УДК 519.6

ВЕКТОРНАЯ ОПТИМИЗАЦИЯ ПРОГРАММНОГО МОДУЛЯ "ЛОГОС-АЭРОГИДРО" СРЕДСТВАМИ БИБЛИОТЕК IAL

И. П. Рыжачкин
(ФГУП "РФЯЦ-ВНИИЭФ", г. Саров Нижегородской области)

В рамках программного модуля "Логос-АэроГидро" пакета программ "Логос" для дискретизации уравнения Навье—Стокса без расщепления используется метод конечных объемов на неструктурированных сетках, приводящий к решению СЛАУ с мелкоблочной матрицей для нахождения нескольких взаимозависимых неизвестных в каждой ячейке сетки. При распараллеливании вычислительных алгоритмов решения СЛАУ используются технологии SSE2-SSE4.2 (Intel®LegacySSE), Intel®AVX, Intel®AVX512. Описываются разработанные в РФЯЦ-ВНИИЭФ библиотеки IAL, обсуждаются проблемы, связанные с созданием оптимизированных библиотек. Приводятся результаты применения библиотек IAL для векторной оптимизации программного модуля "Логос-АэроГидро".

Ключевые слова: векторная оптимизация, SSE, AVX, пакет программ "Логос".

Введение

CAE (Computed-aided engineering) приложения в вычислительном плане являются одними из самых ресурсоемких. Все возрастающие возможности вычислительных систем не обеспечивают нужной динамики увеличения мощности, связанной как с постоянным усложнением алгоритмов, так и с необходимостью увеличения размерности расчетных сеток при моделировании сложных инженерных изделий.

CAE-приложения работают с большими массивами однотипных данных, что, с одной стороны, как нельзя лучше укладывается в парадигму векторных или SIMD (одна инструкция — много данных) вычислений, а с другой стороны, приводит к относительно невысокой плотности арифметических операций на единицу данных.

Последнее обстоятельство обосновывало определенный скептицизм относительно перспектив векторной оптимизации CAE-приложений, но появившиеся в последнее время публикации на эту тему [1] и собственный опыт автора убеждают в обратном.

Поскольку оптимизация на уровне классических C/C++, а также распараллеливание методом деления на потоки и MPI-процессы ограничены числом процессоров и ядер ЭВМ, векторные расширения архитектуры x86-64 открывают новые возможности ускорения CAE-приложений.

Вместе с тем оптимизация с использованием векторных расширений SSE—AVX существенно сложнее методов оптимизации, применяемых в случае скалярных архитектур, и требует разработки новых низкоуровневых вычислительных алгоритмов, позволяющих учесть латентность используемых векторных операций и максимально загрузить векторные типы данных.

К настоящему времени имеется уже достаточно большое количество векторных архитектур, таких как, например, SSE2, ..., SSE4.2, AVX, AVX2, AVX512, KNC, KNL. В составе одной суперкомпьютерной системы можно наблюдать если не все, то достаточно большое их разнообразие. Наличие большого числа векторных расширений ставит задачу разработки технологий, позволяющих приложению динамически настраиваться на тип задействованного процессора и использовать наиболее эффективные для данной архитектуры варианты кода.

Матрично-векторные операции в программном модуле "Логос-АэроГидро"

Матрично-векторные операции являются одними из самых ресурсоемких во множестве современных приложений, включая нейронные сети, используемые в системах искусственного интеллекта.

Исключением не являются и CAE-системы. Моделирование течений вязкого сжимаемого газа в программном модуле "Логос-АэроГидро" производится при помощи методик, которые основываются на решении полной системы уравнений Навье—Стокса [2] посредством алгебраического многосеточного решателя систем линейных алгебраических уравнений (СЛАУ), — его описание для решения СЛАУ с одним неизвестным в каждой ячейке сетки для методики "Логос Simple" приводится в статье [3]. Решение уравнений Навье—Стокса порождает СЛАУ с блочной матрицей, элементами которой являются матрицы небольших размеров — от 3×3 до 16×16. Это обуславливает интенсивное использование матричных, матрично-матричных, матрично-векторных и векторно-векторных операций.

Проблемы повышения качества программных комплексов

Векторная оптимизация больших программных комплексов подразумевает использование как возможностей компилятора путем задания опций и соответствующего перестроения алгоритмов и структур данных, так и оптимизированных библиотек, таких как Intel®IPP и MKL.

