



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. Особенности набора инструкций AVX-512 позволяют применять векторизацию для сложного программного контекста, в частности для гнезд циклов и циклов с сильно разветвленным управлением. При использовании векторных инструкций для контекста с неизвестным профилем исполнения существует опасность низкой эффективности векторизации. Особенно ярко это проявляется при векторизации гнезд циклов с нерегулярным числом итераций внутреннего цикла.

В статье рассматривается практический подход к векторизации гнезд циклов, основанный на предикатном представлении программы. В качестве примера приводится реализация сортировки Шелла, компактная реализация которой состоит из гнезда циклов, в котором количество итераций внутреннего цикла носит нерегулярный характер и зависит от номеров итераций внешних циклов. Такой контекст является крайне неудобным для векторизации.

Приводится сравнение теоретической и практической эффективности векторизации сортировки Шелла, рассматриваются особенности этого программного контекста и объясняется их негативное влияние на производительность векторизованного кода. Полученные результаты могут быть использованы исследователями и разработчиками программного обеспечения для обнаружения причин низкой эффективности векторизации программного кода с похожими особенностями.

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, представляющему собой 512-битное

расширение 256-битных AVX инструкций из набора команд Intel x86 [1], поддержанное в семействах микропроцессоров Intel, начиная с 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, поддерживаемые в 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 [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 присутствует операция `shuf` и пересылка по маске. Для большинства операций AVX-512 реализованы соответствующие интринсики, раскрываемые

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).

Появление набора инструкций AVX-512 вызвало большой интерес со стороны исследователей и разработчиков программного обеспечения, в настоящее время ведутся работы по их использованию для оптимизации приложений из разных научных областей. Можно отметить, например, работы по векторизации ядра программного кода LAMMPS [5], операций с разреженными матрицами [6] и матрицами специального вида [7]. В работе [8] описан подход векторизации гнезда циклов на примере построения множества Мандельброта.

Можно встретить работы, в которых описывается применение векторизации с использованием набора инструкций AVX-512 для других приложений из широкого круга, начиная от задач ядерной физики [9], и численного решения уравнений мелкой воды [10] и заканчивая задачами дискретной математики, включая векторизацию задач сортировки [11] или генератора псевдослучайных чисел [12].

Целью данной работы является исследование эффективности векторизации сложных контекстов программы и обнаружение „подводных камней“, приводящих к снижению эффективности векторизованного кода. В качестве объекта исследования для данной статьи была выбрана сортировка Шелла, обладающая крайне неудобным контекстом исполнения для векторизации с помощью инструкций AVX-512.

Главным вкладом данной статьи является объяснение объективных причин низкой эффективности векторизации гнезд циклов с нерегулярным количеством итераций внутреннего цикла. Отсутствие информации о профиле выполнения программы и внешне привлекательная простая реализация может ввести в заблуждение исследователей и разработчиков программного обеспечения, поэтому знание описанной специфики может оказаться полезным при выполнении векторизации программного кода.

В первом разделе приводится краткое описание сортировки Шелла, рассматриваемой в качестве тестового примера. Во втором разделе

описан подход к векторизации гнезда циклов, представляющего ядро сортировки, и объяснены узкие места векторизации данного контекста. В третьем разделе приводятся формулы вычисления идеального теоретического ускорения от векторизации сортировки Шелла для разных шагов сортировки. Четвертый раздел содержит описание эксперимента, его результаты и сравнение с теоретическими данными. В пятом разделе приводится сравнение с близкими работами, в которых также затрагиваются вопросы векторизации циклов с нерегулярным числом итераций.

