  int A[n] __attribute__((aligned(64)));
  int B[n] __attribute__((aligned(64)));

  for (int i = 0; i < n; i++)
    A[i] = B[i] = i;

  // This loop will be auto-vectorized
  for (int i = 0; i < n; i++)
    A[i] = A[i] + B[i];

  for (int i = 0; i < n; i++)
    printf("%2d %2d %2d\n",i,A[i],B[i]);
}
```

```
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
...
```


© Colfax International, 2013–2017

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

© Colfax International, 2013–2017

GCC SUPPORT FOR AVX-512

GCC ≥ 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.

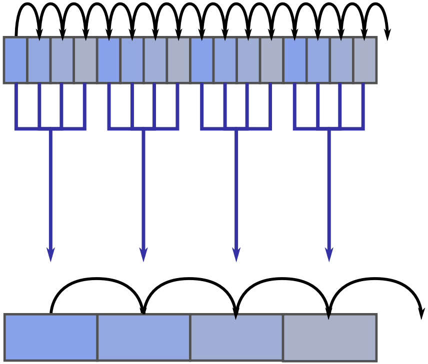
```
// ... foo.cc ... //
for(int i = 0; i < n; i++)
  B[i] = A[i] + B[i];
```

```
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)
```

 © Colfax International, 2013–2017

LIMITATIONS ON AUTOMATIC VECTORIZATION

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



* `#pragma omp simd` to override

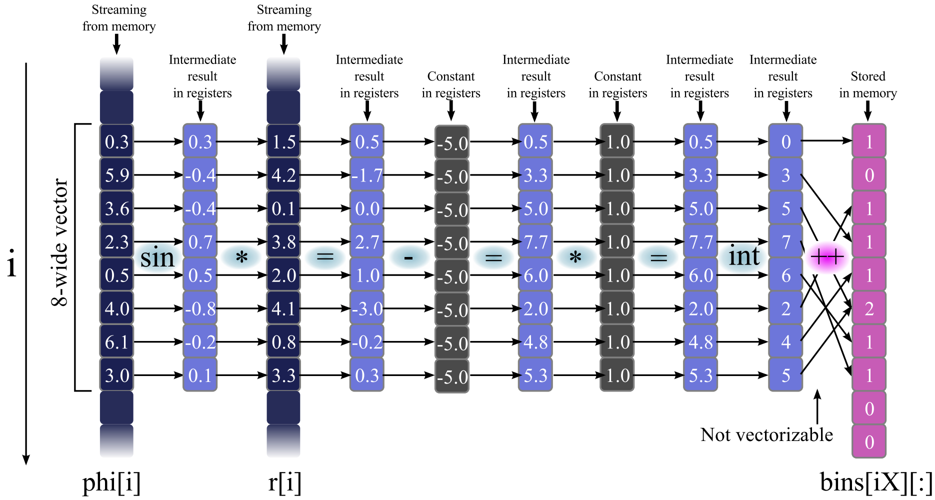
© Colfax International, 2013–2017

AUTO-VECTORIZED LOOPS MAY BE COMPLEX

```
for (int i = ii; i < ii + tileSize; i++) { // Auto-vectorized

  // Newton's law of universal gravity
  const float dx = particle.x[j] - particle.x[i]; // x[j] is a const
  const float dy = particle.y[j] - particle.y[i]; // x[i] -> vector
  const float dz = particle.z[j] - particle.z[i];
  const float rr = 1.0f/sqrtf(dx*dx + dy*dy + dz*dz + softening);
  const float drPowerN32  = rr*rr*rr;

  // Calculate the net force
  Fx[i-ii] += dx * drPowerN32;
  Fy[i-ii] += dy * drPowerN32;
  Fz[i-ii] += dz * drPowerN32;
}
```


© Colfax International, 2013–2017

AUTO-VECTORIZED LOOPS MAY BE COMPLEX (EXAMPLE 2)



See this paper for more details

© Colfax International, 2013–2017

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`

© Colfax International, 2013–2017

EXAMPLE FOR #pragma omp simd

```
const int N=128, T=4;
float A[N*N], B[N*N], C[T*T];

for (int jj = 0; jj < N; jj+=T) // Tile in j
  for (int ii = 0; ii < N; ii+=T) // and tile in i
#pragma omp simd   // Vectorize outer loop
  for (int k = 0; k < N; ++k) // long loop, vectorize it
    for (int i = 0; i < T; i++) { // Loop between ii and ii+T
      // Instead of a loop between jj and jj+T, unrolling that loop:
      C[0*T + i] += A[(jj+0)*N + k]*B[(ii+i)*N + k];
      C[1*T + i] += A[(jj+1)*N + k]*B[(ii+i)*N + k];
      C[2*T + i] += A[(jj+2)*N + k]*B[(ii+i)*N + k];
      C[3*T + i] += A[(jj+3)*N + k]*B[(ii+i)*N + k];
}
```


© Colfax International, 2013–2017

ARRAY NOTATION

EXTENSIONS FOR ARRAY NOTATION

Array notation is a method for specifying

- slices of arrays (begin, length)

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

- a stride (begin, length, stride)

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

- Multi-dimensional arrays

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

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

© Colfax International, 2013–2017

Expressions with Array Notation May Be Complex

Example from http://xeonphi.com/papers/efft

