Применение AVX512-векторизации для увеличения производительности генератора псевдослучайных чисел

[1,2] *М.С. Гуськова <maria.guskova@rambler.ru>*
[1,2,3] *Л.Ю. Бараш <barash@itp.ac.ru>*
[1,2,3,4] *Л.Н. Щур <shchur@chg.ru>*
[1] *Научный центр РАН в Черноголовке*
142432 Черноголовка, Россия
[2] *Национальный исследовательский университет «Высшая школа экономики»,*
101000 Москва, Россия
[3] *Институт теоретической физики им. Л.Д. Ландау,*
142432 Черноголовка, Россия
[4] *Вычислительный центр им. А.А. Дородницына ФИЦ ИУ РАН, 119333*
Москва, Россия

Аннотация. В работе описывается применение наборов SIMD инструкций (Single Instruction Multiple Data) для параллелизации алгоритма генерации псевдослучайных чисел. Дан обзор расширений MMX, SSE, AVX2, AVX512, реализующих принцип SIMD. Приведен пример реализации генератора LFSR113 с использованием расширения AVX512. Приведен сравнительный анализ скоростей работы различных реализаций алгоритма.

Ключевые слова: псевдослучайные числа; SIMD инструкции; технология AVX-512.

DOI: 10.15514/ISPRAS-2018-30(1)-8

Для цитирования: Гуськова М.С., Бараш Л.Ю., Щур Л.Н. Применение AVX512-векторизации для увеличения производительности генератора псевдослучайных чисел. Труды ИСП РАН, том 30, вып. 1, 2018 г., стр. 115-126. 10.15514/ISPRAS-2018-30(1)-8

1. Введение

Генерация равномерно распределенных случайных чисел необходима для компьютерного моделирования методами Монте-Карло и молекулярной динамики [1]. Для генерации случайных чисел используются генераторы псевдослучайных чисел (ГПСЧ). ГПСЧ использует детерминированные алгоритмы для вычисления чисел, но полученная таким способом последовательность обладает свойствами случайной последовательности. При этом к ГПСЧ предъявляются следующие требования [2].

- Последовательность псевдослучайных чисел должна быть равномерно распределенной, а подпоследовательности одинаковой длины должны быть равновероятными. С практической точки зрения, последовательность псевдослучайных чисел должна пройти набор статистических тестов на равномерное распределение и отсутствие корреляций, кроме того, желательно наличие теоретического обоснования хороших статистических свойств генератора.
- Период генератора должен быть достаточно большим.
- Последовательность должна быть воспроизводимой, чтобы можно было повторить эксперимент.
- Должен быть алгоритм пропуска кусков. Параллельные потоки могут использовать различные куски одной и той же последовательности.
- Должна существовать эффективная реализация генератора с точки зрения скорости вычислений и использования оперативной памяти.

Для ряда задач, использующих методы Монте-Карло, генерация случайных чисел занимает значительную часть вычислительного времени, и увеличение производительности генерации является важной задачей.

В настоящей работе для повышения производительности генератора случайных чисел будут использоваться операции SIMD (Single Instruction Multiple Data). Такие операции позволяют применять одну инструкцию одновременно к нескольким элементам данных. Мы детально рассмотрим применение нового расширения AVX512, которое поддерживается процессорами Intel, начиная с 2017 года.

2. Технология SIMD

С 1997 года по сегодняшний день, компания Intel выпустила 9 расширений набора инструкций архитектур Intel 64 and IA-32 для поддержки векторизации. Это MMX технология, расширения SSE, SSE2, SSE3, SSSE3 и SSE4, AVX, AVX2, AVX512 [3].

Каждое расширение содержит набор SIMD-инструкций для работы с упакованными целыми числами и с упакованными числами с плавающей точкой. Каждому расширению также соответствует свой набор регистров (см. рис.1).

Технология Intel MMX(Multimedia Extensions) впервые была реализована в процессорах семейства Intel Pentium MMX в 1997 году и предназначалась для эффективной обработки аудио- и видеопотоков. MMX инструкции позволяют работать с байтами, словами или двойными словами, расположенными в MMX регистрах. Всего таких регистров 8 (MMX0-MMX7), они 64-битные и соответственно могут содержать одновременно 8 байт или 4 слова, или 2 двойных слова, или 1 учетверенное слово. Это расширение может использоваться для обработки целочисленных массивов с помощью SIMD инструкций.

