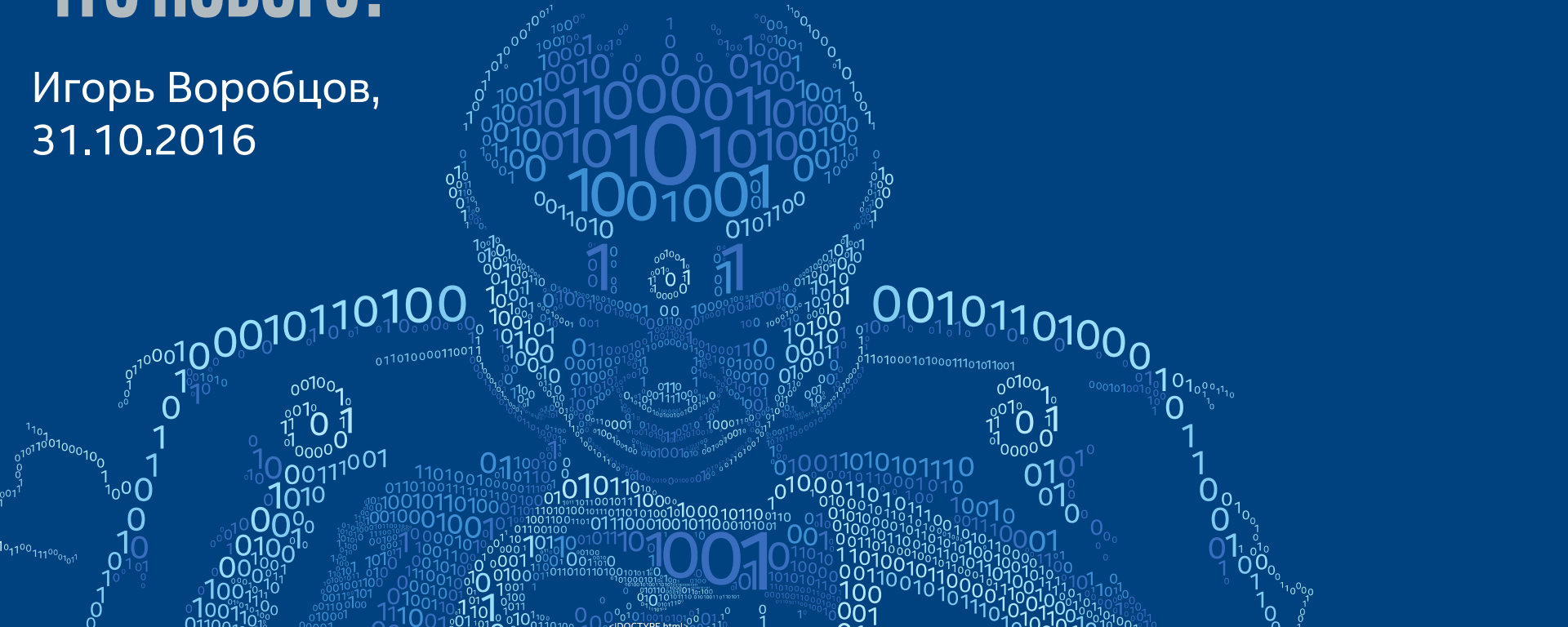
INTEL® PARALLEL STUDIO XE 2017

ЧТО НОВОГО?

Игорь Воробцов,
31.10.2016

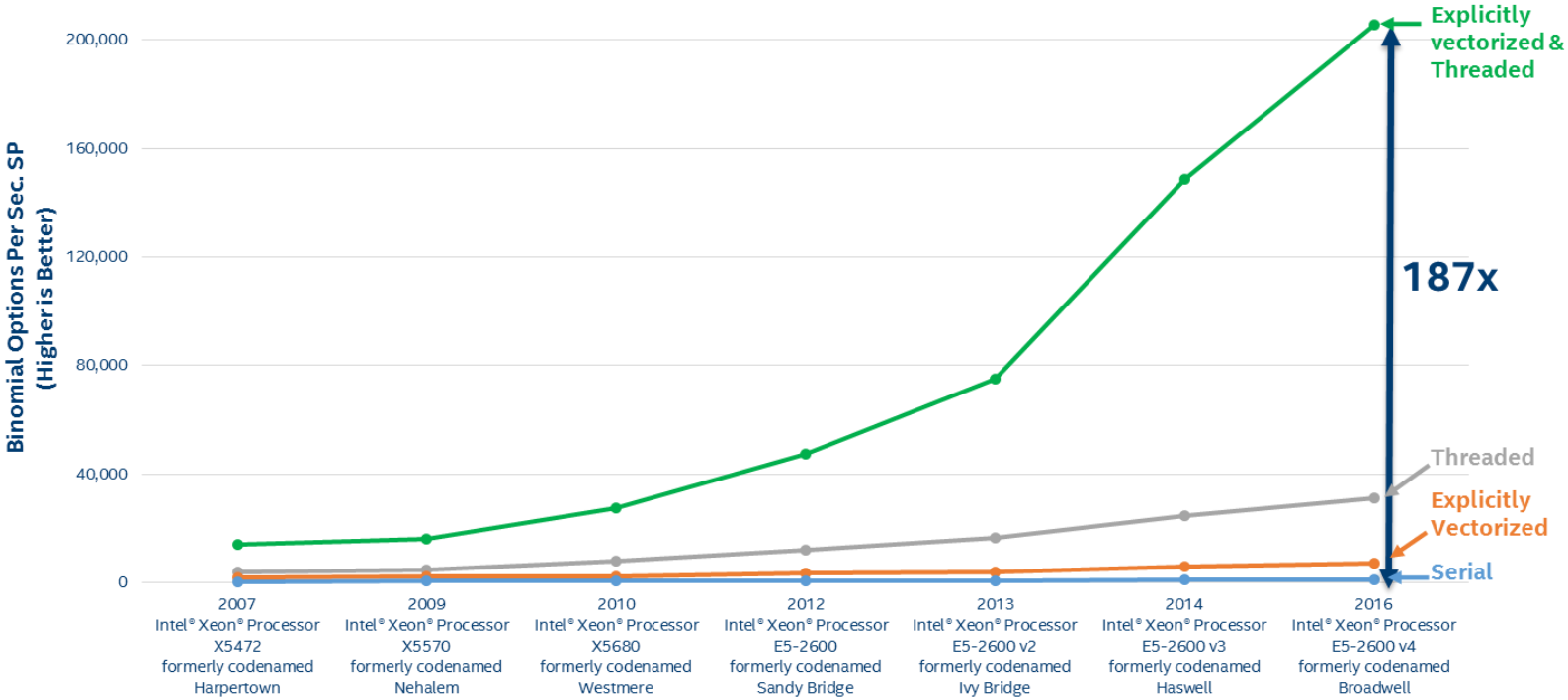


intel Software

INTEL® PARALLEL STUDIO XE 2017

		Composer Edition	Professional Edition	Cluster Edition
BUILD	**Компилятор Intel® C++**	√	√	√
	Компилятор Intel® Fortran	√	√	√
	Intel® Distribution for Python*	√	√	√
	Intel® Math Kernel Library – *быстрая математическая библиотека*	√	√	√
	Intel® Integrated Performance Primitives – *библиотека для обработки данных и мультимедиа*	√	√	√
	Intel® Threading Building Blocks – *библиотека шаблонов C++*	√	√	√
	Intel® Data Analytics Acceleration Library – *машинное обучение & аналитика*	√	√	√
ANALYZE	Intel® VTune™ Amplifier – *профилировщик производительности*		√	√
	Intel® Advisor – *векторизация и прототипирование параллелизма*		√	√
	Intel® Inspector – *отладка проблем с памятью и потоками*		√	√
CLUSTER	Intel® MPI Library – *message passing interface library*			√
	Intel® Trace Analyzer and Collector – *средство анализа MPI приложений*			√
	Intel® Cluster Checker – *система проектирования и диагностики кластера*			√
	Rogue Wave IMSL* Library – *библиотека численного анализа (Фортран)*	**Пакет & Аддон**	**Аддон**	**Аддон**

МОДЕРНИЗИРУЙ СВОЙ КОД!

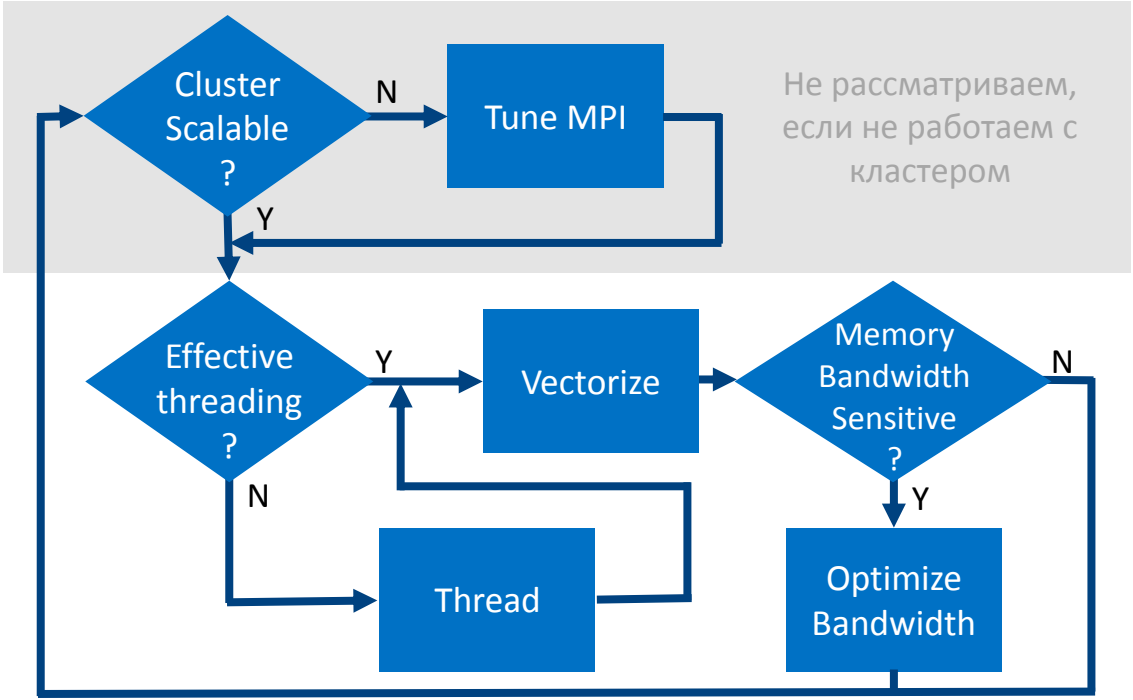


Разница увеличивается с каждым новым поколением «железа»

