Faster Code.... Faster

Intel® Parallel Studio XE 2017
Create Faster Code...Faster

- **Intel® Parallel Studio XE**
  - Design, build, verify, and tune
  - C++, C, Fortran*, Python* and Java*
  - Standards-driven parallel models: OpenMP*, MPI, and TBB

- **Highlights from 2017 edition**
  - **Faster Python application performance** using Intel® Distribution for Python and Intel® VTune™ Amplifier XE.
  - **Faster deep learning on Intel® architecture** using Intel® Math Kernel Library and Intel® Data Analytics Acceleration Library
  - **Quickly assess application performance** using snapshot features of Intel® VTune™ Amplifier XE and Intel® Trace Analyzer and Collector
  - **Scale to next-generation platforms** including the latest Intel® Xeon Phi™ processor. Optimizations for Intel® AVX-512, high bandwidth memory, and explicit vectorization for compiler and analysis tools.

http://intel.ly/perf-tools
## What’s Inside

**Intel® Parallel Studio XE 2017**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C++ Compiler</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Fortran Compiler</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Distribution for Python*</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Math Kernel Library – <em>fast math library</em></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Integrated Performance Primitives – <em>image, signal &amp; data processing</em></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Threading Building Blocks – <em>threading library</em></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Data Analytics Acceleration Library – <em>machine learning &amp; analytics</em></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

| ANALYZE | | | |
| Intel® VTune™ Amplifier XE – *performance profiler* | ✓ | ✓ | ✓ |
| Intel® Advisor – *vectorization optimization and thread prototyping* | ✓ | ✓ | ✓ |
| Intel® Inspector – *memory and thread debugging* | ✓ | ✓ | ✓ |

| SCALE | | | |
| Intel® MPI Library – *message passing interface library* | ✓ | | ✓ |
| Intel® Trace Analyzer and Collector – *MPI Tuning and Analysis* | ✓ | | ✓ |
| Intel® Cluster Checker – *cluster diagnostic expert system* | ✓ | | ✓ |

| Rogue Wave IMSL® Library – *Fortran numerical analysis* | Bundle or Add-on | Add-on | Add-on |

Additional configurations including, floating and academic, are available at: [http://intel.ly/perf-tools](http://intel.ly/perf-tools)
Staying Current with Support for the Latest Standards, Operating Systems, and Processors

Enhanced C11 and C++14 standards support

- Sized deallocation
- Relaxed constexpr restrictions
- Variable templates
- Single-Quotation-Mark as a digit separator

Operating systems

- Debian* 7.0, 8.0; Fedora* 23, 24; Red Hat Enterprise Linux* 6, 7; SuSE LINUX Enterprise Server* 11,12; Ubuntu* 14.04 LTS 16.04 LTS, 16.04
- macOS* 10.11

Enhanced Fortran 2008 and draft 2015 standards support

- Implied-shape PARAMETER arrays
- 2008 bind C internal procedures
- Extended EXIT for all named blocks
- Pointer initialization

Latest processors

- Support and tuning added for the latest Intel® Xeon Phi™ (codenamed Knights Landing) and Intel® AVX-512
Intel® Compilers for Intel® Parallel Studio XE 2017
Intel® C++ 17.0 and Intel® Fortran 17.0

Common Updates

- Enhanced support for the latest Intel® AVX2 and AVX512 instruction sets for the latest Intel® processors (including Intel® Xeon Phi™ processor)
- Enhanced optimization/vectorization reports make Intel compilers indispensable towards modernizing your code
- Support for OpenMP* 4.5, offering improved vectorization control, and new SIMD instructions

**Intel® C++ Compiler**
- SIMD Data Layout Template to facilitate vectorization for your C++ code
- Virtual function vectorization capability
- Full support for the latest C11 and C++14 standards; initial C++17 support

**Intel® Fortran Compiler**
- Substantial coarray performance improvement – up to **twice as fast** as previous versions on non-trivial coarray Fortran programs
- Almost complete Fortran 2008 support
- Further interoperability with C (part of draft Fortran 2015)
Boost Application Performance on Windows* and Linux* Intel® C++ and Fortran Compilers

Boost C++ application performance on Windows* & Linux* using Intel® C++ Compiler
(higher is better)

<table>
<thead>
<tr>
<th>Floating Point</th>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGi 15.10</td>
<td>Visual C++ 170</td>
<td>GCC 6.1.0</td>
</tr>
<tr>
<td>Intel 15.10</td>
<td>Clang 3.8</td>
<td>Intel 17.0</td>
</tr>
</tbody>
</table>