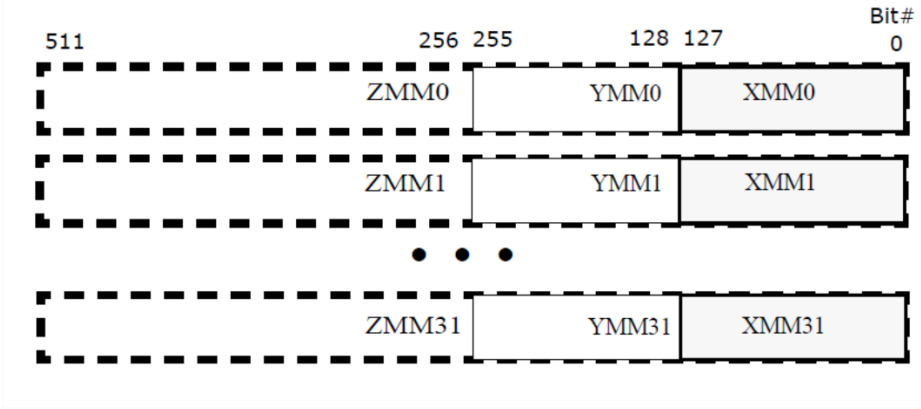

Рис. 1. Регистры XMM, YMM, ZMM
Fig. 1. Registers XMM, YMM, ZMM

Расширение SSE (Streaming SIMD Extensions) появилось в семействе процессоров Intel Pentium III и AMD Athlon XP в 1999 году. SSE является расширением MMX технологии. Расширение включает в себя 8 128-битных XMM регистров (XMM0-XMM7), набор инструкций для работы с числами с плавающей точкой одинарной точности, расположенных в XMM регистрах и набор инструкций для работы с целыми числами в MMX регистрах.

Расширение SSE2 было представлено в процессорах Intel Pentium 4 и Intel Xeon в 2001 году, а также в AMD Opteron и Athlon64 в 2003 году. Инструкции расширений позволяют обрабатывать числа с плавающей точкой двойной точности в XMM регистрах и целые числа, расположенные в регистрах MMX и XMM регистрах. Набор целочисленных инструкций в SSE2 содержит новые IA-32 SIMD операции для работы с 128-битными регистрами и расширяет существующие 64-битные инструкции до работы со 128 битами.

Расширение SSE3 впервые появилось в процессорах Intel Pentium 4 Prescott в 2004 году, а также в AMD Athlon64 в 2005 году. В SSE3 были добавлены 13 новых инструкций, которые улучшают производительность SSE, SSE2 и операций с плавающей точкой.

Расширение SSSE3 (Supplemental SSE) было представлено в процессорах Intel Xeon 5100 серий и семействе процессоров Intel Core 2 в 2006 году. Были добавлены 32 SIMD-инструкции для работы с целочисленными данными.

Расширение SSE4 содержит 54 новые инструкции: 47 из них содержится в SSE4.1 и 7 - в SSE4.2. Оно было впервые реализовано в процессорах семейства Intel Penryn в 2008 году и AMD Bulldozer в 2011 году.

Расширение AVX (Advanced Vector Extensions) является значительным улучшением расширений SSE. Оно поддерживается процессорами Intel и AMD, начиная с 2011 года. Новые YMM регистры - 256-битные. Кроме SIMD

инструкций над MMX регистрами, почти все 128-битные SIMD функции имеют AVX-эквивалент, который имеет трех-операндный синтаксис. Также добавлены совершенно новые инструкции, которых не было в предыдущих расширениях. Расширение использует новый префиксный код VEX, позволяющий работать с новыми YMM регистрами. Большинство инструкций не требует выравнивания по 16- или 32-битным границам. 256-битные AVX-инструкции используют трех-операндный синтаксис, а некоторые – четырех-операндный.

В расширение AVX2 добавлены целочисленные инструкции, ранее доступные только для 128-битных регистров. Здесь же появились новые функциональные операции распространения (broadcast) и перестановок (permute), сдвиговые операции, которые сдвигают элементы вектора на разное количество битов, для извлечения несмежных элементов из памяти. Расширение AVX2 поддерживается процессорами Intel и AMD, начиная с семейств Intel Haswell (2013) и AMD Excavator (2015).

