ББК 32.973
УДК 004.42

А. А. Рыбаков, С. С. Шумилин

Исследование эффективности векторизации гнезд циклов с нерегулярным числом итераций

Аннотация. Векторизация вычислений является важной низкоуровневой оптимизацией, используемой для создания высокоэффективного параллельного кода. Однако, при использовании ее в контексте с неизвестным профилем исполнения существует опасность низкой эффективности применения. Особенно ярко это проявляется при векторизации гнезд циклов с нерегулярным числом итераций внутреннего цикла. В статье рассматривается сравнение теоретической и практической эффективности векторизации на примере сортировки Шелла, так как данный программный код является крайне неудобным с точки зрения применения векторизации.

Ключевые слова и фразы: векторизация, AVX-512, гнезда циклов с нерегулярным числом итераций, сортировка Шелла, теоретическое ускорение.

Введение

В данной статье векторизация рассматривается применительно к набору инструкций AVX-512, представляющих собой 512-битное расширение 256-битных AVX инструкций из набора команд Intel x86 [1], поддержанное в семействах микропроцессоров Intel Xeon Phi KNL [2] и Intel Xeon Skylake. Данный набор состоит из следующих подмножеств: AVX-512F (Foundation) – основной набор векторных инструкций с поддержкой маскирования, AVX-512PF (PreFetch) – инструкции предварительной подкачки данных из памяти, AVX-512ER (Exponential and Reciprocal) – команды для вычисления экспоненты и обратных значений, AVX-512CD (Conflict Detection) – инструкции для определения конфликтов, AVX-512BW и AVX-512DQ, поддержанные в Skylake. В следующих поколениях процессоров набор инструкций AVX-512 расширяется еще больше, в него входят команды

Подготовлено 30 ноября 2018 г. в журнал «Программные системы: теория и приложения».

для работы с 52-битными целочисленными значениями, специальные команды для работы с нейросетями и AES шифрованием, реализация арифметики полей Галуа, имплементация специальных битовых операций, а также новый класс комбинированных операций.

Использование масок в векторных операциях, позволяющих осуществлять выборочную обработку отдельных элементов векторов, позволяет реализовать предикатный режим исполнения операций. В совокупности с многообразием операций перестановки элементов векторов, комбинированными операциями, операциями множественного доступа в память по произвольным адресам, и многими другими особенностями набора инструкций AVX-512 это позволяет создавать качественный параллельный код, приводящий к кратному ускорению.

Для упрощения векторизации исходного кода для компилятора icc доступны специальные функции-интринсики [3, 4], являющиеся обертками к инструкциям или группам инструкций AVX-512. Использование функций-интринсиков и встроенных типов данных для поддержки 512-битных векторов позволяет создавать конкретные векторные инструкции в результирующем коде, оперируя при этом высокоуровневыми сущностями языка программирования C. Из множества интринсиков можно выделить следующие группы функций, схожие по структуре. Функции `swizzle`, `shuffle`, `permute` и `permutevar` осуществляют перестановку элементов вектора и раскрываются в последовательность операций, в которой присутвует операция `shuf` и пересылка по маске. Для большего числа операций AVX-512 реализованы соответствующие инстринсики, раскрывающиеся в одну конкретную операцию. Среди них арифметические операции, побитовые операции, операции чтения из памяти и записи в память, операции конвертации, слияние двух векторов, нахождение обратных значений, получение минимума и максимума из двух значений, операции сравнения, операции с масками, комбинированные операции и другие. Некоторые инстринсики, особенно предназначенные для выполнения упакованных трансцендентных операций, раскрываются просто в вызов библиотечной функции (например `log`, `hypot`, тригонометрические функции).

В качестве объекта исследования для данной статьи была выбрана сортировка Шелла, обладающая крайне неудобным контекстом исполнения для векторизации с помощью инструкций AVX-512.