Конфигурация для Binomial Options SP в конце презентации

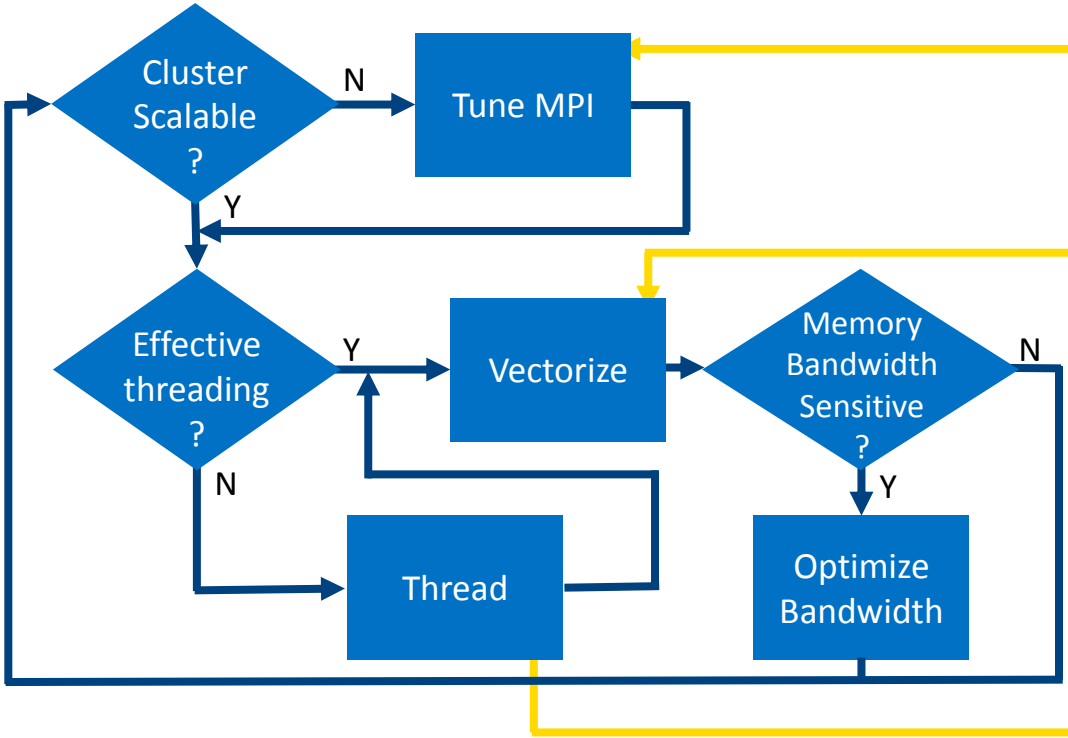
ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ

ИТЕРАТИВНЫЙ ПРОЦЕСС…



ИНСТРУМЕНТЫ ДЛЯ РЕАЛИЗАЦИИ

INTEL® PARALLEL STUDIO XE



Intel® MPI Library
Intel® MPI Benchmarks

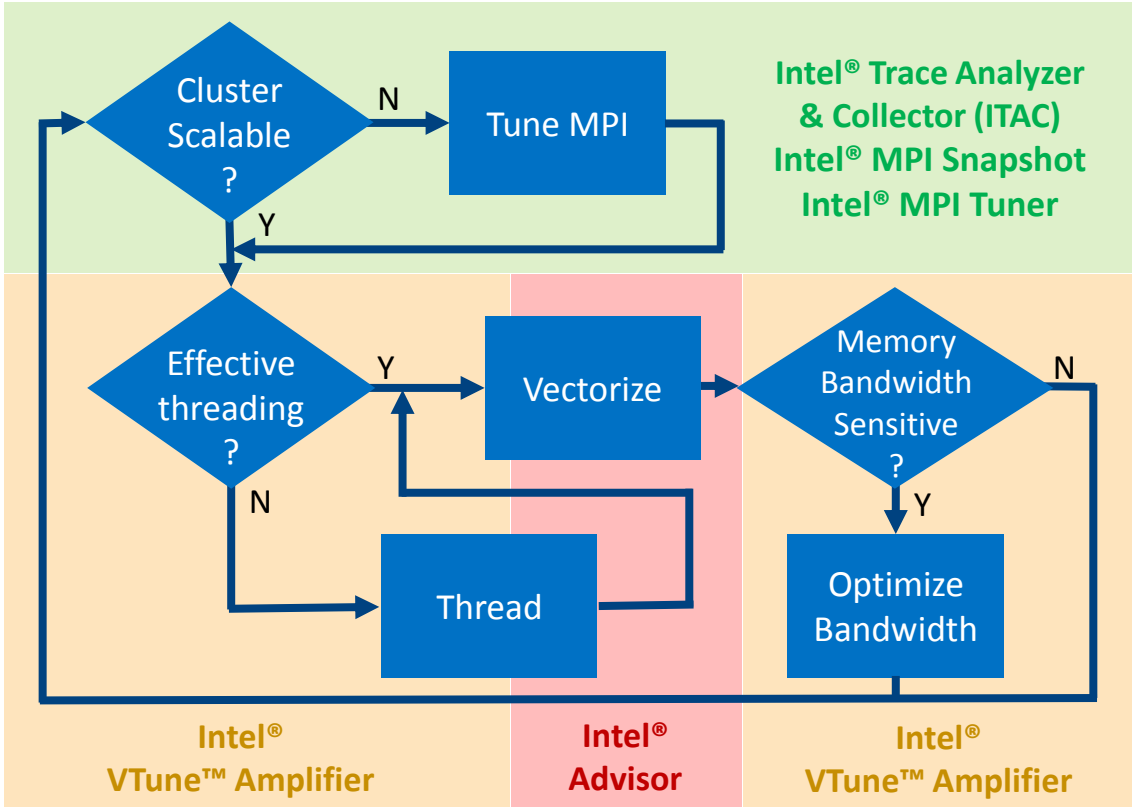
Intel® Compiler

Intel® Math Kernel Library
Intel® IPP – Media & Data Library
Intel® Data Analytics Library
Intel® Cilk™ Plus
Intel® OpenMP*

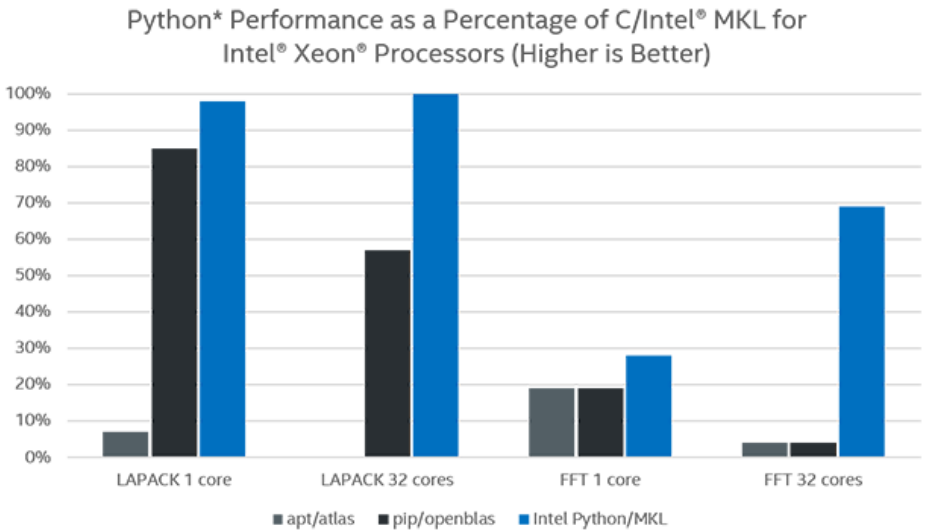
Intel® TBB – Threading Library

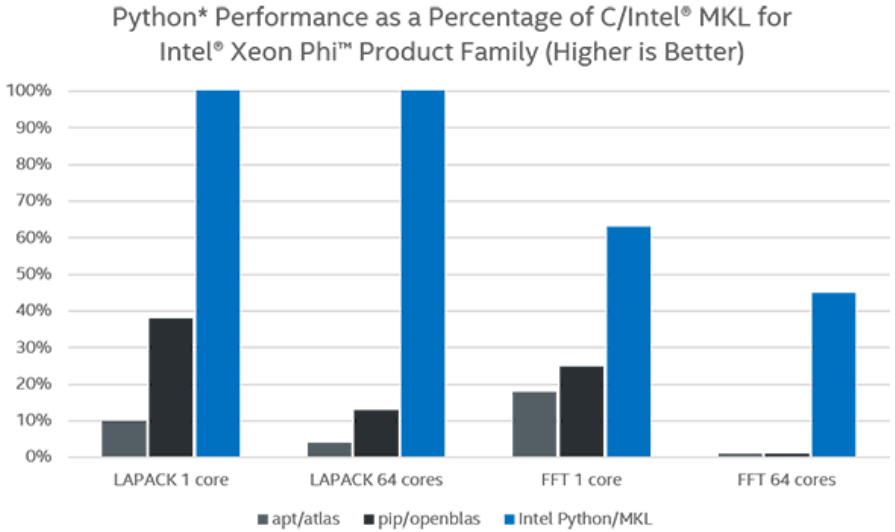
СРЕДСТВА АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ

INTEL® PARALLEL STUDIO XE



INTEL® DISTRIBUTION FOR PYTHON*





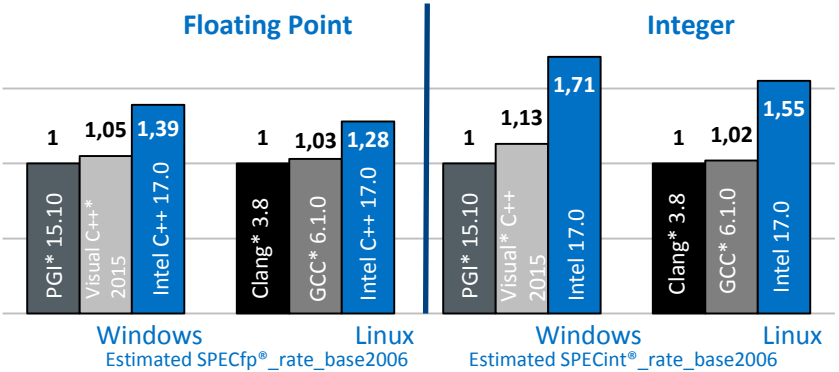
НОВЫЕ ВОЗМОЖНОСТИ КОМПИЛЯТОРА

- Векторизация
- OpenMP* 4.5
- Более широкая поддержка стандартов C++ и Фортран
- Intel® MPX
- Улучшения в Intel Graphics Technology
- Библиотеки
- и многое другое…

ВЫСОКАЯ ПРОИЗВОДИТЕЛЬНОСТЬ НА WINDOWS* И LINUX*

КОМПИЛЯТОРЫ C++ И FORTRAN

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



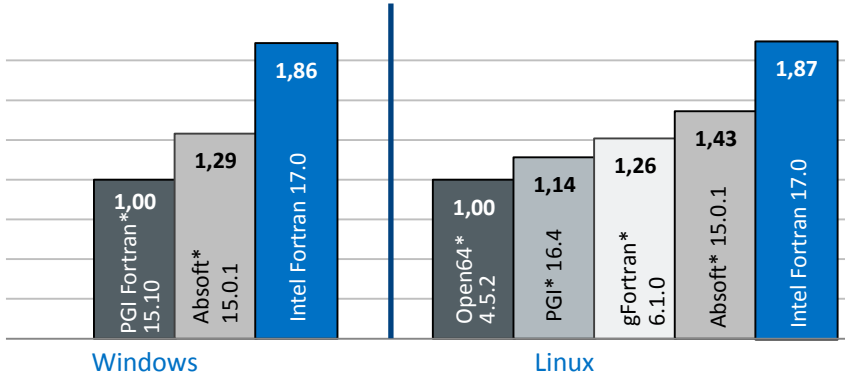
Relative geomean performance, SPEC* benchmark - higher is better

Configuration: Windows hardware: Intel(R) Xeon(R) CPU E3-1245 v5 @ 3.50GHz, HT enabled, TB enabled, 32 GB RAM; Linux hardware: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 256 GB RAM, HyperThreading is on.
Software: Intel compilers 17.0, Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23918 for x86/x64, GCC 6.1.0. PGI 15.10, Clang/LLVM 3.8
Linux OS: Red Hat Enterprise Linux Server release 7.1 (Maipo), kernel 3.10.0-229.el7.x86_64. Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240).
SPEC* Benchmark (www.spec.org). SmartHeap libs 11.3 for Visual* C++ and Intel Compiler were used for SPECint* benchmarks.

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 .

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



Relative geomean performance, Polyhedron* benchmark– higher is better

Configuration: Hardware: Intel(R) Xeon(R) CPU E3-1245 v5 @ 3.50GHz, Hyperthreading enabled, TB enabled, 32 GB RAM. Software: Intel Fortran compiler 170, Absoft*15.0.1,. PGI Fortran* 15.10 (Windows)/16.4 (Linux), Open64* 4.5.2, gFortran* 6.1.0. Linux OS: Red Hat Enterprise Linux Server release 7.2, Kernel 3.10.0-327.4.5.el7.x86_64. Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240). Polyhedron Fortran Benchmark (www.fortran.uk).
Windows compiler switches: Absoft: -m64 -O5 -speed_math=10 -fast_math -march=core -xINTEGER -stack:0x80000000. Intel® Fortran compiler: /fast /Qparallel /QxCORE-AVX2 /nostandard-realloc-lhs /link /stack:64000000. PGI Fortran: -fastsse -Munroll=n:4 -Mipa=fast,inline -Mconcur=numa.
Linux compiler switches: Absoft: -m64 -mavx -O5 -speed_math=10 -march=core -xINTEGER. Gfortran: -Ofast -mfpmath=sse -flto -march=native -funroll-loops -ftree-parallelize-loops=4. Intel Fortran compiler: -fast -parallel -xCORE-AVX2 -nostandard-realloc-lhs. PGI Fortran: -fast -Mipa=fast,inline -Msmartalloc -Mfprelaxed -Mstack_arrays -Mconcur=bind. Open64: -march=auto -Ofast -mso –apo .

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 .

КОМПИЛЯТОР INTEL® C++

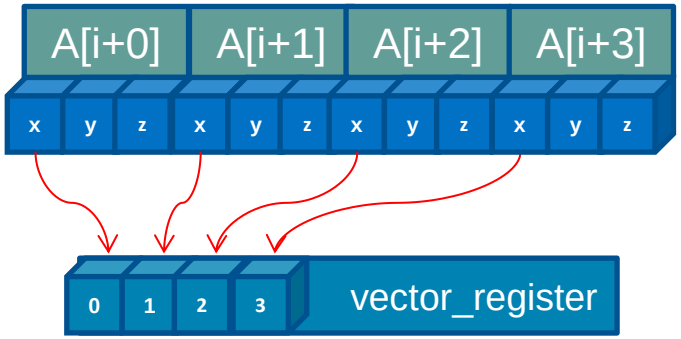
- Библиотека SIMD Data Layout Templates
- Векторизация виртуальных функций
- Расширенная поддержка стандартов C11 и C++14
  - Освобождение памяти определенного размера
  - Упрощенные ограничения на создание constexpr функций
  - Шаблоны переменных
  - Одинарная кавычка как цифровой разделитель
- Улучшена 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

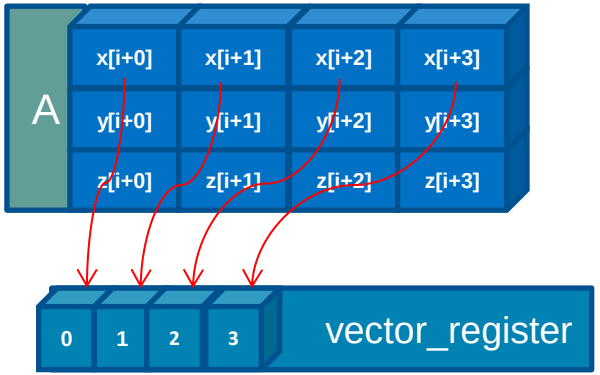
SIMD DATA LAYOUT TEMPLATE
УВЕЛИЧИВАЕМ ПРОИЗВОДИТЕЛЬНОСТЬ C++ ПРИЛОЖЕНИЙ

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

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

‘Array of Structures’ (AOS) vs ‘Structure of Arrays’ (SOA)





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

```
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); }
};
```

ВЫСОКАЯ ПРОИЗВОДИТЕЛЬНОСТЬ

ВЕКТОРИЗАЦИЯ СРЕДСТВАМИ OPENMP*

- Всего 2 строчки кода для использования всех возможностей SSE и AVX
- Директивы игнорируются другими компиляторами, если не поддерживаются

```
typedef float complex fcomplex;
 const uint32_t max_iter = 3000;

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;

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

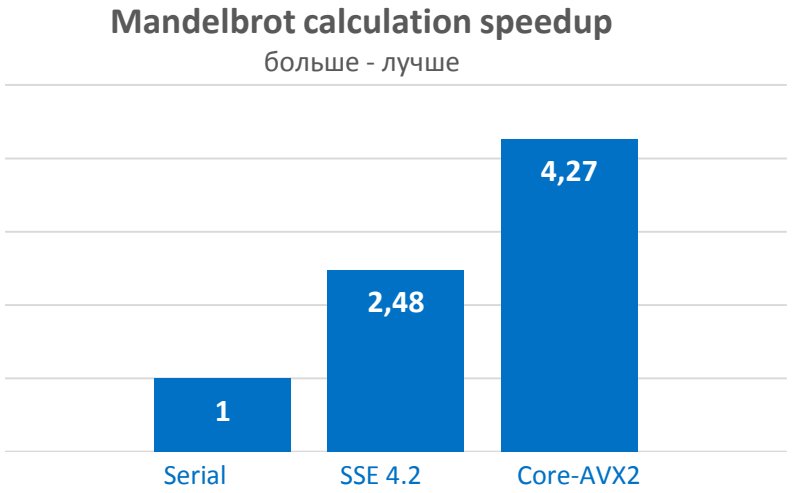
```
typedef float complex fcomplex;
 const uint32_t max_iter = 3000;
#pragma omp declare simd uniform(max_iter), simdlen(16)
uint32_t mandel(fcomplex c, uint32_t max_iter)
{
   uint32_t count = 1; fcomplex z = c;
   while ((cabsf(z) < 2.0f) && (count < max_iter)) {
     z = z * z + c; count++;
   }
   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 2 строчки кода для использования всех возможностей SSE и AVX
- Директивы игнорируются другими компиляторами, если не поддерживаются

```
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);
      }
    }
```



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 –QxSSE4.2 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 .