1. Описание сортировки Шелла

Сортировка Шелла [13] 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 представленные в таблице 1 [14–16]:

ТАБЛИЦА 1. Различные последовательности шагов, используемые в сортировке Шелла

Последовательность	Формула
Последовательность Шелла, 1959 г.	$k_1 = \lfloor \frac{N}{2} \rfloor, k_i = \lfloor \frac{k_{i-1}-1}{2} \rfloor, k_t = 1$
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 (рисунок 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 }
```

РИСУНОК 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 рисунке 2 представлена схема декомпозиции алгоритма сортировки Шелла, в котором явно выделены участки с разной шириной 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).

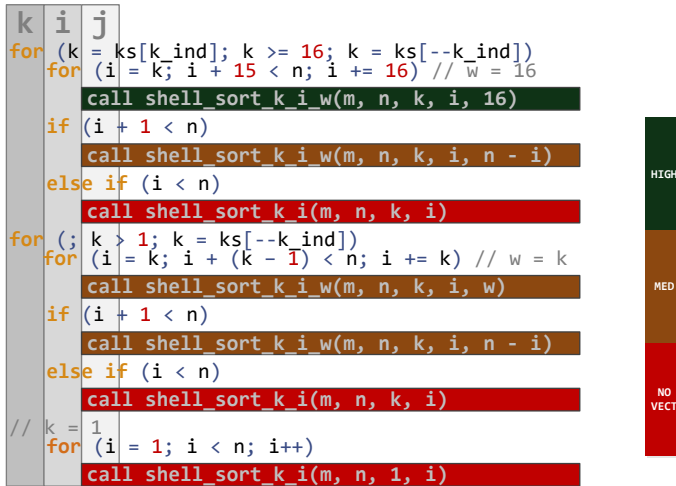


Рисунок 2. Декомпозиция сортировки Шелла для выделения векторизуемых участков кода

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 не векторизуемого остатка.

В последнюю очередь рассматривается не векторизуемая 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 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 [17], после чего заменить все инструкции векторными аналогами, а предикаты – векторными масками (рисунок 3). При этом если до векторизации внутренний цикл завершал работу при

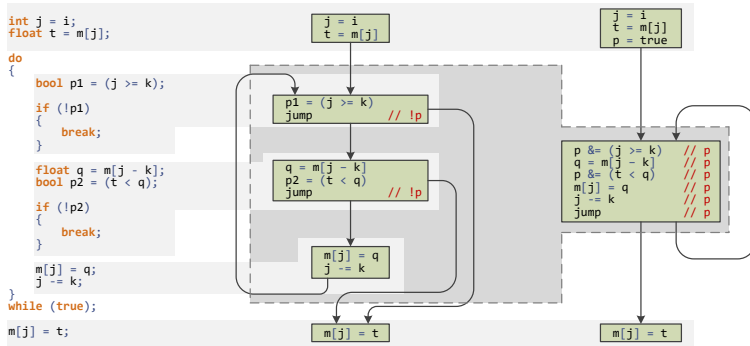


Рисунок 3. Схема перевода тела внутреннего цикла сортировки Шелла в предикатную форму

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 плотности масок векторных инструкций (то есть малый процент элементов векторов на самом деле обрабатывается при выполнении векторной операции). Такой негативный эффект при векторизации гнезд циклов характерен для дискретных задач, в том числе для задач сортировки, поиска, комбинаторного перебора с отсечением ветвей и т.д.

Для задач численного моделирования физических процессов наоборот данный эффект практически незаметен, профиль исполнения отдельных участков программного кода меняется не сильно при небольших изменениях входных данных.

Векторизованная версия ядра сортировки Шелла представлена на рисунке 4:

```

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 }

```

Рисунок 4. Векторизованный вариант ядра сортировки Шелла

Также стоит обратить внимание на появившуюся в векторном коде операцию `scatter` (рисунок 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 (рисунок 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 (рисунок 4, строки 10, 16, 19). Это масочные команды чтения и записи по последовательным адресам, реализованные интринсиками `_mm512_mask_load_ps` и `_mm512_mask_store_ps`, которые раскрываются в операции невыровненного обращения в память **vmovups**.

В данном случае нельзя использовать выровненные операции обращения в память, так как в общем случае адреса обращения в данных командах конечно не являются выровненными. В микропроцессорах Intel Xeon Phi KNL и Intel Xeon Skylake скорость обращения в память команд **vmovaps** не отличается от скорости обращения **vmovups** при условии выровненного обращения, поэтому компилятор `icc` вовсе не генерирует инструкции **vmovaps** [18]. Таким образом, при невыполнении условия выровненности адресов вместо аварийного завершения мы получаем снижение производительности программы, однако данное снижение не сравнимо с медленной работой инструкций **gather** и **scatter**.

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 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. 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 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 Хиббарда, Пратта и Сэджевика [14–16]. 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

Для проведения экспериментов были использованы две версии исходного кода: неоптимизированная функция сортировки и реализованная с помощью функций-интринсиков. Обе версии сортировки были собраны компилятором `icc` с уровнем оптимизации `-O3` и с разрешением генерировать инструкции `AVX-512` (`-xmic-avx512`) для микропроцессора Intel Xeon Phi KNL. Тестовые запуски и замер времени исполнения выполнялись на одном вычислительном узле суперкомпьютера МВС-10П, на сегменте, базирующимся на микропроцессорах Intel Xeon Phi 7290.

На рисунках 5 и 6 представлены результаты теоретических оценок и экспериментальных запусков, исходя из которых получены теоретическое и экспериментальное ускорение сортировки Шелла для последовательностей шагов Шелла, Хиббарда и Сэджевика:

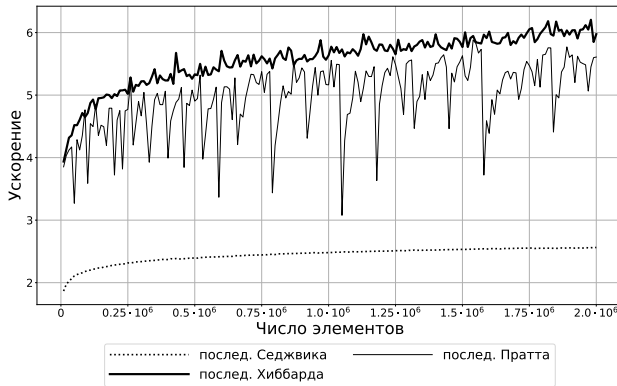


Рисунок 5. Сравнение теоретического ускорения векторизованной версии сортировки Шелла для различных последовательностей шагов

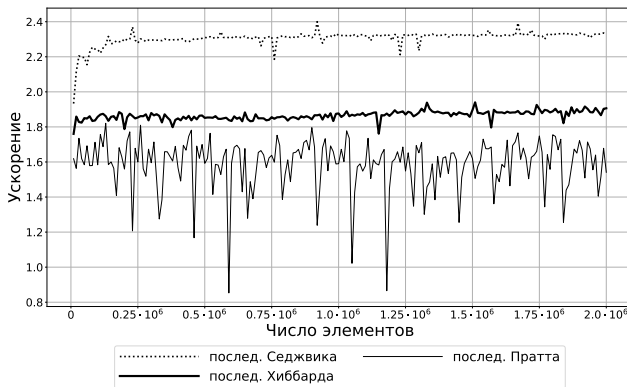


Рисунок 6. Сравнение экспериментального ускорения векторизованной версии сортировки Шелла для различных последовательностей шагов

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 (что, однако, является неплохим результатом).

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

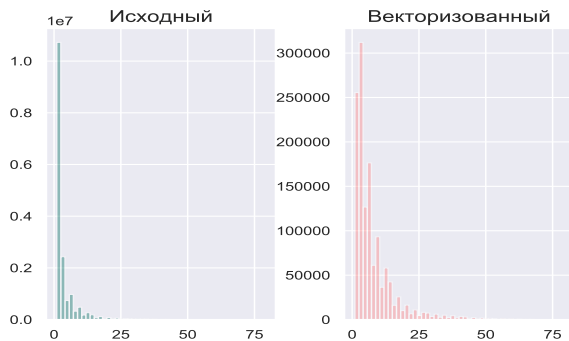


Рисунок 7. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Шелла при $k = 4$

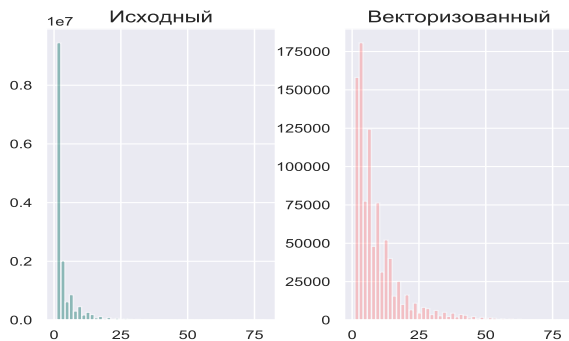


Рисунок 8. Гистограмма распределения количества итераций внутреннего цикла при сортировке с последовательностью Шелла при $k = 15$

Наличие только одной итерации внутреннего цикла в неекто-