1. Описание сортировки Шелла

Сортировка Шелла [5] представляет собой расширение сортировки вставками, которое работает быстрее, так как позволяет на ранних этапах упорядочить далеко расположенные друг от друга элементы массива, это приводит к тому, что массив становится частично упорядоченным. Во время сортировки Шелла выполняется последовательная сортировка подмассивов основного массива, являющихся срезами, при этом шаг среза постоянно уменьшается и на завершающем этапе выполняется обычная сортировка вставками (это соответствует срезу массива с шагом 1). Выполнение сортировки срезов массива с большими шагами облегчает сортировку срезов с меньшими значениями шага, эффективность сортировки существенно зависит от выбранной последовательности шагов. В литературе описано множество существующих последовательностей, из которых мы будем анализировать лишь следующие, представленные в Табл. 1:

Таблица 1. Различные последовательности шагов, используемые в сортировке Шелла

Последовательность	**Формула**
Последовательность Шелла, 1959 г.	$k_1 = \lfloor \frac{N}{2} \rfloor, k_i = \lfloor \frac{k_{i-1}}{2} \rfloor, k_t = 1$
Последовательность Хиббарда, 1963 г.	$2^i - 1 \leq N, i \in \mathbb{N}$
Последовательность Пратта, 1971 г.	$2^i \cdot 3^j \leq \frac{N}{2}, i \in \mathbb{N}, j \in \mathbb{N}$
Последовательность Седжвика, 1986 г.	$k_i = \begin{cases} 9 \cdot 2^i - 9 \cdot 2^{\frac{i}{2}} + 1, k\ even \\ 8 \cdot 2^i - 6 \cdot 2^{\frac{i+1}{2}} + 1, k\ odd \end{cases}$

Каноническая реализация сортировки Шелла состоит из гнезда циклов, содержащего три цикла. Внешний цикл выполняется по всем шагам из используемой последовательности шагов, начиная с максимального и заканчивая единицей. Два внутренних цикла осуществляют сортировку всех подмассивов, являющихся срезами исходного массива с текущим шагом k (Рис. 1).

```
01 void shell_sort(float *m, int n, int *ks, int k_ind)
02 {
03     int i, j, k;
04
05     for (k = ks[k_ind]; k > 0; k = ks[--k_ind])
06     {
07         for (i = k; i < n; i++)
08         {
09             float t = m[i];
10
11             for (j = i; j >= k; j -= k)
12             {
13                 if (t < m[j - k])
14                 {
15                     m[j] = m[j - k];
16                 }
17                 else
18                 {
19                     break;
20                 }
21             }
22
23             m[j] = t;
24         }
25     }
26 }
```

Рис. 1. Реализация сортировки Шелла для произвольной последовательности шагов

2. Векторизация сортировки Шелла с помощью инструкций AVX-512

Рассмотрим возможности по векторизации сортировки Шелла для массива вещественных значений типа `float` (вектор AVX-512 содержит 16 таких значений). Самый вложенный цикл (цикл с счетчиком j, будем называть его просто внутренним) выполняет сортировку одного среза, состоящего из элементов массива, с расстоянием k между соседними элементами. Внутренний цикл не может быть векторизован без выполнения дополнительных модификаций кода, так как между записью элемента $m[j]$ и чтением элемента $m[j-k]$ существует межитерационная зависимость. Однако, можно заметить, что две итерации среднего по вложенности цикла (цикла с индуктивной переменной i, будем называть его промежуточным) с номерами i_1 и i_2 не пересекаются по данным и могут быть выполнены параллельно при выполнении условия $|i_1 - i_2| < k$. Выполним декомпозицию сортировки Шелла для того, чтобы можно было явно выделить ядро, поддающееся векторизации.

На Рис. 2 представлена схема декомпозиции алгоритма сортировки Шелла, в котором явно выделены участки с разной шириной