Специфика алгоритмов пакета программ "Логос" предполагает работу с массивами векторов и плотных матриц небольших размеров — от 3×3 до 16×16. Адресация данных массивов производится в основном при помощи относительных адресов — индексов, сохраняемых во внешней памяти. Адаптация интерфейсов пакета программ "Логос" к интерфейсам стандартных библиотек методом конвертации структур данных во время выполнения приводит к недопустимо высоким издержкам по памяти и времени.

Функциональность указанных библиотек ограничена относительно простыми операциями. В то же время высокую эффективность показывает совмещение нескольких простых операций в одном цикле за счет экономии на обменах *регистр—память* (registry spill).

Комплекс указанных причин в совокупности стимулирует разработку собственных оптимизированных библиотек и всего сопутствующего технологического обеспечения.

Градация пакетов программ на свободные, коммерческие и промышленные в основном определяется статистикой по количеству обнаруженных в них ошибок. Для минимизации количества ошибок необходима разработка специализированных технологий, ориентированных на всестороннюю оценку качества полученного объектного кода.

Технология оптимизации пакета программ подразумевает его деление на относительно небольшие фрагменты по функциональному признаку (функции) с последующим написанием всесторонних тестов для каждого фрагмента. Поскольку каждая функция имеет относительно небольшие размеры, то поиск и исправление ошибок в ней занимают незначительное время.

Технология создания оптимизированных библиотек IAL

Структурно технология создания оптимизированных библиотек IAL может быть представлена в виде совокупности систем, реализованных различными языковыми средствами, куда входят сами библиотеки, написанные с использованием векторного расширения языка C, тесты и тестовые системы, написанные на C++ и ассемблере, а также система построения библиотек, тестов и тестовых систем, написанная на языках make, Perl, bash, PhP (рис. 1).

В рамках разработанной технологии IAL осуществляется построение статических, динамических и диспетчируемых версий библиотек. Диспетчеризация позволяет переключаться на оптимизированные для конкретной архитектуры ветки кода библиотек. При этом основной код приложения является неизменным, что делает воз-

Рис. 1. Компоненты технологии создания оптимизированных библиотек

можным максимально эффективное использование одного исполняемого модуля на любых архитектурах x86-64.

Разработку кода оптимизированной библиотеки можно представить в виде циклического процесса, изображенного на рис. 2.

Процесс состоит из следующих этапов:

1) создание первоначального С-кода функции;
2) создание и модификация тестов функции средствами тестовых систем;
3) скалярная и векторная оптимизация функции;
4) построение библиотеки функции, ее тестирование и переход по необходимости к этапу 2 или 3;
5) построение диспетчируемой версии библиотеки функции для ОС Windows и Linux;
6) тестирование на целевых архитектурах и переход по необходимости к этапу 2.

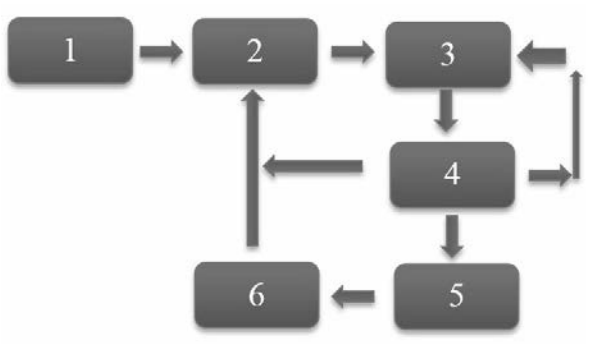

Рис. 2. Этапы создания оптимизированной библиотеки

Векторная оптимизация программного модуля "Логос-АэроГидро"

Рассмотрим оптимизацию "Логос-АэроГидро" в части расчета аэродинамических характеристик сложных конструкций. Характерной особенностью функционирования программных модулей подобного типа является относительно большой объем входных данных в виде сеток, моделей и большой размер множества внутренних переменных, что обуславливает активный обмен данных типа *регистр—память*, а также относительно большое отношение числа таких обменов к числу арифметических операций.

Векторизованная версия программного модуля "Логос-АэроГидро" базируется на оптимизированных библиотеках IAL.

Функции библиотек оптимизированы для архитектур SSE2-SSE4.2, AVX, AVX512.