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



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

ризованной 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 k s 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.

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 принципиально 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 экспериментального ускорения, с низкой эффективностью векторизации и резкими провалами, опускающимися даже ниже единицы.

5. Близкие работы

Во введении к данной работе были упомянуты статьи из разных областей исследований, в которых освещались вопросы векторизации кода с помощью набора инструкций AVX-512. В данном разделе остановимся только на двух из них, в которых исследуется программный контекст, близкий к нашему.

Условно программный контекст можно разделить на непрерывный и дискретный. К непрерывному контексту отнесем такой программный код, профиль исполнения которого не сильно меняется при небольших изменениях входных данных. Для дискретного контекста, наоборот, характерно непредсказуемое изменение профиля исполнения при незначительных изменениях во входных данных. Задача сортировки является дискретной задачей, именно это отражается в свойстве нерегулярности количества итераций внутреннего цикла.

Из упомянутых работ наиболее близкой к данной является векторизация построения множества Мандельброта, описанная в [8]. Ядро построения множества Мандельброта представляет собой гнездо из двух циклов, в котором количество итераций внутреннего цикла нерегулярно и зависит от номера итерации внешнего цикла. Хотя

тело внутреннего цикла гораздо проще (состоит из одного оператора $z = z * z + c$ и условия выхода), однако основной подход остается тем же – заменить скалярное представление внутреннего цикла на векторное с использованием масочных инструкций.

Также в задаче построения множества Мандельброта значительно лучше оценка максимального теоретического ускорения (около 15 раз на вещественных данных одинарной точности против показателей 2,5–6 в текущей работе), что приводит к финальному ускорению кода примерно в 6,5 раз без использования дополнительных оптимизаций алгоритма (в текущей работе этот показатель находится на отметке 1,6–2,3).

В работе [11] авторы приводят исследование алгоритмов сортировки с использованием AVX-512 на процессоре Intel KNL. Они применяют набор инструкций в разных контекстах: новый алгоритм разделения множества, применение битонической сортировки для небольших массивов и предлагают модифицированный вариант быстрой сортировки.

Быстрая сортировка представляет собой двухступенчатый алгоритм, который разделяет исходный массив до тех пор, пока размер подмассивов не станет достаточно малым для использования битонической сортировки. Реализация гибридного алгоритма сортировки, описанного в [11] изначально предполагает векторное исполнение (особенно в части битонической сортировки коротких массивов), поэтому отсутствует ее сравнение с не векторизованной версией, однако на длинных массивах достигнуто ускорение 1,4 раза по сравнению с реализацией C++ STL.

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.