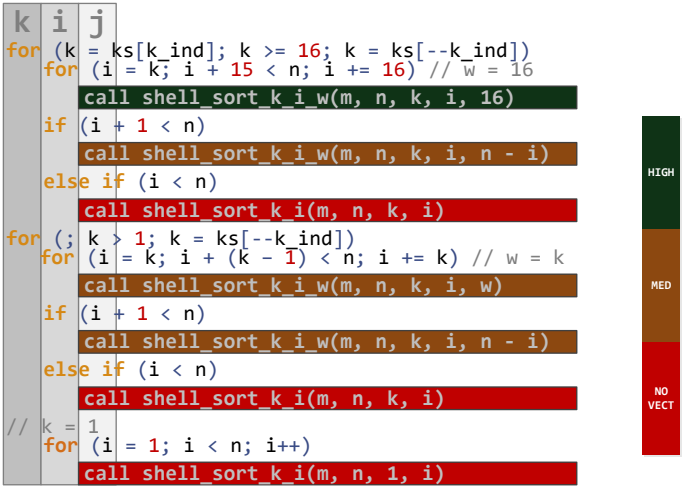


Рис. 2. Декомпозиция сортировки Шелла для выделения векторизуемых участков кода

векторизации. Сначала выделен блок для шагов $k \geq 16$. Для данных значений шагов можно параллельно выполнять 16 соседних итераций промежуточного цикла, при этом достигается максимальная плотность векторизации (показано зеленым цветом на схеме). Все итерации промежуточного цикла разбиваются на группы по 16 соседних итераций и остаток, который векторизуется с шириной меньше 16 (показано желтым цветом, а в том случае, когда остаток состоит всего из одной итерации, то векторизация не требуется, что показано красным цветом на схеме). Далее рассматривается блок значений шагов $1 < k < 16$. При данных значениях ширина векторизации всегда меньше 16, к тому же, как и в предыдущем блоке, возможно появление невекторизуемого остатка. В последнюю очередь рассматривается невекторизуемая финальная сортировка вставками для $k = 1$. Наличие участков кода с шириной векторизации менее 16 приводит к неоптимальному результирующему коду, однако есть более опасная причина низкой эффективности векторизации.

Функция `shell_sort_k_i_w`, появившаяся после декомпозиции алгоритма сортировки Шелла, содержит реализацию сортировки w

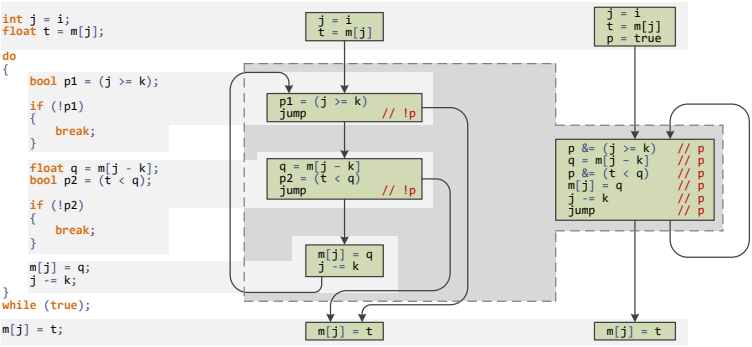


Рис. 3. Схема перевода тела внутреннего цикла сортировки Шелла в предикатную форму

соседних срезов массива, взятых с шагом k. При этом количество итераций внутреннего цикла данной функции является неизвестным. Более того, количество итераций внутреннего цикла при сортировке одного среза никак не связано с количеством итераций внутреннего цикла при сортировке соседнего среза. Это является существенной проблемой при попытке объединить код сортировки соседних срезов, используя векторные инструкции. Для такого объединения необходимо переписать код сортировки среза в предикатной форме [6], после чего заменить все инструкции векторными аналогами, а предикаты – векторными масками (Рис. 3). При этом если до векторизации внутренний цикл завершал работу при обращении предиката в `false`, то после векторизации внутренний цикл завершит работу только если все элементы соответствующей векторной маски обнулятся. Таким образом, количество итераций векторизованного внутреннего цикла равно максимальному количеству итераций всех объединяемых w циклов. Если значения количества итераций соседних объединяемых циклов различаются сильно (а для сортировки Шелла это утверждение верно), то мы получаем потерю эффективности векторизации из за низкой плотнойти масок векторных инструкций (то есть малый процент элементов векторов на самом деле обрабатывается при выполнении векторной операции).