Новый набор инструкций AVX512 для процессоров архитектуры x86-64 компания Intel представила в июле 2013 г., однако первое семейство поддерживающих эту технологию процессоров Intel Xeon Phi x200 (кодовое имя Knights Landing) было выпущено в июне 2016 г.

AVX512 подразделяется на несколько отдельных наборов, для каждого из которых существует свой идентификационный бит. Основной набор AVX512 Foundation включен во все реализации AVX512.

Список отдельных наборов инструкций AVX512:

- F (Foundation) - основной набор инструкций, расширяющий большинство существующих AVX инструкций для работы с 512-битными регистрами общего назначения;
- CDI (Conflict Detection Instructions) обеспечивает лучшую векторизацию циклов;
- ERI (Exponential and Reciprocal Instructions) - инструкции для работы с трансцендентными функциями (экспонента, логарифм, тригонометрические функции);
- PFI (Prefetch Instructions) - инструкции для префетчинга - предварительной загрузки данных и инструкций, что позволяет ускорять дальнейшее выполнение программы;
- BW (Byte and Word Instructions) - инструкции для работы с байтами и словами;
- DQ (Doubleword and Quadword Instructions) - инструкции для работы с двойными и четверными словами (32- и 64-битными целыми числами и числами с плавающей точкой);
- VL (Vector Length Extensions) обеспечивает возможность работы со 128- и 256-битными регистрами (младшими частями 512-битных регистров).

Наборы F, CDI, ERI, PFI впервые реализованы в процессорах Intel Xeon Phi x200 (Knights Landing) в 2016 году, а BW, DQ, VL – в процессорах семейства

Intel Xeon Scalable, выпущенных в 2017 году. Инструкции AVX512 предназначены для работы с 32 новыми 512-битными регистрами общего назначения (ZMM), расширяющими существующие 128- (XMM) и 256-битные (YMM) регистры. Регистры XMM0-XMM15 доступны для инструкций SSE, AVX-128 (AVX, AVX2); регистры YMM0-YMM15 - для AVX256 (AVX, AVX2); регистры ZMM0-ZMM31 - для AVX512F. Для доступа к регистрам XMM16-XMM31 и YMM16-YMM31 необходим набор инструкций AVX512VL.

Микроархитектура процессоров, реализующих AVX512, также подразумевает наличие 8 регистров-масок (k0-k7), позволяющих гибко работать с частями регистров общего назначения. Разрядность этих регистров равна 16 (64 в случае AVX512BW) и позволяет маскировать 16 элементов 512-разрядных регистров при работе с 32-разрядными числами с плавающей точкой или двойными словами. В случае работы с числами с плавающей точкой двойной точности или четверными словами используется только 8 бит масочных регистров.

Мы рассмотрим применение AVX512-векторизации на примере генератора псевдослучайных чисел LFSR113.

3. Генератор псевдослучайных чисел LFSR113

LFSR113 (Linear Feedback Shift Register) является комбинированным генератором из четырех сдвиговых регистров [4]. Метод генерации, основанный на сдвиговых регистрах, использует свойства линейных операций по модулю 2 над битами x_n. Рассмотрим последовательность нулей и единиц:

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \; mod \; 2 \qquad (1)$$

Это линейно-рекуррентное соотношение в поле Z_2 , состоящем из двух элементов, нуля и единицы. Оно называется сдвиговым регистром и имеет период длины $p = 2^k - 1$ тогда и только тогда, когда полином $P(z) = z^k - a_1 z^{k-1} - \cdots - a_k$, называемый характеристическим полиномом рекуррентной последовательности, является примитивным полиномом. Для одного сдвигового регистра выходная последовательность определяется, как $u_n = \sum_{k=1}^{L} x_{ns+k-1} 2^{-k}$, где размер шага s и длина слова $L -$ целые положительные числа.

Генераторы, основанные на сдвиговом регистре, быстрые. Они обладают большой длиной периода при условии правильного выбора примитивных полиномов. Однако в генераторах этого класса были обнаружены корреляции, которые могут привести к систематическим ошибкам в расчетах Монте-Карло [5,6,7]. Для улучшения статистических свойств были разработаны модификации ГСР. LFSR113 является таким комбинированным генератором.

