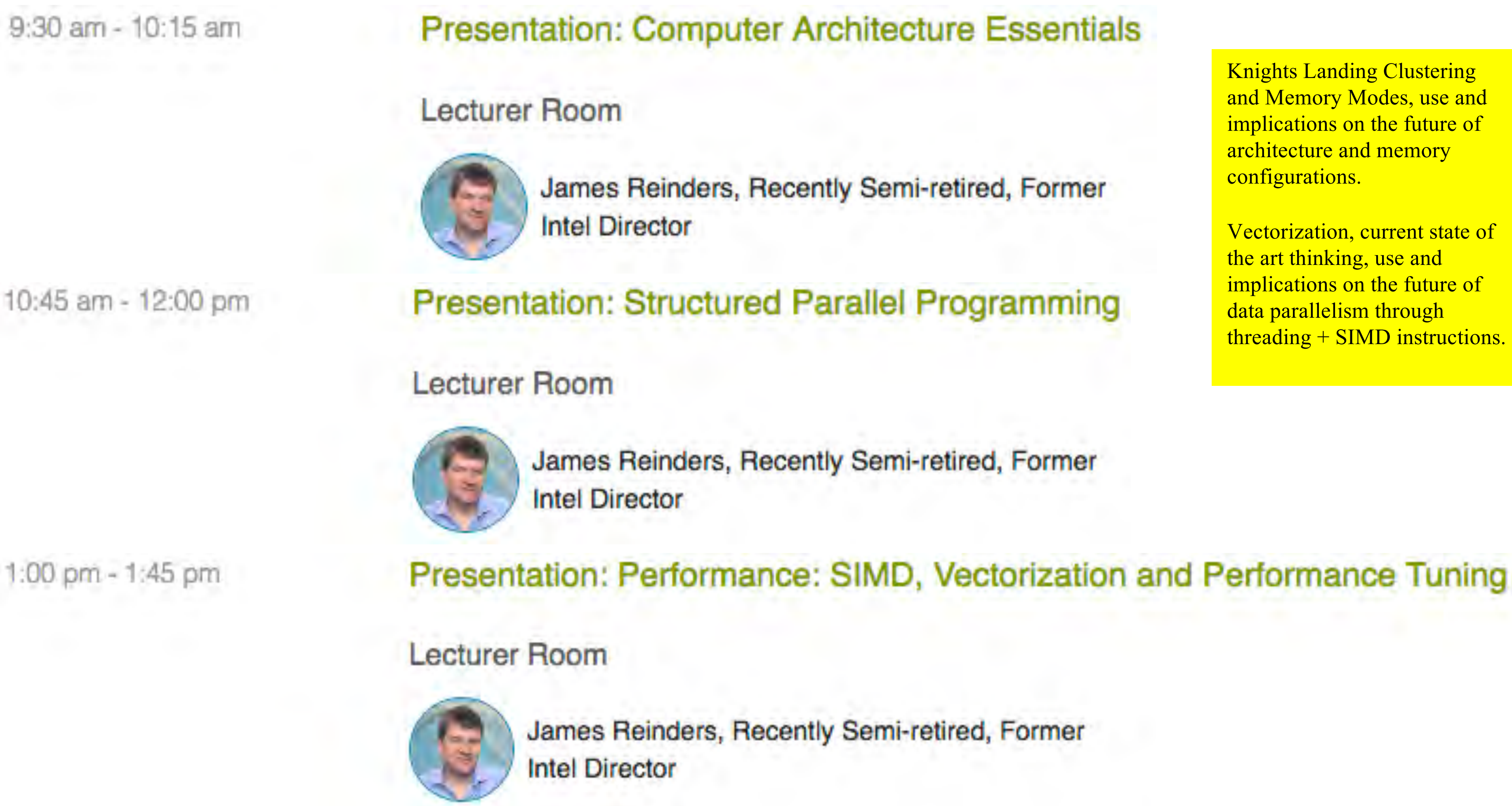
ATPESC

(Argonne Training Program on Extreme-Scale Computing)

Performance, SIMD, Vectorization and Performance Tuning

James Reinders
August 1, 2016, Pheasant Run, St Charles, IL
13:00-13:45

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.



9:30 am - 10:15 am

Presentation: Computer Architecture Essentials

Lecturer Room

James Reinders, Recently Semi-retired, Former Intel Director

10:45 am - 12:00 pm

Presentation: Structured Parallel Programming

Lecturer Room

James Reinders, Recently Semi-retired, Former Intel Director

1:00 pm - 1:45 pm

Presentation: Performance: SIMD, Vectorization and Performance Tuning

Lecturer Room

James Reinders, Recently Semi-retired, Former Intel Director

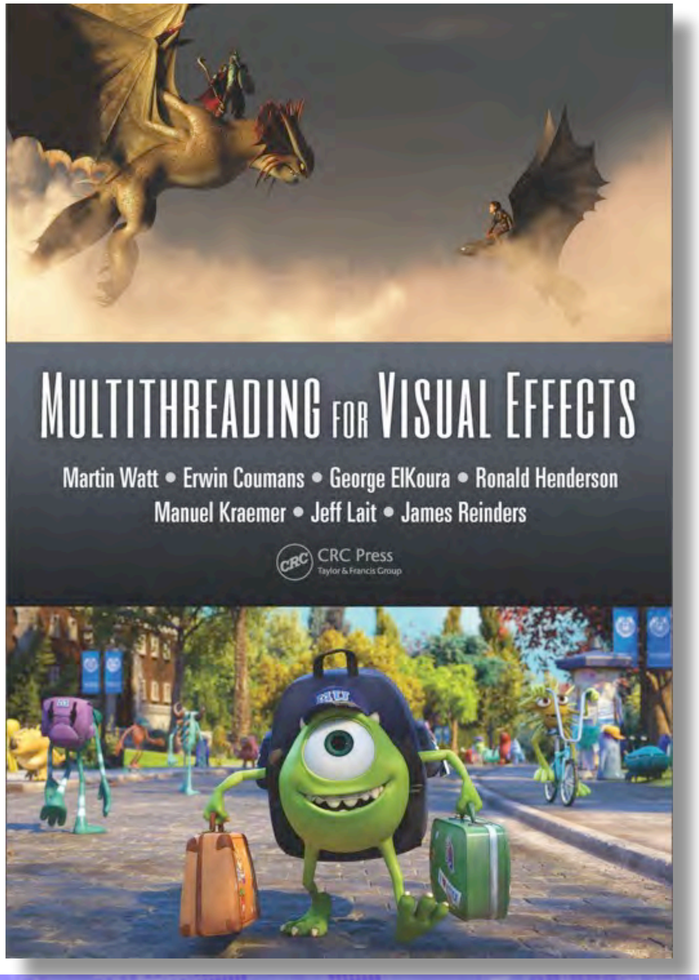
Knights Landing Clustering and Memory Modes, use and implications on the future of architecture and memory configurations.

Vectorization, current state of the art thinking, use and implications on the future of data parallelism through threading + SIMD instructions.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Parallel first

Vectorize second

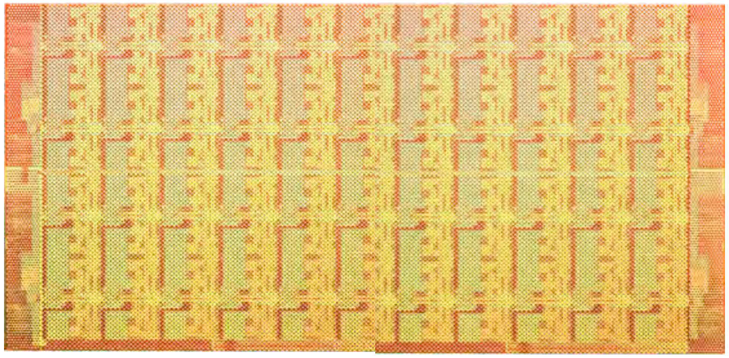

MULTITHREADING FOR VISUAL EFFECTS
Martin Watt • Erwin Coumans • George ElKoura • Ronald Henderson
Manuel Kraemer • Jeff Lait • James Reinders
CRC Press
Taylor & Francis Group

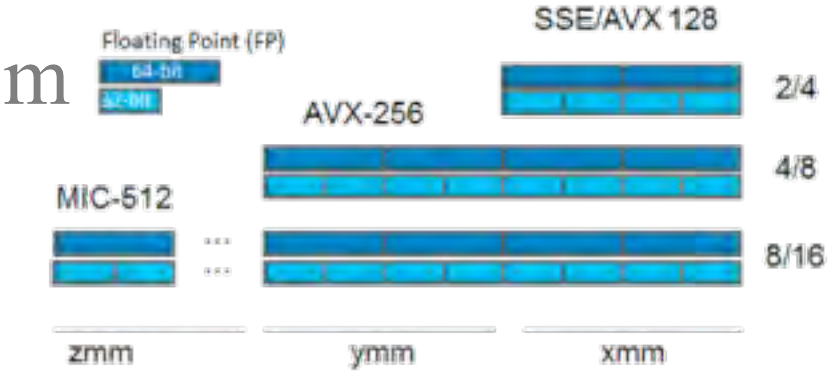
© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

Multithreading is more powerful than vectorization – by simple math:

16 way from vectorization

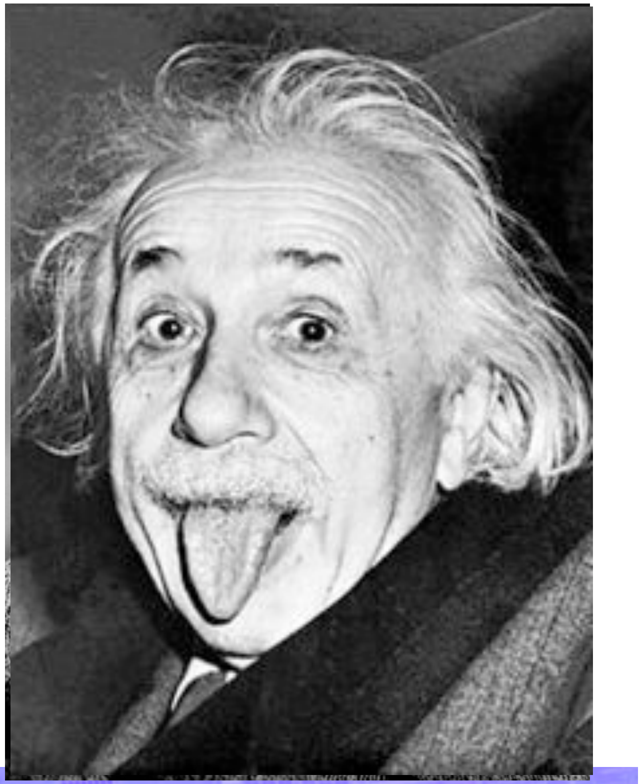
244 way from thread parallelism





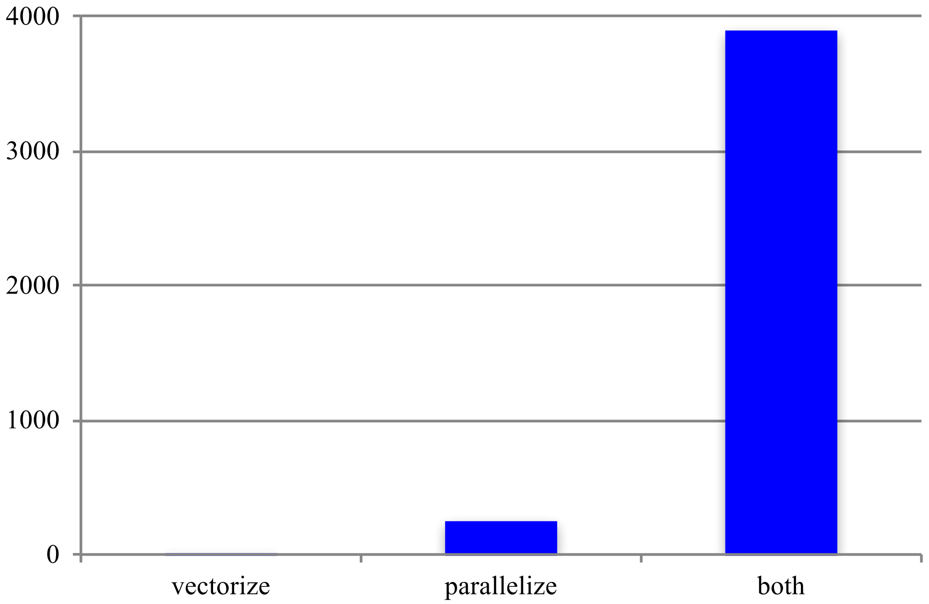
© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

There is an urban legend that Albert Einstein once said that compounding interest is the most powerful force in the universe.



© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

16 x 244 = 3904



MULTIPLICATION
EINSTEIN WAS RIGHT

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Assertion:

We need to embrace *explicit* vectorization in our programming.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Shouldn’t we solve with better tools?

What is vectorization?

Could we just ignore it?

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

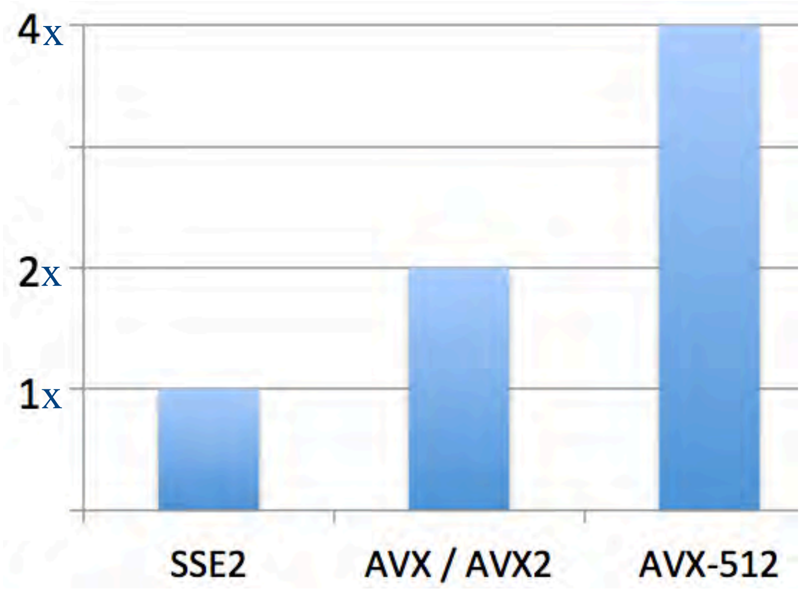
Vectors Instructions (SIMD instructions) Make things Faster

(that’s the premise)

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Up to 4x Performance

with Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Support

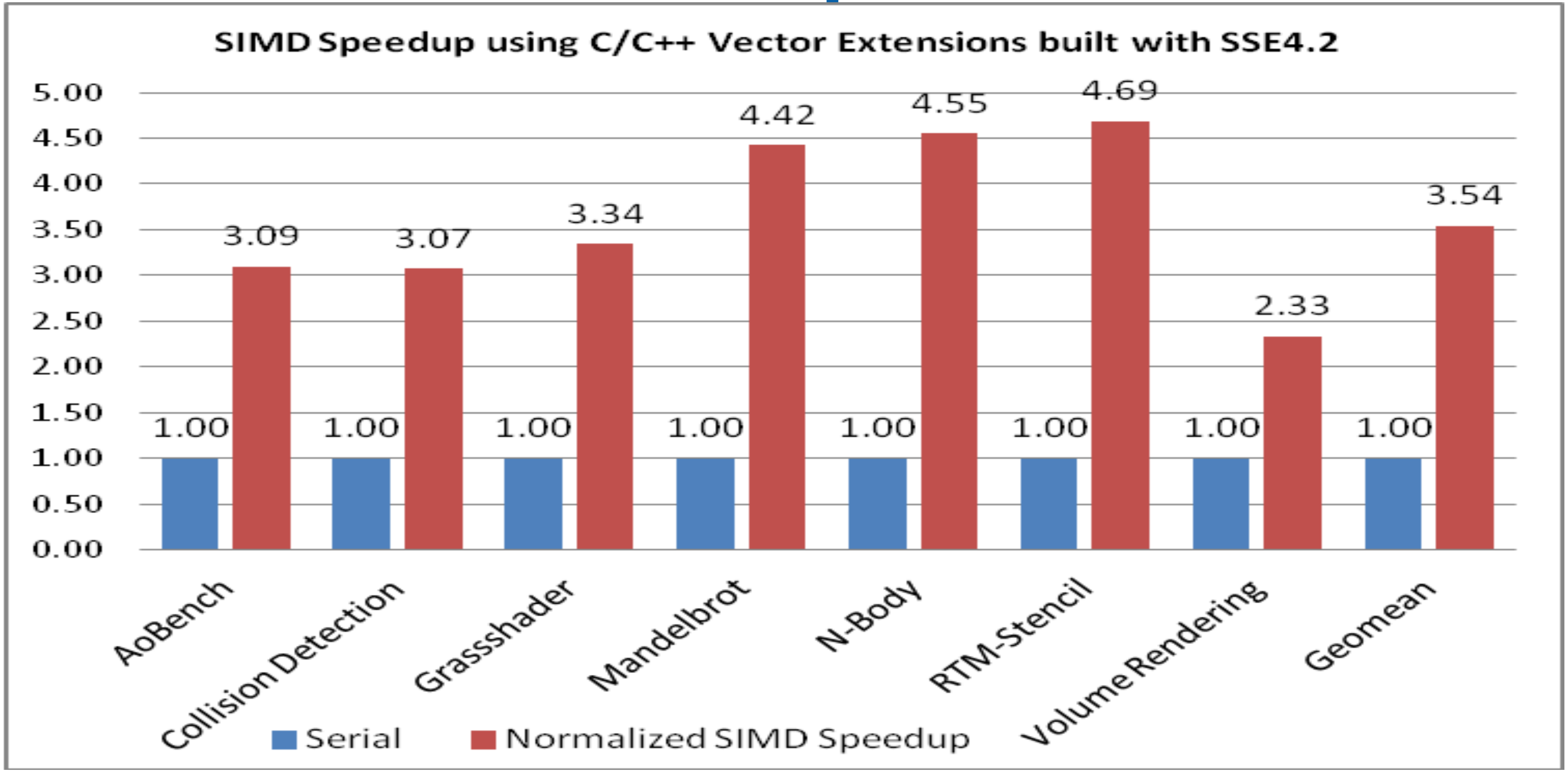


- Significant leap to 512-bit SIMD support for processors
- Intel® Compilers and Intel® Math Kernel Library include AVX-512 support
- Strong compatibility with AVX
- Added EVEX prefix enables additional functionality
- Appears first in future Intel® Xeon Phi™ coprocessor, code named Knights Landing

Higher performance for the most demanding computational tasks

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Performance with Explicit Vectorization



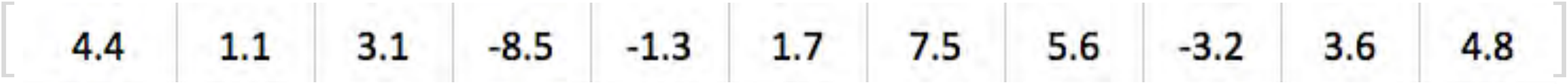
Configuration: Intel® Core™ i7 CPU X980 system (6 cores with Hyper-Threading On), running at 3.33GHz, with 4.0GB RAM, 12M smart cache, 64-bit Windows Server 2008 R2 Enterprise SP1. **For more information go to** http://www.intel.com/performance

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

What is a Vector?

