CERN openlab

Impressions from experiments with Cilk+

April 2nd 2014, Annual Concurrency Forum Meeting

Andrzej Nowak, CERN openlab CTO office

(with material from Juan Jose Fumero and Laurent Duhem)

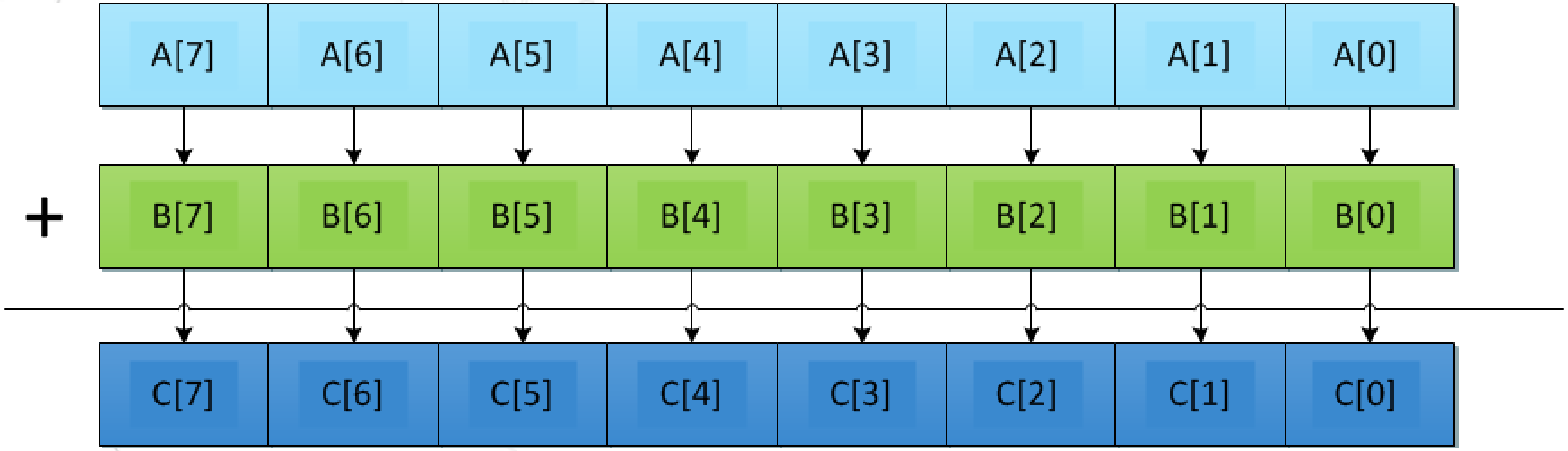
Cilk+

- Extension to the C/C++ languages to support data and tasks parallelism
  - Support for task parallelism (spawn)
  - New syntax to express data parallelism (CEAN – C Extended Array Notation)
  - Single “way” of harnessing the power of both multicore and vector processing
- Implementations (Linux/OSX):
  - Intel Compiler
  - GCC >=4.8.1 cilkplus branch
  - LLVM support still young