ВЫСОКАЯ ПРОИЗВОДИТЕЛЬНОСТЬ

ВЕКТОРИЗАЦИЯ СРЕДСТВАМИ OPENMP*

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

```
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

```
#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;
  }
}
```

ВЫСОКАЯ ПРОИЗВОДИТЕЛЬНОСТЬ

ВЕКТОРИЗАЦИЯ СРЕДСТВАМИ OPENMP*

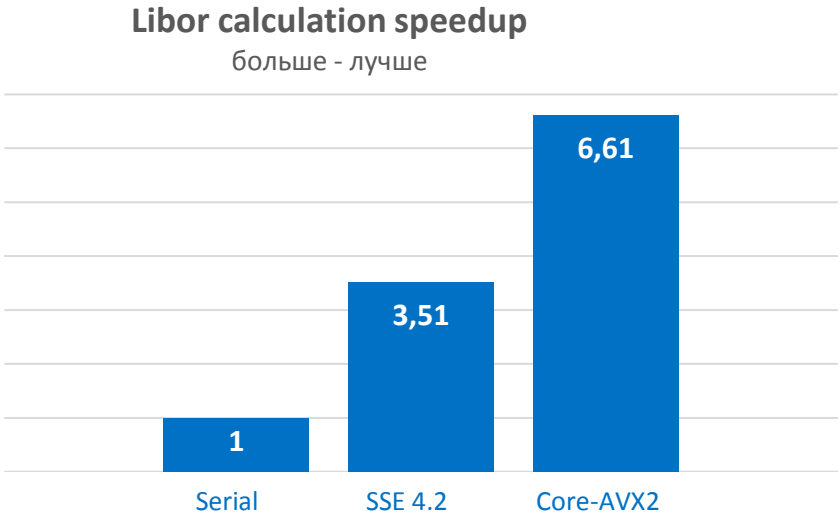
- Всего 3 строчки кода для использования всех возможностей SSE и AVX
- Директивы игнорируются другими компиляторами, если не поддерживаются

```
#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;
  }
}
```



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 –QxSSE4.2 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 .

СТАНДАРТ C++14

- Освобождение памяти определенного размера

```
T* p = new T;
…
delete p;      // C++11
operator delete(p, sizeof(T));  // C++14
```

- Упрощенные ограничения на создание constexpr функций

```
constexpr int prev(int x)
{
     return -x;
}   // now allowed
```

- Шаблоны переменных

```
template<typename T>
constexpr T pi = T(3.14159);
template<class T> T area(T r)
{
     return pi<T> * r * r;
}
```

СТАНДАРТ C++14

Упрощенные ограничения на создание constexpr функций

- Объявление переменных (за исключением static, thread_local и неинициализированных переменных)
- if и switch (но не goto)
- Изменение состояния объектов, являющихся результатом constexpr вычислений
- for (включая range-based for), while и do-while

```
constexpr int uninit() {
      int a; // ошибка:
      // неинициализированная переменная
      return a;
}
```

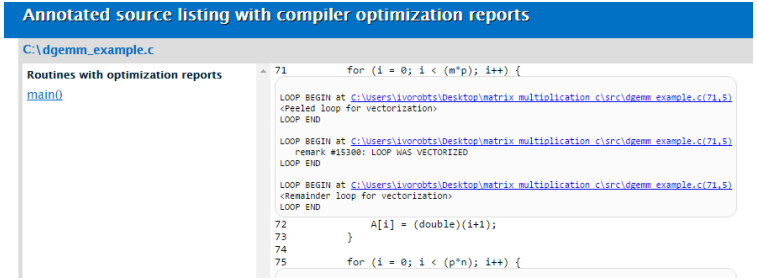
```
constexpr int abs(int x){
      if (x < 0)
            x = -x;
      return x;
}
```

```
constexpr int g(int x, int n) {
      int r = 1;
      while (--n > 0) r *= x;
      return r;
}
```

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

```
vmovups      64(%rsp), %ymm15                    #86.7 [spill]
vfmadd231ps  (%rbx,%rdx,4), %ymm14, %ymm15       #86.7
vmulps       %ymm2, %ymm2, %ymm14                #86.7
vsubps       (%r8,%rdx,4), %ymm8, %ymm2          #86.7
vmovups      %ymm15, 64(%rsp)                    #86.7 [spill]
```

- 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
КЛАУЗА 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

```
!$omp declare simd
REAL FUNCTION FOO(X, Y)
REAL, VALUE :: Y   << by reference
REAL, VALUE :: X   << by reference
FOO = X + Y      << gathers!!!!
END FUNCTION FOO
…
!omp$ simd private(X,Y)
DO I= 0, N
    Y = B(I)
    X = A(I)
    C(I) += FOO(X, Y)
ENDDO
```

OPENMP* 4.5
КЛАУЗА 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

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

OPENMP* 4.5
РЕДУКЦИИ МАССИВОВ В C/C++

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

```
#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

OPENMP* 4.5
ДИРЕКТИВА TASKLOOP

- Параллелизация цикла используя задачи OpenMP

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

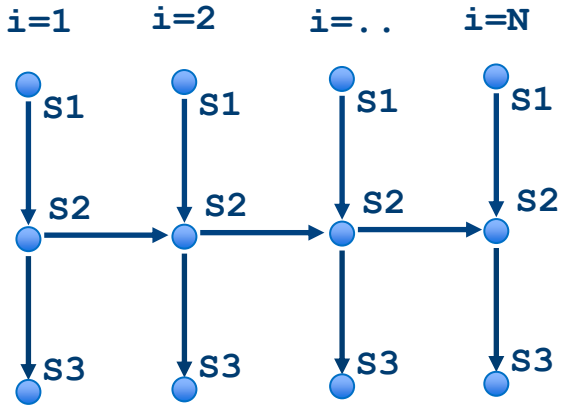
  - Делит цикл на куски (chunk'и)
  - Клауза **grainsize** и **num_tasks** для управления созданием задач
  - Похож на 'cilk_for' из Intel® Cilk Plus
  - Создается задача для каждого куска цикла

```
void CG_mat(Matrix *A, double *x, double *y)
{
  // ...
  #pragma omp taskloop private(j, is, ie, j0, y0) \
              grain_size(500)
  for (i = 0; i < A->n; i++) {
  y0 = 0;
  is = A->ptr[i];
  ie = A->ptr[i + 1];
  for (j = is; j < ie; j++) {
    j0 = index[j];
    y0 += value[j] * x[j0];
  }
  y[i] = y0;
  // ...
}
// ...
```

OPENMP* 4.5
ЦИКЛЫ “DOACROSS”

- Типы зависимостей **source** и **sink** добавлены в клаузу **depend** для поддержки “doacross” циклов вместе с клаузой **ordered**
- **depend(source)** и **depend(sink:<iteration-vector>)**
- Распараллеливание циклов с хорошо структурированными зависимостями

```
#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;
}
```



OPENMP* 4.5
ЦИКЛЫ “DOACROSS”

```
#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 процессоры поддерживают транзакционную память, например Intel® TSX (Intel® Transactional Synchronization Extensions)

Новые функции:

```
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:

```
omp_lock_hint_none
omp_lock_hint_uncontended
omp_lock_hint_contended
omp_lock_hint_nonspeculative
omp_lock_hint_speculative
```

Клауза **hint(type)** для критической секции

C++:

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

Фортран:

```
!$ 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

[Q]check-pointers

[Q]check-pointers-mpx

АСИНХРОННЫЙ ОФФЛОД НА GPU

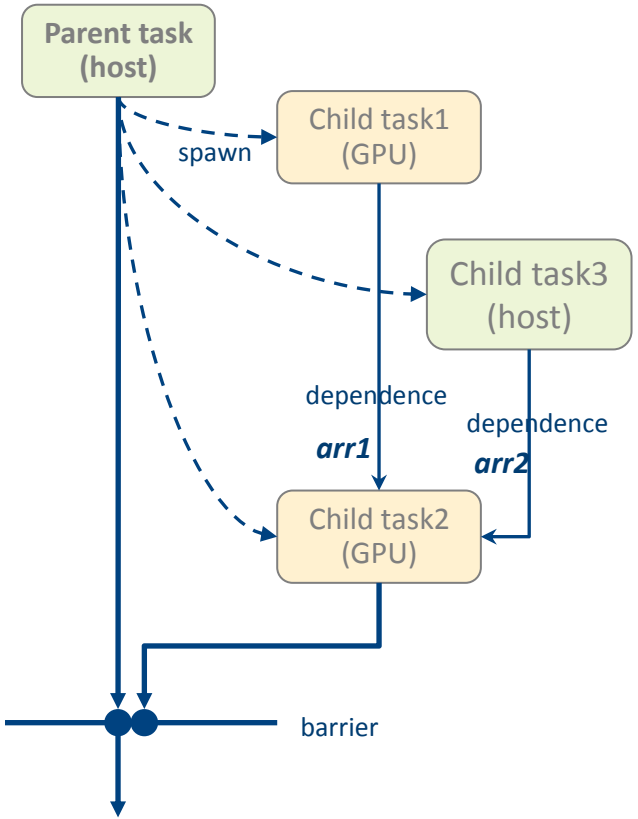
OPENMP КЛАУЗА DEPEND В TARGET ДИРЕКТИВЕ

