

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, поддержанное в семействах микропроцессоров Intel Xeon Phi KNL 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

TODO.

© А. А. Рыбаков, С. С. Шумилин, 2018

© Межведомственный суперкомпьютерный центр Российской академии наук - филиал Федерального государственного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук» (МСЦ РАН - филиал ФГУ ФНЦ НИИСИ РАН), 2018

Подготовлено 1 ноября 2018 г. в журнал «Программные системы: теория и приложения».

Skylake. В следующих поколениях процессоров набор инструкций AVX-512 расширяется еще больше, в него входят команды 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, 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 присутствует `shuf` и пересылка по маске. Для большего числа операций AVX-512 реализованы соответствующие интринсики, 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 интринсики, 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 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.	$k_i = \begin{cases} 9 \cdot 2^i - 9 \cdot 2^{\frac{i}{2}} + 1, k \text{ even} \\ 8 \cdot 2^i - 6 \cdot 2^{\frac{i+1}{2}} + 1, k \text{ odd} \end{cases}$

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     for (k = ks[k_ind]; k > 0; k = ks[--k_ind])
05     {
06         for (i = k; i < n; i++)
07         {
08             float t = m[i];
09             for (j = i; j >= k; j -= k)
10             {
11                 if (t < m[j - k])
12                 {
13                     m[j] = m[j - k];
14                 }
15                 else
16                 {
17                     break;
18                 }
19             }
20             m[j] = t;
21         }
22     }
23 }
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

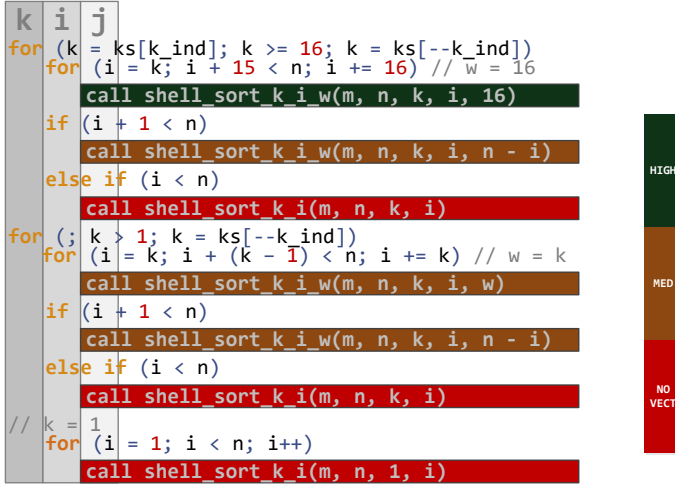


Рис. 2. Декомпозиция сортировки Шелла для выделения векторизуемых участков кода

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 неvectorизуемого остатка. В последнюю очередь рассматривается неvectorизуемая 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` со-

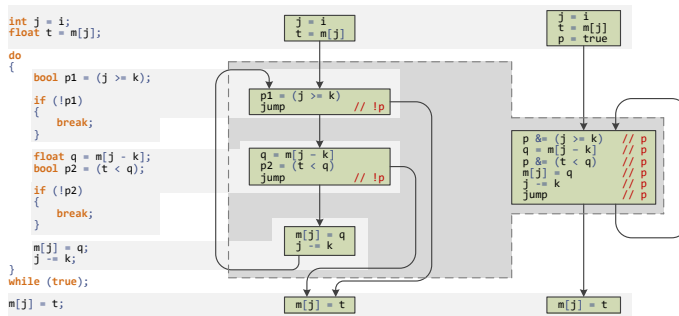


Рис. 3. Схема перевода тела внутреннего цикла сортировки Шелла в предикатную форму