Exploiting in-core parallelism

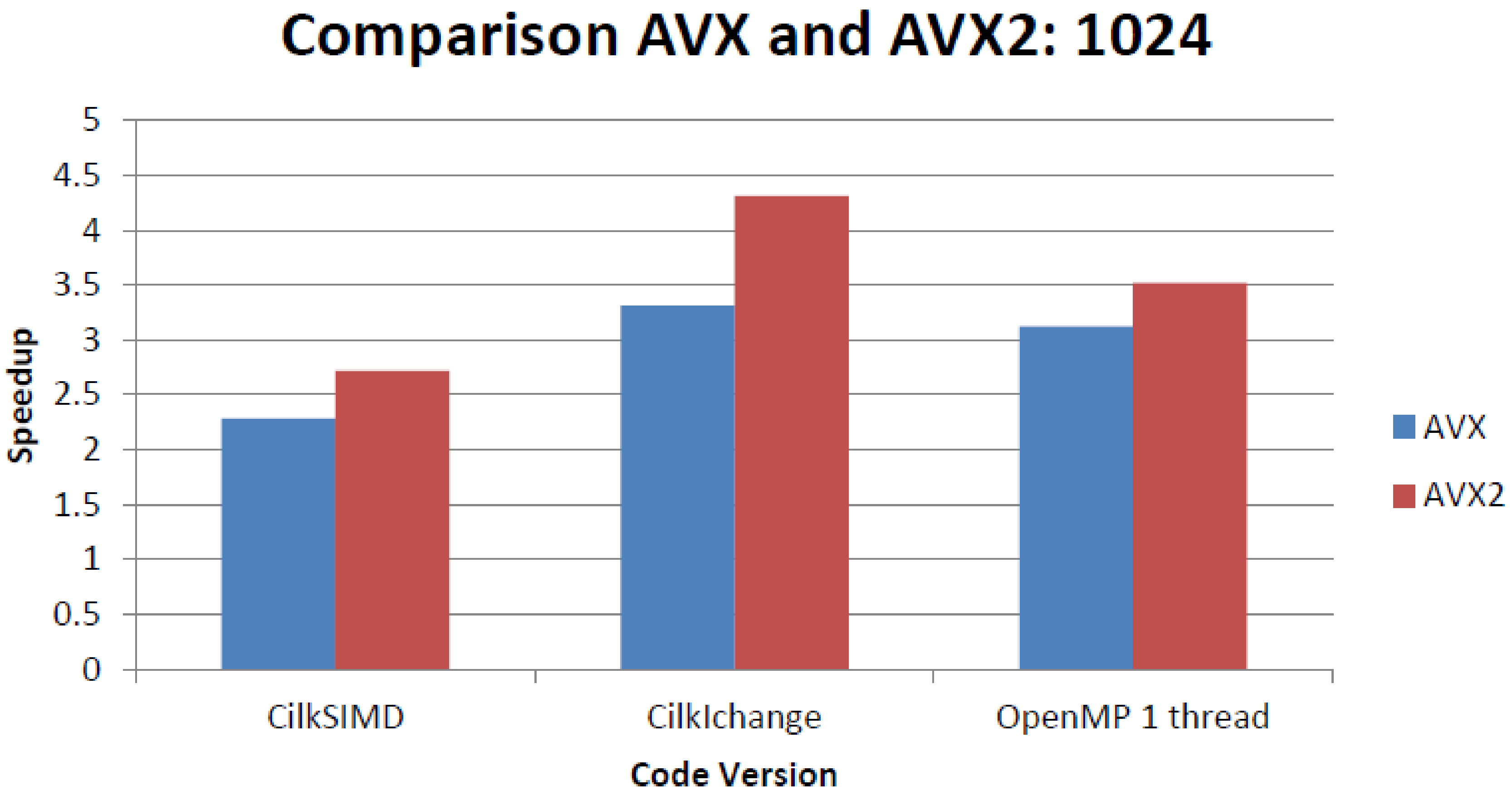
```
for (int i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
}
```

```
vmovups   .L8(%rip), %ymm0
vaddps    .L.9(%rip), %ymm0, %ymm1
vmovups   %ymm1, 32(%rsp)
vmovups   %ymm1, 64(%rsp)
```



This is the architectural spec

Intermission



This is what you get

vs. novec, HSW single socket

Cool features

Simple assignments	A[:] = 5;
Range assignment	A[0:7] = 5;
Assignment w/ stride	A[0:5:2] = 5;
Increments	A[:] = B[:] + 5;

2D arrays	C[:][:] = 12;
	C[0:5:2][:] = 12;

Function calls	func (A[:]); A[:] = pow(c, B[:]) *operators*

Conditions	if (5 == a[:])
	results[:] = „y”
	else
	results[:] = „n”

Reductions	__sec_reduce_mul (A[:])
Gather	C[:] = A[B[:]]
Scatter	A[B[:]] = C[:]