Векторизованная версия ядра сортировки Шелла представлена на Рис. 4. Также стоит обратить внимание на появившуюся в вектор-

```
01 void shell_sort_k_i_w(float *m, int n, int k, int i, int w)
02 {
03     int j = i;
04     __mmask16 ini_mask = ((unsigned int)0xFFFF) >> (16 - w);
05     __mmask16 mask = ini_mask;
06     __m512i ind_j = _mm512_add_epi32(_mm512_set1_epi32(i),
07                                      ind_straight);
08     __m512 t, q;
09
10     t = _mm512_mask_load_ps(t, mask, &m[j]);
11
12     do
13     {
14         mask = mask & _mm512_mask_cmp_epi32_mask(mask, ind_j, ind_k,
15                                                  _MM_CMPINT_GE);
16         q = _mm512_mask_load_ps(q, mask, &m[j - k]);
17         mask = mask & _mm512_mask_cmp_ps_mask(mask, t, q,
18                                               _MM_CMPINT_LT);
19         _mm512_mask_store_ps(&m[j], mask, q);
20         ind_j = _mm512_mask_sub_epi32(ind_j, mask, ind_j, ind_k);
21         j -= k;
22     }
23     while (mask != 0x0);
24
25     _mm512_mask_i32scatter_ps(m, ini_mask, ind_j, t, _MM_SCALE_4);
26 }
```

Рис. 4. Векторизованный вариант ядра сортировки Шелла

ном коде операцию `scatter` (Рис. 4, строка 25) множественной записи данных в память с произвольными смещениями относительно базового адреса. Данная команда появилась как векторный аналог операции записи в память из оригинального кода (Рис. 1, строка 23) ввиду той же причины - нерегулярности количества итераций внутреннего цикла. Стоит отметить, что команды `scatter` являются крайне медленными, что также является причиной снижения эффективности векторизации.

Кроме обозначенной команды `scatter` в векторизованнос коде присутствуют другие команды обращения в память (Рис. 4, строки 10, 16, 19). Это обычные команды `load` и `store`, которые, вообще говоря, должны обращаться по выровненным адресам в памяти. В общем случае подаваемые им адреса конечно не являются выровненными, однако использование вместо них невыровненных обращений существенно замедляет код, поэтому было решено оставить эти инструкции.

3. Вычисление теоретического ускорения

В данном разделе произведем вычисление теоретического ускорения, которое может быть достигнуто при векторизации сортировки

Шелла предложенным в этой статье способом. При этом под теоретическим ускорением будем подразумевать просто отношение количества итераций внутреннего цикла невекторизованной версии к количеству итераций внутреннего цикла в оптимизированной векторизованной версии кода. Определим данное ускорение более формально.

Сначала рассмотрим невекторизованный код. Обозначим через $T(k,i)$ количество итераций внутреннего цикла при фиксированных k и i. Тогда не составляет труда вычислить общее количество итераций внутреннего цикла при выполнении сортировки (обозначим эту велечину через T).

$$T = \sum_{k \in ks} \sum_{i=k}^{n-1} I(k,i) \tag{1}$$

Теперь рассмотрим векторизованную версию кода. Аналогично обозначим через $T_v(k,i)$ количество итераций внутреннего цикла при фиксированных k и i. Нам известно, что ширина векторизации не может превышать k (из-за зависимостей по обращению к массиву), и 16 (размер вектора), то есть $w(k) = \min(k, 16)$. При этом весь диапазон значений i от k до $n-1$, длина которого равна $n-k$ разбивается на $\lfloor \frac{n-k}{w(k)} \rfloor$ групп по w элементов в каждой, и количество итераций в векторизованном цикле, соответствующем каждой группе равно максимальному значению из количеств итераций невекторизованных циклов, объединяемых в данный векторизованный цикл. С учетом векторизации хвостовой части цикла, получим следующую формулу для общего количества итераций векторизованного внутреннего цикла.

$$T_v = \sum_{k \in ks} \left(\left(\sum_{g=0}^{G(k)-1} \max_{i=k+w(k)g}^{k+w(k)(g+1)-1} I(k,i) \right) + \max_{i=k+w(k)G(k)}^{n-1} I(k,i) \right) \tag{2}$$