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

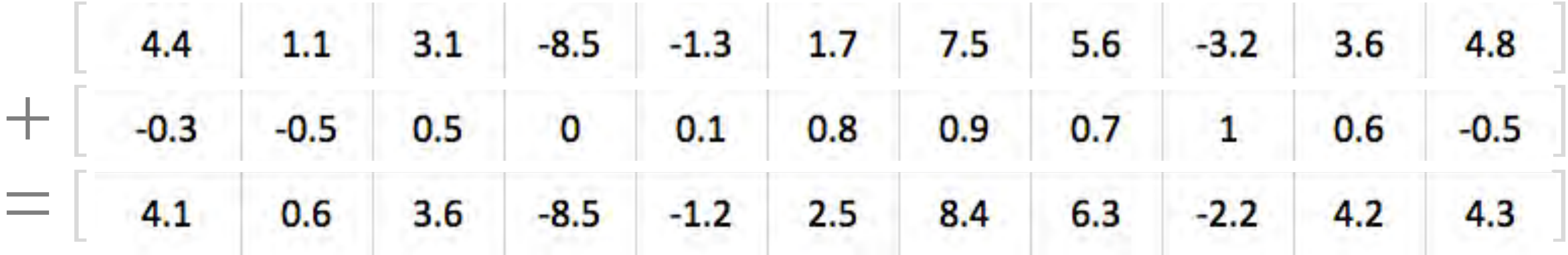
Vector of numbers



4.4	1.1	3.1	-8.5	-1.3	1.7	7.5	5.6	-3.2	3.6	4.8	

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

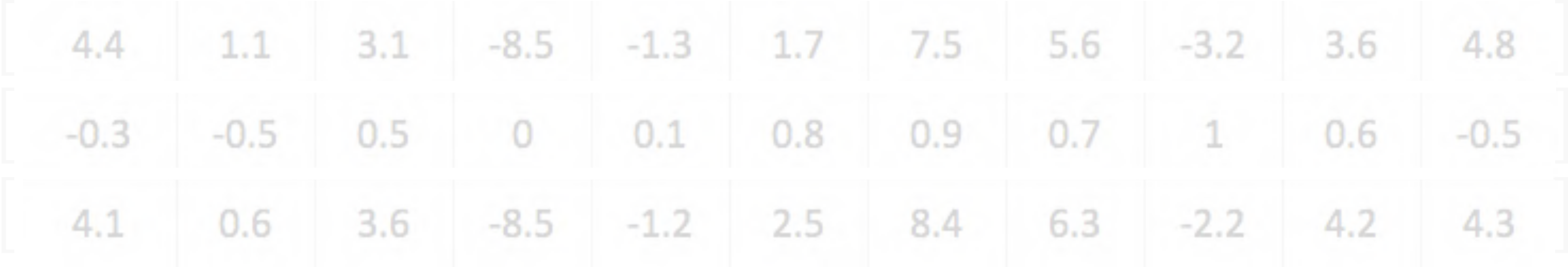
Vector addition



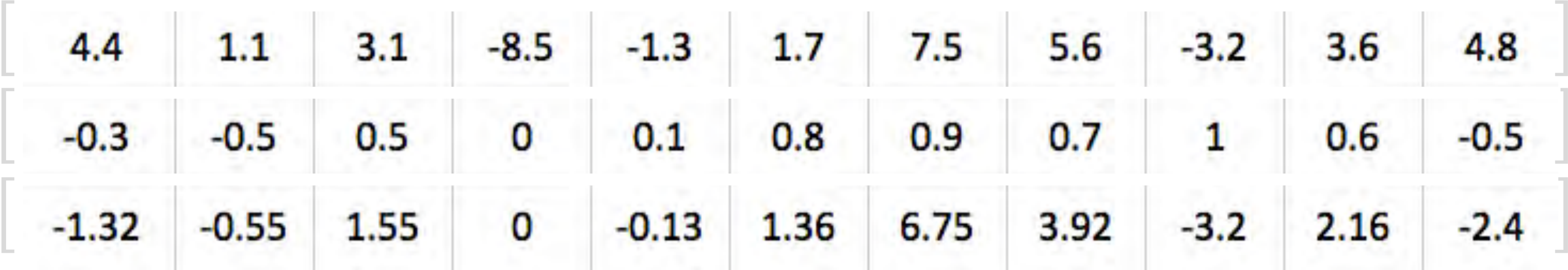
	4.4	1.1	3.1	-8.5	-1.3	1.7	7.5	5.6	-3.2	3.6	4.8
+	-0.3	-0.5	0.5	0	0.1	0.8	0.9	0.7	1	0.6	-0.5
=	4.1	0.6	3.6	-8.5	-1.2	2.5	8.4	6.3	-2.2	4.2	4.3

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

…and Vector multiplication



	4.4	1.1	3.1	-8.5	-1.3	1.7	7.5	5.6	-3.2	3.6	4.8
+	-0.3	-0.5	0.5	0	0.1	0.8	0.9	0.7	1	0.6	-0.5
=	4.1	0.6	3.6	-8.5	-1.2	2.5	8.4	6.3	-2.2	4.2	4.3



	4.4	1.1	3.1	-8.5	-1.3	1.7	7.5	5.6	-3.2	3.6	4.8
×	-0.3	-0.5	0.5	0	0.1	0.8	0.9	0.7	1	0.6	-0.5
=	-1.32	-0.55	1.55	0	-0.13	1.36	6.75	3.92	-3.2	2.16	-2.4

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

An example

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vector data operations: data operations done in parallel

```
void v_add (float *c,
        float *a,
        float *b)
{
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vector data operations:
data operations done in parallel

```
void v_add (float *c,
            float *a,
            float *b)
{
      for (int i=0; i<= MAX; i++)
           c[i]=a[i]+b[i];
}
```

Loop:

1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vector data operations:
data operations done in parallel

```
void v_add (float *c,
            float *a,
            float *b)
{
    for (int i=0; i<= MAX; i++)
        c[i]=a[i]+b[i];
}
```

Loop:

1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i

Loop:

1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vector data operations:
data o

We call this “vectorization”

```
void v_add (float *c,
            float *a,
            float *b)
{
  for (int i=0; i<= MAX; i++)
    c[i]=a[i]+b[i];
}
```

Loop:
1. LOADv4 a[i:i+3] -> Rva
2. LOADv4 b[i:i+3] -> Rvb
3. ADDv4 Rva, Rvb -> Rvc
4. STOREv4 Rvc -> c[i:i+3]
5. ADD i + 4 -> i

Loop:
1. LOAD a[i] -> Ra
2. LOAD b[i] -> Rb
3. ADD Ra, Rb -> Rc
4. STORE Rc -> c[i]
5. ADD i + 1 -> i

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vector data operations:
data operations done in parallel

```
void v_add (float *c, float *a, float *b)
{
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vector data operations:
data operations done in parallel

```
void v_add (float *c, float *a, float *b)
{
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

PROBLEM:
This LOOP is NOT LEGAL to (automatically) VECTORIZE in C / C++ (without more information).

Arrays *not* really in the language

Pointers are, evil pointers!

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Choice 1:
use a compiler switch for auto-vectorization

(and *hope* it vectorizes)

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Choice 2:
give your compiler hints

(and *hope* it vectorizes)

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

C99 *restrict* keyword

```
void v_add (float *restrict c,
        float *restrict a,
        float *restrict b)
{
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

IVDEP (ignore assumed vector dependencies)

```
void v_add (float *c,
        float *a,
        float *b)
{
#pragma ivdep
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Choice 3:
code explicitly for vectors

(mandatory vectorization)

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

OpenMP* 4.0: #pragma omp simd

```
void v_add (float *c,
        float *a,
        float *b)
{
#pragma omp simd
    for (int i=0; i<= MAX; i++)
       c[i]=a[i]+b[i];
}
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

OpenMP* 4.0: #pragma omp declare simd

```
#pragma omp declare simd
void v1_add (float *c,
        float *a,
        float *b)
{

       *c=*a+*b;
}
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

SIMD instruction intrinsics

```
void v_add (float *c,
            float *a,
            float *b)
{
    __m128* pSrc1 = (__m128*) a;
    __m128* pSrc2 = (__m128*) b;
    __m128* pDest = (__m128*) c;
    for (int i=0; i<= MAX/4; i++)
        *pDest++ = _mm_add_ps(*pSrc1++, *pSrc2++);
}
```

Hard coded to 4 wide !

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

array operations (Cilk™ Plus)

```
void v_add (float *c,
            float *a,
            float *b)
{
           c[0:MAX]=a[0:MAX]+b[0:MAX];
}
```

Challenge: long vector slices can cause cache issues; fix is to keep vector slices short.

Cilk™ Plus is supported in Intel compilers, and gcc (4.9).

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vectorization solutions

1. auto-vectorization (use a compiler switch and hope it vectorizes)
   - sequential languages and practices gets in the way
2. give your compiler hints and hope it vectorizes
   - C99 restrict (implied in FORTRAN since 1956)
   - #pragma ivdep
3. code explicitly
   - OpenMP 4.0 #pragma omp simd
   - Cilk™ Plus array notations
   - SIMD instruction intrinsics
   - Kernels: OpenMP 4.0 #pragma omp declare simd; OpenCL; CUDA kernel functions

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

vectorization solutions

1. auto-vectorization (use a compiler switch and hope it vectorizes)
   - sequential languages and practices gets in the way
2. give your compiler hints and hope it vectorizes
   - C99 restrict (implied in FORTRAN since 1956)
   - #pragma ivdep
3. code explicitly
   - OpenMP 4.0 #pragma omp simd
   - Cilk™ Plus array notations
   - SIMD instruction intrinsics
   - Kernels: OpenMP 4.0 #pragma omp declare simd; OpenCL; CUDA kernel functions

Best at being
Reliable, predictable and portable

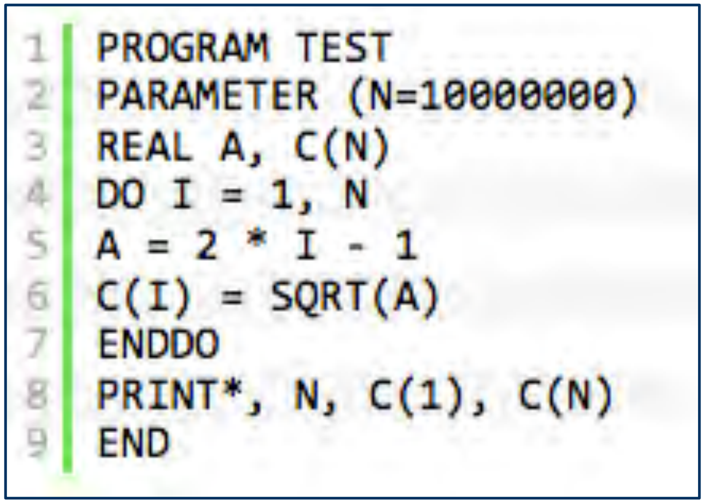
© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

Explicit parallelism

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

parallelization

Try auto-parallel capability:
-parallel (Linux* or OS X*)
-Qparallel (Windows*)



```
PROGRAM TEST
PARAMETER (N=10000000)
REAL A, C(N)
DO I = 1, N
A = 2 * I - 1
C(I) = SQRT(A)
ENDDO
PRINT*, N, C(1), C(N)
END
```

Or explicitly use…
Fortran directive (!DIR$ PARALLEL)
C pragma (#pragma parallel)
Intel® Threading Building Blocks (TBB)

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

parallelization

Try auto-parallel capability:

-parallel (Linux or OS X*)

-Qparallel (Windows)

Best at being
Reliable, predictable and portable

Or explicitly use…

OpenMP

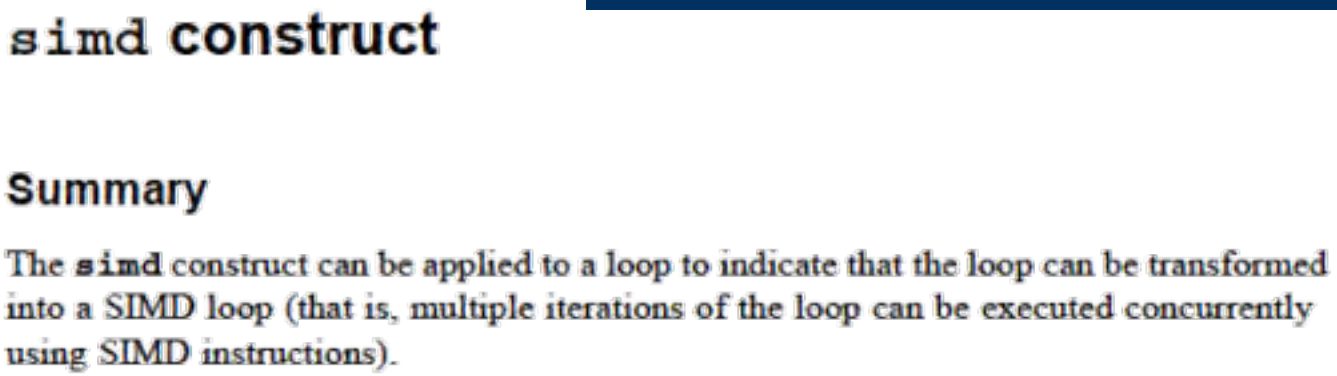
Intel® Threading Building Blocks (T)

```
c$OMP PARALLEL DO
 DO I=1,N B(I) = (A(I) + A(I-1)) / 2.0
 END DO
c$OMP END PARALLEL DO
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

OpenMP 4.0

Based on a proposal from Intel based on customer success with the Intel® Cilk™ Plus features in Intel



simd construct

Summary

The **simd** construct can be applied to a loop to indicate that the loop can be transformed into a SIMD loop (that is, multiple iterations of the loop can be executed concurrently using SIMD instructions).

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

OpenMP 4.0

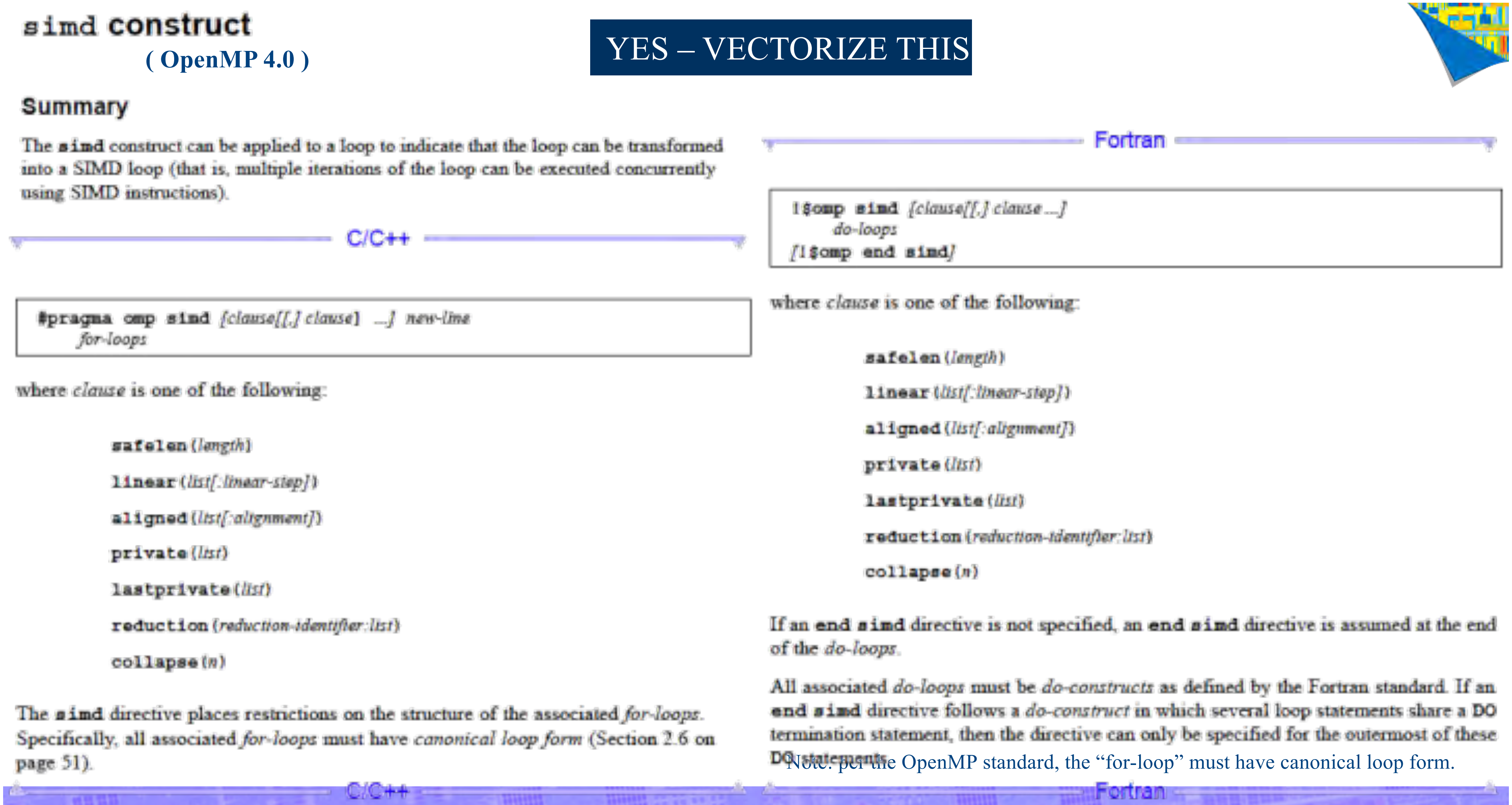
Based on a proposal from Intel based on customer success with the Intel® Cilk™ Plus features in Intel

simd construct

```
#pragma omp simd reduction(+:val) reduction(+:val2)
  for(int pos = 0; pos < RAND_N; pos++) {
     float callValue=
              expectedCall(Sval,Xval,MuByT,VBySqrtT,l_Random[pos]);
     val  += callValue;
     val2 += callValue * callValue;
}
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

YES – VECTORIZE THIS



simd construct

(OpenMP 4.0)

Summary

The **simd** construct can be applied to a loop to indicate that the loop can be transformed into a SIMD loop (that is, multiple iterations of the loop can be executed concurrently using SIMD instructions).

C/C++

```
#pragma omp simd [clause[[,] clause] ...] new-line
      for-loops
```

where *clause* is one of the following:

```
safelen(length)
linear(list[:linear-step])
aligned(list[:alignment])
private(list)
lastprivate(list)
reduction(reduction-identifier:list)
collapse(n)
```

The **simd** directive places restrictions on the structure of the associated *for-loops*. Specifically, all associated *for-loops* must have *canonical loop form* (Section 2.6 on page 51).

C/C++

Fortran

```
!$omp simd [clause[[,] clause ...]
      do-loops
[!$omp end simd]
```

where *clause* is one of the following:

```
safelen(length)
linear(list[:linear-step])
aligned(list[:alignment])
private(list)
lastprivate(list)
reduction(reduction-identifier:list)
collapse(n)
```

If an **end simd** directive is not specified, an **end simd** directive is assumed at the end of the *do-loops*.

All associated *do-loops* must be *do-constructs* as defined by the Fortran standard. If an **end simd** directive follows a *do-construct* in which several loop statements share a **DO** termination statement, then the directive can only be specified for the outermost of these **DO** statements.

Fortran

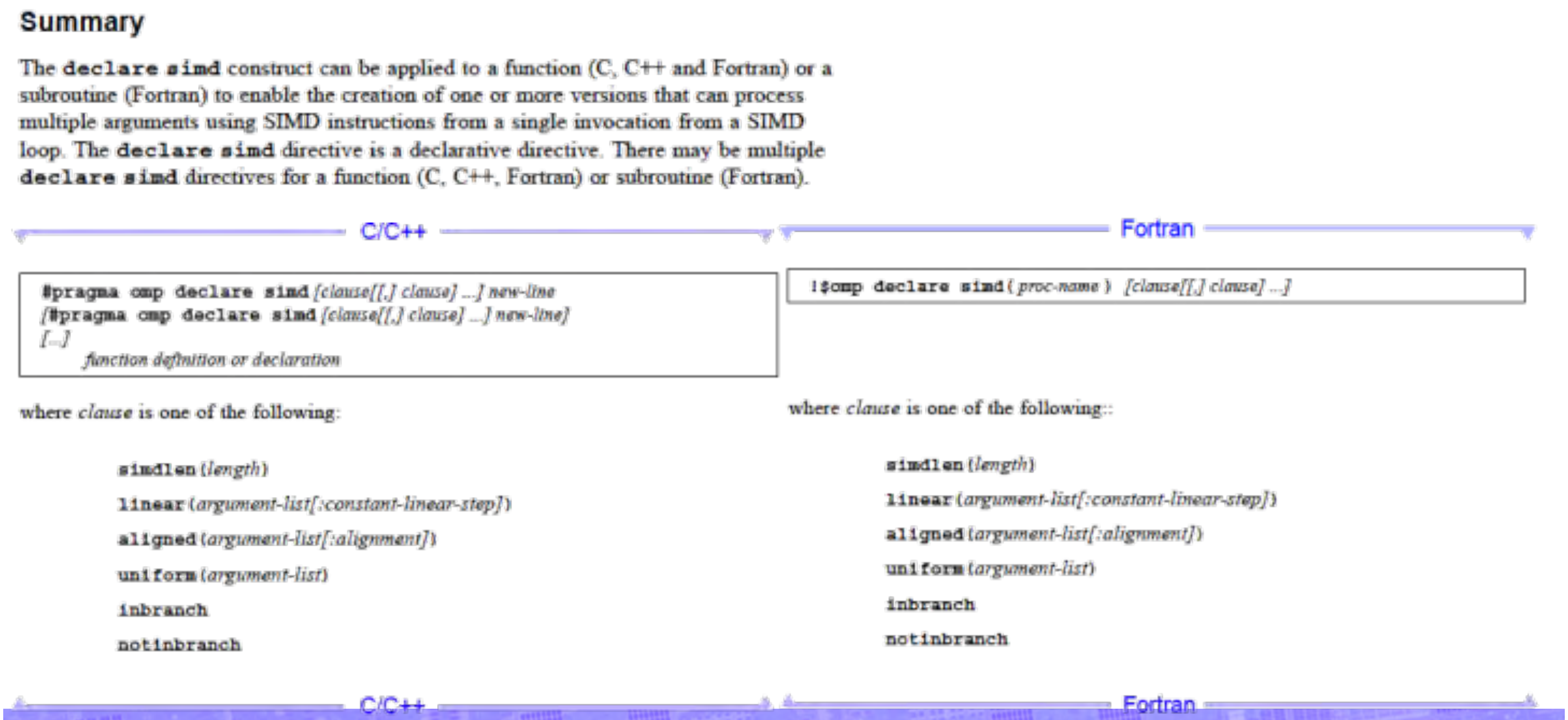
Note: per the OpenMP standard, the "for-loop" must have canonical loop form.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

declare simd construct

(OpenMP 4.0)

Make VECTOR versions of this function.



Summary

The **declare simd** construct can be applied to a function (C, C++ and Fortran) or a subroutine (Fortran) to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop. The **declare simd** directive is a declarative directive. There may be multiple **declare simd** directives for a function (C, C++, Fortran) or subroutine (Fortran).

C/C++

```
#pragma omp declare simd [clause[[,] clause] ...] new-line
[#pragma omp declare simd [clause[[,] clause] ...] new-line]
[...]
    function definition or declaration
```

where *clause* is one of the following:

- **simdlen** (*length*)
- **linear** (*argument-list[:constant-linear-step]*)
- **aligned** (*argument-list[:alignment]*)
- **uniform** (*argument-list*)
- **inbranch**
- **notinbranch**

C/C++

Fortran