Boost Fortran application performance on Windows* & Linux* using Intel® Fortran Compiler
(higher is better)

<table>
<thead>
<tr>
<th>Integer</th>
<th>Windows</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGi 15.10</td>
<td>Visual C++ 170</td>
<td>GCC 6.1.0</td>
</tr>
<tr>
<td>Intel 15.10</td>
<td>Clang 3.8</td>
<td>Intel 17.0</td>
</tr>
</tbody>
</table>

Relative geometric performance, SPEC® benchmark - higher is better

Windows Estimated SPECf®_rate_base2006

Linux Estimated SPECint®_rate_base2006

Optimization Notice: Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of the optimization on non-Intel microprocessors. Optimizations may vary for different component technologies, update versions, processor family, processor model, and processor version.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SPECmark and TuLiP mark, are measured in specific computer systems, components, software, operations, and functions. Any change to any of these factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchase.

All Microprocessor instruction sets covered by this notice. Notice revision #20110804.

Copyright © 2016, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

9
Компилятор Intel® C++

- Библиотека SIMD Data Layout Templates
- Векторизация виртуальных функций
- Расширенная поддержка стандартов C11 и C++14
  - Освобождение памяти определенного размера
  - Упрощенные ограничения на создание constexpr функций
  - Шаблоны переменных
  - Одинарная кавычка как цифровой разделитель
- Улучшена совместимость с GNU* и Microsoft*
- Отчеты оптимизации
- Приведение типов SSE
- Более детальная диагностика компилятора по аргументам шаблонов
- Широкий набор поддерживаемых ОС, включая Android* и embedded версии Linux
SIMD Data Layout Template
Увеличиваем производительность C++ приложений

- Что “не так” в объектно-ориентированном подходе?

```cpp
struct YourStruct{
    float x;
    float y;
    float z;
};
```

‘Array of Structures’ (AOS) vs ‘Structure of Arrays’ (SOA)
SoA -> AoS transformation with SDLT

#include <iostream>
#include <vector>

typedef struct RGBs {
    float r, g, b;
} RGBTy;

void main()
{
    std::vector<RGBTy> a(N);

    #pragma omp simd
    for (int k = 0; k < N; ++k) {
        a[k].r = k * 1.5; // non-unit stride access
        a[k].g = k * 2.5; // non-unit stride access
        a[k].b = k * 3.5; // non-unit stride access
    }
    std::cout << "k = " << 10 << " , a[10].r = " << a[10].r << " , a[10].g = " << a[10].g << " , a[10].b = " << a[10].b << std::endl;
}

#include <iostream>
#include <sdlt/sdlth.h>

typedef struct RGBs {
    float r, g, b;
} RGBTy;

SDLT_PRIMITIVE(RGBTy, r,g,b)

void main()
{
    sdlt::soa1d_container<RGBTy> aContainer(N);
    auto a = aContainer.access();

    #pragma omp simd
    for (int k = 0; k < N; ++k) {
        a[k].r() = k * 1.5; // Access is managed by SDLT
        a[k].g() = k * 2.5; // Access is managed by SDLT
        a[k].b() = k * 3.5; // Access is managed by SDLT
    }
    std::cout << "k = " << 10 << " , a[10].r() = " << a[10].r() << " , a[10].g() = " << a[10].g() << " , a[10].b() = " << a[10].b() << std::endl;
}

Performance improvement is 6x on HSW-EP
SIMD Data Layout Template
Увеличиваем производительность C++ приложений

- Быстрый переход от AOS к SOA
- Увеличиваем продуктивность
  Используем подготовленные шаблоны с минимальными затратами, и пусть SDLT «векторизует» наш код.
- Улучшаем производительность
  SDLT векторизует код, делая доступ к памяти последовательным, что приводит к лучшей производительности.
- Легкая интеграция:
  SDLT следует известной векторной модели программирования от Intel

"We used SDLT to vectorize the deformer code in Premo, the in-house animation tool for DreamWorks Animation. The performance improvements we were able to achieve were dramatic, and these improvements will translate directly into higher quality characters that will be seen on-screen in future movies. Also the library itself was easy to use and integrate into our existing codebase."