где $w(k) = \min(k, 16)$, $G(k) = \lfloor \frac{n-k}{w(k)} \rfloor$. Значения $T = T(n)$ и $T_v = T_v(n)$ были вычислены при сортировке псевдослучайных массивов с количеством элементов от 10 тыс. до 2 млн. для каждой из последовательностей шагов Хиббарда, Пратта и Сэджвика [7–9]. На основе этого вычислялось теоретическое ускорение $s(n) = T(n)/T_v(n)$. Кроме того, аналогичные характеристики рассчитывались и без учета шага $k = 1$ (данные величины обозначены $T'(n)$, $T'_v(n)$ и $s'(n)$

соответственно). Полученные результаты сравнивались с результатами экспериментальных запусков на микропроцессоре Intel Xeon Phi 7290 KNL.

4. Экспериментальные результаты

На Рис. 5, Рис. 6 представлены результаты теоретических оценок и экспериментальных запусков, исходя из которых получены теоретическое и экспериментальное ускорение сортировки Шелла для последовательностей шагов Шелла, Хиббарда и Сэджвика. Из графиков видно, что даже максимальное теоретическое ускорение далеко от идеальной верхней границы (которая равна 16 для значений типа `float`), достигаемого при векторизации гнезд циклов с регулярным количеством итераций. Экспериментальные же результаты ожидаемо оказываются еще ниже, и в конечном итоге финальное ускорение сортировки Шелла редко превышает отметку 2.

Проанализируем гистограммы распределения количества итераций внутреннего цикла для некоторых фиксированных значений k. Из гистограмм, приведенных на Рис. 7, Рис. 8, Рис. 9, Рис. 10 видно, что при переходе к векторизованной версии кода меняется характер распределения и количество выполнений внутреннего цикла с малым количеством итераций резко сокращается.

Наличие только одной итерации внутреннего цикла в невекторизованной версии кода означает, что текущий рассматриваемый элемент $m[i]$ уже стоит на своем месте в сортируемом срезе. Для векторизованной версии одна итерация внутреннего цикла означает, что на своем месте стоят w рассматриваемых элементов w соседних одновременно сортируемых срезов, вероятность чего значительно ниже. Таким образом, итерации внутренних циклов с малым количеством итераций растворяются в векторизованном коде, что снижает его производительность.

Теоретическая оценка эффективности векторизации с использованием последовательности Сэджвика оказалась ближе всего к экспериментальным результатам. Это может быть объяснено тем, что последовательность шагов ks в данном случае достаточно короткая, и сортируемые срезы с разными значениями k слабо коррелируют друг с другом. В последовательности шагов Сэджвика присутствует только один шаг меньше 16 (это последний шаг $k = 1$), для всех остальных шагов доступно применение векторизации с максимальной плотностью.

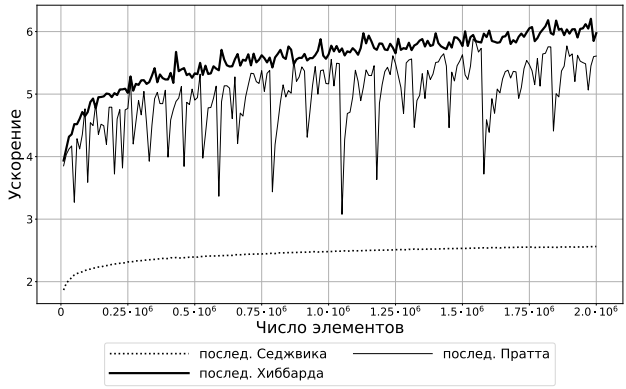


Рис. 5. Сравнение теоретического ускорения векторизованной версии сортировки Шелла для различных последовательностей шагов

Рис. 6. Сравнение экспериментального ускорения векторизованной версии сортировки Шелла для различных последовательностей шагов

Противоположностью последовательности Сэджвика является последовательность шагов Пратта, содержащая всевозможные шаги вида $2^i 3^j$. Во-первых, эта последовательность очень длинная и содержит сразу 8 значений, которые меньше максимальной ширины векторизации, что негативно сказывается на производительности. Во-вторых, срезы, индуцированные такими шагами, являются принци-