```
!$omp declare simd( proc-name ) [clause[[,] clause] ...]
```

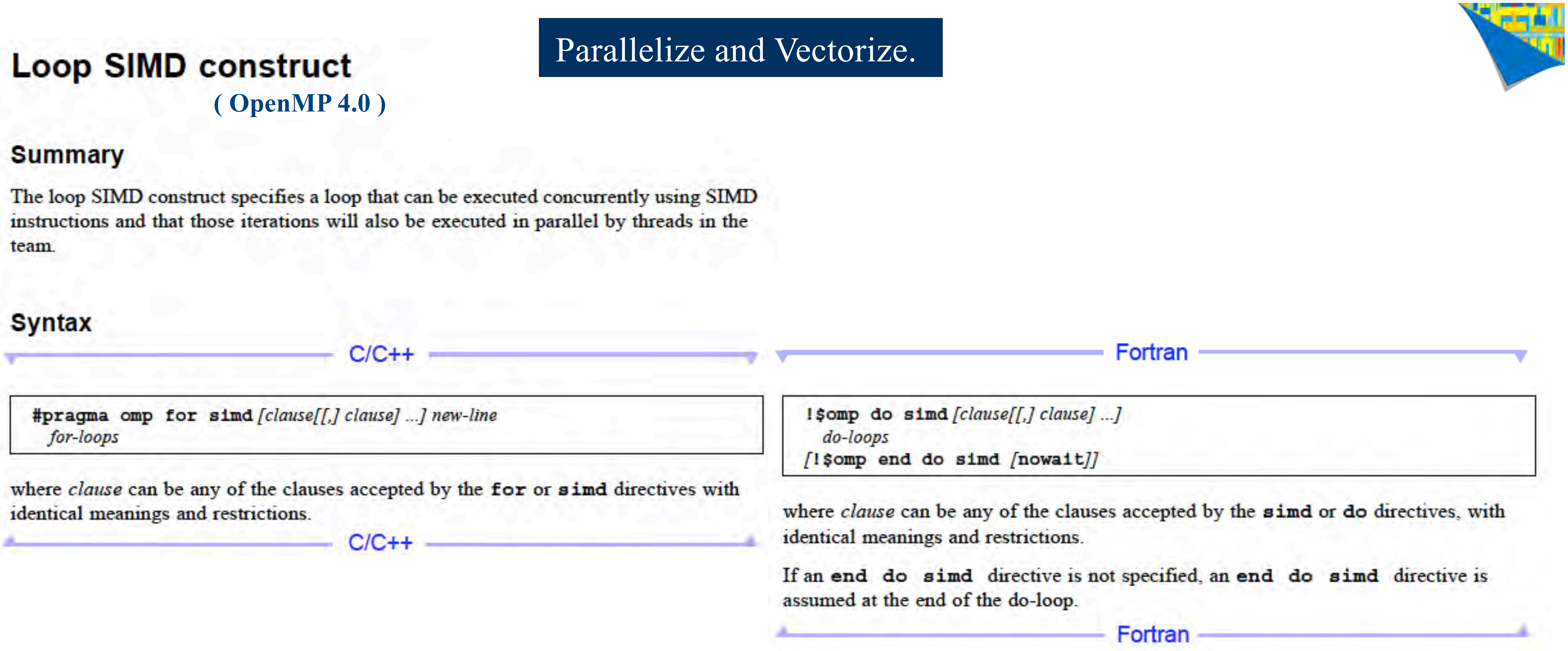
where *clause* is one of the following::

- **simdlen** (*length*)
- **linear** (*argument-list[:constant-linear-step]*)
- **aligned** (*argument-list[:alignment]*)
- **uniform** (*argument-list*)
- **inbranch**
- **notinbranch**

Fortran

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Parallelize and Vectorize.



Loop SIMD construct

(OpenMP 4.0)

Summary

The loop SIMD construct specifies a loop that can be executed concurrently using SIMD instructions and that those iterations will also be executed in parallel by threads in the team.

Syntax

C/C++

```
#pragma omp for simd [clause[[,] clause] ...] new-line
   for-loops
```

where *clause* can be any of the clauses accepted by the **for** or **simd** directives with identical meanings and restrictions.

C/C++

Fortran

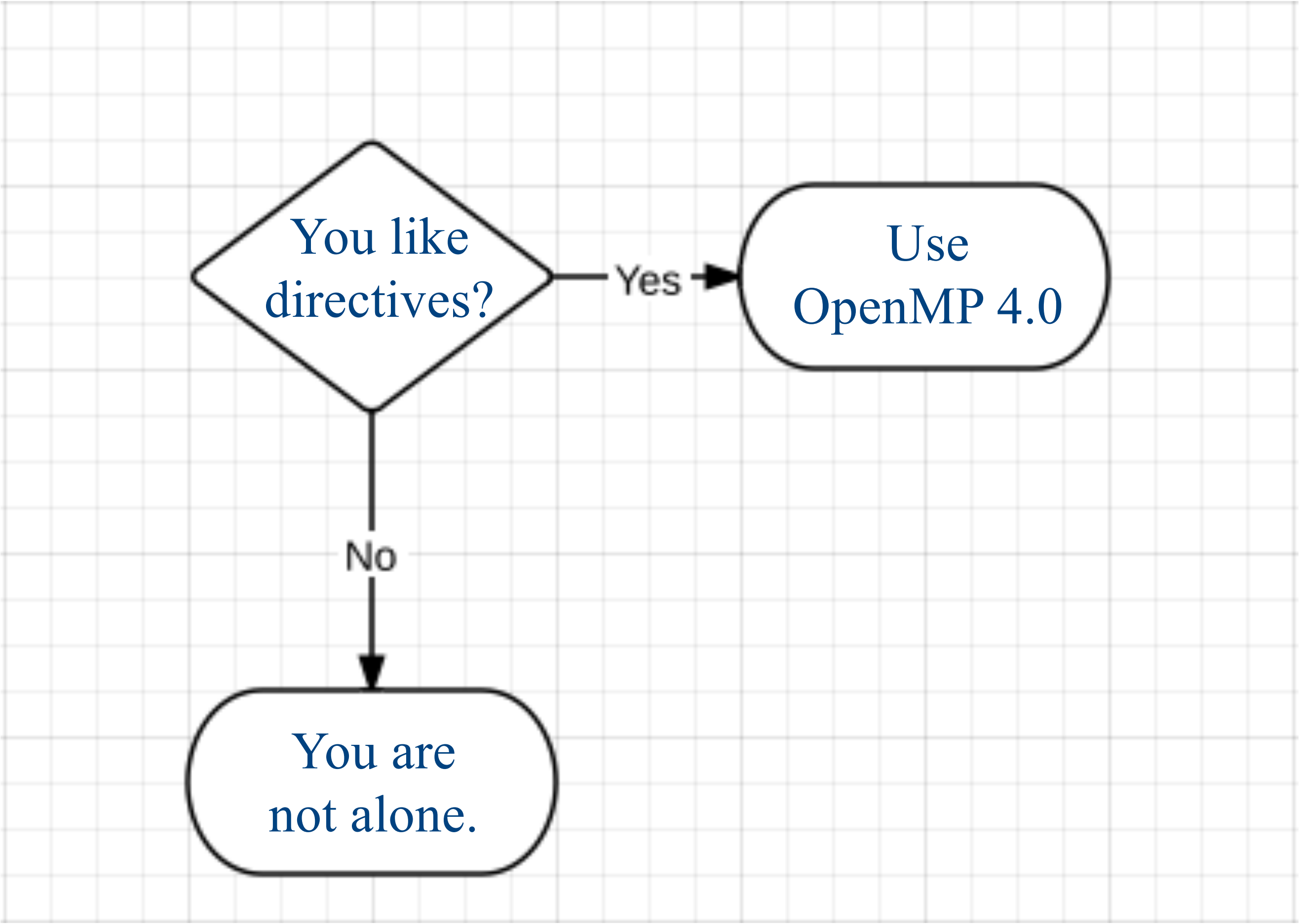
```
!$omp do simd [clause[[,] clause] ...]
   do-loops
[!$omp end do simd [nowait]]
```

where *clause* can be any of the clauses accepted by the **simd** or **do** directives, with identical meanings and restrictions.

If an **end do simd** directive is not specified, an **end do simd** directive is assumed at the end of the do-loop.

Fortran

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.



© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

for your consideration:
Intel Compilers support **keywords** as an alternative

- Keyword versions of SIMD pragmas added:
  `_Simd, _Safelen, _Reduction`
- `__intel_simd_lane()` intrinsic for SIMD enabled functions

Keywords / library interfaces being discussed for SIMD constructs in C and C++ standards

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

History of Intel vector instructions

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Intel Instruction Set Vector Extensions from 1997-2008

1997	1998	1999	2004	2006	2007	2008
Intel® MMX™ technology	**Intel® SSE**	**Intel® SSE2**	**Intel® SSE3**	**Intel® SSSE3**	**Intel® SSE4.1**	**Intel® SSE4.2**
57 new instructions 64 bits Overload FP stack Integer only media extensions	**70 new instructions** 128 bits 4 single-precision vector FP scalar FP instructions cacheability instructions control & conversion instructions media extensions	**144 new instructions** 128 bits 2 double-precision vector FP 8/16/32/64 vector integer 128-bit integer memory & power management	**13 new instructions** 128 bits FP vector calculation x87 integer conversion 128-bit integer unaligned load thread sync.	**32 new instructions** 128 bits enhanced packed integer calculation	**47 new instructions** 128 bits packed integer calculation & conversion better vectorization by compiler load with streaming hint	**7 new instructions** 128 bits string (XML) processing POP-Count CRC32

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Intel Instruction Set Vector Extensions since 2011

2011	2011	2012	2013	TBD
Intel® AVX	Co-processor only 512	**"AVX-1.5"**	**Intel® AVX-2**	**Intel® AVX-512**
Promotion of 128 bit FP vector instructions to 256 bit	**Coprocessor predecessor to AVX-512. New 512 bit vector instructions for MIC architecture, binary compt. not supported by processors –** mostly source compatible with AVX-512	**7 new instructions** 16 bit FP support RDRAND …	**Promotion of integer instruction to 256 bit** - FMA - Gather - TSX/RTM	**Promotion of vector instructions to 512 bits** **Xeon Phi: FI, CDI, ERI, PFI** **Xeon: FI, CDI, BWI, DQI, VLE**

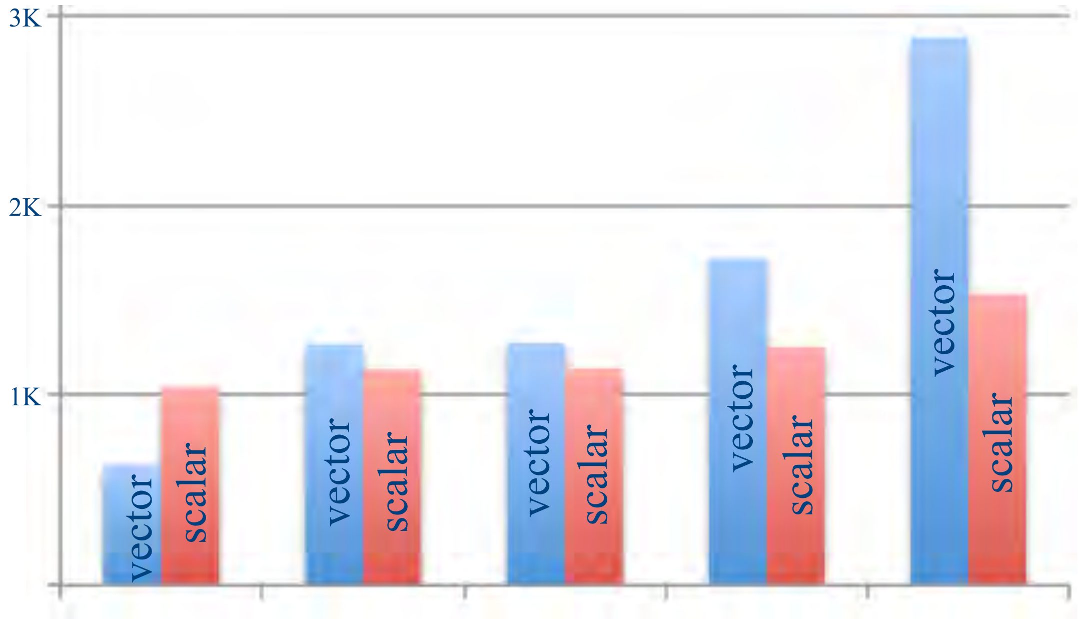
Reinders blogs announced – July 2013, and June 2014.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

		width	Int.	SP	DP
1997	MMX	64	✔		
1999	SSE	128	✔	✔(x4)	
2001	SSE2	128	✔	✔	✔(x2)
2004	SSE3	128	✔	✔	✔
2006	SSSE 3	128	✔	✔	✔
2006	SSE 4.1	128	✔	✔	✔
2008	SSE 4.2	128	✔	✔	✔
2011	AVX	256	✔	✔(x8)	✔(x4)
2013	AVX2	256	✔	✔	✔
future	AVX-512	512	✔	✔(x16)	✔(x8)

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Growth is in vector instructions



Disclaimer: Counting/attributing instructions is in inexact science. The exact numbers are easily debated, the trend is quite real regardless.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Motivation for AVX-512 Conflict Detection

Sparse computations are common in HPC, but hard to vectorize due to race conditions

Consider the “histogram” problem:

```
for(i=0; i<16; i++) {  A[B[i]]++; }
```

```
index = vload &B[i]                 // Load 16 B[i]
old_val = vgather A, index          // Grab A[B[i]]
new_val = vadd old_val, +1.0        // Compute new values
vscatter A, index, new_val          // Update A[B[i]]
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Motivation for AVX-512 Conflict Detection

Sparse computations are common in HPC, but hard to vectorize due to race conditions

```
for(i=0; i<16; i++) {  A[B[i]]++; }
```

Consider the "histogram" problem:

```
index = vload &B[i]                   // Load 16 B[i]
old_val = vgather A, index            // Grab A[B[i]]
new_val = vadd old_val, +1.0          // Compute new values
vscatter A, index, new_val            // Update A[B[i]]
```

- Code above is wrong if any values within B[i] are duplicated
  - Only one update from the repeated index would be registered!
- A solution to the problem would be to avoid executing the sequence gather-op-scatter with vector of indexes that contain conflicts

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Conflict Detection Instructions in AVX-512

improve vectorization!

VPCONFLICT instruction detects elements with previous conflicts in a vector of indexes

- Allows to generate a mask with a subset of elements that are guaranteed to be conflict free
- The computation loop can be re-executed with the remaining elements until all the indexes have been operated upon

CDI instr.
VPCONFLICT{D,Q} zmm1{k1}, zmm2/mem
VPBROADCASTM{W2D,B2Q} zmm1, k2
VPTESTNM{D,Q} k2{k1}, zmm2, zmm3/mem
VPLZCNT{D,Q} zmm1 {k1}, zmm2/mem

```
index = vload &B[i]                                // Load 16 B[i]
pending_elem = 0xFFFF;                             // all still remaining
do {
   curr_elem = get_conflict_free_subset(index, pending_elem)
   old_val = vgather {curr_elem} A, index          // Grab A[B[i]]
   new_val = vadd old_val, +1.0                    // Compute new values
   vscatter A {curr_elem}, index, new_val          // Update A[B[i]]
   pending_elem = pending_elem ^ curr_elem         // remove done idx
} while (pending_elem)
```

for illustration: this not even the fastest version

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

-vec-report

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

“Dear compiler, did you vectorize my loop?”

We heard your feedback……

-vec-report output was hard to understand;

Messages were too cryptic to understand;

Information about one loop showing up at many places of report;

Was easy to be confused about multiple versions of one loop created by the compiler.

We couldn’t do everything you asked, but here are the improvements made for 15.0 compiler (in 2014).

Expect more changes to come, during beta and in future versions.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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 Reports (since 2014)

- Old functionality implemented under **`-opt-report`**, **`-vec-report, -openmp-report, -par-report`** replaced by unified **`-opt-report`** compiler options
  - **`[vec,openmp,par]-report`** options deprecated and map to equivalent opt-report-phase

- Can still select phase with **`-opt-report-phase`** option. For example, to only get vectorization reports, use **`-opt-report-phase=vec`**

- Output now defaults to a **`<name>.optrpt`** file where **`<name>`** corresponds to the output object name. This can be changed with **`-opt-report-file=[<name>|stdout|stderr]`**

- Windows*: **`/Qopt-report, /Qopt-report-phase=<phase> etc.`**
  - Optimization report integration with Microsoft* Visual Studio planned to appear in beta update 1

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

“Vectorization Advisor” – Advisor XE

1. “All the data you need in one place”

*Leverages **Intel Compiler** opt-report+ and **dynamic profile**. Support for other compilers, C, C++, Fortran, for MPI env.*

2. Detects “hot” un-vectorized or “under vectorized” loops.

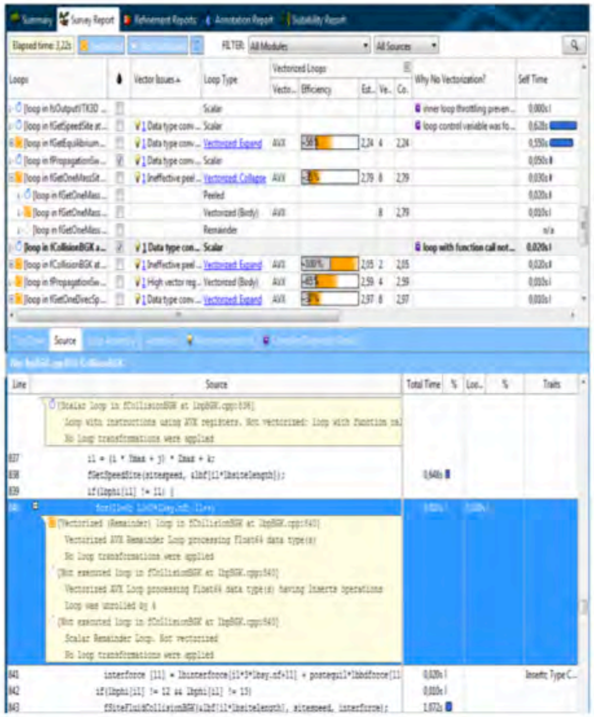
Identifies what is blocking efficient vectorization, where to add it

3. Identify performance penalties and recommend fixes

*Explicit **advices** with “true intelligence”, covering OpenMP4.x.*

4. Memory layout analysis

5. Increase the confidence that vectorization is safe



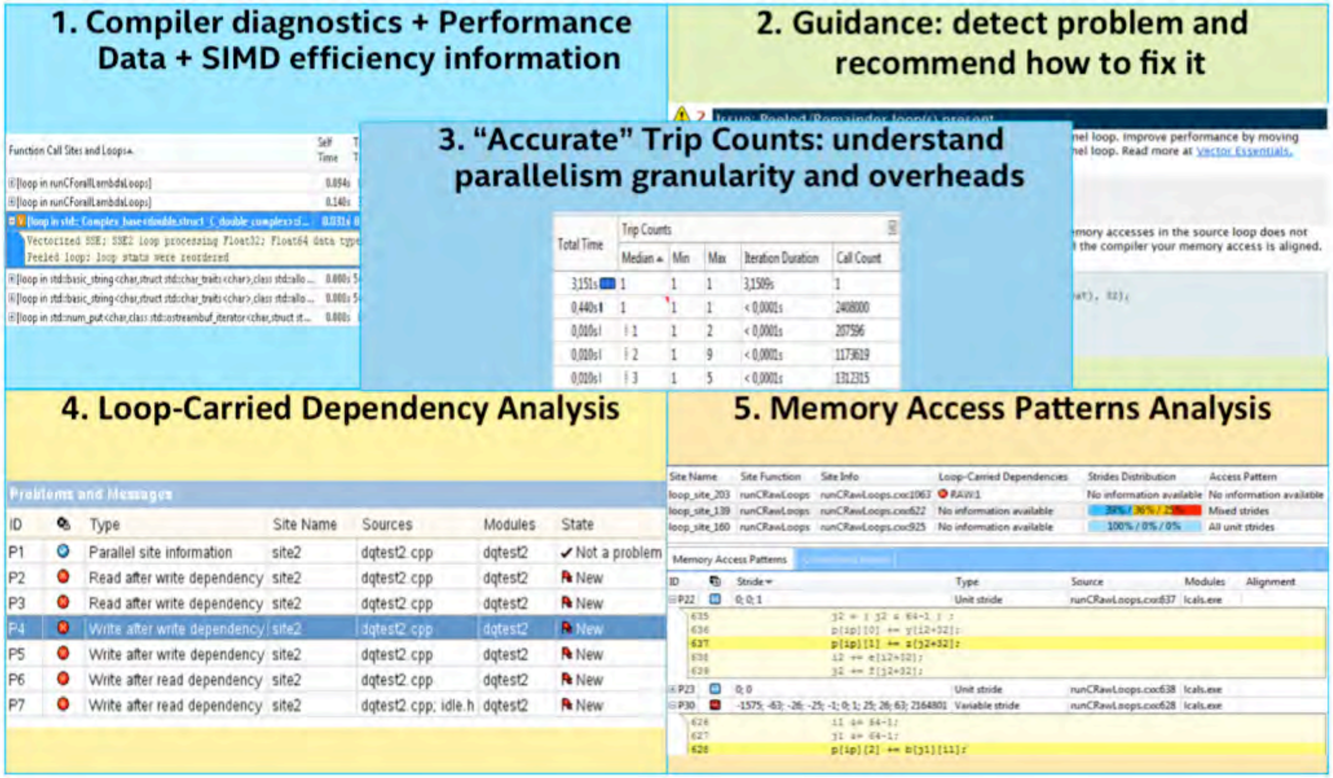
Loops		Vector Issues	Loop Type	Vecto...	Efficiency	Est...	Ve...	Co...	Why No Vectorization?	Self Time
[loop in fsOutputVTK3D ...	□		Scalar						inner loop throttling preven...	0,000s
[loop in fGetSpeedSite at...	□	1 Data type conv...	Scalar						loop control variable was fo...	0,620s
[loop in fGetEquilibrium...	□	1 Data type conv...	Vectorized Expand	AVX	~56%	2,24	4	2,24		0,550s
[loop in fPropagationSw...	☑	1 Data type conv...	Scalar							0,050s
[loop in fGetOneMassSit...	□	1 Ineffective peel...	Vectorized Collapse	AVX	[illegible]	2,79	8	2,79		0,030s
[loop in fGetOneMass...	□		Peeled							0,020s
[loop in fGetOneMass...	□		Vectorized (Body)	AVX			8	2,79		0,010s
[loop in fGetOneMass...	□		Remainder							n/a
[loop in fCollisionBGK a...	☑	**1 Data type con...**	**Scalar**						**loop with function call not...**	**0,020s**
[loop in fCollisionBGK at...	□	1 Ineffective peel...	Vectorized Expand	AVX	~100%	2,05	2	2,05		0,020s
[loop in fPropagationSw...	□	1 High vector reg...	Vectorized (Body)	AVX	~65%	2,59	4	2,59		0,010s
[loop in fGetOneDirecSp...	□	1 Data type conv...	Vectorized Expand	AVX	[illegible]	2,97	8	2,97		0,010s