Martin Watt
Principal Engineer,
DreamWorks Animation
Векторизация виртуальных функций

- Синтаксис как для обычных векторных функций
- Набор версий наследуется и не может быть изменен в переопределениях
- «Векторность» должна быть введена с виртуальным методом, не в переопределениях

```cpp
class A {
    public:
#pragma omp declare simd linear(X)
#pragma omp declare simd uniform(this) linear(X)
    virtual int foo(int X);
}

#pragma omp declare simd uniform(this) linear(X)
int A::foo(int X) { return X + 1; }

class B : public A {
    public:
        // #pragma omp declare simd linear(X) - inherited
        // #pragma omp declare simd uniform(this) linear(X)
        int foo(int X) { return (X*X); }
};
```
typedef float complex fcomplex;
const uint32_t max_iter = 3000;
#pragma omp declare simd uniform(max_iter), simdlen(16)
uint32_t mandel(fcomplex c, uint32_t max_iter)
{
    uint32_t count = 1; fcomplex z = c;
    while ((cabsf(z) < 2.0f) && (count < max_iter)) {
        z = z * z + c; count++;
    }
    return count;
}
uint32_t count[ImageWidth][ImageHeight];
...... .... ....
for (int32_t y = 0; y < ImageHeight; ++y) {
    float c_im = max_imag - y * imag_factor;
    #pragma omp simd safelen(16)
    for (int32_t x = 0; x < ImageWidth; ++x) {
        fcomplex in_vals_tmp = (min_real + x * real_factor) + (c_im * 1.0iF);
        count[y][x] = mandel(in_vals_tmp, max_iter);
    }
}
Высокая производительность
Векторизация средствами OpenMP*

- Всего 3 строчки кода для использования всех возможностей SSE и AVX
- Директивы игнорируются другими компиляторами, если не поддерживаются

```c
float path_calc(float *z, float L[][VLEN], int k, int N, int Nmat)

float portfolio(float L[][VLEN], int k, int N, int Nopt, int Nmat)
...
... for (path=0; path<NPATH; path+=VLEN) {
    /* Initialise forward rates */
    z = z0 + path * Nmat;
    for(int k=0; k < VLEN; k++) {
        for(i=0;i<N;i++) {
            L[i][k] = L0[i];
        }
        /* LIBOR path calculation */
        float temp = path_calc(z, L, k, N, Nmat);
        v[k+path] = portfolio(L, k, N, Nopt, Nmat);
    } /* move pointer to start of next block */
    z += Nmat;
}
```
Высокая производительность
Векторизация средствами OpenMP*

- Всего 3 строчки кода для использования всех возможностей SSE и AVX
- Директивы игнорируются другими компиляторами, если не поддерживаются

```c
#pragma omp declare simd linear(z:40) uniform(L, N, Nmat) linear(k)
float path_calc(float *z, float L[][VLEN], int k, int N, int Nmat)

#pragma omp declare simd uniform(L, N, Nopt, Nmat) linear(k)
float portfolio(float L[][VLEN], int k, int N, int Nopt, int Nmat)

... ...
for (path=0; path<NPATH; path+=VLEN) {
    /* Initialise forward rates */
    z = z0 + path * Nmat;
    #pragma omp declare simd linear(z:Nmat)
    for(int k=0; k < VLEN; k++) {
        for(i=0;i<N;i++) {
            L[i][k] = L0[i];
        }
    }
    /* LIBOR path calculation */
    float temp = path_calc(z, L, k, N, Nmat);
    v[k+path] = portfolio(L, k, N, Nopt, Nmat);
    /* move pointer to start of next block */
    z += Nmat;
}
```
Высокая производительность

Векторизация средствами OpenMP*

- Всего 3 строчки кода для использования всех возможностей SSE и AVX
- Директивы игнорируются другими компиляторами, если не поддерживаются

```c
#pragma omp declare simd linear(z:40) uniform(L, N, Nmat) linear(k)
float path_calc(float *z, float L[][VLEN], int k, int N, int Nmat)

#pragma omp declare simd uniform(L, N, Nopt, Nmat) linear(k)
float portfolio(float L[][VLEN], int k, int N, int Nopt, int Nmat)

... ...
for (path=0; path<NPATH; path+=VLEN) {
    /* Initialise forward rates */
    z = z0 + path * Nmat;
    #pragma omp simd linear(z:Nmat)
    for(int k=0; k < VLEN; k++) {
        for(i=0;i<N;i++) {
            L[i][k] = L0[i];
        }
    }
    /* LIBOR path calculation */
    float temp = path_calc(z, L, k, N, Nmat);
    v[k+path] = portfolio(L, k, N, Nopt, Nmat);
    /* move pointer to start of next block */
    z += Nmat;
}
```