Пусть имеется J рекуррентных последовательностей (1) и $j-$ая последовательность имеет характеристический полином $P_j(z)$ степени k_j. Пусть он является примитивным, тогда период последовательности равен $p_j =$

$2^{k_j} - 1$. Предположим, что $P_j(z)$ не имеют общих делителей, и p_j являются взаимно простыми. Для каждой последовательности нулей и единиц определим выходную последовательность при помощи формулы $u_{j,n} = \sum_{k=1}^{L} x_{ns_j+k-1} 2^{-k}$. Тогда для комбинированного генератора выходная последовательность определяется, как $u_n = u_{1,n} \oplus \cdots \oplus u_{J,n}$ [4]. Период генератора равен $p = (2^{k_1} - 1) \times \cdots \times (2^{k_J} - 1)$.

Для LFSR113 длина слова $L = 32$, количество сдвиговых регистров $J = 4$,

$$x_{1,n} = x_{1,n-31} \oplus x_{1,n-25}, s_1 = 18,$$
$$x_{2,n} = x_{2,n-29} \oplus x_{2,n-27}, s_2 = 2,$$
$$x_{3,n} = x_{3,n-28} \oplus x_{3,n-15}, s_3 = 7,$$
$$x_{4,n} = x_{4,n-25} \oplus x_{4,n-22}, s_4 = 13.$$

Таким образом, период генератора LFSR113 равен $p = (2^{31} - 1)(2^{29} - 1)(2^{28} - 1)(2^{25} - 1) \approx 10^{34}$. Для генератора доказано точное свойство равнораспределения вероятностей в размерности порядка 30 [4]. Статистические тесты выявляют дополнительные корреляции в выходных последовательностях LFSR113, но найденные корреляции связаны исключительно с тем, что выходные биты данных генераторов имеют линейную структуру по построению, что не является серьезным недостатком данных генераторов [8]. Для генератора LFSR113 был разработан эффективный метод пропуска кусков последовательностей и инициализации параллельных потоков случайных чисел [9,10]. Для генератора ранее были разработаны эффективные реализации с использованием SSE- и AVX2-векторизаций [11,13,14,15], а также реализации для графических процессоров [9,16]. Генератор включен в библиотеки генерации случайных чисел GNU Scientific Library [12], RNGAVXLIB [13,14,15], PRAND [9], clRNG [16], и др.

Реализация алгоритма на языке Си представлена в табл. 1. Перед первым вызовом генератора необходимо задать (u_1, u_2, u_3, u_4) – начальное состояние, состоящее из первых четырех выходных значений. u_1, u_2, u_3, u_4 могут принимать любые значения, большие, чем $1, 7, 15, 127$ соответственно.

4. AVX512-реализация параллельной генерации четырех выходных последовательностей псевдослучайных чисел для алгоритма LFSR113

Для реализации алгоритмов использовался набор инструкций AVX512F. Описание использованных команд можно найти в [3].

В таблицах 1 и 2 представлены реализации алгоритма для генератора LFSR113. В таблице 1 приведен алгоритм на языке ANSI C, в таблице 2 – алгоритм, использующий AVX512-векторизацию. Первая часть AVX512-алгоритма, приведенного в таблице 2, выполняет те же самые действия, что и алгоритм на ANSI C, приведенный в таблице 1, но каждая операция применяется сразу к вектору из шестнадцати чисел. Состояние генератора

задается четырьмя числами, а в регистры ZMM поместятся одновременно 4 таких состояний. Таким образом, AVX512-векторизация позволяет параллельно вычислять четыре выходные последовательности. Вторая выходная последовательность инициализируется при помощи пропуска 2^{108} чисел в первой выходной последовательности, третий поток – 2^{109}, четвертый поток – 2^{110} .

Чтобы вычислить новое состояние генератора, старое записывается в регистр *ZMM1* при помощи команды **vmovaps**. Эта команда работает с данными, выровненными по 64-битным границам, и выполняется гораздо быстрее, чем аналогичная команда **vmovups,** для работы которой не требуется выравнивание. В регистр *ZMM5* помещаются первые 16 чисел массива *lfsr113_const512*. Трех-операндный синтаксис позволяет записывать результат в регистр, отличный от входных операндов. Поэтому следующая команда **vpandd** вычисляет

$z1 \,\&\, 4294967294U, z2 \,\&\, 4294967288U, z3 \,\&\, 4294967280U, z4 \,\&\, 4294967168U$

одновременно для всех четырех потоков и помещает результат в регистр *ZMM2*. Далее команд **vpsllvd** производит сдвиг влево каждого слова регистра *ZMM2* на заданное количество бит. Теперь регистр *ZMM2* содержит

$(z1 \,\&\, 4294967294U) \ll 18,\ (z2 \,\&\, 4294967288U) \ll 2,$

$(z3 \,\&\, 4294967280U) \ll 7,\ (z4 \,\&\, 4294967168U) \ll 13$ для всех четырех потоков. Следующие команды завершают вычисление новых состояний.