Line	Source	Total Time	%	Loo...	%	Traits

```
      [Scalar loop in fCollisionBGK at lbpBGK.cpp:838]
        Loop with instructions using AVX registers. Not vectorized: loop with function cal
        No loop transformations were applied
837        i1 = (i * Ymax + j) * Zmax + k;
838        fGetSpeedSite(sitespeed, &lbf[i1*lbsitelength]);                    0,640s
839        if(lbphi[i1] != 11) {
840          for(l=0; l<3*lbsy.nf; l++)                                        [illegible]   [illegible]
      [Vectorized (Remainder) loop in fCollisionBGK at lbpBGK.cpp:840]
        Vectorized AVX Remainder Loop processing float64 data type(s)
        No loop transformations were applied
      [Not executed loop in fCollisionBGK at lbpBGK.cpp:840]
        Vectorized AVX Loop processing float64 data type(s) having Inserts operations
        Loop was unrolled by 4
      [Not executed loop in fCollisionBGK at lbpBGK.cpp:840]
        Scalar Remainder Loop. Not vectorized
        No loop transformations were applied
841          interforce [l1] = lbinterforce[i1*3*lbsy.nf+l1] + posteqil*lbbdforce[l1    0,020s   Inserts; Type C...
842        if(lbphi[i1] != 12 && lbphi[i1] != 13)                              0,010s
843          fSiteFluidCollisionBGK(&lbf[i1*lbsitelength], sitespeed, interforce);   1,072s
```

Vectorization Advisor.

Assist code modernization for x86 SIMD



1. Compiler diagnostics + Performance Data + SIMD efficiency information

Function Call Sites and Loops	Self Time
[loop in runCForallLambdaLoops]	0.094s
[loop in runCForallLambdaLoops]	0.140s
[loop in std::Complex_base<double,struct _C_double_complex>::...	0.031s
Vectorized SSE; SSE2 loop processing Float32; Float64 data type Peeled loop; loop stats were reordered	
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s

2. Guidance: detect problem and recommend how to fix it

Issue: Peeled/Remainder loop(s) present

...nel loop. Improve performance by moving ...nel loop. Read more at Vector Essentials.

...mory accesses in the source loop does not ... the compiler your memory access is aligned.

3. "Accurate" Trip Counts: understand parallelism granularity and overheads

Total Time	Trip Counts				
	Median	Min	Max	Iteration Duration	Call Count
3,151s	1	1	1	3,1509s	1
0,440s	1	1	1	< 0,0001s	2408000
0,010s	1	1	2	< 0,0001s	207596
0,010s	2	1	9	< 0,0001s	1173619
0,010s	3	1	5	< 0,0001s	1312315

4. Loop-Carried Dependency Analysis

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1	●	Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2	●	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3	●	Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4	●	Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5	●	Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6	●	Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7	●	Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

5. Memory Access Patterns Analysis

Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	● RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	38% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns

ID		Stride	Type	Source	Modules	Alignment
P22	■	0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	
P23	■	0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30	■	-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	

```
635    j2 = i j2 < 64-1 ; ;
636    p[ip][0] += y[i2+32];
637    p[ip][1] += z[j2+32];
638    i2 += e[i2+32];
639    j2 += f[j2+32];
```

```
626    i1 &= 64-1;
627    j1 &= 64-1;
628    p[ip][2] += b[j1][i1];
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Summary

We need to embrace explicit vectorization in our programming.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Summary

We need to embrace explicit vectorization in our programming.

But, generally use parallelism first (tasks, threads, MPI, etc.)

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

KEEP CALM AND LOOK FOR WHAT YOU DO NOT SEE

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

[redacted]

It is hard to “see” if you do not look.

It is hard to “see” if you do not look.

We could guess,
after all – we are smart enough
to *believe* we know what is happening.

Look for:

?

Look for:

- Confirmation

Look for:

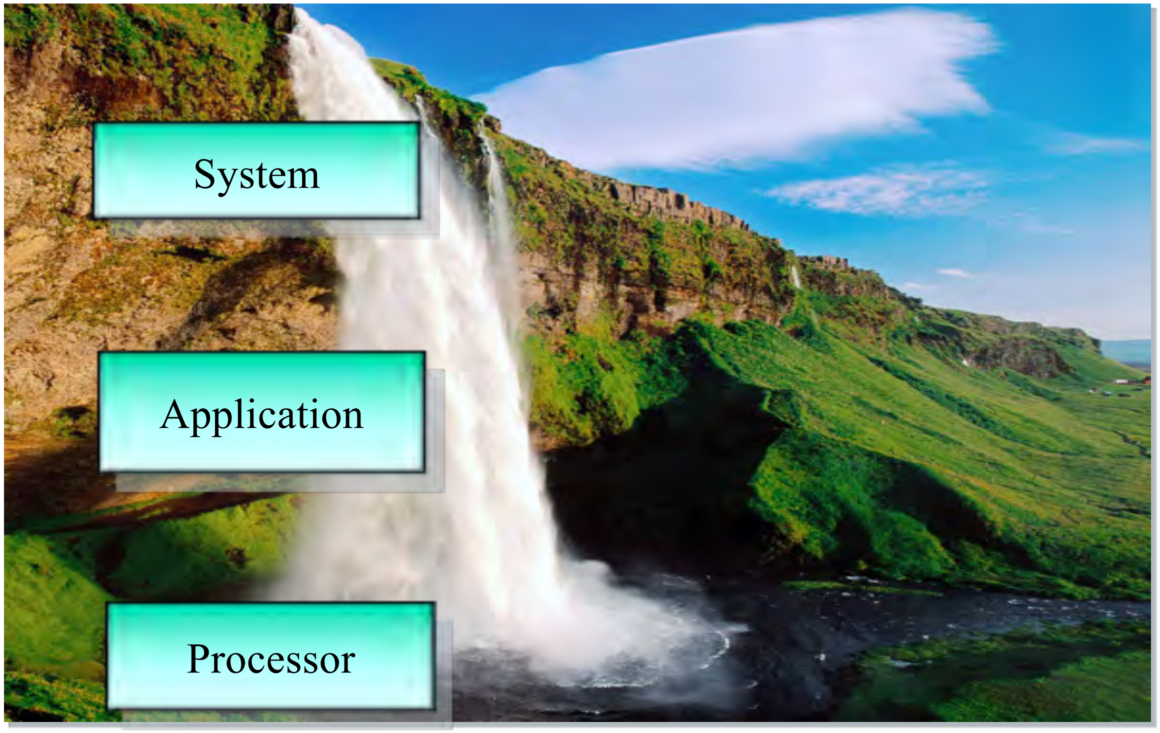
- Confirmation
- Surprises

Look for:

- Confirmation
- Surprises

Your EXPERTISE will grow as you investigate.

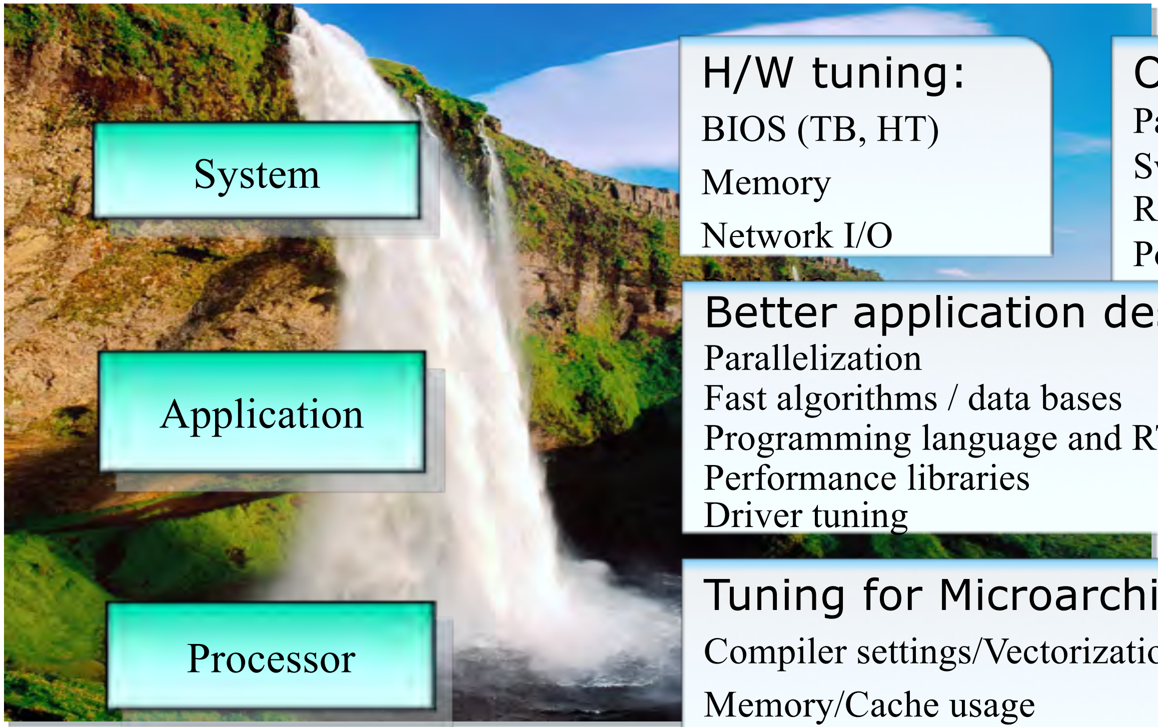
Optimization: A Top-down Approach



https://software.intel.com/en-us/articles/de-mystifying-software-performance-optimization

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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: A Top-down Approach



System

Application

Processor

H/W tuning:
BIOS (TB, HT)
Memory
Network I/O

OS tuning:
Page size
Swap file
RAM Disk
Power settings

Better application design:
Parallelization
Fast algorithms / data bases
Programming language and RT libs
Performance libraries
Driver tuning

Tuning for Microarchitecture:
Compiler settings/Vectorization
Memory/Cache usage
CPU pitfalls

OS, System Expertise SW/uArch

https://software.intel.com/en-us/articles/de-mystifying-software-performance-optimization

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Who: Software Developers, Performance Engineers, Domain Experts

How:

- Workload selection
  - Repeatable results
  - Steady stat
- Define Metrics and Collect Baseline
  - Wall-clock time, FLOPS, FPS
  - <insert your metric here>
- Identify Hotspots
  - Focus effort where it counts
  - Use Tools
- Determine inefficiencies
  - Is there parallelism?
  - Are you memory bound?
  - Will better algorithms or programming languages help?

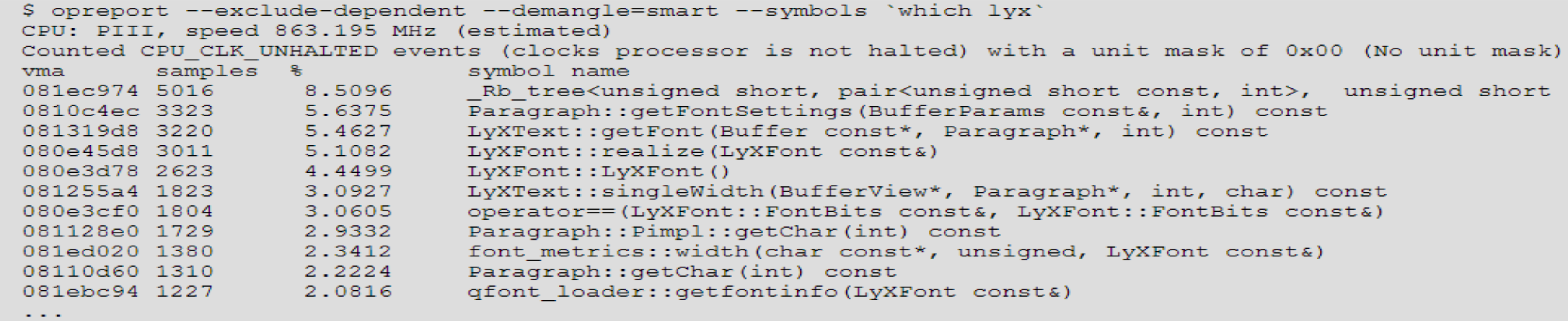
This step often requires some knowledge of the application and its algorithms

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity



```
$ opreport --exclude-dependent --demangle=smart --symbols `which lyx`
CPU: PIII, speed 863.195 MHz (estimated)
Counted CPU_CLK_UNHALTED events (clocks processor is not halted) with a unit mask of 0x00 (No unit mask)
vma      samples  %          symbol name
081ec974 5016      8.5096    _Rb_tree<unsigned short, pair<unsigned short const, int>,  unsigned short
0810c4ec 3323      5.6375    Paragraph::getFontSettings(BufferParams const&, int) const
081319d8 3220      5.4627    LyXText::getFont(Buffer const*, Paragraph*, int) const
080e45d8 3011      5.1082    LyXFont::realize(LyXFont const&)
080e3d78 2623      4.4499    LyXFont::LyXFont()
081255a4 1823      3.0927    LyXText::singleWidth(BufferView*, Paragraph*, int, char) const
080e3cf0 1804      3.0605    operator==(LyXFont::FontBits const&, LyXFont::FontBits const&)
081128e0 1729      2.9332    Paragraph::Pimpl::getChar(int) const
081ed020 1380      2.3412    font_metrics::width(char const*, unsigned, LyXFont const&)
08110d60 1310      2.2224    Paragraph::getChar(int) const
081ebc94 1227      2.0816    qfont_loader::getfontinfo(LyXFont const&)
...
```

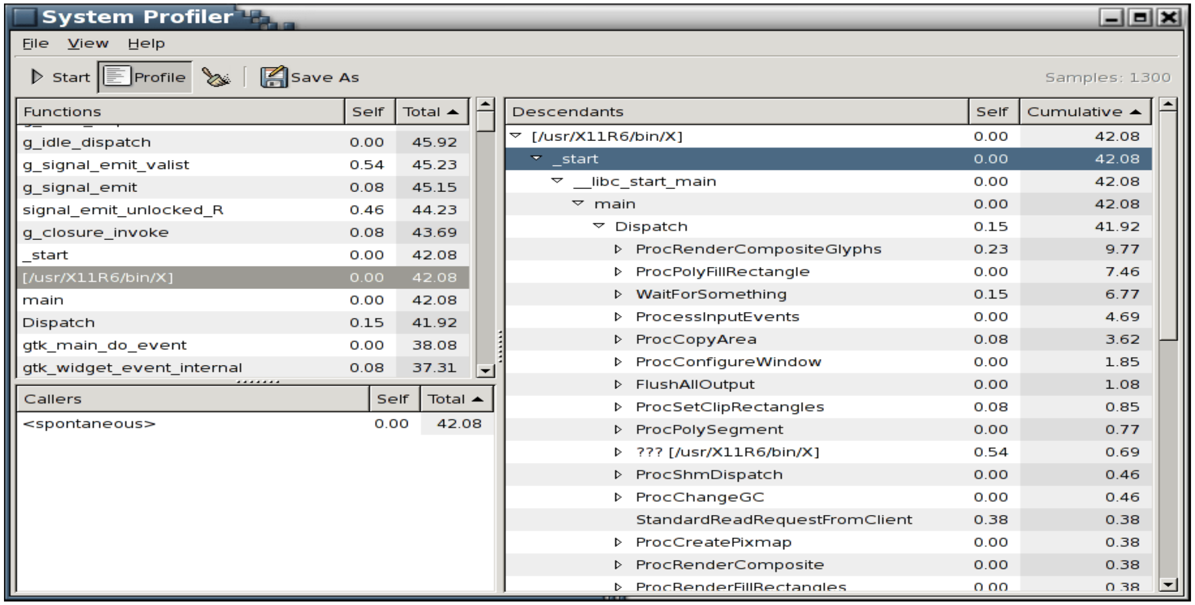
oprofile: http://oprofile.sourceforge.net/

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity



Functions	Self	Total
g_idle_dispatch	0.00	45.92
g_signal_emit_valist	0.54	45.23
g_signal_emit	0.08	45.15
signal_emit_unlocked_R	0.46	44.23
g_closure_invoke	0.08	43.69
_start	0.00	42.08
[/usr/X11R6/bin/X]	0.00	42.08
main	0.00	42.08
Dispatch	0.15	41.92
gtk_main_do_event	0.00	38.08
gtk_widget_event_internal	0.08	37.31

Callers	Self	Total
<spontaneous>	0.00	42.08

Descendants	Self	Cumulative
[/usr/X11R6/bin/X]	0.00	42.08
_start	0.00	42.08
__libc_start_main	0.00	42.08
main	0.00	42.08
Dispatch	0.15	41.92
ProcRenderCompositeGlyphs	0.23	9.77
ProcPolyFillRectangle	0.00	7.46
WaitForSomething	0.15	6.77
ProcessInputEvents	0.00	4.69
ProcCopyArea	0.08	3.62
ProcConfigureWindow	0.00	1.85
FlushAllOutput	0.00	1.08
ProcSetClipRectangles	0.08	0.85
ProcPolySegment	0.00	0.77
??? [/usr/X11R6/bin/X]	0.54	0.69
ProcShmDispatch	0.00	0.46
ProcChangeGC	0.00	0.46
StandardReadRequestFromClient	0.38	0.38
ProcCreatePixmap	0.00	0.38
ProcRenderComposite	0.00	0.38
ProcRenderFillRectangles	0.00	0.38

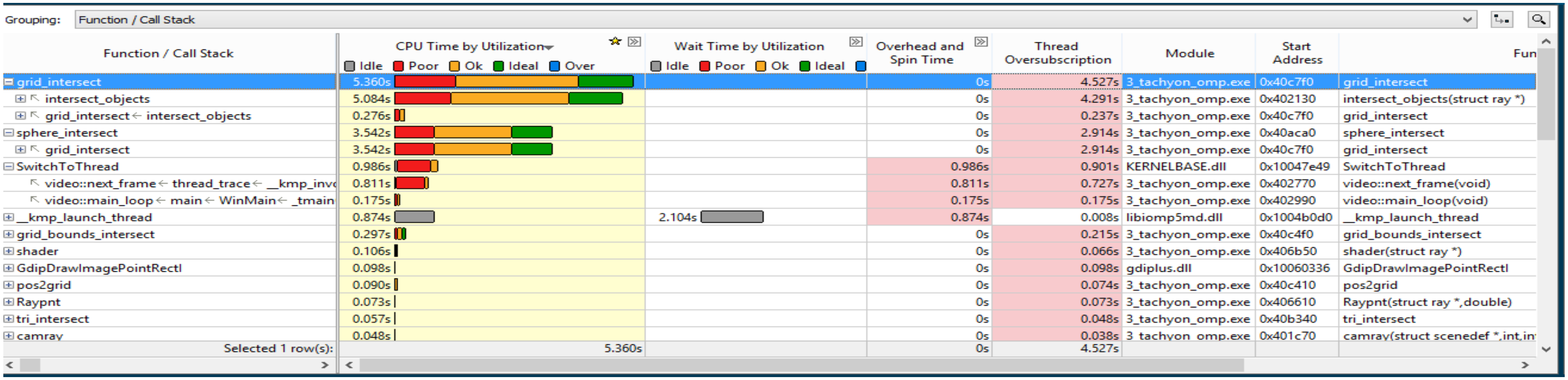
sysprof: http://sysprof.com

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity



Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization	Wait Time by Utilization	Overhead and Spin Time	Thread Oversubscription	Module	Start Address	Fun
	■ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	■ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦					
⊟ grid_intersect	5.360s		0s	4.527s	3_tachyon_omp.exe	0x40c7f0	grid_intersect
⊞ ↖ intersect_objects	5.084s		0s	4.291s	3_tachyon_omp.exe	0x402130	intersect_objects(struct ray *)
⊞ ↖ grid_intersect ← intersect_objects	0.276s		0s	0.237s	3_tachyon_omp.exe	0x40c7f0	grid_intersect
⊟ sphere_intersect	3.542s		0s	2.914s	3_tachyon_omp.exe	0x40aca0	sphere_intersect
⊞ ↖ grid_intersect	3.542s		0s	2.914s	3_tachyon_omp.exe	0x40c7f0	grid_intersect
⊟ SwitchToThread	0.986s		0.986s	0.901s	KERNELBASE.dll	0x10047e49	SwitchToThread
↖ video::next_frame ← thread_trace ← __kmp_inv	0.811s		0.811s	0.727s	3_tachyon_omp.exe	0x402770	video::next_frame(void)
↖ video::main_loop ← main ← WinMain ← _tmain	0.175s		0.175s	0.175s	3_tachyon_omp.exe	0x402990	video::main_loop(void)
⊞ __kmp_launch_thread	0.874s	2.104s	0.874s	0.008s	libiomp5md.dll	0x1004b0d0	__kmp_launch_thread
⊞ grid_bounds_intersect	0.297s		0s	0.215s	3_tachyon_omp.exe	0x40c4f0	grid_bounds_intersect
⊞ shader	0.106s		0s	0.066s	3_tachyon_omp.exe	0x406b50	shader(struct ray *)
⊞ GdipDrawImagePointRectI	0.098s		0s	0.098s	gdiplus.dll	0x10060336	GdipDrawImagePointRectI
⊞ pos2grid	0.090s		0s	0.074s	3_tachyon_omp.exe	0x40c410	pos2grid
⊞ Raypnt	0.073s		0s	0.073s	3_tachyon_omp.exe	0x406610	Raypnt(struct ray *,double)
⊞ tri_intersect	0.057s		0s	0.048s	3_tachyon_omp.exe	0x40b340	tri_intersect
⊞ camray	0.048s		0s	0.038s	3_tachyon_omp.exe	0x401c70	camray(struct scenedef *,int,in
Selected 1 row(s):	5.360s		0s	4.527s			

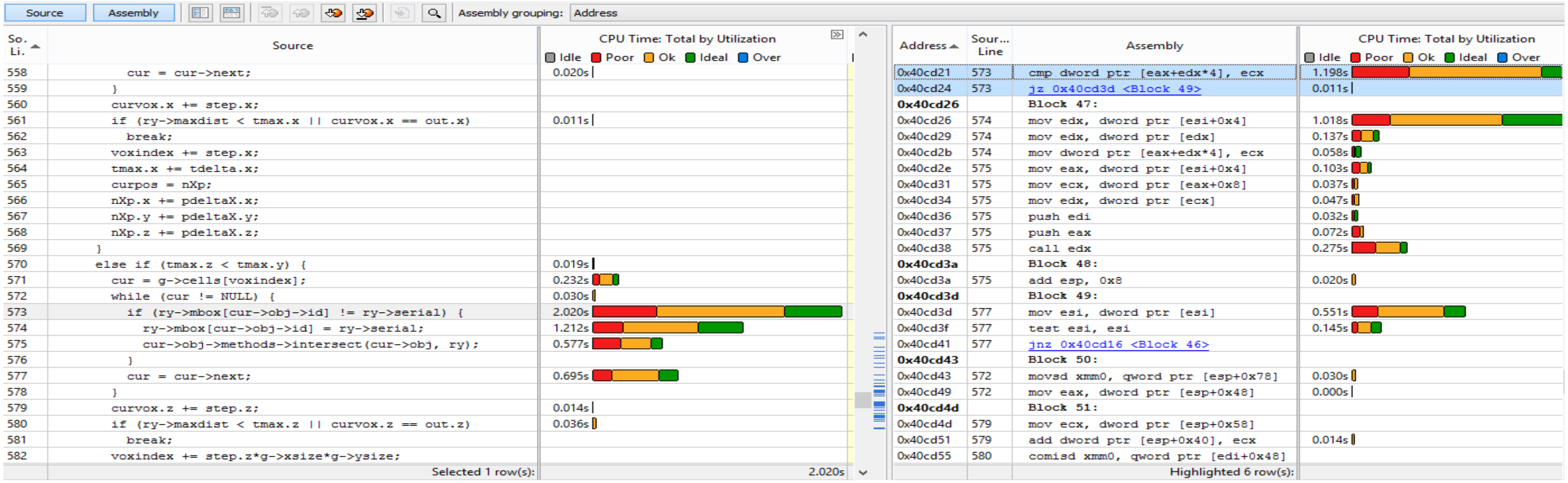
Intel® VTune™ Amplifier XE: http://intel.ly/vtune-amplifier-xe

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity



Assembly grouping: Address

So. Li.	Source	CPU Time: Total by Utilization Idle Poor Ok Ideal Over
558	cur = cur->next;	0.020s
559	}	
560	curvox.x += step.x;	
561	if (ry->maxdist < tmax.x || curvox.x == out.x)	0.011s
562	break;	
563	voxindex += step.x;	
564	tmax.x += tdelta.x;	
565	curpos = nXp;	
566	nXp.x += pdeltaX.x;	
567	nXp.y += pdeltaX.y;	
568	nXp.z += pdeltaX.z;	
569	}	
570	else if (tmax.z < tmax.y) {	0.019s
571	cur = g->cells[voxindex];	0.232s
572	while (cur != NULL) {	0.030s
573	if (ry->mbox[cur->obj->id] != ry->serial) {	2.020s
574	ry->mbox[cur->obj->id] = ry->serial;	1.212s
575	cur->obj->methods->intersect(cur->obj, ry);	0.577s
576	}	
577	cur = cur->next;	0.695s
578	}	
579	curvox.z += step.z;	0.014s
580	if (ry->maxdist < tmax.z || curvox.z == out.z)	0.036s
581	break;	
582	voxindex += step.z*g->xsize*g->ysize;	
	Selected 1 row(s):	2.020s

Address	Sour... Line	Assembly	CPU Time: Total by Utilization Idle Poor Ok Ideal Over
0x40cd21	573	cmp dword ptr [eax+edx*4], ecx	1.198s
0x40cd24	573	jz 0x40cd3d <Block 49>	0.011s
0x40cd26		Block 47:	
0x40cd26	574	mov edx, dword ptr [esi+0x4]	1.018s
0x40cd29	574	mov edx, dword ptr [edx]	0.137s
0x40cd2b	574	mov dword ptr [eax+edx*4], ecx	0.058s
0x40cd2e	575	mov eax, dword ptr [esi+0x4]	0.103s
0x40cd31	575	mov ecx, dword ptr [eax+0x8]	0.037s
0x40cd34	575	mov edx, dword ptr [ecx]	0.047s
0x40cd36	575	push edi	0.032s
0x40cd37	575	push eax	0.072s
0x40cd38	575	call edx	0.275s
0x40cd3a		Block 48:	
0x40cd3a	575	add esp, 0x8	0.020s
0x40cd3d		Block 49:	
0x40cd3d	577	mov esi, dword ptr [esi]	0.551s
0x40cd3f	577	test esi, esi	0.145s
0x40cd41	577	jnz 0x40cd16 <Block 46>	
0x40cd43		Block 50:	
0x40cd43	572	movsd xmm0, qword ptr [esp+0x78]	0.030s
0x40cd49	572	mov eax, dword ptr [esp+0x48]	0.000s
0x40cd4d		Block 51:	
0x40cd4d	579	mov ecx, dword ptr [esp+0x58]	
0x40cd51	579	add dword ptr [esp+0x40], ecx	0.014s
0x40cd55	580	comisd xmm0, qword ptr [edi+0x48]	
		Highlighted 6 row(s):	

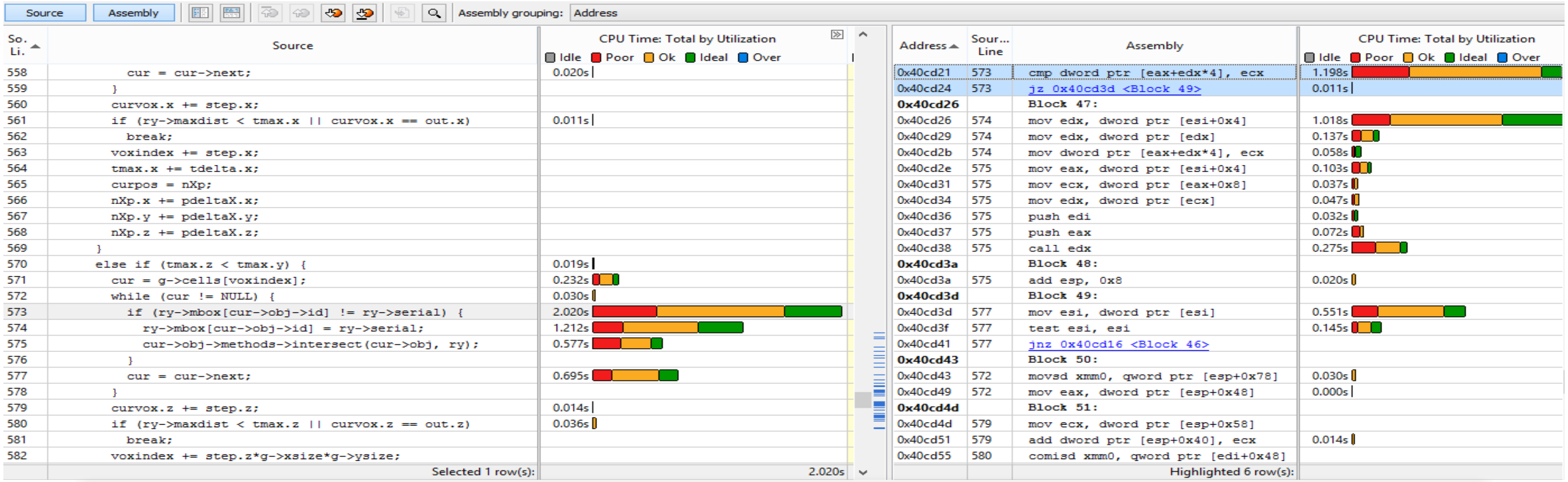
Intel® VTune™ Amplifier XE: http://intel.ly/vtune-amplifier-xe

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity



So. Li.	Source	CPU Time: Total by Utilization Idle / Poor / Ok / Ideal / Over
558	`cur = cur->next;`	0.020s
559	`}`	
560	`curvox.x += step.x;`	
561	`if (ry->maxdist < tmax.x || curvox.x == out.x)`	0.011s
562	`break;`	
563	`voxindex += step.x;`	
564	`tmax.x += tdelta.x;`	
565	`curpos = nXp;`	
566	`nXp.x += pdeltaX.x;`	
567	`nXp.y += pdeltaX.y;`	
568	`nXp.z += pdeltaX.z;`	
569	`}`	
570	`else if (tmax.z < tmax.y) {`	0.019s
571	`cur = g->cells[voxindex];`	0.232s
572	`while (cur != NULL) {`	0.030s
573	`if (ry->mbox[cur->obj->id] != ry->serial) {`	2.020s
574	`ry->mbox[cur->obj->id] = ry->serial;`	1.212s
575	`cur->obj->methods->intersect(cur->obj, ry);`	0.577s
576	`}`	
577	`cur = cur->next;`	0.695s
578	`}`	
579	`curvox.z += step.z;`	0.014s
580	`if (ry->maxdist < tmax.z || curvox.z == out.z)`	0.036s
581	`break;`	
582	`voxindex += step.z*g->xsize*g->ysize;`	
	Selected 1 row(s):	2.020s

Address	Sour... Line	Assembly	CPU Time: Total by Utilization Idle / Poor / Ok / Ideal / Over
0x40cd21	573	`cmp dword ptr [eax+edx*4], ecx`	1.198s
0x40cd24	573	`jz 0x40cd3d <Block 49>`	0.011s
0x40cd26		`Block 47:`	
0x40cd26	574	`mov edx, dword ptr [esi+0x4]`	1.018s
0x40cd29	574	`mov edx, dword ptr [edx]`	0.137s
0x40cd2b	574	`mov dword ptr [eax+edx*4], ecx`	0.058s
0x40cd2e	575	`mov eax, dword ptr [esi+0x4]`	0.103s
0x40cd31	575	`mov ecx, dword ptr [eax+0x8]`	0.037s
0x40cd34	575	`mov edx, dword ptr [ecx]`	0.047s
0x40cd36	575	`push edi`	0.032s
0x40cd37	575	`push eax`	0.072s
0x40cd38	575	`call edx`	0.275s
0x40cd3a		`Block 48:`	
0x40cd3a	575	`add esp, 0x8`	0.020s
0x40cd3d		`Block 49:`	
0x40cd3d	577	`mov esi, dword ptr [esi]`	0.551s
0x40cd3f	577	`test esi, esi`	0.145s
0x40cd41	577	`jnz 0x40cd16 <Block 46>`	
0x40cd43		`Block 50:`	
0x40cd43	572	`movsd xmm0, qword ptr [esp+0x78]`	0.030s
0x40cd49	572	`mov eax, dword ptr [esp+0x48]`	0.000s
0x40cd4d		`Block 51:`	
0x40cd4d	579	`mov ecx, dword ptr [esp+0x58]`	
0x40cd51	579	`add dword ptr [esp+0x40], ecx`	0.014s
0x40cd55	580	`comisd xmm0, qword ptr [edi+0x48]`	
		Highlighted 6 row(s):	

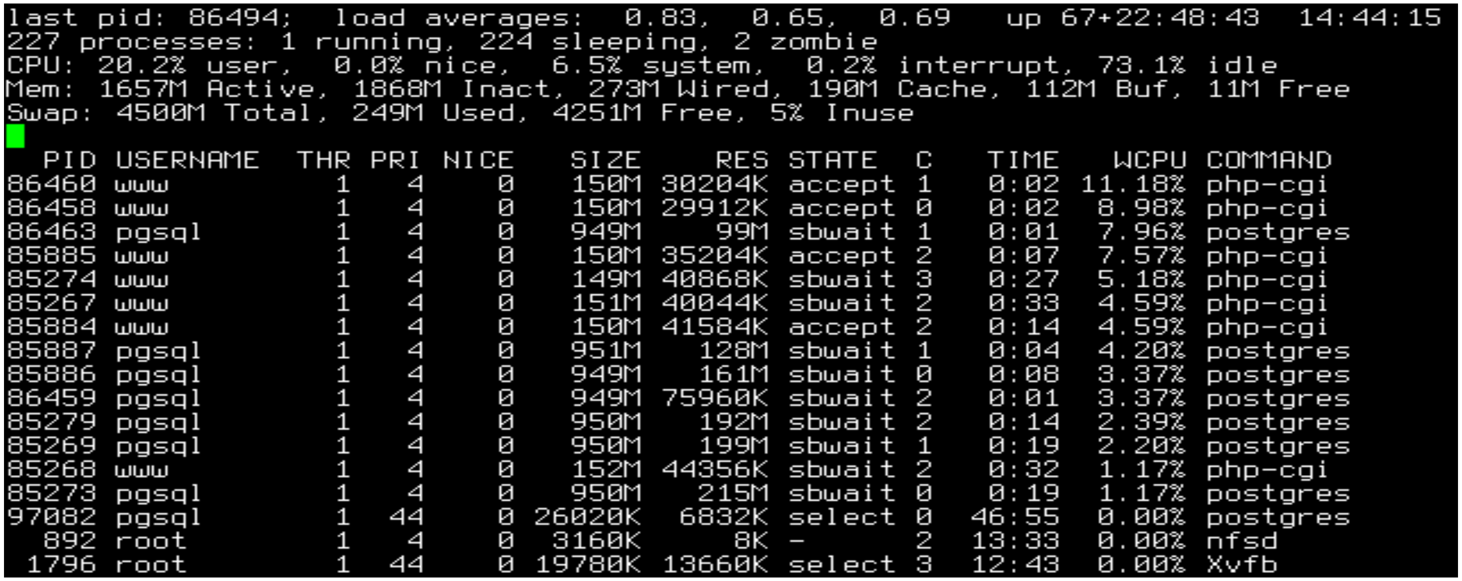
This may reinforce your understanding of the application but often reveals surprises

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Resource Utilization

- Is the application parallel?
- Multi-thread vs. Multi-process
- Memory Bound?

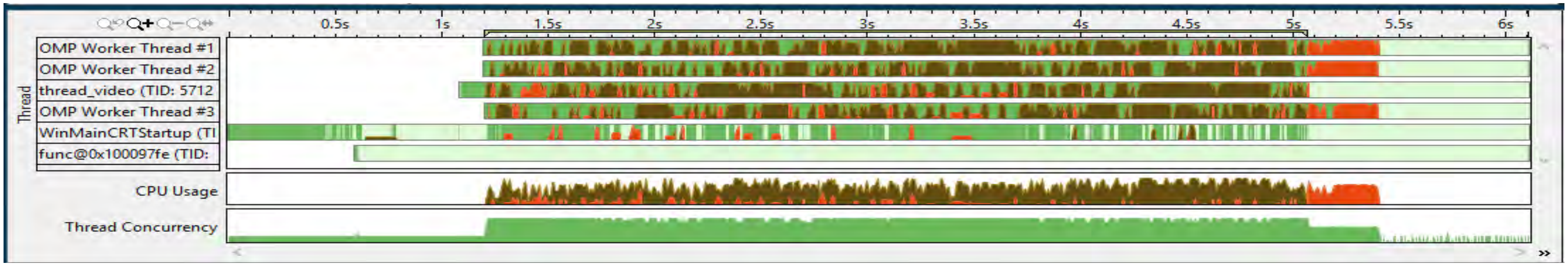


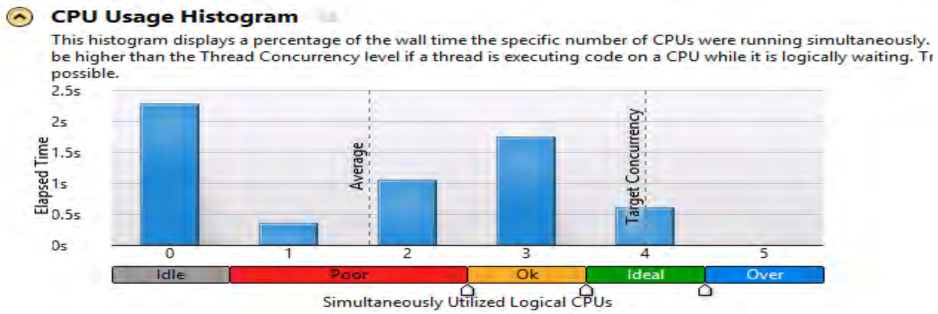
```
last pid: 86494;  load averages:  0.83,  0.65,  0.69    up 67+22:48:43  14:44:15
227 processes: 1 running, 224 sleeping, 2 zombie
CPU: 20.2% user,  0.0% nice,  6.5% system,  0.2% interrupt, 73.1% idle
Mem: 1657M Active, 1868M Inact, 273M Wired, 190M Cache, 112M Buf, 11M Free
Swap: 4500M Total, 249M Used, 4251M Free, 5% Inuse