Libor calculation speedup

более - лучше

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>SSE 4.2</th>
<th>Core-AVX2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3,51</td>
<td>6,61</td>
</tr>
</tbody>
</table>

Конфигурация: Intel® Xeon® CPU E3-1270 @ 3.50 GHz Haswell система (4 ядра с Hyper-Threading On), работающая в 3.50GHz, с 32GB RAM, L1 Cache 256KB, L2 Cache 1.0MB, L3 Cache 8.0MB; 64-bit Windows® Server 2012 R2 Datacenter. Компилятор: GCC 4.8.3 —O3 -Qopenmp -Qsimd -QxSSE4.2 or AVX2. Для более точных результатов можно использовать инструкции AVX2. More information can be found at http://www.intel.com/performance

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 to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110834.
Больше отчетов, красивых и разных

- Исходный код с «встроенными» отчетами об оптимизации
  
  `[Q | -q]opt-report-annotate=[text | html]`
  
  `[Q | -q]opt-report-annotate-position=[caller|callee|both]`
  
- Информация по распределению регистров (Register Allocation) и его эффективности
  
  - spill'ы в ассемблере

- Улучшены отчёты по векторизации

- Добавлены имена переменных и обращения к памяти

  16.0: remark #15346: vector dependence: assumed ANTI dependence between line 108 and line 116
  
  17.0: remark #15346: vector dependence: assumed ANTI dependence between *(s1) (108:2) and *(r+4) (116:2)

- Ещё более понятные причины не векторизованных циклов, например

  exception handling for function call prevents vectorization
Компилятор Intel® Fortran

- Производительность Coarray Fortran существенно улучшена
- Выравнивание динамически выделяемых массивов
- Почти полная поддержка Fortran 2008 и частичная драфта 2015
  - массив констант с подразумеваемой формой
  - BIND(C) разрешен для внутренних процедур
  - EXIT для всех именованных блоков
  - Инициализация указателей
- VS2013 Shell вместо VS2010 Shell на Windows
OpenMP* 4.5
Clause linear(ref/val/uval)

For C, the compiler "puts" sequential values in the vector register.

In Fortran, arguments are passed by reference:

- 4 addresses in the register
- LINEAR(REF(X)) tells the compiler that addresses are sequential

---

```!
/*omp declare simd
REAL FUNCTION FOO(X, Y)
REAL, VALUE :: Y << by reference
REAL, VALUE :: X << by reference
FOO = X + Y << gathers!!!!
END FUNCTION FOO
```

```!
omp$ simd private(X,Y)
DO I= 0, N
    Y = B(I)
    X = A(I)
    C(I) += FOO(X, Y)
ENDDO
```
OpenMP* 4.5
Clause linear(ref/val/uval)

Для C, компилятор «кладет» последовательные значения в векторный регистр

В Фортране аргументы передаются по ссылке

- 4 адреса в регистре
- LINEAR(REF(X)) говорит компилятору, что адреса последовательны

```fortran
!$omp declare simd linear(ref(x), ref(y))
REAL FUNCTION FOO(X, Y)
REAL, VALUE :: Y   << by reference
REAL, VALUE :: X   << by reference
FOO = X + Y       << sequential reads!!!!
END FUNCTION FOO
...
!omp$ simd private(X,Y)
DO I= 0, N
   Y = B(I)
   X = A(I)
   C(I) += FOO(X, Y)
ENDDO
```

Для С, компилятор «кладет» последовательные значения в векторный регистр

Авторские права © 2016, Intel Corporation. Все права защищены.
*Other names and brands may be claimed as the property of others.
OpenMP* 4.5

Редукции массивов в C/C++

Для Фортрана, OpenMP 4.0 уже поддерживал массив в качестве переменной редукции

```c
#pragma omp parallel reduction(+:a)
for (i = 0; i < n; i++){
    a[i] += 1;
}
```

OpenMP 4.5 разрешает использовать массив и указатели для C/C++

- Появилось в 17.0 версии
  - Но работает только для всего объекта – секции массива из Intel® Cilk Plus не поддерживаются
Примечание по оптимизации

OpenMP* 4.5 Директива TASKLOOP

Параллелизация цикла используя задачи OpenMP

```c
#pragma omp taskloop [simd] [clauses]
    for-loops
```

- Делит цикл на куски (chunk’и)
- Clause `grainsize` и `num_tasks` для управления созданием задач
- Похож на ‘cilk_for’ из Intel® Cilk Plus
- Создается задача для каждого куска цикла
OpenMP* 4.5
Циклы “doacross”