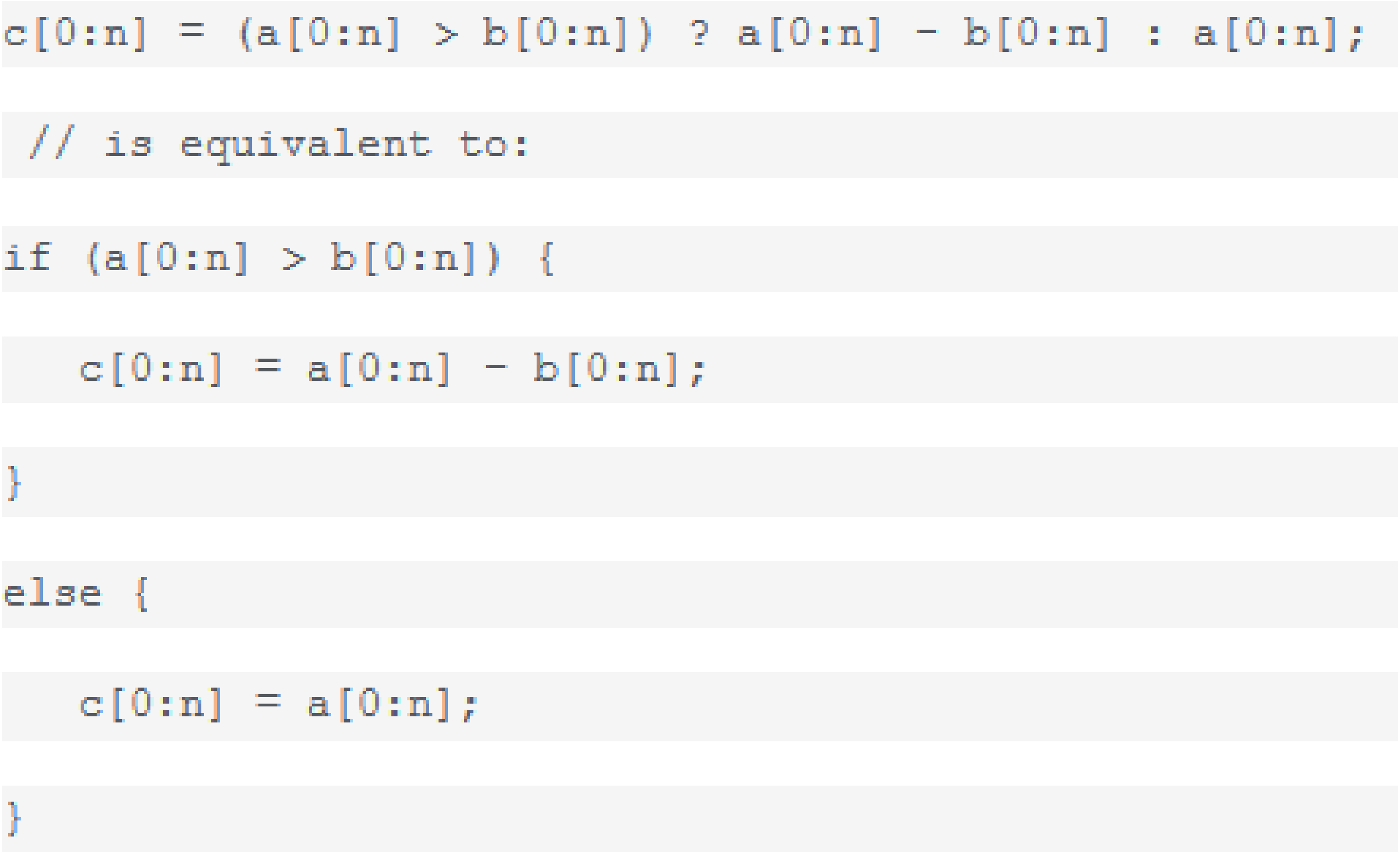
Example – matmul in OpenMP

```
void mxm_omp(double * restrict result, double *a, double *b, int m) {
    int i, j, k;
    #pragma omp parallel for private(i, j, k) firstprivate(m) shared(result,a,b)
    for (i = 0; i < m; i++) {
        for (j = 0; j < m; j++) {
            for (k = 0; k < m; k++) {
                result[i*m+j] += a[i*m+k] * b[k*m+j];
            }
        }
    }
}
```

Example – matmul in Cilk+

```
void mxm_array_notation_interchange(double *restrict result, double
*a, double *b, int n) {
for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            result[i*n:n] += a[i*n+k] * b[k*n:n];
        }
    }
}
```

Conditions



```
c[0:n] = (a[0:n] > b[0:n]) ? a[0:n] - b[0:n] : a[0:n];

 // is equivalent to:

if (a[0:n] > b[0:n]) {

   c[0:n] = a[0:n] - b[0:n];

}

else {

   c[0:n] = a[0:n];

}
```