  PID USERNAME  THR PRI NICE   SIZE    RES STATE  C   TIME   WCPU COMMAND
86460 www         1   4    0   150M 30204K accept 1   0:02 11.18% php-cgi
86458 www         1   4    0   150M 29912K accept 0   0:02  8.98% php-cgi
86463 pgsql       1   4    0   949M    99M sbwait 1   0:01  7.96% postgres
85885 www         1   4    0   150M 35204K accept 2   0:07  7.57% php-cgi
85274 www         1   4    0   149M 40868K sbwait 3   0:27  5.18% php-cgi
85267 www         1   4    0   151M 40044K sbwait 2   0:33  4.59% php-cgi
85884 www         1   4    0   150M 41584K accept 2   0:14  4.59% php-cgi
85887 pgsql       1   4    0   951M   128M sbwait 1   0:04  4.20% postgres
85886 pgsql       1   4    0   949M   161M sbwait 0   0:08  3.37% postgres
86459 pgsql       1   4    0   949M 75960K sbwait 2   0:01  3.37% postgres
85279 pgsql       1   4    0   950M   192M sbwait 2   0:14  2.39% postgres
85269 pgsql       1   4    0   950M   199M sbwait 1   0:19  2.20% postgres
85268 www         1   4    0   152M 44356K sbwait 2   0:32  1.17% php-cgi
85273 pgsql       1   4    0   950M   215M sbwait 0   0:19  1.17% postgres
97082 pgsql       1  44    0 26020K  6832K select 0  46:55  0.00% postgres
  892 root        1   4    0  3160K     8K -      2  13:33  0.00% nfsd
 1796 root        1  44    0 19780K 13660K select 3  12:43  0.00% Xvfb
```

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

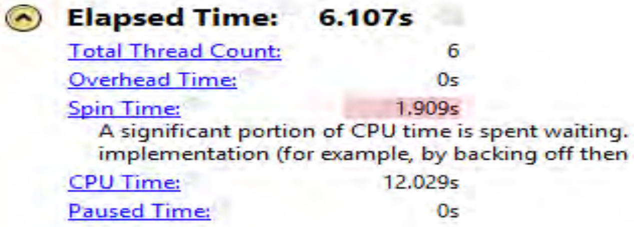
Application Tuning
Resource Utilization

- Is the application parallel?




CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. be higher than the Thread Concurrency level if a thread is executing code on a CPU while it is logically waiting. Tr possible.


Elapsed Time: 6.107s

Total Thread Count: 6

Overhead Time: 0s

Spin Time: 1.909s

A significant portion of CPU time is spent waiting. implementation (for example, by backing off then

CPU Time: 12.029s

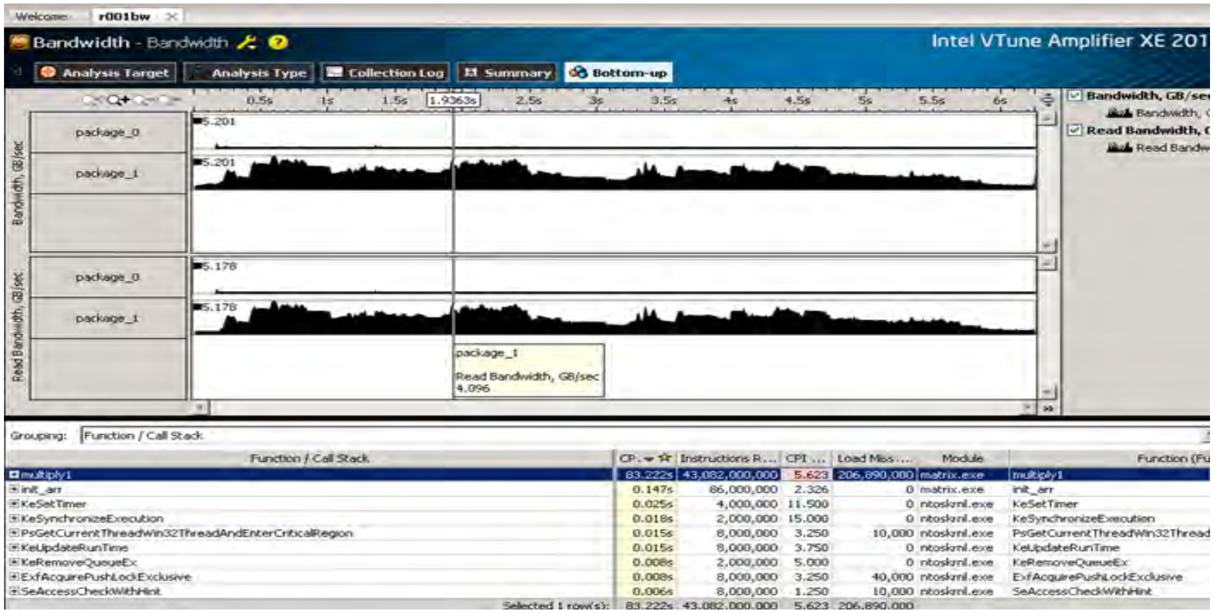
Paused Time: 0s

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Resource Utilization

- Memory Bound?



Grouping: Function / Call Stack

Function / Call Stack	CP... ★	Instructions R...	CPI ...	Load Miss ...	Module	Function (Fu
⊞ multiply1	83.222s	43,082,000,000	5.623	206,890,000	matrix.exe	multiply1
⊞ init_arr	0.147s	86,000,000	2.326	0	matrix.exe	init_arr
⊞ KeSetTimer	0.025s	4,000,000	11.500	0	ntoskrnl.exe	KeSetTimer
⊞ KeSynchronizeExecution	0.018s	2,000,000	15.000	0	ntoskrnl.exe	KeSynchronizeExecution
⊞ PsGetCurrentThreadWin32ThreadAndEnterCriticalRegion	0.015s	8,000,000	3.250	10,000	ntoskrnl.exe	PsGetCurrentThreadWin32Thread
⊞ KeUpdateRunTime	0.015s	8,000,000	3.750	0	ntoskrnl.exe	KeUpdateRunTime
⊞ KeRemoveQueueEx	0.008s	2,000,000	5.000	0	ntoskrnl.exe	KeRemoveQueueEx
⊞ ExfAcquirePushLockExclusive	0.008s	8,000,000	3.250	40,000	ntoskrnl.exe	ExfAcquirePushLockExclusive
⊞ SeAccessCheckWithHint	0.006s	8,000,000	1.250	10,000	ntoskrnl.exe	SeAccessCheckWithHint
Selected 1 row(s):	83.222s	43,082,000,000	5.623	206,890,000		

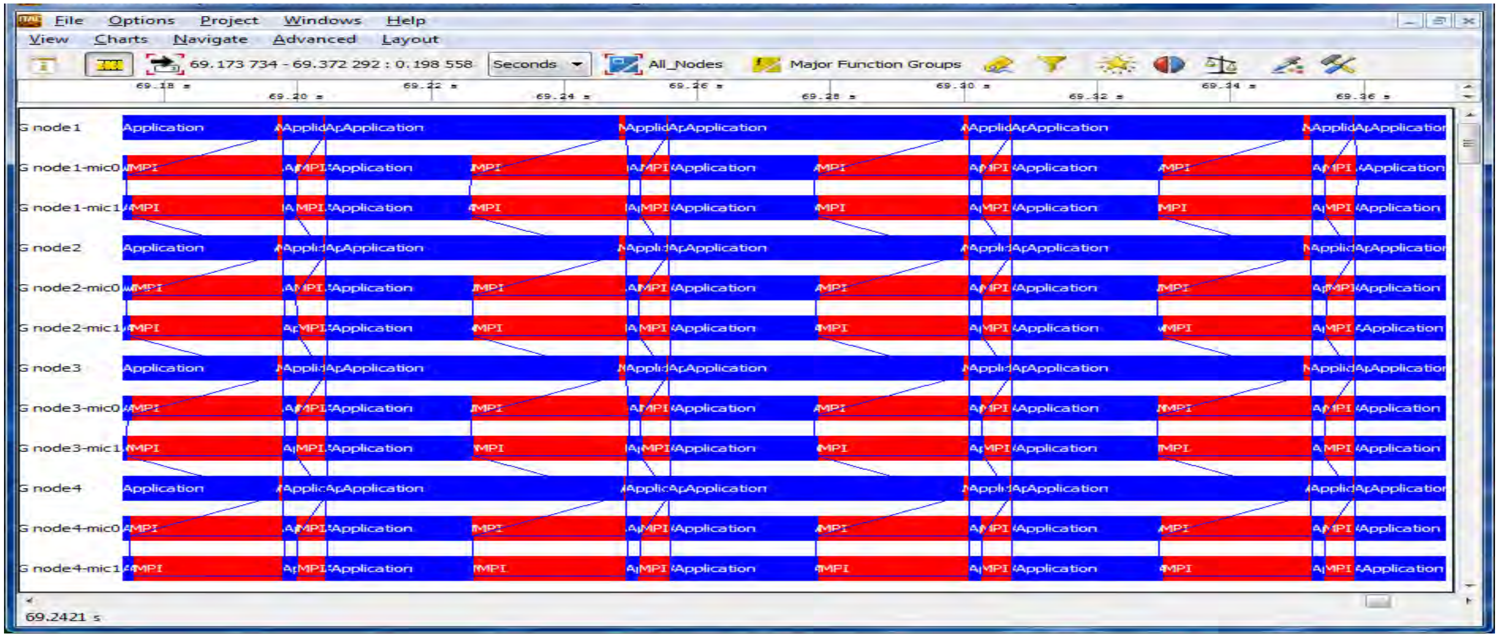
- Know your max theoretical memory bandwidth

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

Resource Utilization

MPI applications have added communication complexity



Intel® Trace Analyzer and Collector: http://intel.ly/traceanalyzer-collector

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Application Tuning

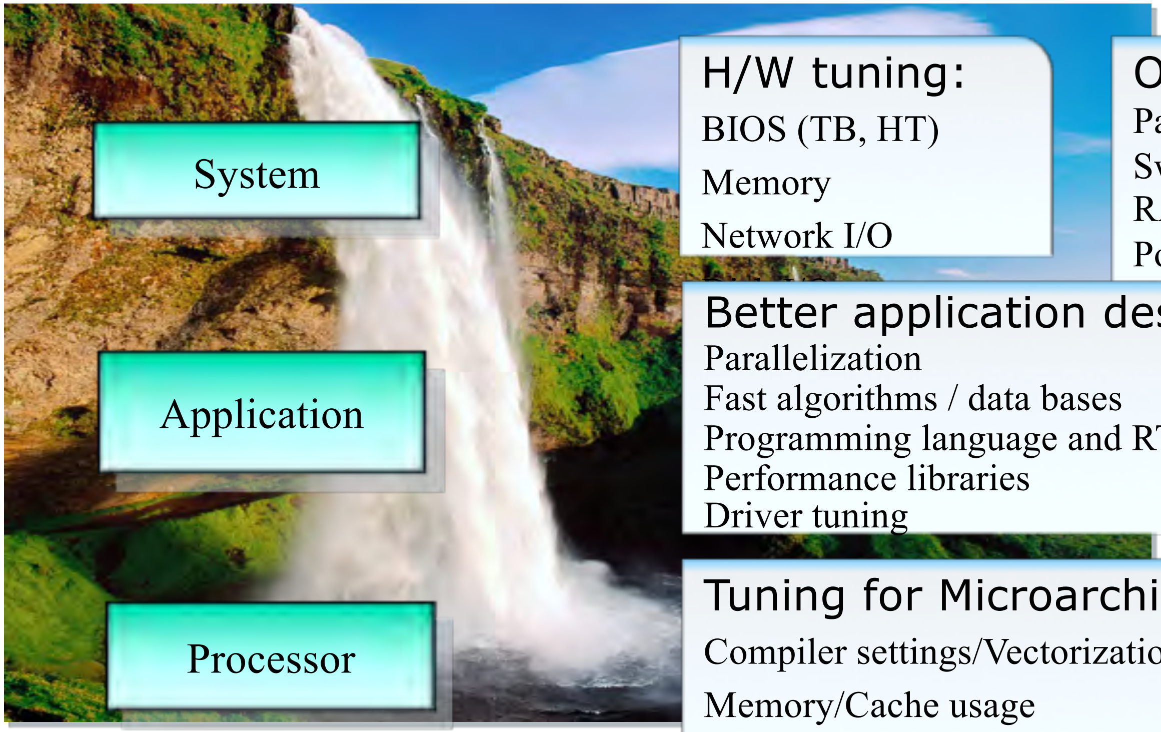
What's Next?

- If your Hotspots are common algorithms:
  - Look for optimized libraries
- If your Hotspots are uncommon:
  - Compiler optimizations
  - Expert analysis and refactoring of an algorithm
    - The opposite of "low-hanging fruit"
  - Deeper analysis of hardware performance
    - More on this later
- If the system is underutilized:
  - Add parallelism - multi-thread or multi-process
    - OpenMP, TBB, Cilk, MPI, etc…

➢ Tools can help you determine where to look and may identify some issues.
➢ Some tools may provide suggestions for fixes.
➢ In the end – the developer and/or expert has to make the changes and decisions – there is no silver bullet.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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: A Top-down Approach



System

Application

Processor

H/W tuning:

BIOS (TB, HT)

Memory

Network I/O

OS tuning:

Page size
Swap file
RAM Disk
Power settings

Better application design:

Parallelization
Fast algorithms / data bases
Programming language and RT libs
Performance libraries
Driver tuning

Tuning for Microarchitecture:

Compiler settings/Vectorization

Memory/Cache usage

CPU pitfalls

OS, System

Expertise

SW/uArch

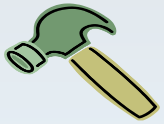
© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

Microarchitecture Tuning

Who: ~~Architecture Experts~~
Software Developers, Performance Engineers, Domain Experts

How:

- Use architecture specific hardware events
- Use predefined metrics and best known methods
  - Often hardware specific
  - (Hopefully) provided by the vendor
- Tools make this possible for the non-expert
  - Linux perf
  - Intel® VTune™ Amplifier XE
- Follow the Top-Down Characterization
  - Locate the hardware bottlenecks
  - Whitepaper here: https://software.intel.com/en-us/articles/how-to-tune-applications-using-a-top-down-characterization-of-microarchitectural-issues



© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Introduction to Performance Monitoring Unit (PMU)

Registers on Intel CPUs to count architectural events

- E.g. Instructions, Cache Misses, Branch Mispredict

Events can be counted or sampled

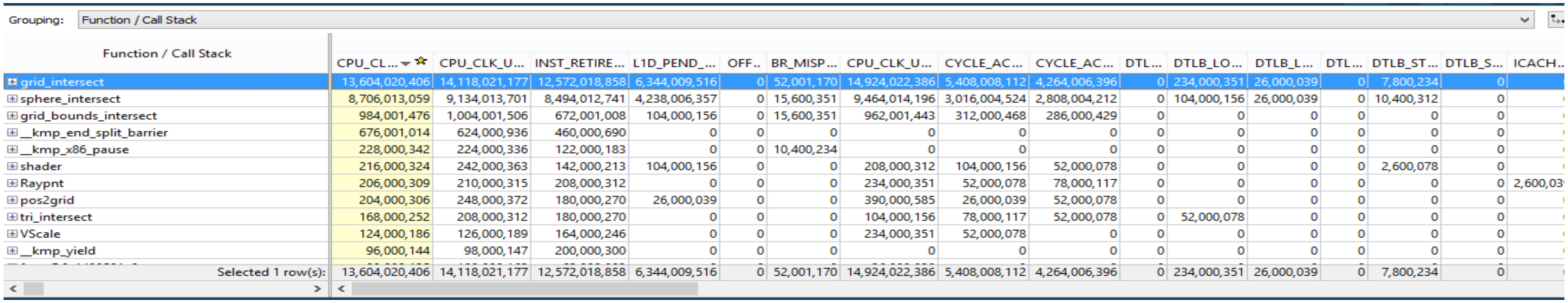
- Sampled events include Instruction Pointer

Raw event counts are difficult to interpret

- Use a tool like VTune or Perf with predefined metrics

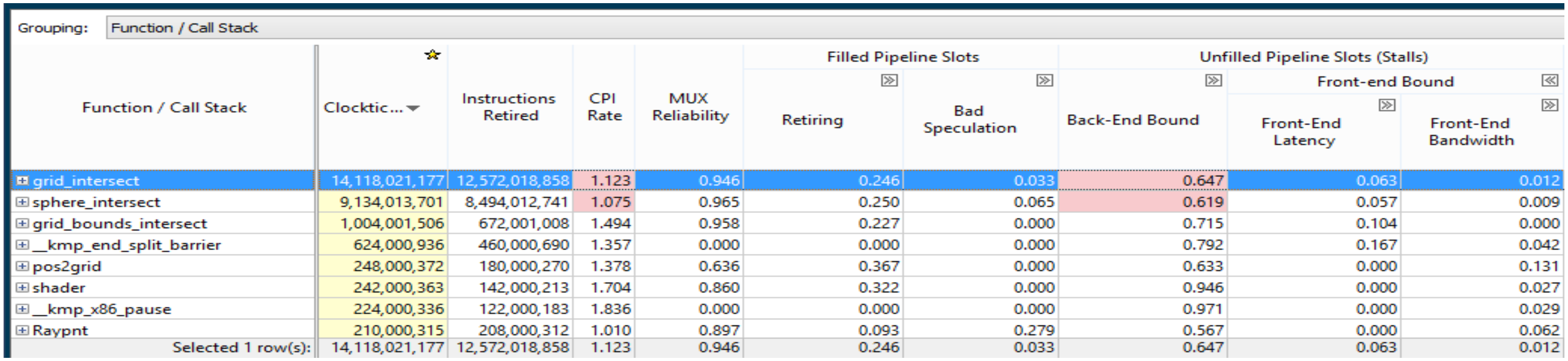
© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

Raw PMU Event Counts vs Metrics



Grouping: Function / Call Stack

Function / Call Stack	CPU_CL...	CPU_CLK_U...	INST_RETIRE...	L1D_PEND_...	OFF..	BR_MISP...	CPU_CLK_U...	CYCLE_AC...	CYCLE_AC...	DTL...	DTLB_LO...	DTLB_L...	DTL...	DTLB_ST...	DTLB_S...	ICACH..
⊞ grid_intersect	13,604,020,406	14,118,021,177	12,572,018,858	6,344,009,516	0	52,001,170	14,924,022,386	5,408,008,112	4,264,006,396	0	234,000,351	26,000,039	0	7,800,234	0	