```
// 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]; }
```



INTEL® ADVISOR

МОДЕЛИРОВАНИЕ ПАРАЛЛЕЛИЗМА ПО ЗАДАЧАМ

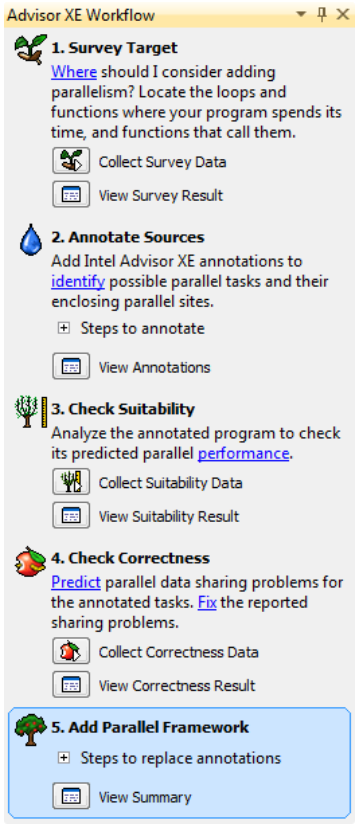
1) Analyze it.

2) Design it.
(Compiler ignores these annotations.)

3) Tune it.

4) Check it.

5) Do it!

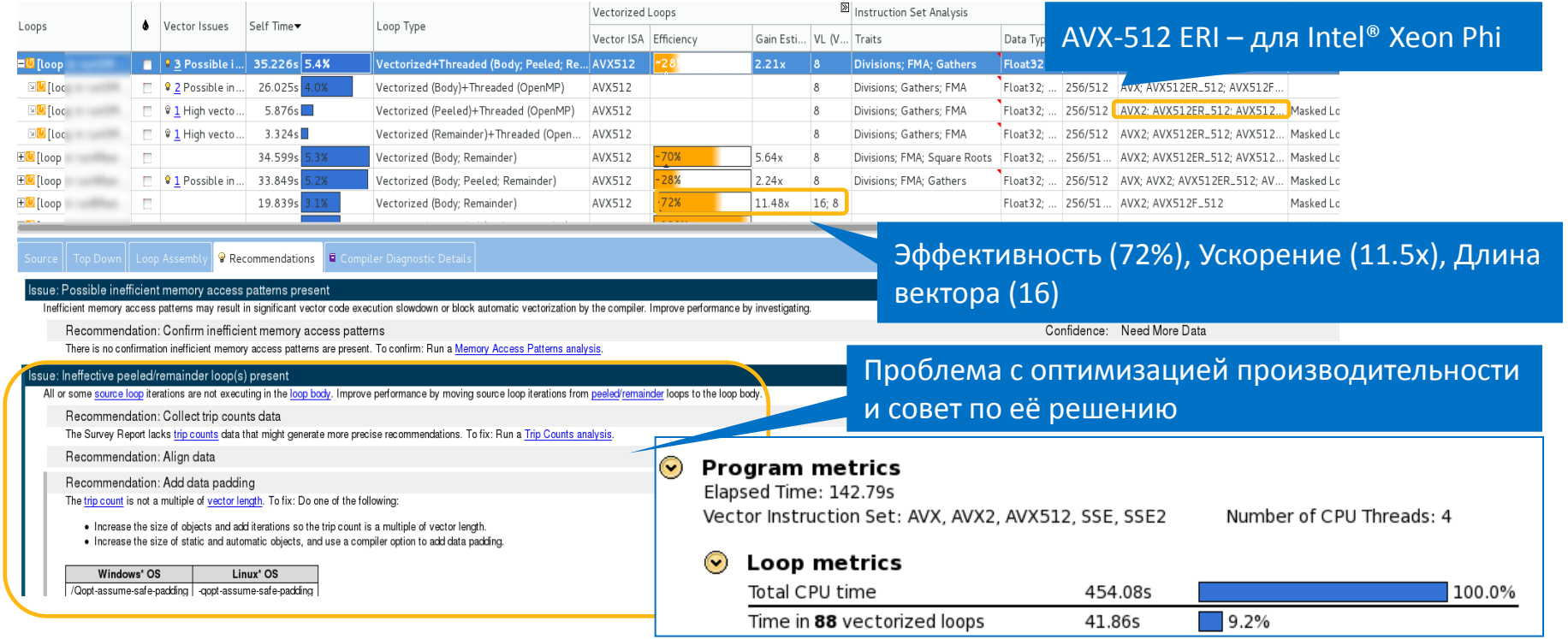


АССИСТЕНТ ВЕКТОРИЗАЦИИ

- Диагностика компилятора + Результаты профилировки = Всё в одном месте
- Советы – Как векторизовать?
- Проверка зависимостей
  - Безопасно ли векторизовать?
- Анализ доступа к памяти
  - Последовательный vs Непоследовательный, невыровненный, …

INTEL® ADVISOR

ПОДДЕРЖКА INTEL® XEON PHI™ ПОСЛЕДНЕГО ПОКОЛЕНИЯ



AVX-512 ERI – для Intel® Xeon Phi

Эффективность (72%), Ускорение (11.5x), Длина вектора (16)

Проблема с оптимизацией производительности и совет по её решению

INTEL® ADVISOR
AVX-512

- Опция компилятора –axCOMMON-AVX512 –xAVX
  - AVX2 code path (для Haswell и более ранних процессоров)
  - AVX-512 code path
- Сравниваем AVX и AVX-512 с помощью Intel Advisor

Loops		Self Time	Loop Type	Vectorized Loops					Instruction Set Analysis				Advanced
				Vect...	Efficiency	Gain...	VL (...	Compiler Es...	Traits	Data T...	Vector W...	Instruction Sets	Vectorization De
[loop in s352_ at loopstl.cpp:5939]		**0,641s**	**Vectorized (Body)**	**AVX2**	**~54%**	**2,15x**	**4**	**2,15x**	**FMA; Inserts**	**Float32**	**128**	**AVX; FMA**	
[loop in s352_ at loopstl.cpp:5939]		n/a	Remainder [Not Executed]				4		FMA				
[loop in s352_ at loopstl.cpp:5939]		0,641s	Vectorized (Body)	AVX2			4	2,15x	Inserts; FMA				
[loop in s352_ at loopstl.cpp:5939]		n/a	Vectorized (Body) [Not Executed]	AVX512			16	3,20x	Gathers; FMA				
[loop in s352_ at loopstl.cpp:5939]		n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	2,70x	Gathers; FMA				
[loop in s125_.Aompparallel_for@ ...		0,496s	Vectorized Versions	AVX2	~100%	13,54x	8	<13,54x	FMA; NT-stores				
[loop in s125_.Aompparallel_for...		n/a	Peeled [Not Executed]				8		FMA				
[loop in s125_.Aompparallel_for...		n/a	Remainder [Not Executed]				8		FMA				
[loop in s125_.Aompparallel_for...		0,465s	Vectorized (Body)	AVX2			8	13,54x					
[loop in s125_.Zompparallel_for...		n/a	Vectorized (Peeled) [Not Executed]	AVX512			16	6,77x	FMA				
[loop in s125_.Zompparallel_for...		n/a	Vectorized (Body) [Not Executed]	AVX512			32	30,61x					
[loop in s125_.Zompparallel_for...		n/a	Vectorized (Remainder) [Not Executed]	AVX512			16	9,78x	FMA				

Inserts (AVX2) vs. Gathers (AVX-512)