седних 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, 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 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 плотности 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 векторном коде операцию scatter (Рис. 4, строка 25) множественной записи

```

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

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 векторизованном 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 теорети-

ческим ускорением будем подразумевать просто отношение количества итераций внутреннего цикла не векторизованной версии к количеству итераций внутреннего цикла в оптимизированной векторизованной версии кода. Определим данное ускорение более формально.

Сначала рассмотрим не векторизованный 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 величину через T).

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

Теперь рассмотрим векторизованную версию кода. Так же 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 не векторизованных циклов, объединяемых в данный векторизованный цикл. С учетом векторизации хвостовой части цикла, получим следующую формулу для общего количества итераций векторизованного внутреннего цикла.

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

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 Шелла, Хиббарда, Пратта и Сэджвика. 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

TODO. Тут должно быть описание экспериментальных запусков. 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), 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.

TODO. Проанализируем гистограммы распределения количества итераций внутреннего цикла для фиксированного значения 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. Таким образом, выполнения внутренних циклов с малым количеством итераций расворяются в векторизованном коде, что снижает его производительность. Стоит отметить, что для последовательности Шелла в сортировке с использованием векторизованного кода количество итераций равно двум встречается с максимальной частотностью. Для последовательности Хиббарда таковым является количество итераций равно четырем. Теоретическая оценка эффективности векторизации с использованием последовательности Сэджвика оказалась достаточно точной, что показывают результаты экспериментов. Разброс эффективности находится в почти схожих пределах (1.5 - 3).

Стоит отметить скачкообразную форму графика распределения эффективности для последовательности шагов Пратта. Это связано с особенностями формулы вычисления шагов, которые приводят к наложению подпоследовательностей и проходам по уже отсортированным элементам, что приводит к малому числу итераций и, соответственно, к низкой эффективности векторизации.

Также на графике теоретического значения эффективности для последовательности Пратта две точки имеют значение меньше единицы. Это связано с зависимостью адресов обращения в память от

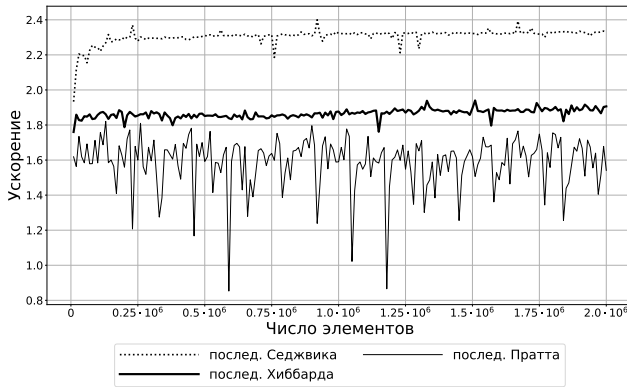


Рис. 5. Сравнение экспериментального ускорения векторизованной версии сортировки Шелла для различных последовательностей шагов

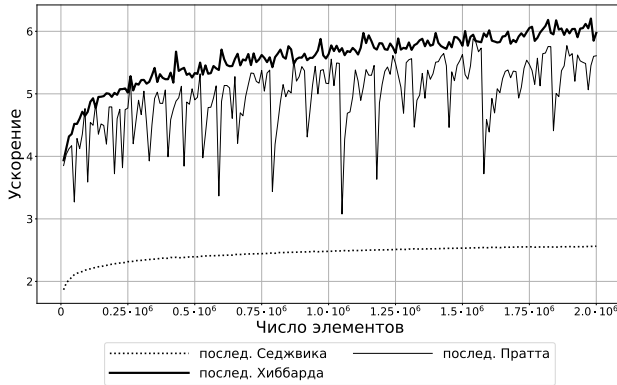


Рис. 6. Сравнение теоретического ускорения векторизованной версии сортировки Шелла для различных последовательностей шагов

количества итераций в циклах с неизвестным числом итераций внутреннего цикла, а также чрезмерным использованием медленных команд (например, `i32gather/i32scatter`).



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

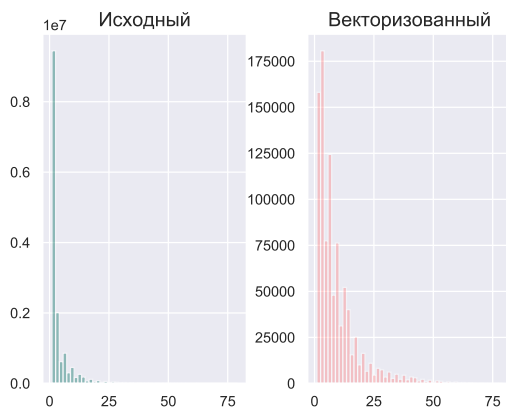