В состав библиотек IAL входят:

1) IALM — матричные и матрично-векторные операции;
2) IALS — операции с векторами;
3) IALL — аэродинамические (гидродинамических) алгоритмы;
4) IALKERNEL — основные операции, такие как определение типа процессора и настройка диспетчера кода, операции с памятью, управление использованием OMP-потоков функциями IAL-библиотек.

Библиотеки IAL дополнительно включают тесты функций и тестовые системы. Система построения служит для построения тестов, тестовых систем и библиотек.

При векторной оптимизации библиотек IAL использовалось векторное расширение языка C [4] для инструкций SSE2-SSE4.2, AVX, AVX512 [5], что существенно ускорило матрично-векторные операции по сравнению с оптимизированным С-кодом. Основным приемом оптимизации является разработка для каждого размера матрицы (в основном от 3×3 до 16×16), входящей в блочно-разреженную матрицу СЛАУ, своего векторного вычислительного алгоритма, учитывающего выровненность данных, а также количество и латентность используемых векторных операций.

На рис. 3 приведен пример С-кода функции умножения матрицы на матрицу с размерами 4×4, оптимизированного методом *раскрутки цикла*. Данный код выгодно отличается от функции общего вида отсутствием цикла и соответственно затрат на управление циклом. Наличие большого числа арифметических операций позволяет компилятору составить код с минимальным количеством простоев конвейера (pipe stalls) по причине зависимости по данным.

На рис. 4 показано, как та же задача решена с использованием векторного расширения SSE.

На рис. 5 показано решение той же задачи с использованием векторного расширения AVX. В реализации с использованием этого векторного расширения задача раскрутки цикла по строкам матрицы, как правило, достаточно успешно решается компилятором.

На рис. 6 приводятся некоторые данные по производительности вышеописанных вариантов реализации умножения матриц. Используются следующие обозначения: *mul_mms* — ал-

```
void MatMat_4 (const float * __restrict matrix1, const float * __restrict matrix2, float * __restrict
result){MEM_ZERO_32( result, 16 );
        result[ 0 ] += matrix1[ 0 ] * matrix2[ 0 ];
        result[ 0 ] += matrix1[ 1 ] * matrix2[ 4 ];
        result[ 0 ] += matrix1[ 2 ] * matrix2[ 8 ];
        result[ 0 ] += matrix1[ 3 ] * matrix2[ 12 ];
        result[ 1 ] += matrix1[ 0 ] * matrix2[ 1 ];
        result[ 1 ] += matrix1[ 1 ] * matrix2[ 5 ];
        result[ 1 ] += matrix1[ 2 ] * matrix2[ 9 ];
        result[ 1 ] += matrix1[ 3 ] * matrix2[ 13 ];
        ..........................
        result[ 15 ] += matrix1[ 12 ] * matrix2[ 3 ];
        result[ 15 ] += matrix1[ 13 ] * matrix2[ 7 ];
        result[ 15 ] += matrix1[ 14 ] * matrix2[ 11 ];
        result[ 15 ] += matrix1[ 15 ] * matrix2[ 15 ];
        return;
}
```

Рис. 3. Оптимизация умножения матрицы на матрицу с размерами 4×4 методом раскрутки цикла

```
void MatMat_4_SSE( const float* p_Orig1, int src1Stride_step1, int src1Width, int src1Height, const float*
p Orig2, int src2Stride step1, int src2Width, int src2Height, float* p Dest, int jDstStride step1 )
{
    __m128   b0, b1, b2, b3;
    __m128  row, rslt, tmprVal, dst;
    /* Загрузка матрицы B. */
    b0      = _mm_loadu_ps(&(((float*)((Ial8u*)p_Orig2))[0]));
    b1      = _mm_loadu_ps(&(((float*)((Ial8u*)p_Orig2 + src2Stride_step1))[0]));
    b2      = _mm_loadu_ps(&(((float*)((Ial8u*)p_Orig2 + 2*src2Stride_step1))[0]));
    b3      = _mm_loadu_ps(&(((float*)((Ial8u*)p_Orig2 + 3*src2Stride_step1))[0]));
    /* Вычисляем первый ряд результирующей матрицы */
    row     = _mm_load1_ps(&(((float*)((Ial8u*)p_Orig1))[0]));
    rslt    = _mm_mul_ps(row, b0);
    row     = _mm_load1_ps(&(((float*)((Ial8u*)p_Orig1))[1]));
    rslt    = _mm_add_ps(rslt, _mm_mul_ps(row, b1));
    row     = _mm_load1_ps(&(((float*)((Ial8u*)p_Orig1))[2]));
    rslt    = _mm_add_ps(rslt, _mm_mul_ps(row, b2));
    row     = _mm_load1_ps(&(((float*)((Ial8u*)p_Orig1))[3]));
    rslt    = _mm_add_ps(rslt, _mm_mul_ps(row, b3));
    _mm_storel_pi((__m64*)&(((float*)((Ial8u*)p_Dest))[0]), rslt);
    _mm_storeh_pi((__m64*)&(((float*)((Ial8u*)p_Dest))[2]), rslt);
    /* Вычисляем второй ряд результирующей матрицы */
.....
    /* Вычисляем третий ряд результирующей матрицы */
 .....
   /* Вычисляем четвёртый ряд результирующей матрицы */
.....
    return ;
}
```