Оценка ускорения: 13.5x (AVX2) vs. 30.6x (AVX-512)

INTEL® ADVISOR

ТОЧНОЕ ИЗМЕРЕНИЕ FLOPS'ОВ

- FLOPSы для циклов и функций
- Все последние процессоры Intel (не со-процессоры)
- Инструментация (FLOP) и сэмплирование (время)
- Учитвает операции маскирования AVX-512

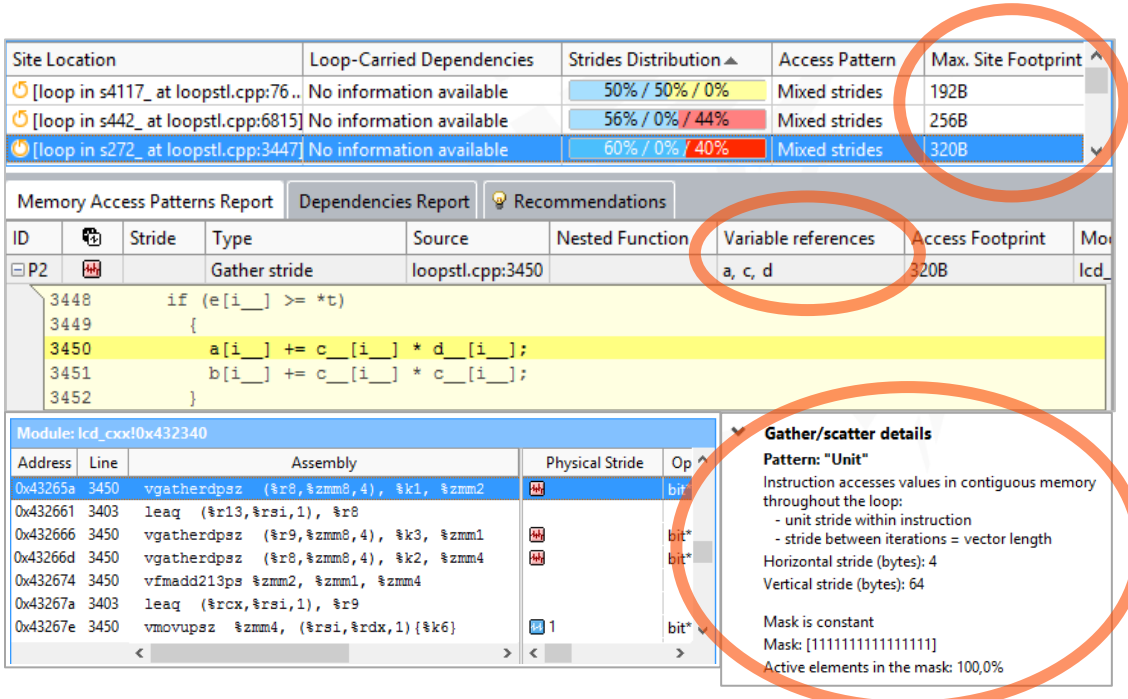
INTEL ADVISOR 2017

Function Call Sites and Loops	FLOPS						
	GFLOPS	AI	L1 GB/s	GFLOP	FLOP Per Iteration	L1 GB	L1 Bytes Per Iteration
[loop in matvec at Multiply.c:69]	0.826	0.1633	5.0586	3.0720	32	18.8160	196
[loop in matvec at Multiply.c:60]	0.912	0.1633	5.5853	3.0720	32	18.8160	196
[loop in matvec at Multiply.c:69]	1.248	0.2500	4.9920	1.3440	4	5.3760	16
[loop in matvec at Multiply.c:60]	1.592	0.2500	6.3699	1.3440	4	5.3760	16
[loop in matvec at Multiply.c:69]	3.055	0.2500	12.2205	0.0960	16	0.3840	64
[loop in matvec at Multiply.c:60]	6.282	0.2500	25.1279	0.0960	16	0.3840	64

INTEL® ADVISOR

УЛУЧШЕННЫЙ АНАЛИЗ ДОСТУПА К ПАМЯТИ

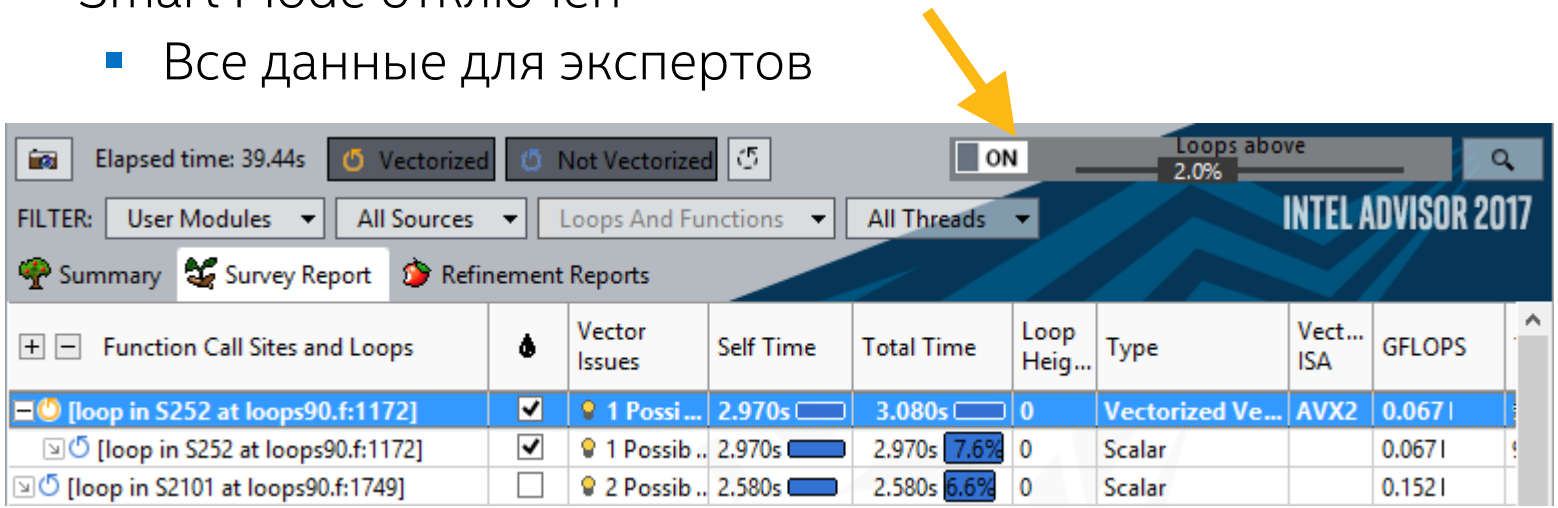
- Подсчет используемой памяти (memory footprint)
- Показывает имена переменных
- Определяет ненужные gather/scatter инструкции



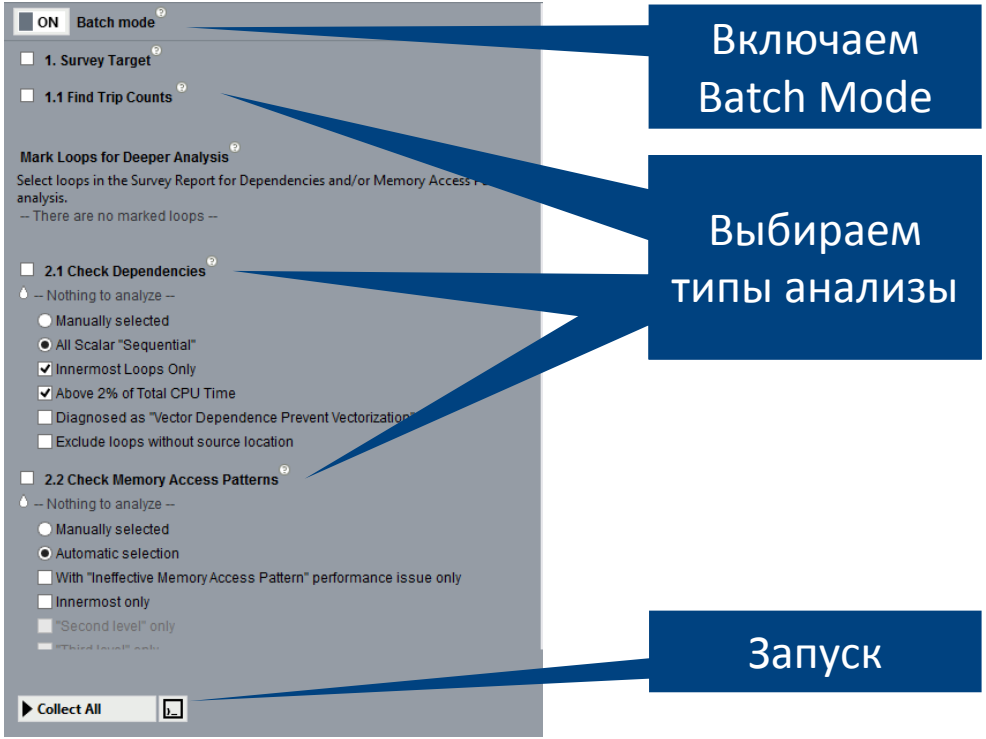
INTEL® ADVISOR

РЕЖИМ SMART

- Smart Mode включен
  - Показывает только «важные» для нас циклы
  - Только необходимые метрики
- Smart Mode отключен
  - Все данные для экспертов



INTEL® ADVISOR
BATCH MODE

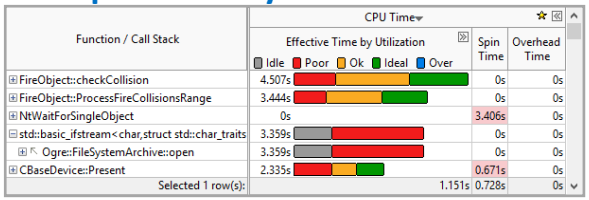


- Запуск нескольких анализов одной кнопкой
- Содержит предустановленные настройки для анализа

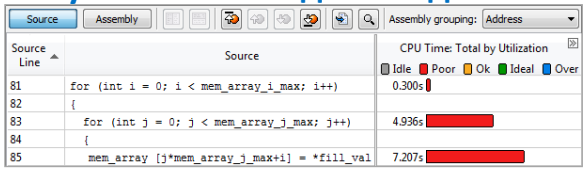
INTEL® VTUNE™ AMPLIFIER XE 2017

- Получаем нужные данные
  - Hotspot, Call counts
  - Профилировка потоков – анализ Concurrency и Lock & Waits
  - Промахи кэша, анализ bandwidth
  - GPU Offload and OpenCL™ Kernel Tracing
- Быстро находим ответы
  - Результаты в исходном коде/ассемблере
  - Анализ масштабируемости OpenMP, анализ кадров для графики
  - Отличные возможности для фильтрации результатов
  - Визуализация активности потоков и задач на временной шкале
- Легко использовать
  - Без специальных компиляторов – C, C++, C#, Fortran, Java, ASM
  - Интеграция в Visual Studio* или свой интерфейс
  - Графический интерфейс & командная строка
  - Локальная & Удаленная профилировка
  - Просмотр данных с Windows* & Linux* на OS X*

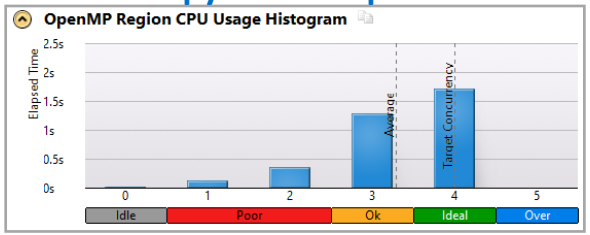
Быстро найти «узкие» места



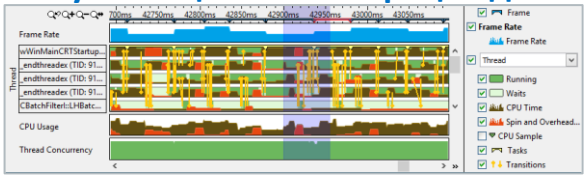
Результаты в исходной коде



Масштабируемость OpenMP

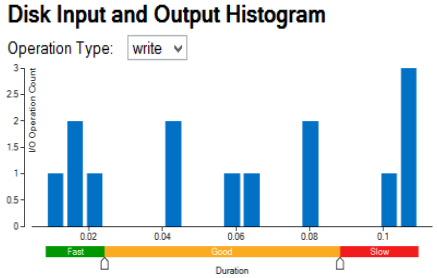