Типы зависимостей *source* и *sink* добавлены в clause *depend* для поддержки “doacross” циклов вместе с clause *ordered*

*depend(source)* и *depend(sink:<iteration-vector>)*

Распараллеливание циклов с хорошо структурированными зависимостями

```c
#pragma omp parallel for ordered
for (i = 1, ...; i <= N) {
    S1;
    #pragma omp ordered depend(sink:i-1)
    S2;
    #pragma omp ordered depend(source)
    S3;
}
```
Циклы “doacross”

```c
#pragma omp parallel for ordered(2)
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
    {
        a[i][j] = foo(i, j);
        #pragma omp ordered depend (sink: i - 1, j) depend (sink: i, j - 1)
        b[i][j] = bar(a[i][j], b[i - 1][j], b[i][j - 1]);
        #pragma omp ordered depend (source)
        baz(a[i][j], b[i][j]);
    }
```
OpenMP* 4.5
Средства синхронизации

Современные процессы поддерживают транзакционную память, например Intel® TSX (Intel® Transactional Synchronization Extensions)

Новые функции:

```c
omp_init_lock_with_hint(omp_lock_t *lock,
                       omp_lock_hint_t hint)
omp_init_nest_lock_with_hint(omp_nest_lock_t *lock,
                             omp_lock_hint_t hint)
```

Типы:

```c
omp_lock_hint_none
omp_lock_hint_uncontended
omp_lock_hint_contended
omp_lock_hint_nonspeculative
omp_lock_hint_speculative
```

Clause `hint(type)` для критической секции

C++:

```c
#pragma omp critical [(name)] [hint(expression)]
structured-block
```

Фортран:

```fortran
!$ omp critical [(name)] [hint(expression)]
structured-block
!$ omp end critical [(name)]
```
Intel® MPX

4 новых 128-битных регистра для хранения границ

- Существующие регистры не затронуты
- Необходима поддержка ОС

Новые инструкции для загрузки и проверки границ до доступа к памяти

- Исключение в случае проблемы

Новые инструкции для чтения/записи границ в участок памяти

Расширения MPX “забиваются” NOP’ами если не поддерживаются

```plaintext
[Q]check-pointers
[Q]check-pointers-mpx
```
Асинхронный OFFLOAD на GPU
Openmp DEPEND Cluase в TARGET директиве

// initialize arr1 - offload to target
#pragma omp target map(from: arr1[0:SIZE]) depend(out:arr1) nowait
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr1[i] += i; }

// initialize arr2
#pragma omp task depend(out:arr2)
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr2[i] += -i; }

// compute intermediate result on target
#pragma omp target map(to: arr1[0:SIZE], arr2[0:SIZE]) \ 
    map(from:arr3[0:SIZE]) \ 
    nowait depend(in:arr1, arr2)
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { arr3[i] = arr1[i] + arr2[i]; }

#pragma omp taskwait
#pragma omp parallel for
    for (int i = 0; i < SIZE; i++) { res[i] += arr3[i]; }
## Impressive Performance Improvement

Intel C++ Explicit Vectorization using OpenMP* SIMD

### SIMD Speedup on Intel® Xeon® Processor

<table>
<thead>
<tr>
<th>Software</th>
<th>Serial</th>
<th>SSE4.2</th>
<th>Core-AVX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AoBench</td>
<td>1.00</td>
<td>2.48</td>
<td>4,27</td>
</tr>
<tr>
<td>Collision Detection</td>
<td>1.00</td>
<td>2.27</td>
<td>4,14</td>
</tr>
<tr>
<td>Grassshader</td>
<td>1.00</td>
<td>2.26</td>
<td>4,15</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>1.00</td>
<td>2.43</td>
<td>4,83</td>
</tr>
<tr>
<td>Libor</td>
<td>1.00</td>
<td>3.51</td>
<td>6,61</td>
</tr>
<tr>
<td>RTM-stencil</td>
<td>1.00</td>
<td>3.91</td>
<td>6,06</td>
</tr>
<tr>
<td>Geomean</td>
<td>1.00</td>
<td>2.74</td>
<td>4,92</td>
</tr>
</tbody>
</table>