Была рассмотрена векторная реализация сортировки Шелла и проанализированы три основные причины ее низкой эффективности. Основной причиной низкой эффективности векторизации гнезда циклов с нерегулярным числом итераций внутреннего цикла является большой

разброс количества итераций копий циклов, которые реализуются одновременно с помощью векторных инструкций. Такой разброс порождает большое количество векторных команд с разреженными масками, что закономерно приводит к снижению производительности.

Другой причиной низкой производительности является наличие в коде межитерационной зависимости по данным, которая ограничивает ширину векторизации при $k < 16$. Наконец третьей важной причиной является наличие в коде команд множественного обращения в память, которые сильно замедляют скорость работы программы, несмотря на свою гибкость и удобство использования.

Таким образом, полученные результаты говорят о том, что при выполнении векторизации программного кода, который содержит циклы с сильно изменяющимся и нерегулярным количеством итераций, необходимо отдельно оценивать возможность векторизации для избежания неожиданной деградации эффективности программного кода.

Работа выполнена в МСЦ РАН в рамках государственного задания по теме 0065-2019-0016 (reg. no. AAAA-A19-119011590098-8). 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. 2019. 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> (access date: 21.10.2019). \uparrow_{78}
- [2] J. Jeffers, J. Reinders, A. Sodani. *Intel Xeon Phi processor high performance programming*, Knights Landing Edition, 2 ed., Morgan Kaufmann Publ., 2016, 632 pp. \uparrow_{78}
- [3] Intel C++ Compiler 16.0 User and Reference Guide. Intel Corporation. 2015. URL: <https://software.intel.com/en-us/download/intel-c-compiler-160-update-4-user-and-reference-guide> (access date: 21.10.2019). \uparrow_{78}
- [4] Intel Intrinsics Guide. URL: <https://software.intel.com/sites/landingpage/IntrinsicsGuide> (access date: 21.10.2019). \uparrow_{78}
- [5] W. McDaniel, M. Hohnerbach, R. Canales. et al.. "LAMMPS' PPPM Long-Range Solver for the Second Generation Xeon Phi", *J.M. Kunkel et al. (Eds.): ISC High Performance 2017, LNCS, 10266* (2017), pp. 61–78. \uparrow_{79}
- [6] T. Malas, T. Kurth, J. Deslippe. "Optimization of the Sparse Matrix-Vector Products of an IDR Krylov Iterative Solver in EMGeo for the Intel KNL

- Manycore Processor”, *M. Tauber et al. (Eds.): ISC High Performance Workshops 2016, LNCS, 9945* (2016), pp. 378–389. ^{↑₇₉}
- [7] Л. А. Бендерский, А. А. Рыбаков, С. С. Шумилин. «Векторизация перемножения малоразмерных матриц специального вида с использованием инструкций AVX-512», *Современные информационные технологии и ИТ-образование*, **14:3** (2018), с. 594–602. ^{↑₇₉}
- [8] O. Krzikalla, F. Wende, M. Hohnerbach. “Dynamic SIMD vector lane scheduling”, *Lect. Notes Comput. Sci.*, 2016, no.9945, pp. 354–365. ^{↑_{79, 90}}
- [9] B. Cook, P. Maris, M. Shao. “High Performance Optimizations for Nuclear Physics Code MFDn on KNL”, *M. Tauber et al. (Eds.): ISC High Performance Workshops 2016, LNCS, 9945* (2016), pp. 366–377. ^{↑₇₉}