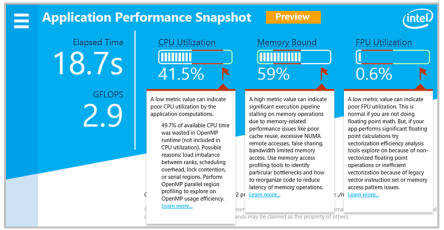
Визуализация & Фильтрация данных



INTEL® VTUNE™ AMPLIFIER XE 2017
ЧТО НОВОГО?

- Профилировка приложений на Python
- Оптимизация под Intel® Xeon Phi™ Knights Landing
- Оценка производительности HPC приложений
- Анализ доступа к памяти
- Анализ дискового ввода-вывода
- Улучшения в профилировке OpenCL™ & GPU
- Простая удалённая профилировка
- Preview: Application & Storage Performance Snapshots

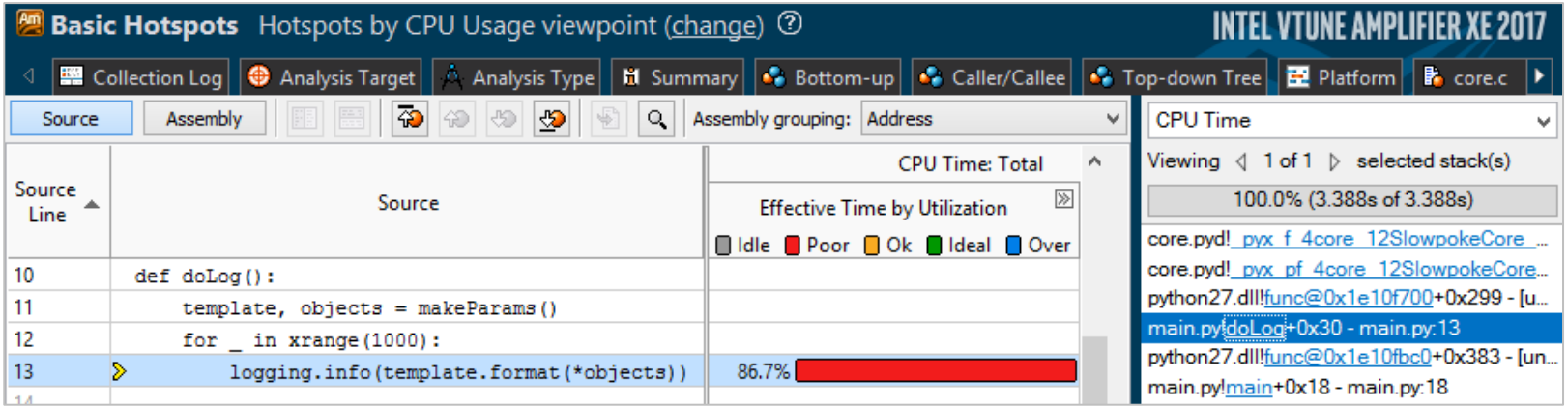




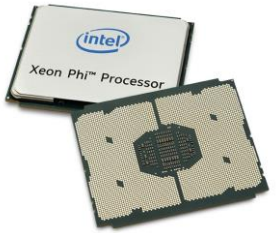
INTEL® VTUNE™ AMPLIFIER XE 2017

ПРОФИЛИРОВКА PYTHON & GO

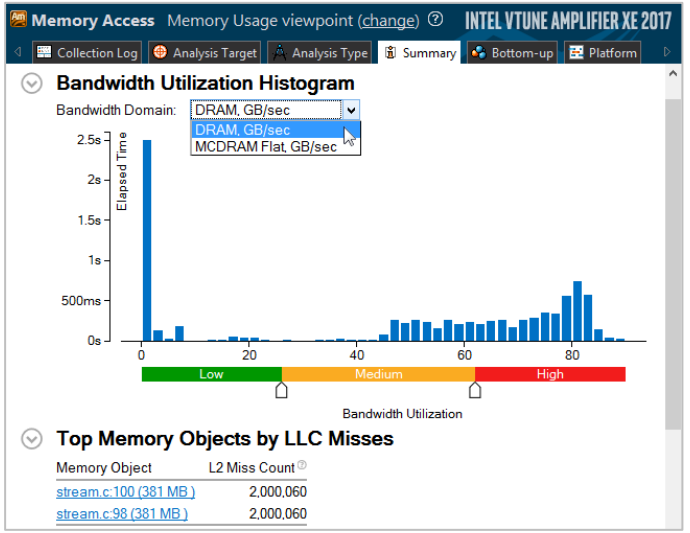
- Сэмлирование с маленьким оверхедом
  - Точные данные без большого оверхеда на инструментацию
  - Запускаем приложение или «подцепляемся» к процессу
- Детали на уровне строчек кода
- «Смешанные» приложения на Python / native C, C++, Fortran…



INTEL® VTUNE™ AMPLIFIER XE 2017
ОПТИМИЗАЦИЯ ПОД INTEL® XEON PHI™ KNIGHTS LANDING



- Память
- Масштабируемость MPI и OpenMP
- Микроархитектурный анализ
- Векторизация – используем Intel® Advisor

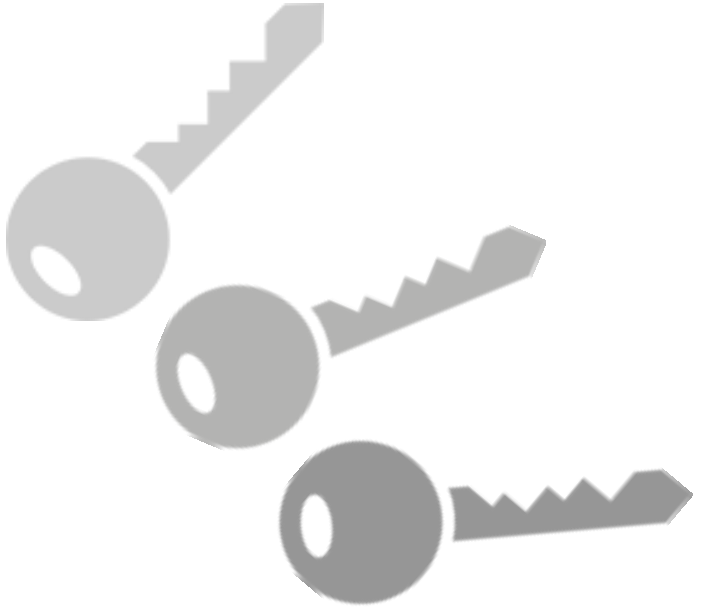


КЛЮЧИ К ВЫСОКОЙ ПРОИЗВОДИТЕЛЬНОСТИ СЕГОДНЯ

1) Threading: CPU Utilization

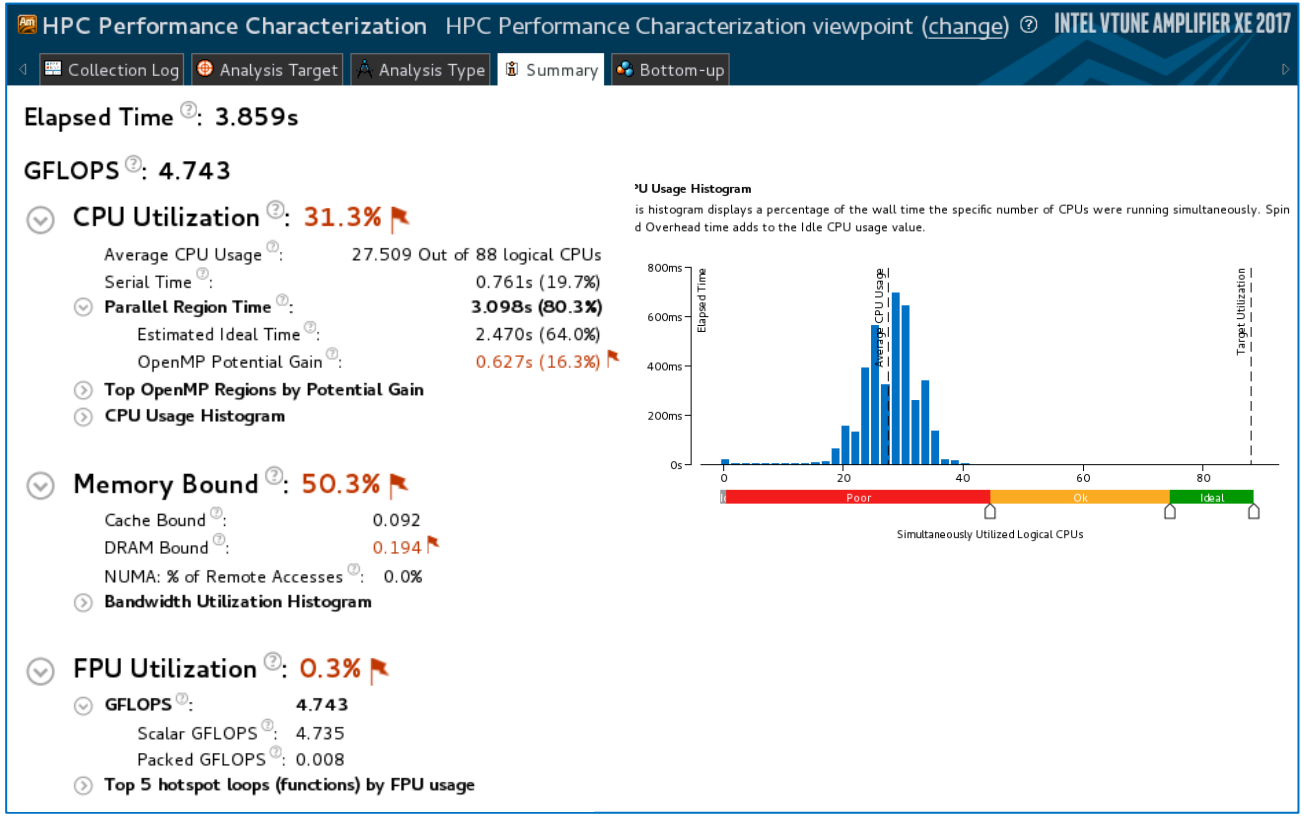
2) Эффективность доступа к памяти

3) Векторизация: FPU Utilization



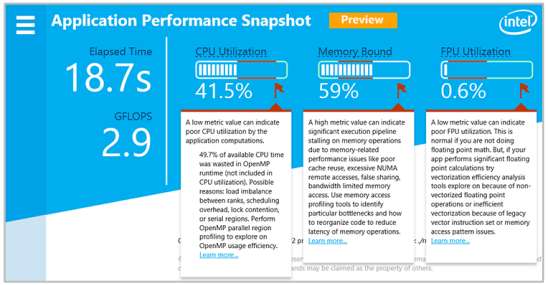
INTEL® VTUNE™ AMPLIFIER XE 2017

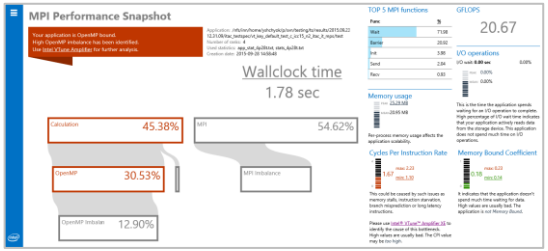
ОПТИМИЗАЦИЯ HPC ПРИЛОЖЕНИЙ

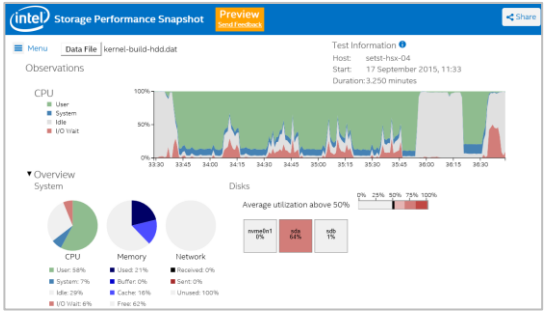


INTEL® PERFORMANCE SNAPSHOTS

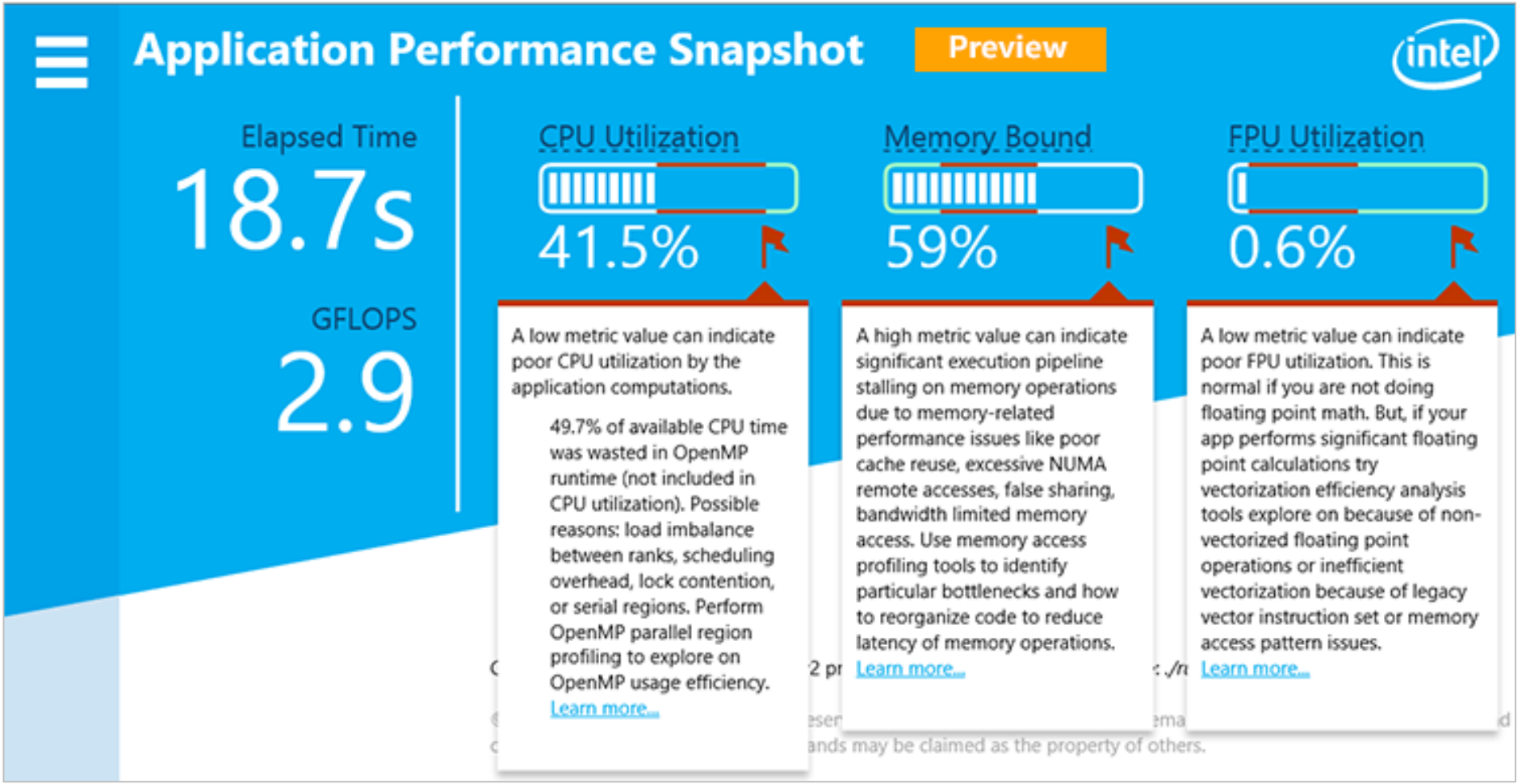
- Насколько хорошо приложение использует возможности «железа»?
  - Запуск теста во время кофе-брейка
  - Высокоуровневая сводка
- Выберете нужный snapshot:
  - **Application** – не MPI приложения
  - **MPI** – MPI приложения
  - **Storage** – для анализа хранилищ







http://www.intel.com/performance-snapshot


Application Performance Snapshot
Preview
intel
Elapsed Time
18.7s
GFLOPS
2.9
CPU Utilization
41.5%
A low metric value can indicate poor CPU utilization by the application computations.
49.7% of available CPU time was wasted in OpenMP runtime (not included in CPU utilization). Possible reasons: load imbalance between ranks, scheduling overhead, lock contention, or serial regions. Perform OpenMP parallel region profiling to explore on OpenMP usage efficiency.
Learn more...
Memory Bound
59%