Configuration: Intel* Xeon* CPU E3-1270 @ 3.50 GHz Haswell system (4 cores with Hyper-Threading On), running at 3.50GHz, with 32.0GB RAM, L1 Cache 256KB, L2 Cache 1.0MB, L3 Cache 8.0MB, 64-bit Windows* Server 2012 R2 Datacenter. Compiler options: SSE4.2: –O3 –Qopenmp -simd –QsSE4.2 or AVX2: -O3 –Qopenmp -simd –QxCORE-AVX2. For more information go to http://www.intel.com/performance

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 to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.
INTEL® DISTRIBUTION FOR PYTHON*
LIBRARIES

Intel® Math Kernel Library
Intel® Data Analytics Acceleration Library
Intel® Integrated Performance Primitives
Intel® Threading Building Blocks
INTEL® THREADING BUILDING BLOCKS
# Rich Feature Set for Parallelism

**Intel® Threading Building Blocks (Intel® TBB)**

<table>
<thead>
<tr>
<th>Generic Parallel Algorithms</th>
<th>Flow Graph</th>
<th>Concurrent Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient scalable way to exploit the power of multi-core without having to start from scratch.</td>
<td>A set of classes to express parallelism as a graph of compute dependencies and/or data flow</td>
<td>Concurrent access, and a scalable alternative to containers that are externally locked for thread-safety</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task Scheduler</th>
<th>Concurrent Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophisticated work scheduling engine that empowers parallel algorithms and the flow graph</td>
<td>Synchronization Primitives</td>
</tr>
<tr>
<td></td>
<td>Atomic operations, a variety of mutexes with different properties, condition variables</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow Graph</th>
<th>Concurrent Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Synchronization Primitives</td>
</tr>
<tr>
<td></td>
<td>Atomic operations, a variety of mutexes with different properties, condition variables</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory Allocation</th>
<th>Timers and Exceptions</th>
<th>Threads</th>
<th>Thread Local Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalable memory manager and false-sharing free allocators</td>
<td>Thread-safe timers and exception classes</td>
<td>OS API wrappers</td>
<td>Efficient implementation for unlimited number of thread-local variables</td>
</tr>
</tbody>
</table>

**Parallel algorithms and data structures**

- Threads and synchronization
- Memory allocation and task scheduling

---

*Other names and brands may be claimed as the property of others.*
Intel® TBB: Scalability and Productivity

Excellent Performance Scalability with Intel® Threading Building Blocks 2017 on Intel® Xeon® Processor

Configuration Info: Software Versions: Intel® C++ Compiler (Version 17.0), Intel® Threading Building Blocks (Intel® TBB) 2017; Hardware: Intel® Xeon® Processor E5-2699 v4 @ 2.20GHz 44 (88T), 128GB Main Memory; Operating System: Red Hat Enterprise Linux Server release 7.2 (Maipo), kernel 3.10.0-327.4.5.el7.x86_64; Benchmark Source: Intel Corp. Note: sudoku, primes and tachyon are included with Intel TBB. Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark limitations.htm.

* Other brands and names are the property of their respective owners.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.
What’s New: Intel® Threading Building Blocks 2017

static_partitioner class
- Helps minimizing overhead of parallel loops

streaming_node class
- Enables heterogeneous streaming computations within the flow graph.

Added method to isolate execution of a group of tasks or an algorithm from other tasks submitted to the scheduler. A preview feature for 2017.

Python* module is added to replace Python's thread pool class.

Graph/stereo example is added.

Improvements to graph/fgbzzip example (added async_msg usage example)
INTEL® INTEGRATED PERFORMANCE PRIMITIVES
# Intel® IPP Domain Applications

<table>
<thead>
<tr>
<th>Image Processing</th>
<th>Signal Processing</th>
<th>Data Compression and Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Medical imaging</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Computer vision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Digital surveillance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Biometric identification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Automated sorting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• ADAS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Visual search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Games (sophisticated audio content or effects)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Echo cancellation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Telecommunications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Energy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Data centers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Enterprise data managements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• ID verification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Smart cards/wallets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Electronic signature</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Information security/cybersecurity</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intel® IPP Data Compression and Decompression Performance Boost vs. ZLIB Library

**Optimization Notice:** Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor–dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.
What’s New: Intel® Integrated Performance Primitives 2017

- Extended optimization for Intel® AVX-512 and Intel® Xeon® and Intel® Xeon Phi™ processors and coprocessors
- Intel® IPP Platform-Aware APIs in the image and signal processing domains are added to support external threading and 64-bit data length
- Significantly improved performance of zlib compression functions is Extension of IPP optimized functionality in OpenCV
- Limited pre-silicon optimizations for next generation Intel Xeon Phi and CNL EP/XE server
INTEL SOFTWARE ANALYSIS TOOLS