Инструкция **vmovaps** записывает их в структуру *state*. Далее необходимо вычислить $z_1 \oplus z_2 \oplus z_3 \oplus z_4,\ z_5 \oplus z_6 \oplus z_7 \oplus z_8,\ z_9 \oplus z_{10} \oplus z_{11} \oplus z_{12},$ $z_{13} \oplus z_{14} \oplus z_{15} \oplus z_{16}$. В регистре *ZMM1* содержатся только что вычисленные состояния $z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}, z_{12}, z_{13}, z_{14}, z_{15}, z_{16}$. Далее команда **vpshufd $78** перемешивает числа в *ZMM1*, и результат сохраняет в *ZMM2:* $z_3, z_4, z_1, z_2, z_7, z_8, z_5, z_6, z_{11}, z_{12}, z_9, z_{10}, z_{15}, z_{16}, z_{13}, z_{14}$. После применения XOR к *ZMM1* и *ZMM2*, регистр *ZMM3* содержит $z_1 \oplus z_3, z_2 \oplus z_4, z_3 \oplus z_1, z_4 \oplus z_2, z_5 \oplus z_7, z_6 \oplus z_8, z_7 \oplus z_5, z_8 \oplus z_6, z_9 \oplus z_{11}, z_{10} \oplus z_{12}, z_{11} \oplus z_9, z_{12} \oplus z_{10}, z_{13} \oplus z_{15}, z_{14} \oplus z_{16}, z_{15} \oplus z_{13}, z_{16} \oplus z_{14}$. После выполнения команды **vpshufd $255** регистр *ZMM2* содержит $z_4 \oplus z_2, z_4 \oplus z_2, z_4 \oplus z_2, z_4 \oplus z_2, z_8 \oplus z_6, z_8 \oplus z_6, z_8 \oplus z_6, z_8 \oplus z_6, z_{12} \oplus z_{10}, z_{12} \oplus z_{10}, z_{12} \oplus z_{10}, z_{12} \oplus z_{10}, z_{16} \oplus z_{14}, z_{16} \oplus z_{14}, z_{16} \oplus z_{14}, z_{16} \oplus z_{14}$. После очередного применения команды XOR, регистр *ZMM3* содержит $z_1 \oplus z_3 \oplus z_4 \oplus z_2, 0, z_3 \oplus z_1 \oplus z_4 \oplus z_2, 0, z_5 \oplus z_7 \oplus z_8 \oplus z_6, 0, z_7 \oplus z_5 \oplus z_8 \oplus z_6, 0, z_9 \oplus z_{11} \oplus z_{12} \oplus z_{10}, 0, z_{11} \oplus z_9 \oplus z_{12} \oplus z_{10}, 0, z_{13} \oplus z_{15} \oplus z_{16} \oplus z_{14}, 0, z_{15} \oplus z_{13} \oplus z_{16} \oplus z_{14}, 0$. Далее при помощи команды **vpermd** выходные числа каждой последовательность помещаются в младшие 128 бит регистра *ZMM3*: $z_1 \oplus z_3 \oplus z_4 \oplus z_2, z_5 \oplus z_7 \oplus z_8 \oplus z_6, z_9 \oplus z_{11} \oplus z_{12} \oplus z_{10}, z_{15} \oplus z_{13} \oplus z_{16} \oplus z_{14}$. Последняя инструкция **vmovaps** записывает их в массив *ans*.