Рис. 8. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Шелла при $k = 15$

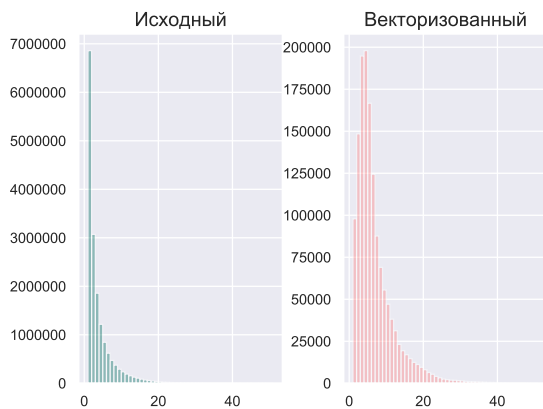


Рис. 9. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Хиббарда при $k = 3$

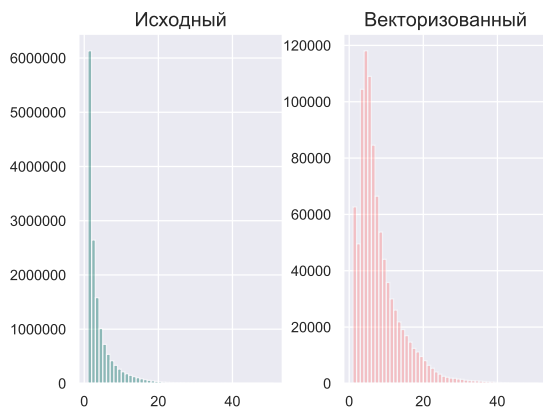


Рис. 10. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Хиббарда при $k = 15$

Заключение

В работе приведены теоретические расчеты эффективности векторизации. Также представлены экспериментальные результаты сортировки Шелла в векторизованной форме, полученные на микропроцессоре Intel Xeon Phi 7290 KNL. Приведен векторизованный вариант ядра сортировки Шелла, написанный на языке C с использованием набора инструкций AVX-512. Результаты приведены для трех последовательностей шагов, представляющих наибольший интерес: Сэджвика, Хиббарда и Пратта.

Из экспериментальных результатов видно, что эффективность векторизации каждой из последовательностей ограничена и значительно меньше 16.

Список литературы

- [1] Krappel T., Riedelbauch S. Scale resolving flow simulations of a Francis turbine using highly parallel CFD simulations. High Performance Computing in Science and Engineering'16. Springer Intern. Publ., 2016, pp. 499–510..↑
- [2] Fu H.H., Liao J.F., Yang J.Z., et al. The Sunway TaihuLight supercomputer: system and applications. Sci. China Inf. Sci., 2016, vol. 59, no. 7, pp. 072001. DOI: 10.1007/s11432-016-5588-7..↑
- [3] Klenk B., Fröning H. An overview of MPI characteristics of exascale proxy applications. Proc. ISC High Performance. J.M. Kunkel et al. (eds.), LNCS, 2017, vol. 10266, pp. 217–236..↑
- [4] Baranov A., Telegin P., Tikhomirov A. Comparison of auction methods for job scheduling with absolute priorities. Proc. Conf. Parallel Computing Technologies (PaCT 2017), Malyshkin V. (eds.), LNCS, 2017, vol. 10421, pp. 387–395..↑
- [5] Аладышев О.С., Киселёв Е.А., Савин Г.И., Телегин П.Н., Шабанов Б.М. Влияние характеристик внешней памяти суперкомпьютерных комплексов на выполнение параллельных программ // Системы и средства информатики. 2014. Т. 24. Вып. 4. С. 111–123..↑
- [6] Рыбаков А.А. Внутреннее представление и механизм межпроцессного обмена для блочно-структурированной сетки при выполнении расчетов на суперкомпьютере // Программные системы: теория и приложения. 2017. Т. 8. Вып. 1. С. 121–134..↑
- [7] Бендерский Л.А., Любимов Д.А., Рыбаков А.А. Анализ эффективности масштабирования при расчетах высокоскоростных турбулентных течений на суперкомпьютере RANS/ILES методом высокого разрешения // Тр. НИИСИ РАН. 2017. Т. 7. № 4. С. 32–40..↑

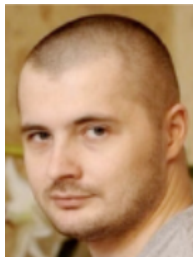
- [8] Дикарев Н.И., Шабанов Б.М., Шмелев А.С. Моделирование параллельной работы ядер векторного потокового процессора с общей памятью // Программные системы: теория и приложения. 2018. № 1 (36). С. 37–52.. ↑
- [9] Jeffers J., Reinders J., Sodani A. Intel Xeon Phi processor high performance programming. Knights Land-ing Edition. Morgan Kaufmann Publ., 2016, 632 p.. ↑
- [10] 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: 18.05.2018).. ↑
- [11] 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).. ↑

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* 1 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

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

ABSTRACT. Computation vectorization 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 efficiency of vectorization of loop nests with an irregular number of iterations”. Prepared for the journal *Program Systems: Theory and Applications* at November 1, 2018 (*In Russian*).