⊞ sphere_intersect	8,706,013,059	9,134,013,701	8,494,012,741	4,238,006,357	0	15,600,351	9,464,014,196	3,016,004,524	2,808,004,212	0	104,000,156	26,000,039	0	10,400,312	0	
⊞ grid_bounds_intersect	984,001,476	1,004,001,506	672,001,008	104,000,156	0	15,600,351	962,001,443	312,000,468	286,000,429	0	0	0	0	0	0	
⊞ __kmp_end_split_barrier	676,001,014	624,000,936	460,000,690	0	0	0	0	0	0	0	0	0	0	0	0	
⊞ __kmp_x86_pause	228,000,342	224,000,336	122,000,183	0	0	10,400,234	0	0	0	0	0	0	0	0	0	
⊞ shader	216,000,324	242,000,363	142,000,213	104,000,156	0	0	208,000,312	104,000,156	52,000,078	0	0	0	0	2,600,078	0	
⊞ Raypnt	206,000,309	210,000,315	208,000,312	0	0	0	234,000,351	52,000,078	78,000,117	0	0	0	0	0	0	2,600,03
⊞ pos2grid	204,000,306	248,000,372	180,000,270	26,000,039	0	0	390,000,585	26,000,039	52,000,078	0	0	0	0	0	0	
⊞ tri_intersect	168,000,252	208,000,312	180,000,270	0	0	0	104,000,156	78,000,117	52,000,078	0	52,000,078	0	0	0	0	
⊞ VScale	124,000,186	126,000,189	164,000,246	0	0	0	234,000,351	52,000,078	0	0	0	0	0	0	0	
⊞ __kmp_yield	96,000,144	98,000,147	200,000,300	0	0	0	0	0	0	0	0	0	0	0	0	
Selected 1 row(s):	13,604,020,406	14,118,021,177	12,572,018,858	6,344,009,516	0	52,001,170	14,924,022,386	5,408,008,112	4,264,006,396	0	234,000,351	26,000,039	0	7,800,234	0	



Grouping: Function / Call Stack

Function / Call Stack	Clocktic...	Instructions Retired	CPI Rate	MUX Reliability	Filled Pipeline Slots		Unfilled Pipeline Slots (Stalls)		
								Front-end Bound	
					Retiring	Bad Speculation	Back-End Bound	Front-End Latency	Front-End Bandwidth
⊞ grid_intersect	14,118,021,177	12,572,018,858	1.123	0.946	0.246	0.033	0.647	0.063	0.012
⊞ sphere_intersect	9,134,013,701	8,494,012,741	1.075	0.965	0.250	0.065	0.619	0.057	0.009
⊞ grid_bounds_intersect	1,004,001,506	672,001,008	1.494	0.958	0.227	0.000	0.715	0.104	0.000
⊞ __kmp_end_split_barrier	624,000,936	460,000,690	1.357	0.000	0.000	0.000	0.792	0.167	0.042
⊞ pos2grid	248,000,372	180,000,270	1.378	0.636	0.367	0.000	0.633	0.000	0.131
⊞ shader	242,000,363	142,000,213	1.704	0.860	0.322	0.000	0.946	0.000	0.027
⊞ __kmp_x86_pause	224,000,336	122,000,183	1.836	0.000	0.000	0.000	0.971	0.000	0.029
⊞ Raypnt	210,000,315	208,000,312	1.010	0.897	0.093	0.279	0.567	0.000	0.062
Selected 1 row(s):	14,118,021,177	12,572,018,858	1.123	0.946	0.246	0.033	0.647	0.063	0.012

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Adding Regression Tests for Performance

Regression testing isn't just for bugs

1. Create a baseline performance characterization
2. After each change or at a regular interval
    1. Compare new results to baseline
    2. Compare new results to previous results
    3. Evaluate the change
3. goto (1)

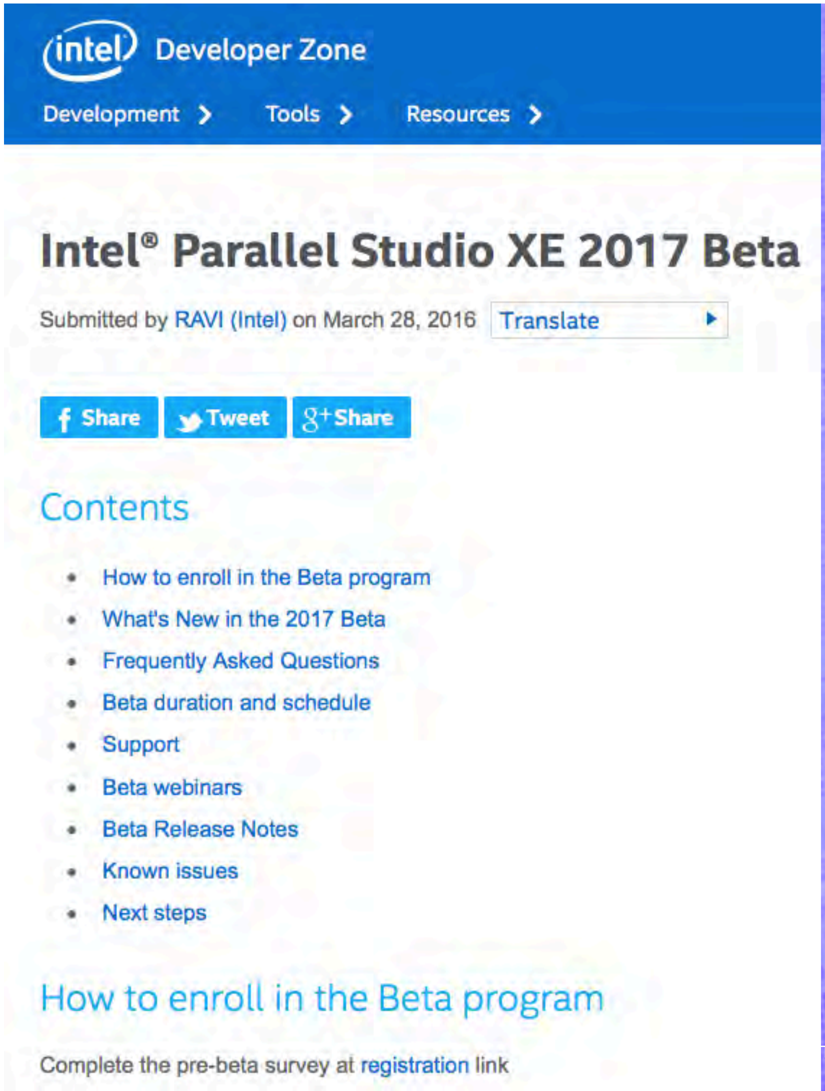
Performance tuning is easier if it's always on your mind and integrated into your development

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Scientific Approach to Analysis

- None of the tools provide exact results
  - Data collection overhead or dropping details
  - Define what results need to be precise
- Low overhead tools provide statistical results
  - Statistical theory is applicable
  - Think of proper sampling frequency (for data bandwidth)
  - Think of proper length of data collection (for process)
  - Think of proper number of experiments and results deviation
- Take into account other processes in a system
  - Anti-virus
  - Daemons and services
  - System processes
- Start early – tune often!

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.



Intel® Parallel Studio XE 2017 Beta

Submitted by RAVI (Intel) on March 28, 2016

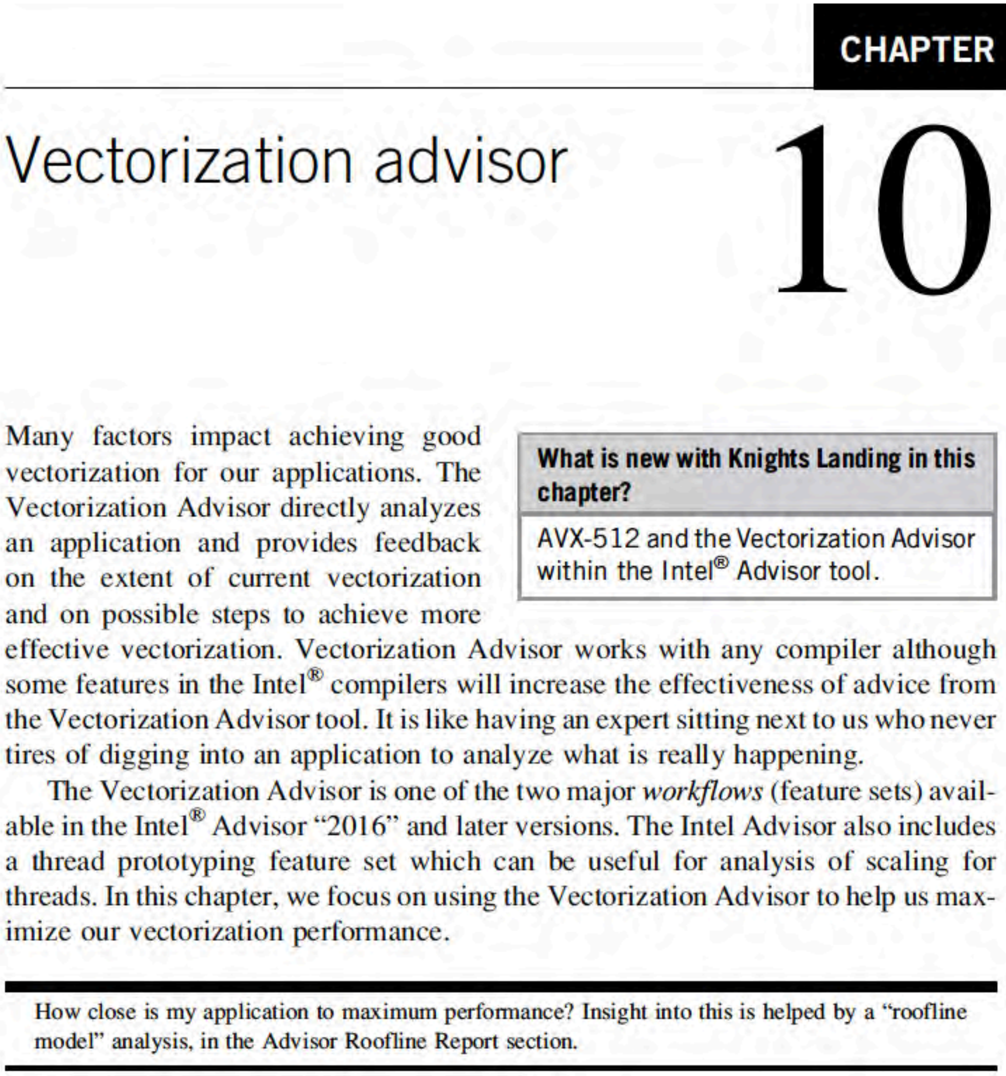
Contents

How to enroll in the Beta program

Complete the pre-beta survey at registration link

http://software.intel.com/articles/intel-parallel-studio-xe-2017-beta

BETA for "2017" Product – NOW



CHAPTER

10

Vectorization advisor

What is new with Knights Landing in this chapter?

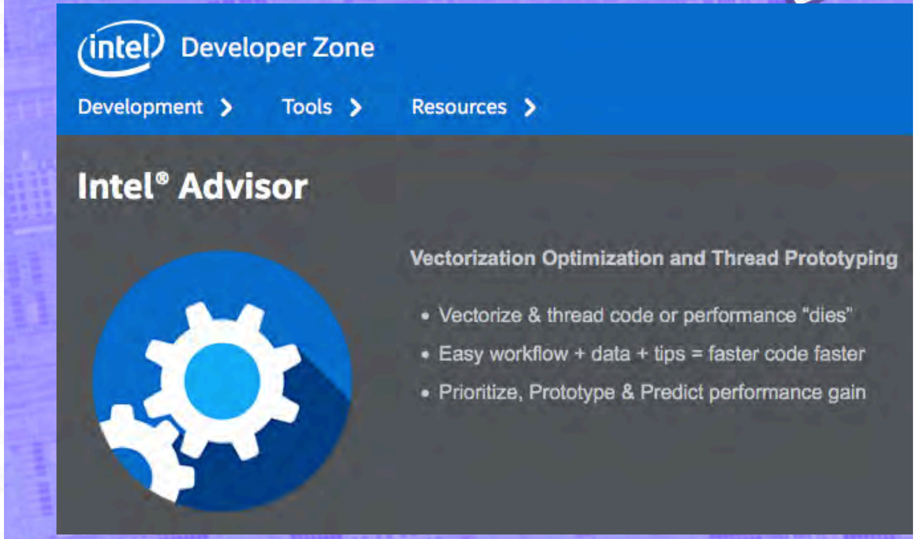
AVX-512 and the Vectorization Advisor within the Intel® Advisor tool.

Many factors impact achieving good vectorization for our applications. The Vectorization Advisor directly analyzes an application and provides feedback on the extent of current vectorization and on possible steps to achieve more effective vectorization. Vectorization Advisor works with any compiler although some features in the Intel® compilers will increase the effectiveness of advice from the Vectorization Advisor tool. It is like having an expert sitting next to us who never tires of digging into an application to analyze what is really happening.

The Vectorization Advisor is one of the two major *workflows* (feature sets) available in the Intel® Advisor "2016" and later versions. The Intel Advisor also includes a thread prototyping feature set which can be useful for analysis of scaling for threads. In this chapter, we focus on using the Vectorization Advisor to help us maximize our vectorization performance.

How close is my application to maximum performance? Insight into this is helped by a "roofline model" analysis, in the Advisor Roofline Report section.

Intel® Advisor



Intel® Advisor

Vectorization Optimization and Thread Prototyping

- Vectorize & thread code or performance "dies"
- Easy workflow + data + tips = faster code faster
- Prioritize, Prototype & Predict performance gain

I will talk about some NEW
"2017" features – which help
Intel Xeon processors tuning
and Intel Xeon Phi processor tuning
BOTH – Of Course!

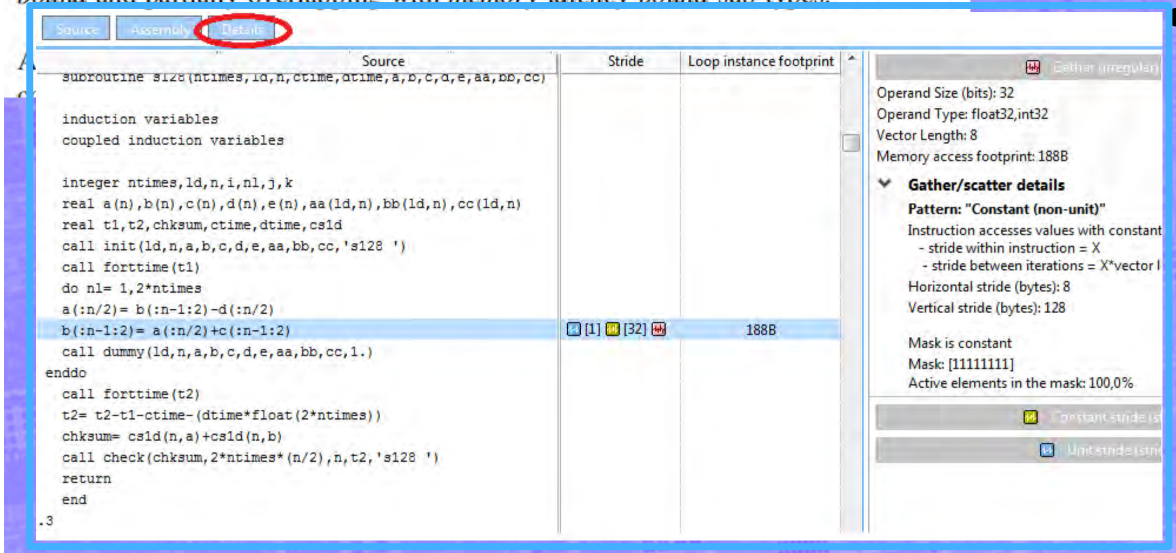
are trademarks of Intel Corporation in the U.S. and/or other countries.

Memory Access Pattern Report

MEMORY ACCESS PATTERN REPORT

An initial **Survey** analysis of hot loops often identifies inefficient memory access patterns as a main bottleneck. Memory access patterns issues are the toughest and most frequent performance problem in code not yet modernized for vector SIMD parallelism.

Applying *straightforward* SIMD and threading optimizations often does not provide desirable speedups because some parts of applications (including vectorized hot loops) become *memory bound*. Memory-access-patterns-bound code is just one subtype of a larger memory-bound class of problems, along with *memory-bandwidth-bound* and partially overlapping with *memory-latency-bound* sub-types.



Source	Stride	Loop instance footprint
b(:n-1:2)= a(:n/2)+c(:n-1:2)	[1] [32]	188B