Intel® VTune™ Amplifier XE Performance Profiler
Intel® Inspector XE Memory & Thread Debugger
Intel® Advisor XE Vectorization Optimization and Thread Prototyping
INTEL® PERFORMANCE SNAPSHOTS
## Application Performance Snapshot

**Discover Opportunities for Better Performance with Vectorization and Threading**

### Objectives
- Simple enough to run during a coffee break
- Highlight where code modernization can help

### Users
- Performance teams – fast prioritization of which apps will benefit most
- All Developers – size the potential performance gain from code modernization

### Non-Objectives
- Actionable tuning data – that is another tool. Snapshot is just a fast “health” check.

### Free download:
http://www.intel.com/performance-snapshot

Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.
**MPI Performance Snapshot**

Your application is OpenMP bound. High OpenMP imbalance has been identified. Use Intel VTune Amplifier for further analysis.

**Wallclock time**

1.78 sec

**Calculation**

45.38%

**OpenMP**

30.53%

**OpenMP Imbalance**

12.90%

**MPI**

54.62%

**Memory usage**

- Peak: 25.29 MB
- Real: 20.95 MB

Per-process memory usage affects the application scalability.

**Cycles Per Instruction Rate**

- Max: 2.23
- Min: 1.19

This could be caused by such issues as memory stalls, instruction stalling, branch misprediction or long latency instructions.

**Memory Bound Coefficient**

- Max: 0.23
- Min: 0.18

It indicates that the application doesn't spend much time waiting for data. High values are usually bad. The application is not Memory Bound.

**Optimization Notice**

Storage Performance Snapshot
Discover if Faster Storage can Improve Server/Workstation Performance

Learn It On One Coffee Break
- Easy setup
- Quickly see meaningful data
- System view of workload
- Any architecture

Targeted Systems
- Servers and workstations with directly attached storage
- Not scale out storage clusters
- Linux kernel 2.6 or newer, dstat 0.7 or newer
- Windows Server* 2012, Windows* 8, or newer Windows OS

Free download: http://www.intel.com/performance-snapshot
Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.
INTEL® VTUNE™ AMPLIFIER XE
PERFORMANCE PROFILER
Intel® VTune™ Amplifier Tunes Knights Landing Processors

Four Critical Optimizations for Intel® Xeon Phi™ Processors

1) High-bandwidth memory
   - Decide which data structures to place in MCDRAM.
   - See performance problems by memory hierarchy.
   - Measure DRAM and MCDRAM bandwidth.

2) Scalability of MPI* and OpenMP*
   - Serial versus parallel time
   - Imbalance, overhead cost, parallel loop parameters

3) Micro architecture efficiency
   - See the efficiency of your code in the core pipeline.
   - Zero in on details with custom PMU events.

4) Vectorization efficiency: Use Intel® Advisor
   - Optimize for Intel® AVX-512 with or without Intel® AVX-512 hardware.
Optimize Memory Access

Memory Access Analysis: Intel® VTune™ Amplifier 2017

Tune data structures for performance
- Attribute cache misses to data structures (not just the code causing the miss)
- Support for custom memory allocators

Optimize NUMA latency and scalability
- True and false sharing optimization
- Auto detect max system bandwidth
- Easier tuning of inter-socket bandwidth

Easier install, latest processors
- No special drivers required on Linux*
- Intel® Xeon Phi™ processor MCDRAM (high-bandwidth memory) analysis

*Other names and brands may be claimed as the property of others.
Storage Device Analysis (HDD, SATA, or NVMe SSD)

Intel® VTune™ Amplifier

Are you I/O bound or CPU bound?

- Explore imbalance between I/O operations (async and sync) and compute.
- Storage accesses mapped to the source code.
- See when CPU is waiting for I/O.
- Measure bus bandwidth to storage.

Latency analysis

- Tune storage accesses with latency histogram.
- Distribution of I/O over multiple devices.
Intel® Performance Snapshots

Three Fast Ways to Discover Untapped Performance

Is your application making good use of modern computer hardware?

- Run a test case during your coffee break.
- High-level summary shows which apps can benefit most from code modernization and faster storage.

Pick a performance snapshot:
- **Application**: For non-MPI apps
- **MPI**: For MPI apps
- **Storage**: For systems, servers, and workstations with directly attached storage.

Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.
New for 2017: Python*, FLOPS, Storage, and More...