- [10] C. R. Ferreira, K. T. Mandli, M. Bader. “Vectorization of Riemann solvers for the single- and multi-layer shallow water equations”, *Proceedings of the 2018 International Conference on High Performance Computing and Simulation, HPCS 2018*, 2018, pp. 415–422. ^{↑₇₉}
- [11] B. Bramas. “Fast Sorting Algorithms using AVX-512 on Intel Knights Landing”, *International Journal of Advanced Computer Science and Applications*, **8:10** (2017), pp. 337–344. ^{↑_{79, 91}}
- [12] М. С. Гуськова, Л. Ю. Бараш, Л. Н. Шур.. «Применение AVX512-векторизации для увеличения производительности генератора псевдослучайных чисел», *Труды ИСП РАН*, **30:1** (2018), с. 115–126. ^{↑₇₉}
- [13] Д. Кнут. *Искусство программирования*. Т. 3: *Сортировка и поиск*, Вильямс, М., 1994, 832 с. ^{↑₈₀}
- [14] A168604, последовательность Хиббарда. URL: <https://oeis.org/A168604> (дата обращения 21.10.19). ^{↑_{80, 86}}
- [15] A003586, последовательность Пратта. <https://oeis.org/A003586> (дата обращения 21.10.19). ^{↑_{80, 86}}
- [16] A033622, последовательность Седжвика. URL: <https://oeis.org/A033622> (дата обращения 21.10.19). ^{↑_{80, 86}}
- [17] В. Ю. Волконский, С. К. Окунев. «Предикатное представление как основа оптимизации программы для архитектур с явно выраженной параллельностью», *Информационные технологии*, 2003, №4, с. 36–45. ^{↑₈₃}
- [18] Using Intel AVX without Writing AVX. URL: <https://software.intel.com/en-us/articles/using-intel-avx-without-writing-avx> (access date: 21.10.2019). ^{↑₈₅}


Поступила в редакцию 28.02.2019
 Переработана 21.10.2019
 Опубликовано 30.11.2019


Рекомендовал к публикации

к.т.н. С. А. Амелькин

Пример ссылки на эту публикацию:

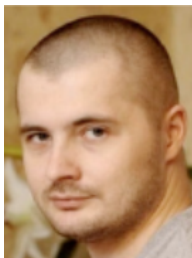
А. А. Рыбаков, С. С. Шумилин. «Исследование эффективности векторизации гнезд циклов с нерегулярным числом итераций». *Программные системы: теория и приложения*, 2019, **10**:4(43), с. 77–96.

 10.25209/2079-3316-2019-10-4-77-96

 http://psta.psiras.ru/read/psta2019_4_77-96.pdf

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.



0000-0002-9755-8830

e-mail: rybakov.aax@gmail.com

Сергей Сергеевич Шумилин



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



0000-0002-3953-7054

e-mail: shumilin@jscs.ru

CSCSTI 50.07.05, 50.33.04

UDC 519.681.5:004.272.25

Alexey Rybakov, Sergei Shumilin. *Study of the vectorization efficiency 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.

Key words and phrases: vectorization, AVX-512, loop sockets with an irregular number of iterations, Shell sorting, theoretical acceleration.

2010 *Mathematics Subject Classification:* 68W10; 65P99, 68M07


References

- [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. 2019. 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> (data obrashcheniya: 21.10.2019).^{↑₇₈}
- [2] J. Jeffers, J. Reinders, A. Sodani. *Intel Xeon Phi processor high performance programming*, Knights Landing Edition, 2 ed., Morgan Kaufmann Publ., 2016, 632 pp.^{↑₇₈}
- [3] Intel C++ Compiler 16.0 User and Reference Guide. Intel Corporation. 2015. URL: <https://software.intel.com/en-us/download/intel-c-compiler-160-update-4-user-and-reference-guide> (data obrashcheniya: 21.10.2019).^{↑₇₈}
- [4] Intel Intrinsics Guide. URL: <https://software.intel.com/sites/landingpage/IntrinsicsGuide> (data obrashcheniya: 21.10.2019).^{↑₇₈}