Рис. 4. Реализация операции умножения квадратных матриц 4×4 при помощи векторного расширения SSE, оптимизированная методом раскрутки цикла (суффикс step1 означает длину буфера строки в байтах; step2 — длину буфера числа в байтах)

горитм умножения матриц общего вида на C; *mul_mmo* — алгоритм умножения матриц общего вида на C, оптимизированный методом фиксации числа циклов, что подразумевает создание отдельного кода для каждого размера матриц; *mul_mmi* — векторизованный алгоритм умножения матриц общего вида, оптимизированный при помощи векторного расширения C.

Результаты приведены в тактах процессора и рассчитывались по формуле

$$N = \frac{o}{(2n-1)\,n^2},$$

где o — число тактов процессора, потраченных на вычисление алгоритма nul_mm^*; n — размер матрицы; N — число тактов процессора, потраченных на одну арифметическую операцию.

Для наиболее часто используемых размеров матриц 5×5 скорость векторизованного кода в 1,5 раза выше скалярного оптимизированного

```
void MatMat_4_AVX( const float* p_Orig1, int src1Stride_step1, int src1Width, int src1Height, const float*
p_Orig2, int src2Stride_step1, int src2Width, int src2Height, float* p_Dest, int jDstStride_step1 )
{
        __m256i mask0;
        __m256          row0, row1, row2, row3 ;
        __m256          r0, r1, r2, r3 ;
        __m256          d0; int *p = (int*)&mask0;
        int             i;
        p[0] = p[1] = p[2] = p[3] = -1;
        p[4] = 0 p[5] = p[6] = p[7] = 0;
        row0 = _mm256_maskload_ps((float*)((char*)p_Orig2), mask0);
        row1 = _mm256_maskload_ps((float*)((char*)p_Orig2 +   src2Stride_step1), mask0);
        row2 = _mm256_maskload_ps((float*)((char*)p_Orig2 + 2*src2Stride_step1), mask0);
        row3 = _mm256_maskload_ps((float*)((char*)p_Orig2 + 3*src2Stride_step1), mask0);
      for( i=0; i<4; i++) {
          r0  = _mm256_broadcast_ss((float*)((char*) p_Orig1    + i*src1Stride_step1));
          r1  =  mm256 broadcast ss((float*)((char*)(p Orig1+1) + i*src1Stride step1));
          r2  = _mm256_broadcast_ss((float*)((char*)(p_Orig1+2) + i*src1Stride_step1));
          r3  = _mm256_broadcast_ss((float*)((char*)(p_Orig1+3) + i*src1Stride_step1));
          d0  = _mm256_mul_ps(row0, r0);
          d0  = _mm256_add_ps(d0, _mm256_mul_ps(row1, r1));
          d0  = _mm256_add_ps(d0, _mm256_mul_ps(row2, r2));
          d0  = _mm256_add_ps(d0, _mm256_mul_ps(row3, r3));
          _mm256_maskstore_ps((float*)((char*)p_Dest + i*jDstStride_step1), mask0, d0);
      }
        _mm256_zeroupper();
        return 0;
}
```

Рис. 5. Реализация операции умножения квадратных матриц 4×4 при помощи векторного расширения AVX