Source: intel.com

Slightly more complex examples (1)

Map:

```
safx[:] = std::abs(newptx[:]) - dx;
safy[:] = std::abs(newpty[:]) - dy;
safz[:] = std::abs(newptz[:]) - dz;
```

More complex map:

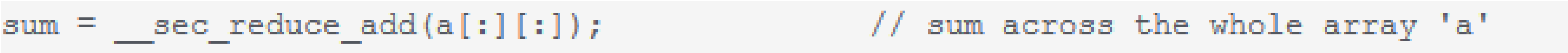
```
snxtx[:] = safx[:]/std::abs(vdirx[0][:]+tiny);
```

Will it vectorize?

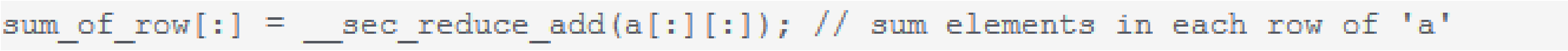
Slightly more complex examples (2)

```
mask1[:] = (x[:] >= vstepmax[0][:] ||
            y[:] >= vstepmax[0][:] ||
            z[:] >= vstepmax[0][:]) ? 1.0 : 0.0;
int faraway = __sec_reduce_any_nonzero(mask1[:]);
if (faraway) return;
```

Important to use reductions and vector operations where possible

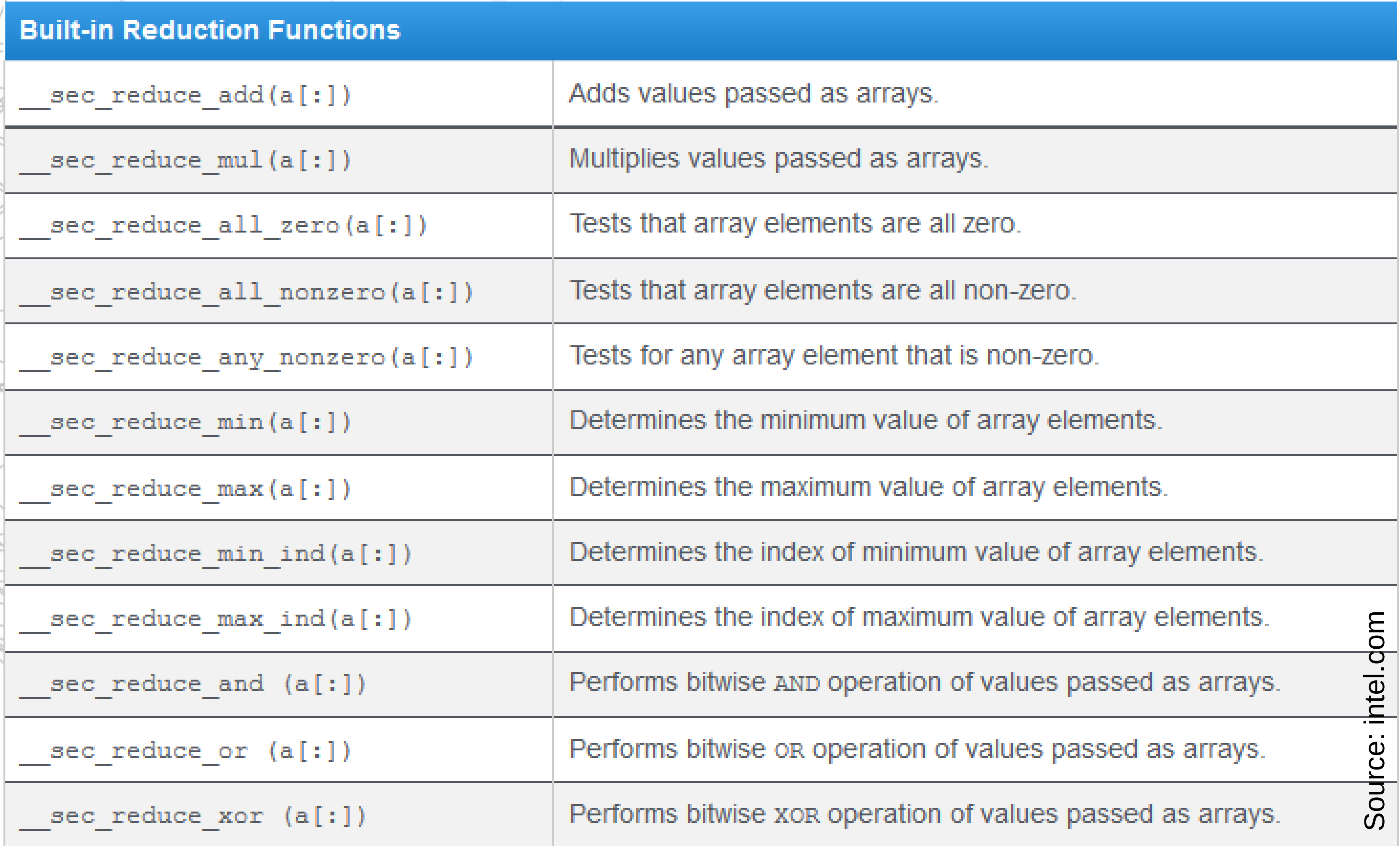


```
sum = __sec_reduce_add(a[:][:]);                    // sum across the whole array 'a'
```



```
sum_of_row[:] = __sec_reduce_add(a[:][:]); // sum elements in each row of 'a'
```