- [5] W. McDoniel, M. Hohnerbach, R. Canales. et al.. "LAMMPS' PPPM Long-Range Solver for the Second Generation Xeon Phi", *J.M. Kunkel et al. (Eds.): ISC High Performance 2017, LNCS, 10266* (2017), pp. 61—78.^{↑₇₉}
- [6] T. Malas, T. Kurth, J. Deslippe. "Optimization of the Sparse Matrix-Vector Products of an IDR Krylov Iterative Solver in EMGeo for the Intel KNL Manycore Processor", *M. Taufer et al. (Eds.): ISC High Performance Workshops 2016, LNCS, 9945* (2016), pp. 378–389.^{↑₇₉}




© A. A. RYBAKOV, S. S. SHUMILIN, 2019

© JOINT SUPERCOMPUTER CENTER OF RAS, 2019

© PROGRAM SYSTEMS: THEORY AND APPLICATIONS (DESIGN), 2019


 10.25209/2079-3316-2019-10-4-77-96




- [7] L. A. Benderskiy, A. A. Rybakov, S. S. Shumilin. “Vektorizatsiya peremnozheniya malorazmernykh matrits spetsial’nogo vida s ispol’zovaniyem instruktsiy AVX-512”, *Sovremennyye informatsionnyye tekhnologii i IT-obrazovaniye*, **14:3** (2018), pp. 594–602.   ^{↑₇₉}
- [8] O. Krzikalla, F. Wende, M. Hohnerbach. “Dynamic SIMD vector lane scheduling”, *Lect. Notes Comput. Sci.*, 2016, no.9945, pp. 354–365. ^{↑_{79,90}}
- [9] B. Cook, P. Maris, M. Shao. “High Performance Optimizations for Nuclear Physics Code MFDn on KNL”, *M. Taufer et al. (Eds.): ISC High Performance Workshops 2016, LNCS, 9945* (2016), pp. 366–377. ^{↑₇₉}
- [10] C. R. Ferreira, K. T. Mandli, M. Bader. “Vectorization of Riemann solvers for the single- and multi-layer shallow water equations”, *Proceedings of the 2018 International Conference on High Performance Computing and Simulation, HPCS 2018*, 2018, pp. 415–422. ^{↑₇₉}
- [11] B. Bramas. “Fast Sorting Algorithms using AVX-512 on Intel Knights Landing”, *International Journal of Advanced Computer Science and Applications*, **8:10** (2017), pp. 337–344.  ^{↑_{79,91}}
- [12] M. S. Gus’kova, L. Yu. Barash, L. N. Shchur.. “Primeneniye AVX512-vektorizatsii dlya uvelicheniya proizvoditel’nosti generatora psevdosluchaynykh chisel”, *Trudy ISP RAN*, **30:1** (2018), pp. 115–126. ^{↑₇₉}
- [13] D. Knut. *Iskusstvo programmirovaniya*. V. 3: *Sortirovka i poisk*, Vil’yams, M., 1994, 832 pp. ^{↑₈₀}
- [14] A168604, posledovatel’nost’ Khibbarda. URL: <https://oeis.org/A168604> (data obrashcheniya 21.10.19). ^{↑_{80,86}}
- [15] A003586, posledovatel’nost’ Pratta. <https://oeis.org/A003586> (data obrashcheniya 21.10.19). ^{↑_{80,86}}
- [16] A033622, posledovatel’nost’ Sedzhvika. URL: <https://oeis.org/A033622> (data obrashcheniya 21.10.19). ^{↑_{80,86}}
- [17] V. Yu. Volkonskiy, S. K. Okunev. “Predikatnoye predstavleniye kak osnova optimizatsii programmy dlya arkhitektury s yavno vyrazhennoy parallel’nost’yu”, *Informatsionnyye tekhnologii*, 2003, no.4, pp. 36–45. ^{↑₈₃}
- [18] Using Intel AVX without Writing AVX. URL: <https://software.intel.com/en-us/articles/using-intel-avx-without-writing-avx> (data obrashcheniya: 21.10.2019). ^{↑₈₅}

Sample citation of this publication:

A. A. Rybakov, S. S. Shumilin. “Study of the vectorization efficiency of loop nests with an irregular number of iterations”. *Program Systems: Theory and Applications*, 2019, **10:4**(43), pp. 77–96. (In Russian).

 10.25209/2079-3316-2019-10-4-77-96

 http://psta.psras.ru/read/psta2019_4_77-96.pdf