```
evenrek[:] = evens[kk  :kTILE:2];
evenimk[:] = evens[kk+1:kTILE:2];
oddrek [:] = odds [kk  :kTILE:2];
oddimk [:] = odds [kk+1:kTILE:2];

evens[kk  :kTILE:2] = evenrek[:] + coslist[:]*oddrek[:] - sinlist[:]*oddimk[:];
evens[kk+1:kTILE:2] = evenimk[:] + sinlist[:]*oddrek[:] + coslist[:]*oddimk[:];

oddmirrek[:] = odds[size-kk  :kTILE:-2];
oddmirimk[:] = odds[size-kk+1:kTILE:-2];

odds[size-kk  :kTILE:-2] =
                      evenrek[:] - coslist[:]*oddrek[:] + sinlist[:]*oddimk[:];
odds[size-kk+1:kTILE:-2] =
                     -evenimk[:] + sinlist[:]*oddrek[:] + coslist[:]*oddimk[:];
// ...
```

 © Colfax International, 2013–2017

SIMD-ENABLED FUNCTIONS

SIMD-ENABLED FUNCTIONS

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

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

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


© Colfax International, 2013–2017

SIMD-ENABLED FUNCTIONS MAY BE COMPLEX

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


© Colfax International, 2013–2017

MULTIVERSIONING, POINTER DISAMBIGUATION

TRUE VECTOR DEPENDENCE

- True vector dependence – vectorization impossible:

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

- Safe to vectorize:

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

- May be safe to vectorize:

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


© Colfax International, 2013–2017

ASSUMED VECTOR DEPENDENCE

Not enough information to confirm or rule out vector dependence:

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

- 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

© Colfax International, 2013–2017

MULTIVERSIONING

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

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

© Colfax International, 2013–2017

POINTER DISAMBIGUATION

Prevent multiversioning or allow vectorization with a directive:

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

```
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.

© Colfax International, 2013–2017

ADDTIONAL CONTROLS

Vectorization Directives

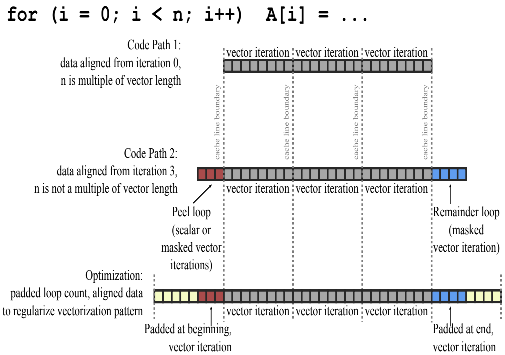
- `#pragma omp simd`
- `#pragma vector always`
- `#pragma vector aligned | unaligned`
- `__assume_aligned` keyword
- `#pragma vector nontemporal | temporal`
- `#pragma novector`
- `#pragma ivdep`
- `restrict` qualifier and `-restrict` command-line argument
- `#pragma loop count`

© Colfax International, 2013–2017

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**



© Colfax International, 2013–2017

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.

© Colfax International, 2013–2017

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

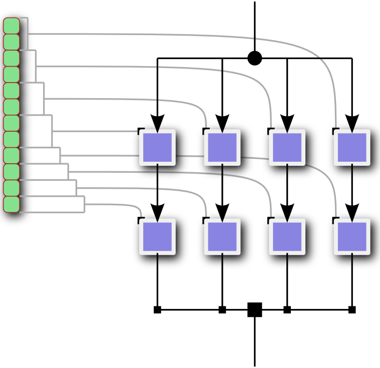
Later in the course – tuning automatic vectorization:

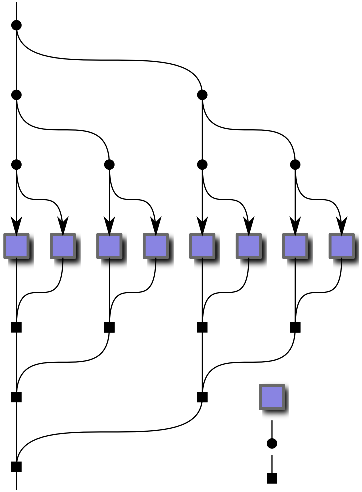
- ▷ alignment
- ▷ unit-stride data structures
- ▷ vectorization pattern regularization
- ▷ programming techniques for exposing automatic vectorization
- ▷ compiler hints.

© Colfax International, 2013–2017

WHAT'S NEXT

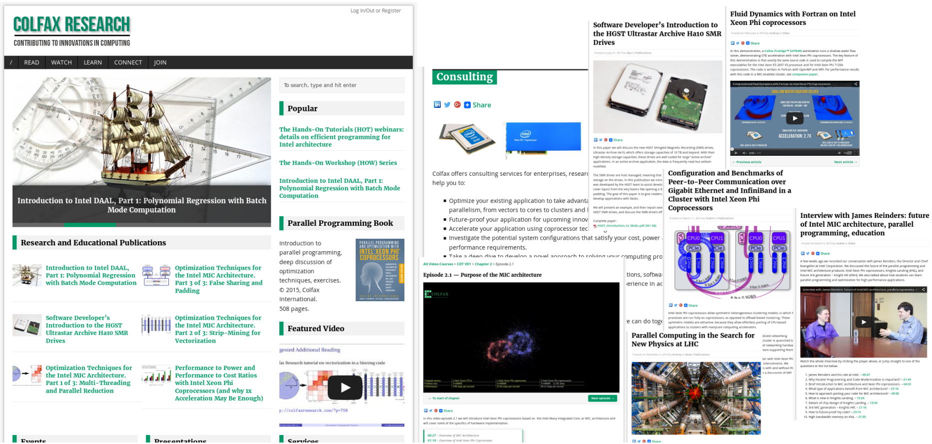
Next session: expressing thread parallelism with OpenMP.





© Colfax International, 2013–2017

COLFAX RESEARCH



http://colfaxresearch.com/