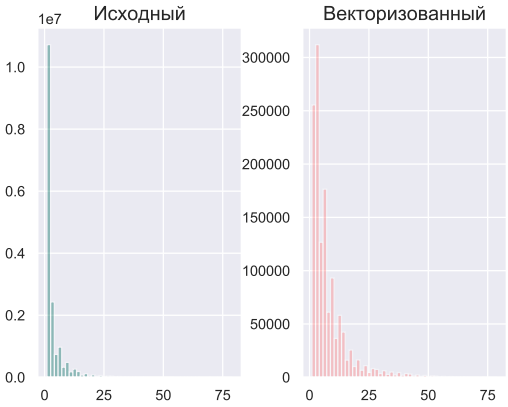


Рис. 7. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Шелла при $k = 4$

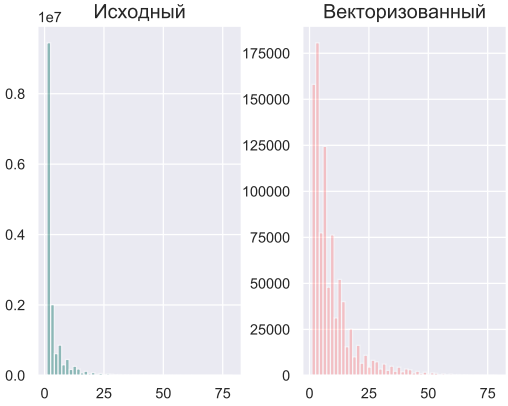


Рис. 8. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Шелла при $k = 15$

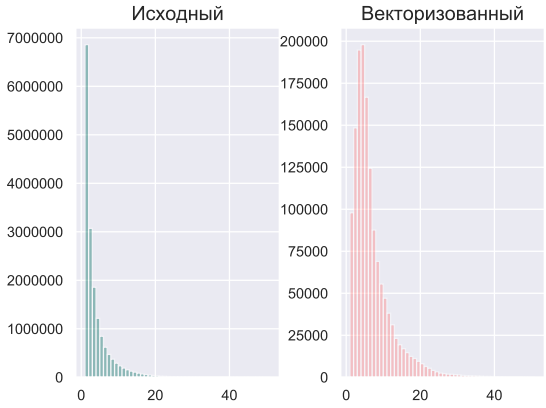


Рис. 9. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Хиббарда при $k = 3$

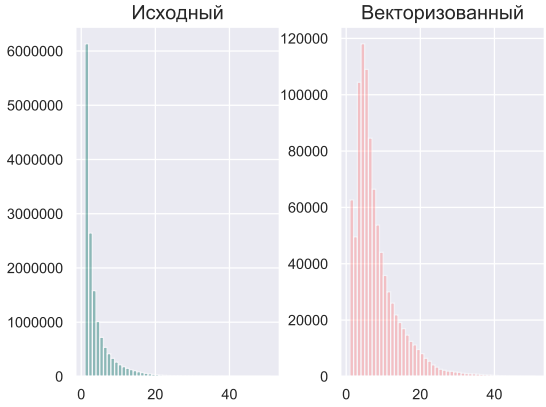


Рис. 10. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Хиббарда при $k = 15$

пиально сильно коррелирующими, это приводит к нерегулярному снижению количества итераций во внутреннем цикле невекторизованной версии. В итоге мы имеем очень непостоянный график как теоретического, так и и экспериментального ускорения, с низкой эффективностью векторизации и резкими провалами, опускающимися даже ниже единицы.

Заключение

На примере сортировки Шелла был рассмотрен крайне неудобный для векторизации контекст исполнения, представленный гнездом циклов с нерегулярным количеством итераций внутреннего цикла. Даже теоретическое ускорение для такого контекста далеко от идеальной верхней границы, а на практике и вовсе может быть получен негативный эффект от применения векторизации. Такие свойства характерны прежде всего для дискретных задач, к числу которых относятся задачи сортировки, комбинаторики и перечисления. При использовании векторизации для таких задач характер их исполнения должен быть оценен отдельно в каждом конкретном случае для избежания неожиданной деградации эффективности программного кода.