A high metric value can indicate significant execution pipeline stalling on memory operations due to memory-related performance issues like poor cache reuse, excessive NUMA remote accesses, false sharing, bandwidth limited memory access. Use memory access profiling tools to identify particular bottlenecks and how to reorganize code to reduce latency of memory operations.
Learn more...
FPU Utilization
0.6%
A low metric value can indicate poor FPU utilization. This is normal if you are not doing floating point math. But, if your app performs significant floating point calculations try vectorization efficiency analysis tools explore on because of non-vectorized floating point operations or inefficient vectorization because of legacy vector instruction set or memory access pattern issues.
Learn more...
ands may be claimed as the property of others.

CALL TO ACTION

- Пробуем сегодня!

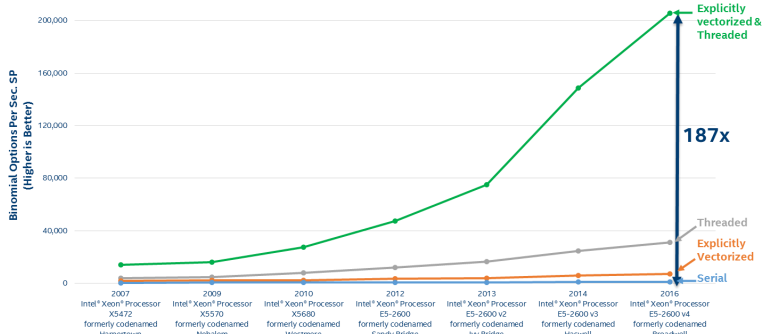
  https://software.intel.com/en-us/intel-parallel-studio-xe/try-buy

- Intel® Premier Support

- Ваш отзыв по использованию

Intel® Parallel Studio XE 2017 важен!

КОНФИГУРАЦИЯ ДЛЯ BINOMIAL OPTIONS SP



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

Performance measured in Intel Labs by Intel employees

Platform Hardware and Software Configuration

Platform	Unscaled Core Frequency	Cores/Socket	Num Sockets	L1 Data Cache	L1 I Cache	L2 Cache	L3 Cache	Memory	Memory Frequency	Memory Access	H/W Prefetchers Enabled	HT Enabled	Turbo Enabled	C States	O/S Name	Operating System	Compiler Version
Intel® Xeon™ 5472 Processor	3.0 GHZ	4	2	32K	32K	12 MB	None	32 GB	800 MHZ	UMA	Y	N	N	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5570 Processor	2.93 GHZ	4	2	32K	32K	256K	8 MB	48 GB	1333 MHZ	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ X5680 Processor	3.33 GHZ	6	2	32K	32K	256K	12 MB	48 MB	1333 MHZ	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2690 Processor	2.9 GHZ	8	2	32K	32K	256K	20 MB	64 GB	1600 MHZ	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 2697v2 Processor	2.7 GHZ	12	2	32K	32K	256K	30 MB	64 GB	1867 MHZ	NUMA	Y	Y	Y	Disabled	Fedora 20	3.11.10-301.fc20	icc version 14.0.1
Intel® Xeon™ E5 26xxv3 Processor	2.2 GHz	14	2	32K	32K	256K	35 MB	64 GB	2133 MHZ	NUMA	Y	Y	Y	Disabled	Fedora 20	3.13.5-202.fc20	icc version 14.0.1
Intel® Xeon™ E5 26xxv4 Processor																	

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. For more information go to

Legal Disclaimer & Optimization Notice

- INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
- 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.
-

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