Intel® VTune™ Amplifier Performance Profiler

- Profile Python* and Mixed Python / C++ / Fortran*
- Tune latest Intel® Xeon Phi™ processors
- Quickly see three keys to HPC performance
- Optimize memory access
- Storage analysis: I/O bound or CPU bound?
- Enhanced OpenCL* and GPU profiling
- Easier remote and command line usage
- Add custom counters to the timeline
- Preview: Application and storage performance snapshots
- Intel® Advisor: Optimize vectorization for Intel® AVX-512 (with or without hardware)
Find and Debug Memory and Threading Errors

Intel® Inspector: Memory and Thread Debugger

Correctness tools increase ROI by 12%-21%¹
- Errors found earlier are less expensive to fix.
- Several studies, ROI% varies, but earlier is cheaper.

Diagnosing some errors can take months
- Races and deadlocks not easily reproduced.
- Memory errors hard to find without a tool.

Debugger integration speeds diagnosis
- Breakpoint set just before the problem.
- Examine variables and threads with the debugger.

Debugger Breakpoints

Diagnose in Hours Instead of Months

¹ Cost Factors – Square Project Analysis
CERT: U.S. Computer Emergency Readiness Team, and Carnegie Mellon CyLab NIST: National Institute of Standards & Technology : Square Project Results

Part of Intel® Parallel Studio XE Professional Ed. For Windows* and Linux* From $1,599

Peter von Kaenel, Director, Software Development, Harmonic Inc.

Intel® Inspector dramatically sped up our ability to track down difficult to isolate threading errors before our packages are released to the field.
New for 2017: New Processors, New C++ Language Features
Intel® Inspector 2017: Memory and Thread Debugger

New C++ language features
- Full C++ 11 support including `std::mutex` and `std::atomic`

Easier identification of threading bugs
- Variable name causing error is shown (global, static, and stack) in addition to the code lines

Run native on Intel® Xeon Phi™ processors
- This simplifies workflow for Intel Xeon Phi processor development
- Tip: Reduce thread count to ≤ 30 for best KNL performance while running Intel® Inspector.
INTEL® ADVISOR XE
VECTORIZATION OPTIMIZATION AND THREAD PROTOTYPING
FOR SOFTWARE ARCHITECTS
New for 2017: AVX-512, FLOPS, & More...

Intel® Advisor: Vectorization Optimization

- Next-gen Intel® Xeon Phi™ support
- Tune for Intel® AVX-512 with or without Intel AVX-512 hardware
- Precise FLOPS calculation
- Enhanced memory access analysis
- Easier selection of high-impact loops
- Batch mode workflow saves time
- Fast answers with loop aAnalytics
CLUSTER TOOLS

Intel® MPI Library

Intel® Trace Analyzer and Collector
Optimized MPI application performance

- Application-specific tuning
- Automatic tuning
- New! - Support for Intel® Xeon Phi™ processor (code-named Knights Landing)
- New: Support for Intel® Omni-Path Architecture Fabric

Lower-latency and multi-vendor interoperability

- Industry leading latency
- Performance optimized support for the fabric capabilities through OpenFabrics*(OFI)

Faster MPI communication

- Optimized collectives

Sustainable scalability up to 340K cores

- Native InfiniBand* interface support allows for lower latencies, higher bandwidth, and reduced memory requirements

More robust MPI applications

- Seamless interoperability with Intel® Trace Analyzer and Collector
What’s New: Intel® MPI Library 2017

- Support for Intel® Xeon Phi™ processors (code-named Knights Landing (KNL))
- Support for Intel® Omni-Path Architecture fabric
- Usage of specially optimized memcpy for KNL
- Tuning of shared memory collectives on single KNL nodes
- General optimization of RMA
- General optimization and speed up startup time and MPI tune utility
Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling statistics and load balancing
- Identify communication hotspots

Features

- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Idealizer

Automatically detect performance issues and their impact on runtime
**Lightweight**: Low overhead profiling for 100K+ ranks.

**Scalable**: Performance variation at scale can be detected sooner.

**Identify Key Metrics**: Shows MPI*/OpenMP* imbalances.
What’s New: Intel® Trace Analyzer and Collector

- Intel Trace Analyzer and Collector will be ready for KNL
- Improved scalability of imbalance profiler by up to 10x
- Improved MPI Snapshot feature HTML output
Additional Material

- **Product page** – overview, features, FAQs, support...
- **Training materials** – movies, tech briefs, documentation...
- **Evaluation guides** – step by step walk through
- **Reviews**

Additional Development Products:

- **Intel® Software Development Products**