Array section reductions



Built-in Reduction Functions	
`__sec_reduce_add(a[:])`	Adds values passed as arrays.
`__sec_reduce_mul(a[:])`	Multiplies values passed as arrays.
`__sec_reduce_all_zero(a[:])`	Tests that array elements are all zero.
`__sec_reduce_all_nonzero(a[:])`	Tests that array elements are all non-zero.
`__sec_reduce_any_nonzero(a[:])`	Tests for any array element that is non-zero.
`__sec_reduce_min(a[:])`	Determines the minimum value of array elements.
`__sec_reduce_max(a[:])`	Determines the maximum value of array elements.
`__sec_reduce_min_ind(a[:])`	Determines the index of minimum value of array elements.
`__sec_reduce_max_ind(a[:])`	Determines the index of maximum value of array elements.
`__sec_reduce_and (a[:])`	Performs bitwise AND operation of values passed as arrays.
`__sec_reduce_or (a[:])`	Performs bitwise OR operation of values passed as arrays.
`__sec_reduce_xor (a[:])`	Performs bitwise XOR operation of values passed as arrays.

Source: intel.com

Obstacles

- In some cases, need to use hints for performance
- **`#pragma ivdep`** – ignore assumed dependencies
- **`#pragma nontemporal`** – use nontemporal stores
- **`double (*vdistance)[4] = (double (*)[4]) &(distance[i]);`**
- **`__assume_aligned(vdistance, 32);`**
- **`double in[4] __attribute__((aligned(32)));`**

Conclusions

- Cilk+ is easy to learn and use
  - Array notation is convenient and intuitive
  - Easy to add task parallelism in the same package
- Ease of use != performance
  - Exclusive use of high level abstraction is insufficient - hints from the programmer required for performance
  - Compiler support crucial – white spots exist, but improving rapidly!
- Overall, vectorization still requires some technical knowledge
- In our opinion, this is one of the best options available today
- Several bugs reported to the GCC team and others to Intel (and fixed)

A full report and two presentations (Aug 14th and Aug 28th 2013) are available on the openlab website, written by Juan Jose Fumero