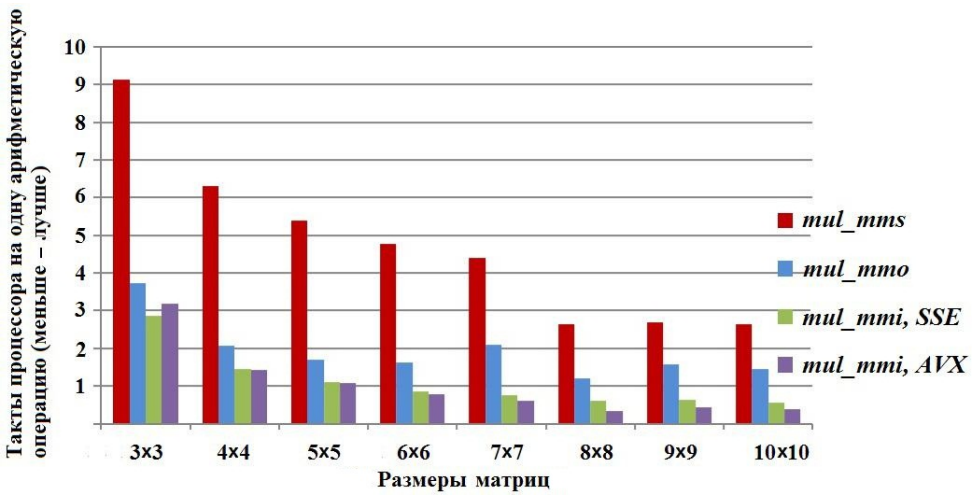

Рис. 6. Производительность алгоритмов *mul_ mm** на серверной архитектуре Intel®Sandy Bridge

(*mul_ mmo*) и в 5,5 раз выше С-кода общего вида (*mul_ mms*).

Начиная с размеров матриц 8×8, скорость векторизованного кода в 4 раза (SSE) и 8 раз (AVX) выше С-кода общего вида (*mul_ mms*) и соответственно в 2 и 4,5 раза выше скалярного оптимизированного кода (*mul_ mmo*).

Эффективность векторных операций SSE—AVX растет вплоть до размеров матриц 8×8, что связано как с наполняемостью векторных регистров, так и с уменьшением удельных затрат на управление циклом.

Эффективность скалярного кода (*mul_ mmo*) растет вплоть до размеров 8×8, что связано с уменьшением удельных затрат на управление циклом.

AVX-код начинает опережать SSE, начиная с размеров матриц 5×5, что связано с размерами регистров (32 и 16 байт соответственно).

Результаты применения библиотек IAL

Далее приведены результаты сравнения базового варианта программного модуля "Логос-АэроГидро" без оптимизации при помощи библиотек IAL и с применением библиотек IAL.

На рис. 7 показан профиль матрично-векторных операций "Логос-АэроГидро" на задаче сверхзвукового обтекания крыла самолета, полученный при помощи профилировщика Intel VTune Amplifier XE 2013. Было выполнено 5 000 итераций, начиная с итерации с номером 5 000. Профилирование производилось в один поток на ЭВМ Intel(R)Core(TM)i5-2310CPU@2,90 GHz.

Функции умножения матрицы на вектор с последующим вычитанием (или прибавлением) результата (`MatVecSub (Add)`), умножения матрицы на матрицу (`Mul`), умножения матрицы на вектор (`MatVec`), инверсии матриц (`MatInv`) занимают около 32 % от общего времени счета задачи.

На рис. 7 слева приведены данные по вкладу функций, оптимизированных методом раскрутки цикла (см. рис. 3). Справа — данные по вкладу функций, оптимизированных при помощи векторных расширений SSE—AVX. В обозначениях функций 32f соответствует одинарной точности, 64f — двойной. Из рисунка видно, что суммарные затраты процессорного времени перечисленных функций снизились в 2,1 раза.

На рис. 8 представлено распределение процессорного времени "Логос-АэроГидро" до и после оптимизации SSE—AVX при размерах матриц 14×14, где ускорение счета задачи достигло 18 %.

Заключение

Векторная оптимизация матрично-векторных операций программного модуля "Логос-АэроГидро" пакета программ "Логос" при помощи библиотек IAL сокращает время расчета, обеспечивает архитектурную универсальность и существенно облегчает процесс построения и использования приложения, обеспечивая таким образом его необходимые рыночные качества.