Работа выполнена в МСЦ РАН в рамках государственного задания по теме 0065-2018-0409 «Разработка архитектур, системных решений и методов для создания вычислительных комплексов и распределенных сред мультипетафлопсного диапазона производительности, в том числе нетрадиционных архитектур микропроцессоров». Для расчетов при проведении исследований использовался суперкомпьютер МВС-10П, находящийся в МСЦ РАН.

Список литературы

[1] Intel 64 and IA-32 Architectures Software Developer's Manual. Combined volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4. Intel Corporation. 2017. URL: https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4 (дата обращения: 25.10.2018). ↑1

[2] Jeffers James, Reinders James, Sodani Avinash. *Intel Xeon Phi processor high performance programming*, Knights Landing Edition, 2 ed., Morgan Kaufmann Publ., 2016, 632 p. ↑1

[3] Intel C++ Compiler 16.0 User and Reference Guide. Intel Corporation. 2015. URL: https://software.intel.com/en-us/articles/intel-c-compiler-160-for-windows-release-notes-for-intel-parallel-studio-xe-2016 (дата обращения: 25.10.2018). ↑2

[4] Intel Intrinsics Guide. URL: https://software.intel.com/sites/landingpage/IntrinsicsGuide (дата обращения: 02.11.2018). ↑2

[5] Кнут Дональд. *Искусство программирования*. Т. 3: *Сортировка и поиск*, Вильямс, М., 1994, 832 с. ↑3

[6] Волконский В.Ю., Окунев С.К.. «Предикатное представление как основа оптимизации программы для архитектур с явно выраженной параллельностью», *Информационные технологии*, 2003, №4, с. 36–45. ↑6

[7] A168604, последовательность Хиббарда. URL: https://oeis.org/A168604 (дата обращения 02.11.18). ↑8

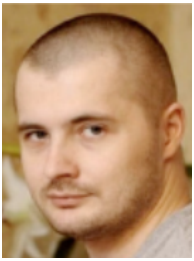
[8] A003586, последовательность Пратта. https://oeis.org/A003586 (дата обращения 02.11.18). ↑8

[9] A033622, последовательность Седжвика. URL: https://oeis.org/A033622 (дата обращения 02.11.18). ↑8

Пример ссылки на эту публикацию:

А. А. Рыбаков, С. С. Шумилин. «Исследование эффективности векторизации гнезд циклов с нерегулярным числом итераций». Подготовлено в журнал *Программные системы: теория и приложения* 30 ноября 2018 г.

Об авторах:



Алексей Анатольевич Рыбаков

Рыбаков Алексей Анатольевич – к ф.-м. н., внс МСЦ РАН – филиала ФГУ ФНЦ НИИСИ РАН. Области научных интересов – математическое моделирование задач газовой динамики с использованием суперкомпьютеров, методы построения и управления расчетными сетками, дискретная математика, теория графов, модели случайных графов, параллельное программирование, функциональное программирование.

iD: 0000-0002-9755-8830
e-mail: rybakov.aax@gmail.com

Сергей Сергеевич Шумилин

Шумилин Сергей Сергеевич, старший инженер МСЦ РАН – филиала ФГУ ФНЦ НИИСИ РАН. Область научных интересов: машинное обучение, анализ данных, алгоритмы, параллельное программирование.

iD: 0000-0002-3953-7054
e-mail: shumilin@jscc.ru

UDC 004.42

A. Rybakov, S. Shumilin. *Study of the vectorization efficiency of loop nests with an irregular number of iterations.*

Abstract. Vectorization of computation is an important low-level optimization used to create highly efficient parallel code. However, when used in context with an unknown program execution profile, a danger of low effectiveness of the application emerges. This is especially pronounced when vectorizing nests of cycles with an irregular number of iterations of the inner loop. The article discusses a comparison of the theoretical and practical efficiency of vectorization on the example of Shell sorting, since this program code is extremely inconvenient for vectorization.

2010 *Mathematics Subject Classification*: 68N19

Sample citation of this publication:

A. Rybakov, S. Shumilin. "Study of the vectorization efficiency of loop nests with an irregular number of iterations". Prepared for the journal *Program Systems: Theory and Applications* at November 30, 2018 (*In Russian*).