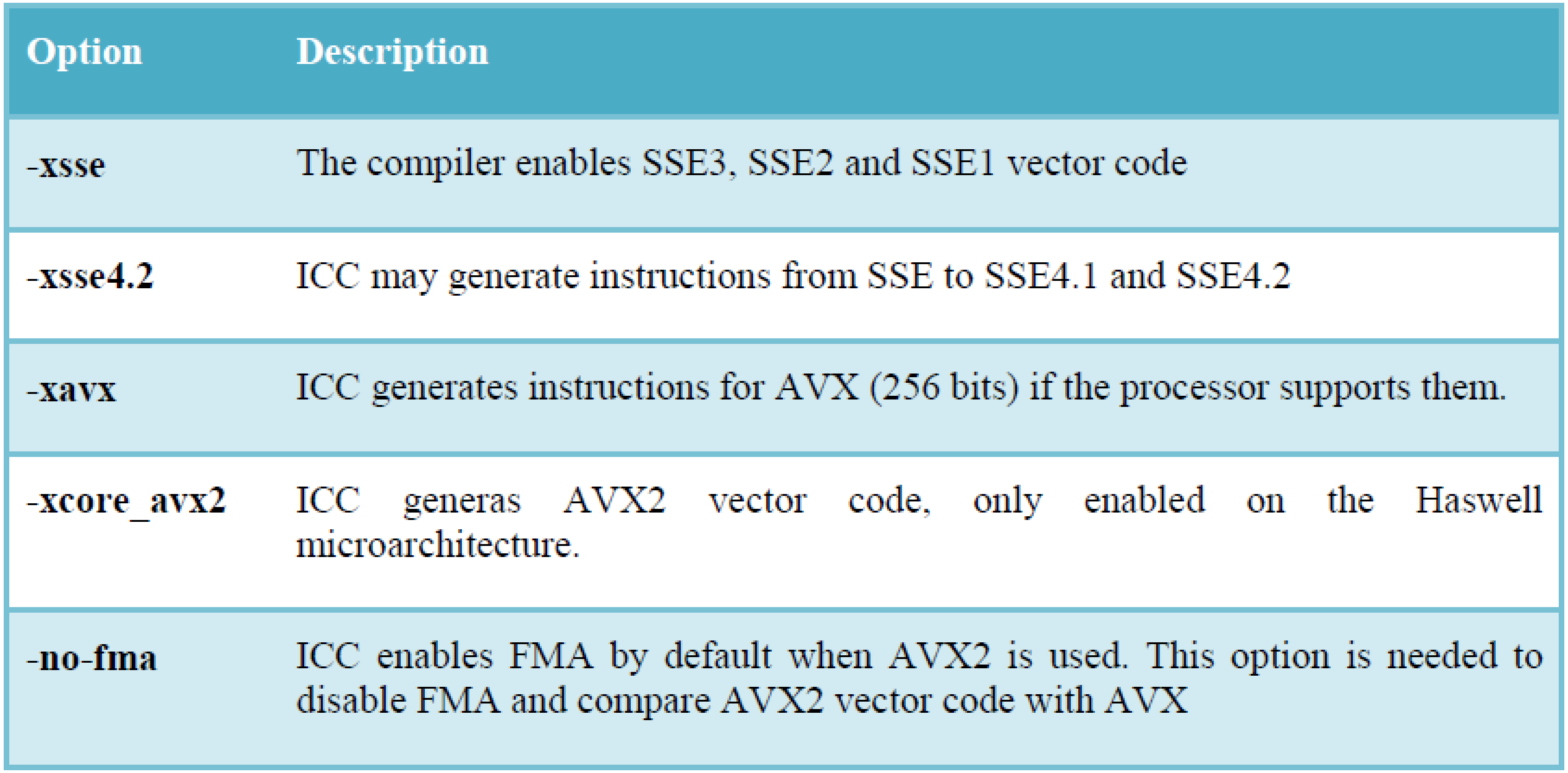
Thank you

CERN
openlab

Andrzej.Nowak@cern.ch

Backup – CEAN link & icc options

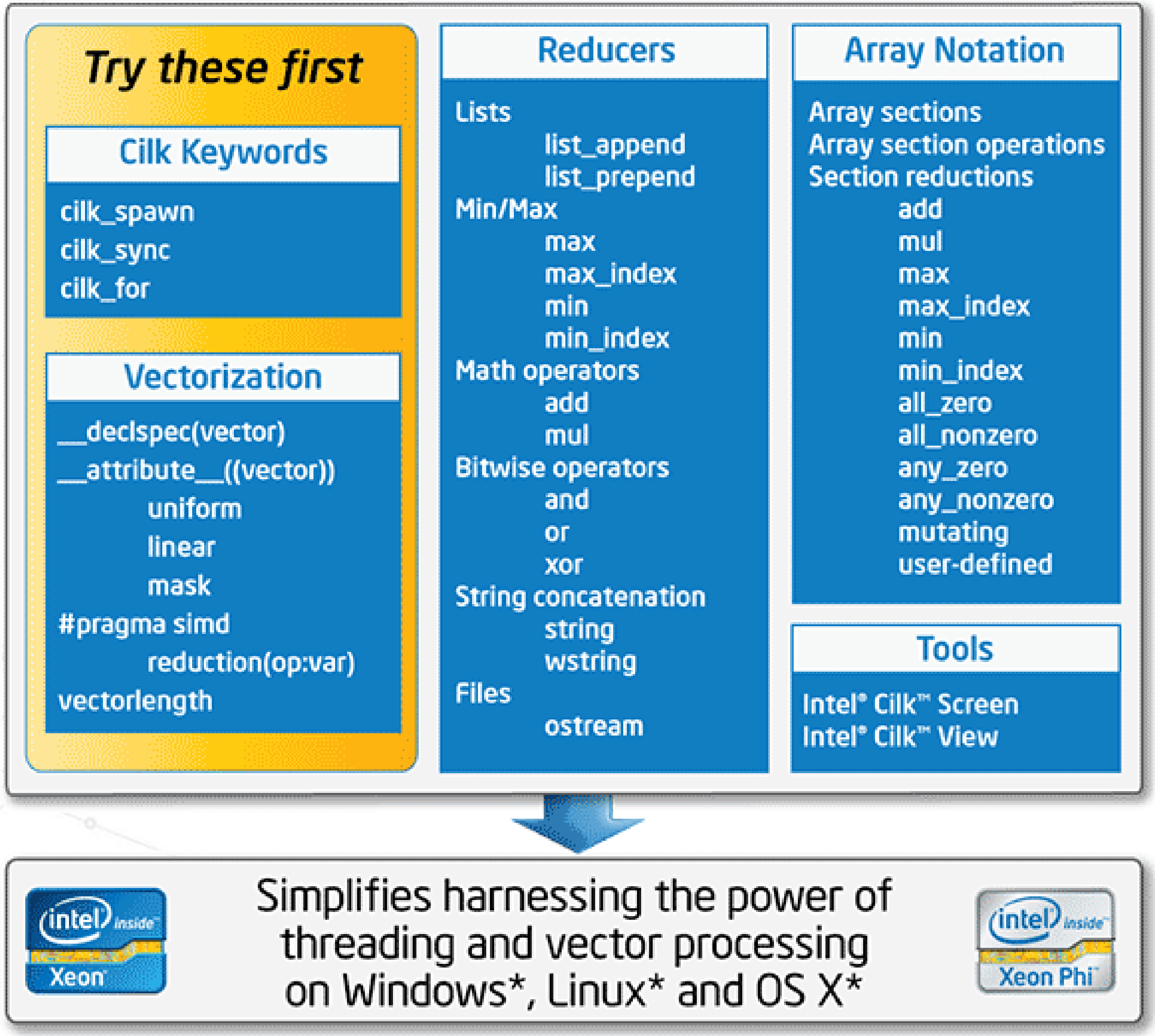
Cilk CEAN docs: http://software.intel.com/en-us/node/459410



Option	Description
-xsse	The compiler enables SSE3, SSE2 and SSE1 vector code
-xsse4.2	ICC may generate instructions from SSE to SSE4.1 and SSE4.2
-xavx	ICC generates instructions for AVX (256 bits) if the processor supports them.
-xcore_avx2	ICC generas AVX2 vector code, only enabled on the Haswell microarchitecture.
-no-fma	ICC enables FMA by default when AVX2 is used. This option is needed to disable FMA and compare AVX2 vector code with AVX

Intel® Cilk™ Plus

C/C++ compiler extension for simplified parallelism



Try these first

Cilk Keywords

- cilk_spawn
- cilk_sync
- cilk_for

Vectorization

- __declspec(vector)
- __attribute__((vector))
  - uniform
  - linear
  - mask
- #pragma simd
  - reduction(op:var)
- vectorlength

Reducers

- Lists
  - list_append
  - list_prepend
- Min/Max
  - max
  - max_index
  - min
  - min_index
- Math operators
  - add
  - mul
- Bitwise operators
  - and
  - or
  - xor
- String concatenation
  - string
  - wstring
- Files
  - ostream

Array Notation

- Array sections
- Array section operations
- Section reductions
  - add
  - mul
  - max
  - max_index
  - min
  - min_index
  - all_zero
  - all_nonzero
  - any_zero
  - any_nonzero
  - mutating
  - user-defined

Tools

- Intel® Cilk™ Screen
- Intel® Cilk™ View

Simplifies harnessing the power of threading and vector processing on Windows*, Linux* and OS X*