```
  subroutine s128(ntimes,ld,n,ctime,dtime,a,b,c,d,e,aa,bb,cc)

  induction variables
  coupled induction variables

  integer ntimes,ld,n,i,nl,j,k
  real a(n),b(n),c(n),d(n),e(n),aa(ld,n),bb(ld,n),cc(ld,n)
  real t1,t2,chksum,ctime,dtime,cs1d
  call init(ld,n,a,b,c,d,e,aa,bb,cc,'s128 ')
  call forttime(t1)
  do nl= 1,2*ntimes
  a(:n/2)= b(:n-1:2)-d(:n/2)
  b(:n-1:2)= a(:n/2)+c(:n-1:2)
  call dummy(ld,n,a,b,c,d,e,aa,bb,cc,1.)
enddo
  call forttime(t2)
  t2= t2-t1-ctime-(dtime*float(2*ntimes))
  chksum= cs1d(n,a)+cs1d(n,b)
  call check(chksum,2*ntimes*(n/2),n,t2,'s128 ')
  return
  end
.3
```

Operand Size (bits): 32
Operand Type: float32,int32
Vector Length: 8
Memory access footprint: 188B

Gather/scatter details

Pattern: "Constant (non-unit)"

Instruction accesses values with constant
- stride within instruction = X
- stride between iterations = X*vector l

Horizontal stride (bytes): 8
Vertical stride (bytes): 128

Mask is constant
Mask: [11111111]
Active elements in the mask: 100,0%

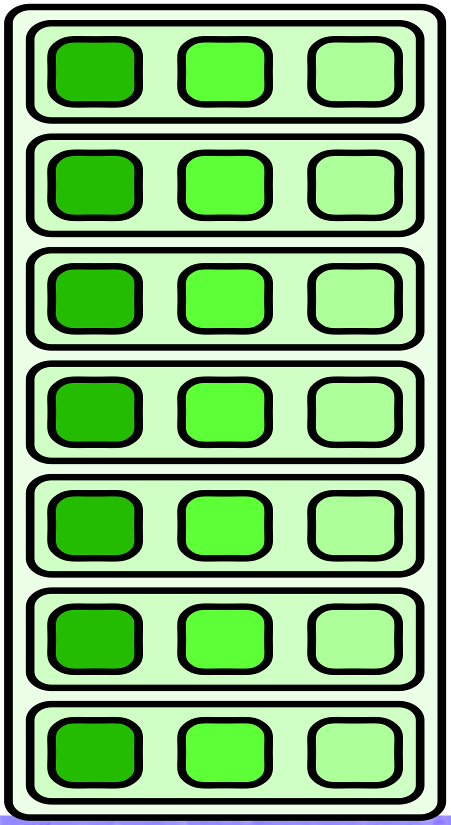
Can recommend:

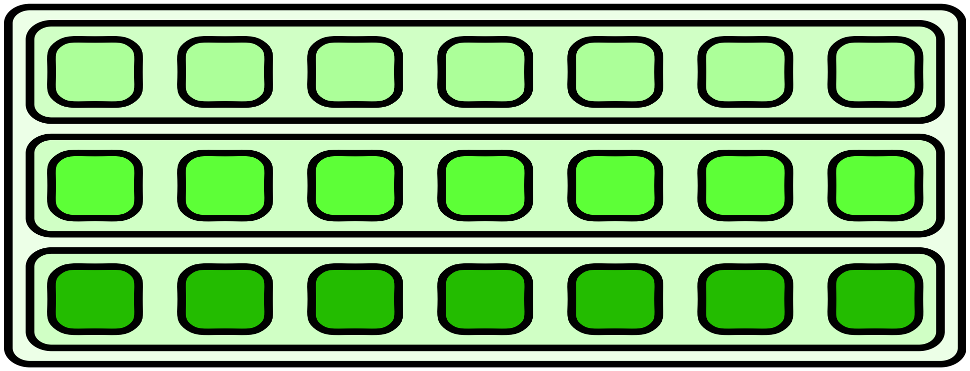
- AoS to SoA
- AoSoA
- Use of SDLT
- Use of MCDRAM

Gather/Scatter Report

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Data Layout: AoS vs. SoA





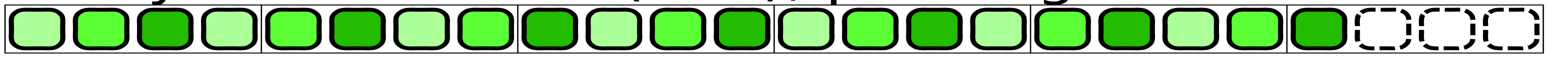
Array of structures (AoS) tends to cause cache alignment problems, and is hard to vectorize.

Structure of arrays (SoA) can be easily aligned to cache boundaries and is vectorizable.

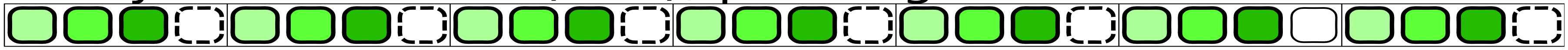
© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.
*Other names and brands may be claimed as the property of others.

Data Layout: Alignment

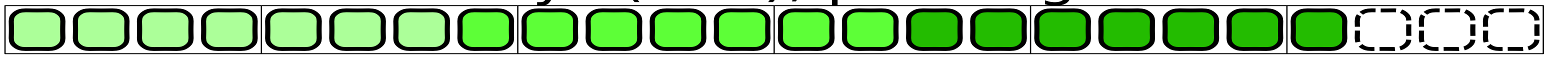
Array of Structures (AoS), padding at end.



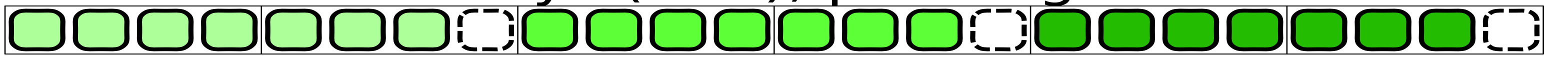
Array of Structures (AoS), padding after each structure.



Structure of Arrays (SoA), padding at end.

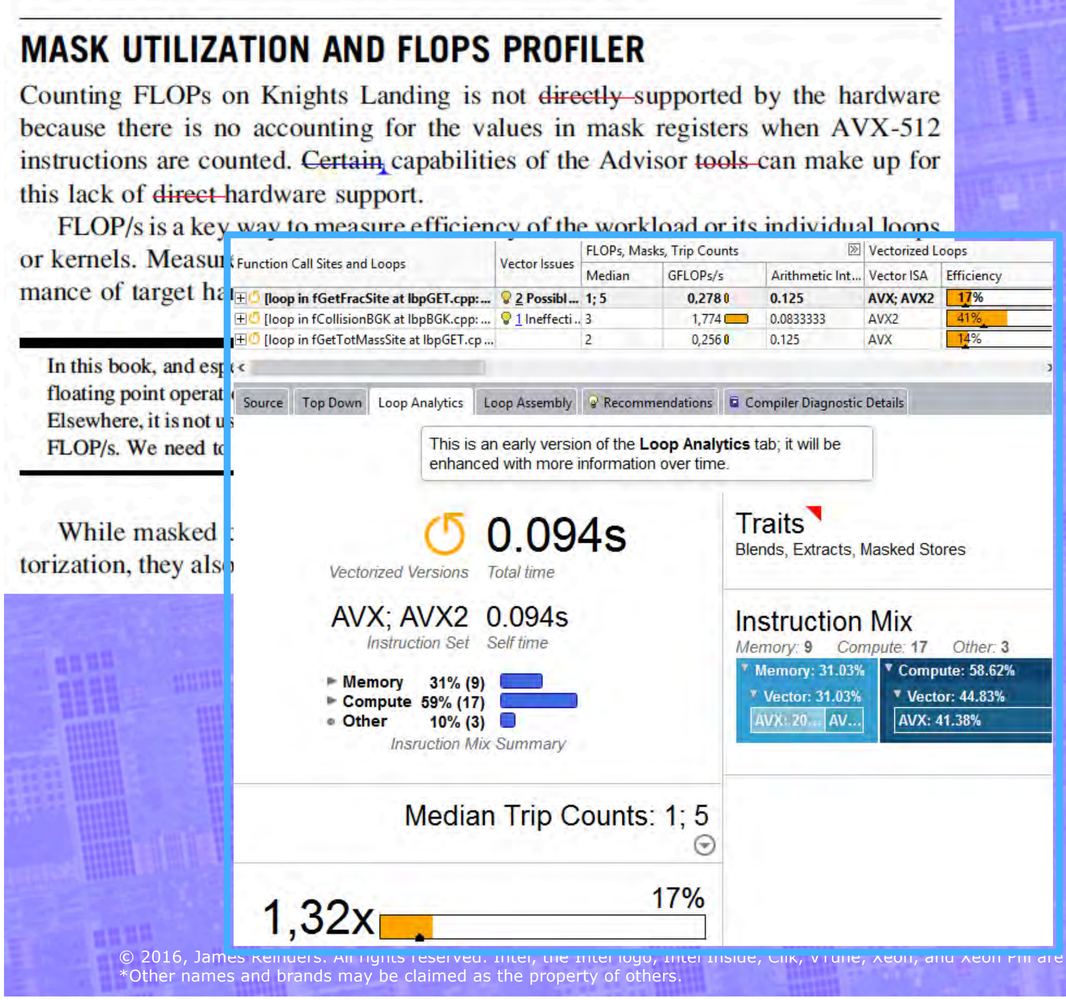


Structure of Arrays (SoA), padding after each component.



© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Mask Utilization and FLOPS Profiler



MASK UTILIZATION AND FLOPS PROFILER

Counting FLOPs on Knights Landing is not ~~directly~~ supported by the hardware because there is no accounting for the values in mask registers when AVX-512 instructions are counted. ~~Certain~~ capabilities of the Advisor ~~tools~~ can make up for this lack of ~~direct~~ hardware support.

FLOP/s is a key way to measure efficiency of the workload or its individual loops or kernels. Measur mance of target ha

In this book, and esp floating point operat Elsewhere, it is not u FLOP/s. We need to

While masked torization, they also

Function Call Sites and Loops	Vector Issues	FLOPs, Masks, Trip Counts			Vectorized Loops	
		Median	GFLOPs/s	Arithmetic Int...	Vector ISA	Efficiency
[loop in fGetFracSite at lbpGET.cpp:...	**2 Possibl...**	**1; 5**	**0,278**	**0.125**	**AVX; AVX2**	**17%**
[loop in fCollisionBGK at lbpBGK.cpp: ...	1 Ineffecti...	3	1,774	0.0833333	AVX2	41%
[loop in fGetTotMassSite at lbpGET.cp ...		2	0,256	0.125	AVX	14%

This is an early version of the **Loop Analytics** tab; it will be enhanced with more information over time.

0.094s

Vectorized Versions *Total time*

AVX; AVX2 0.094s

Instruction Set *Self time*

Memory	31% (9)	
Compute	59% (17)	
Other	10% (3)	

Insruction Mix Summary

Median Trip Counts: 1; 5

1,32x 17%

Traits

Blends, Extracts, Masked Stores

Instruction Mix

Memory: **9** *Compute:* **17** *Other:* **3**

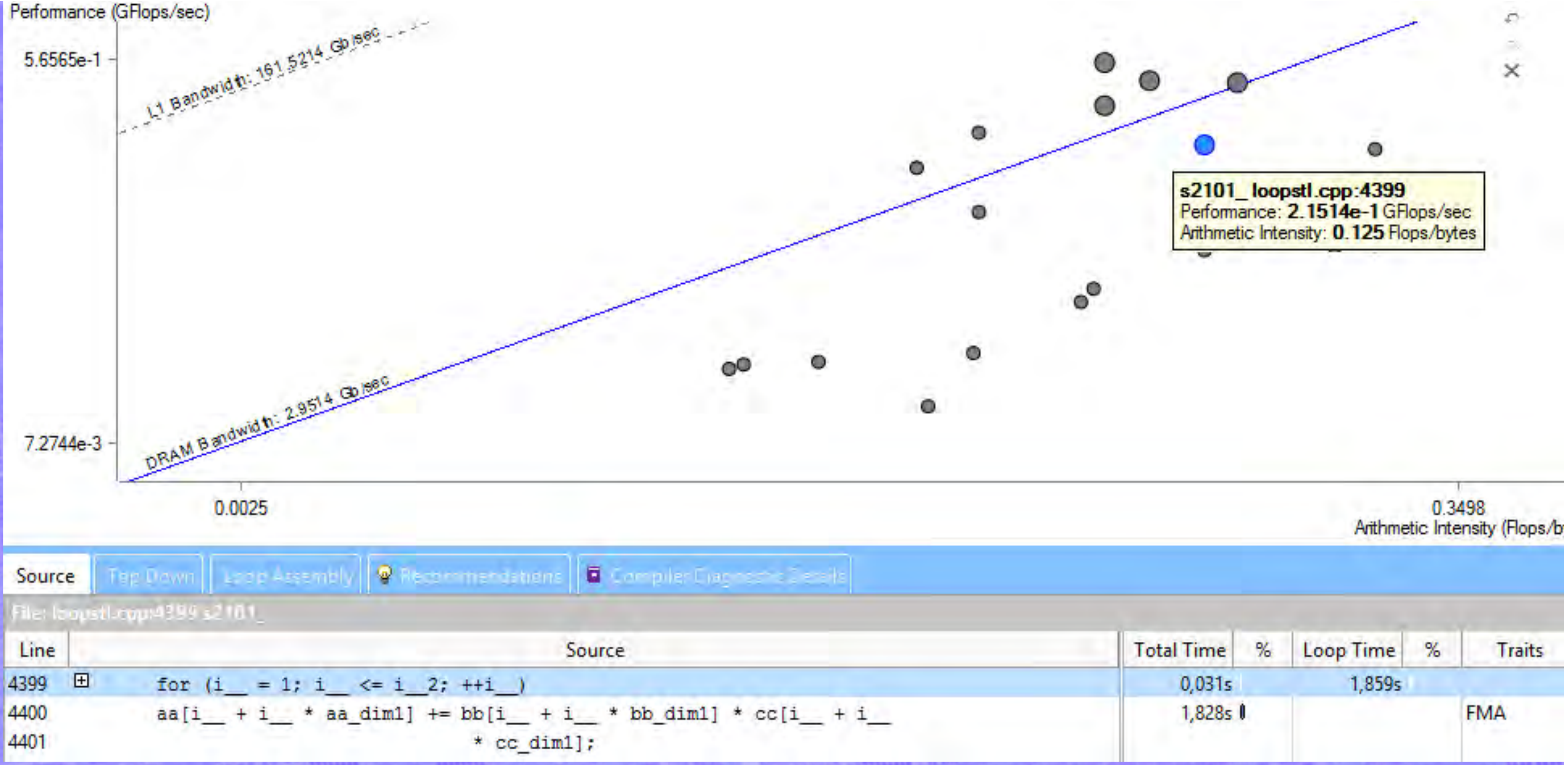
Mask-aware:

- FLOPs Report
- Vector Efficiency
- Memory Access Pattern,
- Roofline Analysis Graph

Vectorization efficiency and FLOP/s in Survey Report and Loop Analytics.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Roofline Report in Intel® Advisor 2017 (beta/ preview)


Performance (GFlops/sec)
5.6565e-1
7.2744e-3
L1 Bandwidth: 161.5214 Gb/sec
DRAM Bandwidth: 2.9514 Gb/sec
0.0025
0.3498
Arithmetic Intensity (Flops/b
s2101_ loopstl.cpp:4399
Performance: 2.1514e-1 GFlops/sec
Arithmetic Intensity: 0.125 Flops/bytes
Source
Top Down
Loop Assembly
Recommendations
Compiler Diagnostic Details
File: loopstl.cpp:4399 s2101_

Line	Source	Total Time	%	Loop Time	%	Traits
4399	⊞ `for (i__ = 1; i__ <= i__2; ++i__)`	0,031s		1,859s		
4400	`aa[i__ + i__ * aa_dim1] += bb[i__ + i__ * bb_dim1] * cc[i__ + i__`	1,828s				FMA
4401	`* cc_dim1];`					

…supplements AI-based analysis with a dynamic FLOP/s profile and peak FLOPs and memory sub-system throughput levels providing enlightening "bounds and bottlenecks" analysis for complex workloads.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

References

- Top-Down Performance Tuning Methodology
  - www.software.intel.com/en-us/articles/de-mystifying-software-performance-optimization
- Top-Down Characterization of Microarchitectural Bottlenecks
  - www.software.intel.com/en-us/articles/how-to-tune-applications-using-a-top-down-characterization-of-microarchitectural-issues
- Intel® VTune™ Amplifier XE
  - www.intel.ly/vtune-amplifier-xe
- Tuning Guides
  - www.intel.com/vtune-tuning-guides

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

Look for:

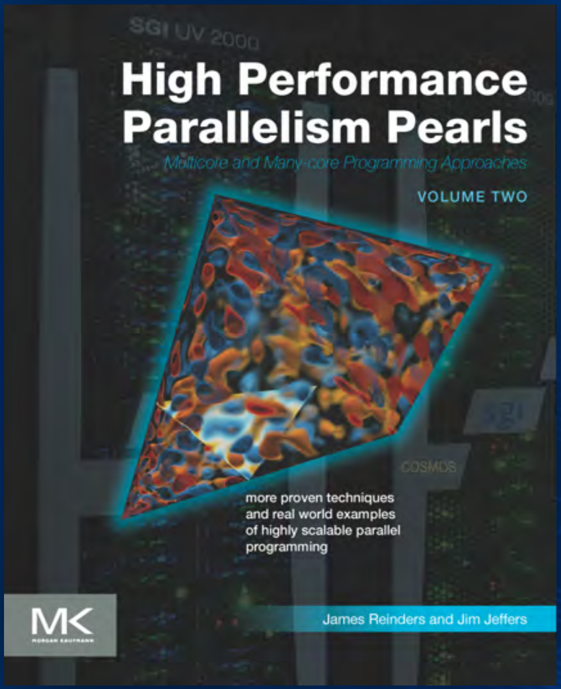
- Confirmation
- Surprises

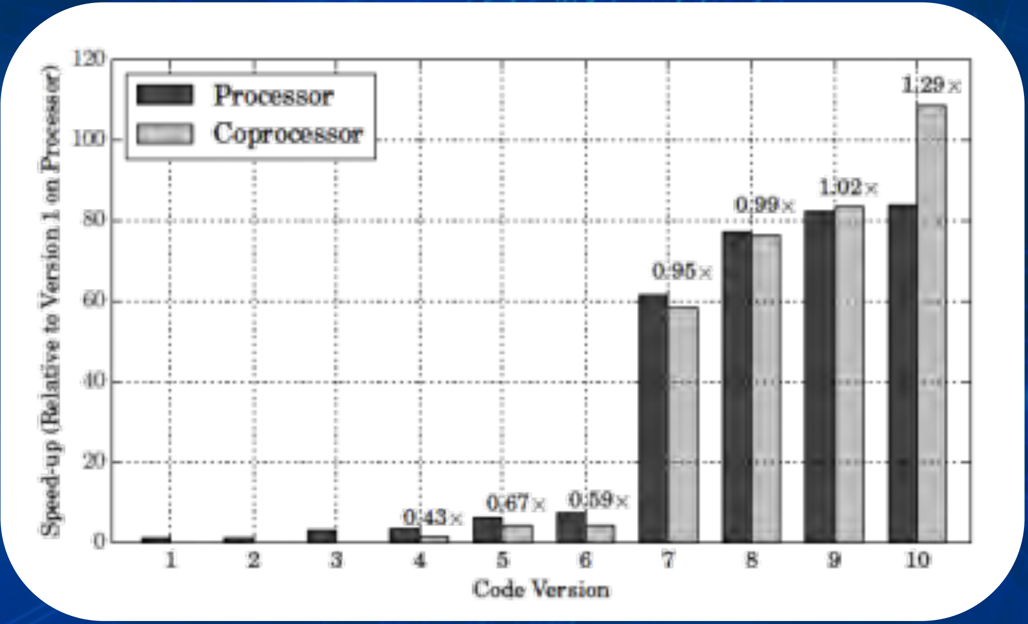
Do not skip either

KEEP CALM AND MIND YOUR ALGORITHMS

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

#Moderncode: COSMOS


SGI UV 2000
High Performance Parallelism Pearls
Multicore and Many-core Programming Approaches
VOLUME TWO
COSMOS
more proven techniques and real world examples of highly scalable parallel programming
MK
James Reinders and Jim Jeffers

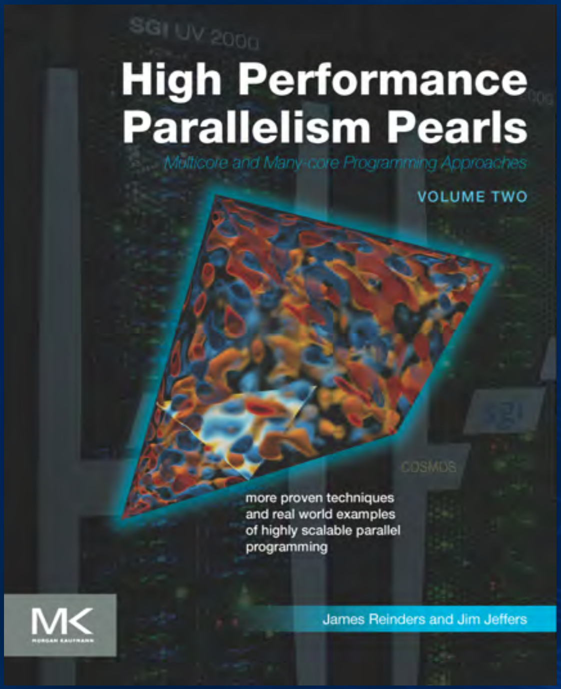


Book Cover Background: Photo of the COSMOS@DiRAC SGI UV2000 based Supercomputer manufactured by SGI, Inc and operated by the Stephen Hawking Centre for Theoretical Cosmology, University of Cambridge. Photo courtesy of Philip Mynott. Book Cover Foreground: 3D visualization of statistical fluctuations in the Cosmic Microwave Background, the remnant of the first measurable light after the Big Bang. CMB data is from the Planck satellite and is the topic of Chapter 10 providing insights into new physics and how the universe evolved. Visualization rendered with Intel's OSPRay ray tracing open source software by Gregory P. Johnson and Timothy Rowley, Intel Corporation.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

#Moderncode: COSMOS

What?



SGI UV 2000

High Performance Parallelism Pearls

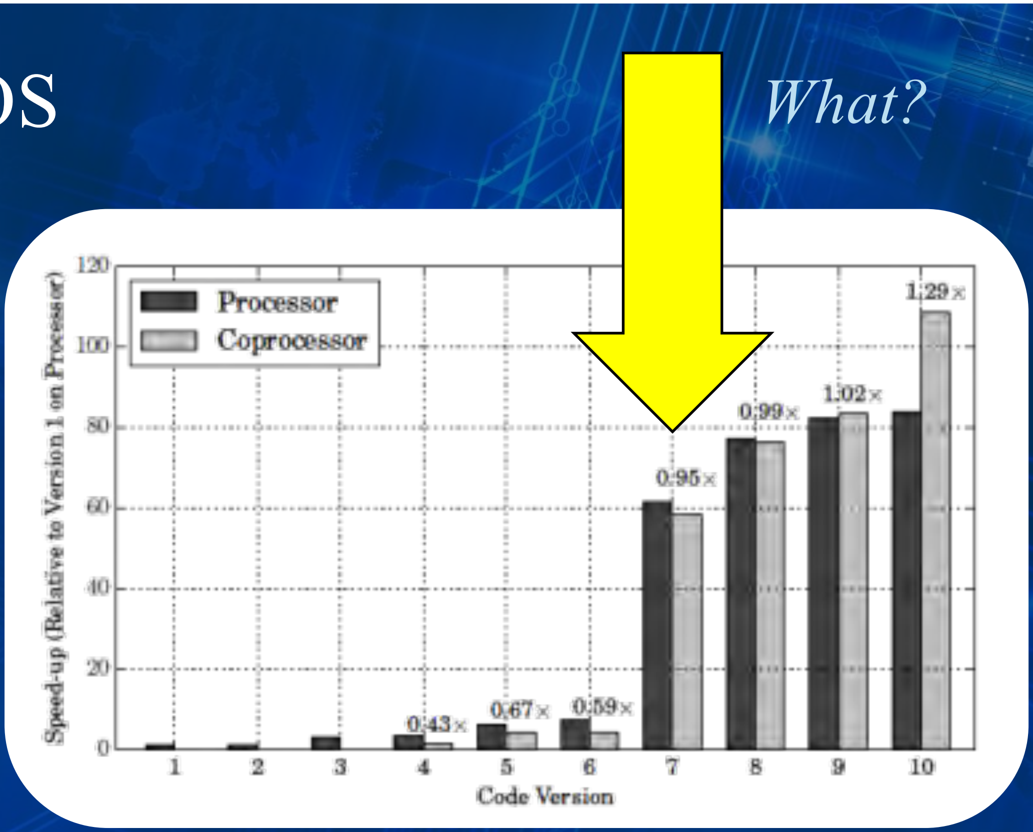
Multicore and Many-core Programming Approaches

VOLUME TWO

more proven techniques and real world examples of highly scalable parallel programming

MK

James Reinders and Jim Jeffers



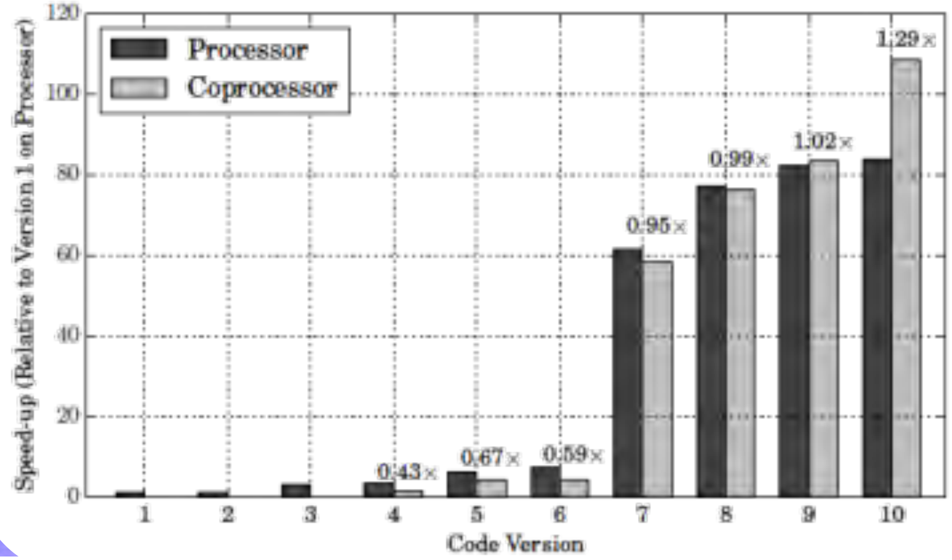
Book Cover Background: Photo of the COSMOS@DiRAC SGI UV2000 based Supercomputer manufactured by SGI, Inc and operated by the Stephen Hawking Centre for Theoretical Cosmology, University of Cambridge. Photo courtesy of Philip Mynott. Book Cover Foreground: 3D visualization of statistical fluctuations in the Cosmic Microwave Background, the remnant of the first measurable light after the Big Bang. CMB data is from the Planck satellite and is the topic of Chapter 10 providing insights into new physics and how the universe evolved. Visualization rendered with Intel's OSPRay ray tracing open source software by Gregory P. Johnson and Timothy Rowley, Intel Corporation.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

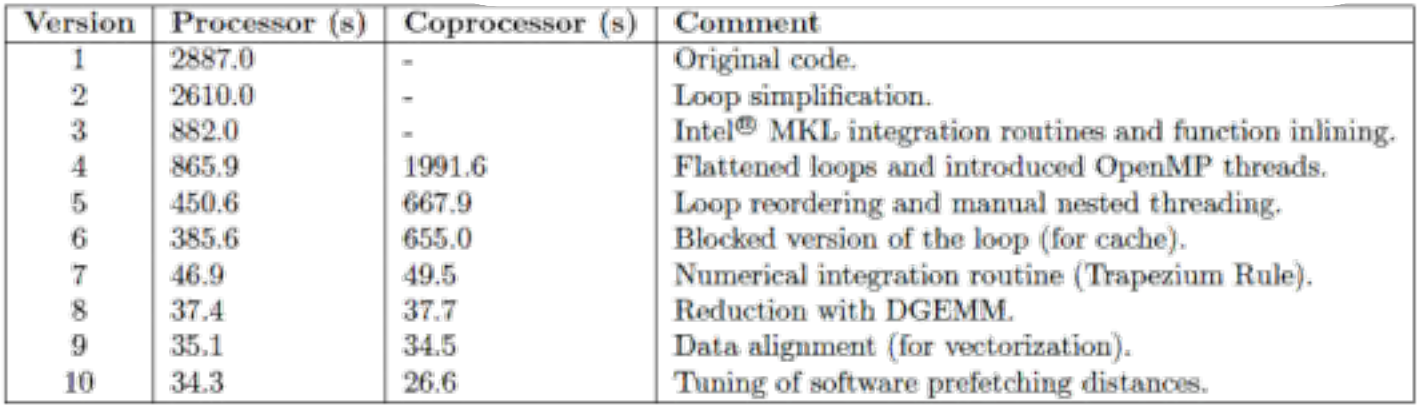
High Performance Parallelism Pearls

Cosmic Microwave Background Analysis: Nested Parallelism in Practice

Volume 2, Chapter 10



We find that using a simple trapezium rule integrator combined with hand-selected sampling points (to improve accuracy in areas of interest) provides sufficient numerical accuracy to obtain a physically meaningful result, and the reduced space and time requirements of this simplified method give a speed-up of O(10x).



Version	Processor (s)	Coprocessor (s)	Comment
1	2887.0	-	Original code.
2	2610.0	-	Loop simplification.
3	882.0	-	Intel® MKL integration routines and function inlining.
4	865.9	1991.6	Flattened loops and introduced OpenMP threads.
5	450.6	667.9	Loop reordering and manual nested threading.
6	385.6	655.0	Blocked version of the loop (for cache).
7	46.9	49.5	Numerical integration routine (Trapezium Rule).
8	37.4	37.7	Reduction with DGEMM.
9	35.1	34.5	Data alignment (for vectorization).
10	34.3	26.6	Tuning of software prefetching distances.

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 [illegible] ned with other products. For more information go to [illegible] where the numbers are published.

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

KEEP CALM HPC WILL SAVE THE WORLD

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.

James Reinders. Parallel Programming Enthusiast

James has been involved in multiple engineering, research and educational efforts to increase use of parallel programming throughout the industry. James worked 10,001 days as an Intel employee 1989-2016, and contributed to numerous projects including the world's first TeraFLOP/s supercomputer (ASCI Red), first 3 TeraFLOP/s supercomputer (ASCI Red upgrade), the world's first TeraFLOP/s microprocessor (Intel® Xeon Phi™ coprocessor) and the world's first 3 TeraFLOP/s microprocessor (Intel® Xeon Phi™ Processor).

James been an author on numerous technical books, including VTune™ Performance Analyzer Essentials (Intel Press, 2005),

Intel® Threading Building Blocks (O'Reilly Media, 2007), Structured Parallel Programming (Morgan Kaufmann, 2012), Intel® Xeon Phi™ Coprocessor High Performance Programming (Morgan Kaufmann, 2013), Multithreading for Visual Effects (A K Peters/CRC Press, 2014),

High Performance Parallelism Pearls Volume 1 (Morgan Kaufmann, Nov. 2014), High Performance Parallelism Pearls Volume 2 (Morgan Kaufmann, Aug. 2015), and Intel® Xeon Phi™ Processor High Performance Programming - Knights Landing Edition (Morgan Kaufmann, 2016).

© 2016, James Reinders. All rights reserved. Intel, the Intel logo, Intel Inside, Cilk, VTune, Xeon, and Xeon Phi 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.