Относительно невысокая средняя плотность арифметических операций на единицу данных

MatVecSub_5	2511.710s	MulSub_maviv_5x5_32f	1082.334s
MatVecAdd_5	484.516s	MulAdd_mavivi_5x5_32f	421.748s
MATRIX_N::Mul	464.569s	Multiply_mm_64f	189.003s
MatVec_5	318.399s	Multiply_mv_32f	146.215s
realLevel::MatInv	281.762s	Reverse_m_64f_5x5_W7	70.119s

Рис. 7. Профилирование матрично-векторных и матричных операций "Логос-АэроГидро" при помощи Intel VTune Amplifier XE 2013 на задаче сверхзвукового обтекания крыла самолета

а *б*

Рис. 8. Распределение процессорного времени "Логос-АэроГидро" до (*а*) и после (*б*) IAL-оптимизации: ■ — матрично-векторные операции; ■ — прочие операции

CAE-приложений не снижает эффективности векторной оптимизации матрично-векторных операций, в том числе за счет сопутствующего уменьшения числа обменов *регистр—память*. В результате векторной оптимизации матрично-векторных операций программного модуля "Логос-АэроГидро" при помощи библиотек IAL, оптимизированных для векторных расширений SSE—AVX, суммарные затраты процессорного времени на выполнение данных операций снизились более чем в 2 раза, а всего расчета — от 9 до 18 % на кластерной машине.

Разработанная технология создания библиотек IAL обеспечивает высокий уровень качества оптимизированных библиотек за счет развитой системы тестирования и построения кода. Указанная технология обеспечивает создание библиотек, способных динамически настраиваться на тип задействованного процессора и использовать наиболее эффективные для данной архитектуры варианты кода.

Список литературы

1. *Getmanskiy V., Andreev A. E., Alekseev S., Gorobtsov A. S., Egunov V., Kharkov E.* Optimization and parallelization of CAE software stress-strain solver for heterogeneous computing hardware // Conf. on Creativity in Intelligent Technologies and Data Science CIT&DS 2017. https://link.springer.com/chapter/10.1007%2F978-3-319-65551-2_41.
2. *Козелков А. С., Дерюгин Ю. Н., Зеленский Д. К.* Многофункциональный пакет программ "Логос Аэро-Гидро" для расчета задач гидродинамики и тепломассопереноса на суперЭВМ. Базовые технологии и алгоритмы // Труды XII Межд. семинара "Супервычисления и математическое моделирование". Саров, 11—15 октября, 2010. Саров: РФЯЦ-ВНИИЭФ. С. 215—230.
   Kozelkov A. S., Deryugin Yu. N., Zelenskiy D. K. Mnogofunktsionalny paket programm "Logos Aero-Gidro" dlya rascheta zadach gidrodinamiki i teplomassoperenosa na superEVM. Bazovye tekhnologii i algoritmy // Trudy XII Mezhd. seminara "Supervychysleniya i matematicheskoe modelirovanie". Sarov, 11—15 oktyabrya, 2010. Sarov: RFYaTs-VNIIEF. S. 215—230.
3. *Козелков А. С., Дерюгин Ю. Н., Лашкин С. В., Силаев Д. П., Симонов П. Г., Тятюшкина Е. С.* Реализация метода расчета вязкой несжимаемой жидкости с использованием многосеточного метода на основе алгоритма SIMPLE в пакете программ ЛОГОС // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2013. Вып. 4. С. 44—56.
   Kozelkov A. S., Deryugin Yu. N., Lashkin S. V., Silaev D. P., Simonov P. G., Tyatyushkina E. S. Realizatsiya metoda rascheta vyazkoy neszhimaemoy zhidkosti s ispolzovaniem mnogosetochnogo metoda na osnove algoritma SIMPLE v pakete programm LOGOS // Voprosy atomnoy nauki i tekhniki. Ser. Matematicheskoe modelirovanie fizicheskikh protsessov. 2013. Vyp. 4. S. 44—56.
4. Intrinsics, USA, 2016. https://software.intel.com.
5. Intel®64 and IA-32 Architectures Optimiztion Reference Manual. Order Number 248966-026, April 2012. https://software.intel.com.

Статья поступила в редакцию 25.03.20.