Табл. 1. Реализации алгоритма LFSR113 генерации псевдослучайных чисел на ANSI C
Table. 1. Implementation of algorithm LFSR113 for generation of pseudo-random numbers with ANSI C

```
unsigned u1, u2, u3, u4;
unsigned int lfsr113_ansi_generate_(){
   unsigned b;
   b = (( u1 <<  6) ^ u1) >> 13;
   u1 = (( u1 & 4294967294U) << 18) ^ b;
   b = (( u2 <<  2) ^  u2) >> 27;
   u2 = (( u2 & 4294967288U) <<  2) ^ b;
   b = (( u3 << 13) ^  u3) >> 21;
   u3 = (( u3 & 4294967280U) <<  7) ^ b;
   b = (( u4 <<  3) ^  u4) >> 12;
   u4 = (( u4 & 4294967168U) << 13) ^ b;
   return ( u1 ^  u2 ^  u3 ^  u4);
}
```

Табл. 2. AVX512-реализация параллельной генерации четырех последовательностей псевдослучайных чисел для алгоритма LFSR113
Table 2. AVX512-реализация параллельной генерации четырех последовательностей псевдослучайных чисел для алгоритма LFSR113

```
typedef struct{
  unsigned u[16] __attribute__ ((aligned(64)));
} lfsr113_state;
unsigned lfsr113_consts512[64] __attribute__ ((aligned(64)))
={4294967294U,4294967288U,4294967280U,4294967168U,
4294967294U,4294967288U,4294967280U,4294967168U,
4294967294U,4294967288U,4294967280U,4294967168U,
4294967294U,4294967288U,4294967280U,4294967168U,     6,2,13,3,
6,2,13,3,   6,2,13,3,   6,2,13,3,   13,27,21,12,   13,27,21,12,
13,27,21,12,  13,27,21,12,  18,2,7,13,  18,2,7,13,  18,2,7,13,
18,2,7,13};
unsigned    ind[16]    __attribute__    ((aligned(64)))    =
{0,4,8,12,0,4,8,12,0,4,8,12,0,4,8,12};
void    lfsr113_avx512_generate_four_(lfsr113_state*    state,
unsigned int * ans){
    asm volatile(
    "vmovaps (%0),%%zmm1\n"\
    "vmovaps (%1),%%zmm5\n"\
     "vpandd %%zmm1, %%zmm5, %%zmm2\n"\
     "vpsllvd 192(%1),%%zmm2,%%zmm2\n"\
    "vpsllvd 64(%1),%%zmm1,%%zmm4\n"\
```

```
      "vpxord %%zmm4,%%zmm1,%%zmm4\n"\
      "vpsrlvd 128(%1),%%zmm4,%%zmm4\n"\
      "vpxord %%zmm4,%%zmm2,%%zmm1\n"\
      "vmovaps %%zmm1,(%0)\n"\
      "vpshufd $78,%%zmm1,%%zmm2\n"\
      "vpxord %%zmm2,%%zmm1,%%zmm3\n"\
      "vpshufd $255,%%zmm3,%%zmm2\n"\
      "vpxord %%zmm2,%%zmm3,%%zmm3\n"\
      "vmovaps (%3),%%zmm4\n"\
      "vpermd %%zmm3, %%zmm4,  %%zmm3\n"\
      "vmovaps %%xmm3, (%2)\n"\
      "": :
"r"(state->u),"r"(lfsr113_consts512),"r"(ans), "r"(ind));
   }
```

5. Скорость генерации

Были измерены скорости генерации псевдослучайных чисел для различных реализаций генератора LFSR113. В таблице 3 представлены скорости генерации для ANSI C версии, и для версий, которые используют наборы команд SSE4.1, AVX2 и AVX512F. Измерения были проведены на Intel Xeon Platinum 8162 (2 GHz). Был использован компилятор GCC версии 4.8.5.

Для данного генератора исходная ANSI C версия весьма эффективна за счет использования только быстрых однотактовых логических и сдвиговых операций. Более того, при компиляции ANSI C версии с ключом оптимизации –O3, компилятор автоматически оптимизирует алгоритм, используя при этом AVX инструкции, поэтому скорость генерации для такой версии превосходит скорость SSE-реализации.

Табл. 3. Производительность реализаций генератора LFSR113
Table. 3. Performance of implementations for LFSR113 generator

	ANSI C Gbit/s	SSE4.1 Gbit/s	AVX2 Gbit/s	AVX512 Gbit/s	AVX512/ ANSI C	AVX512 / SSE	AVX512/ AVX2
-O0	4.17	4.64	15.72	31.92	7.65	6.88	2.03
-O3	14.24	4.67	20.11	39.37	2.76	8.43	1.96

Статья подготовлена в рамках государственного задания ФАНО России № 0033-2014-0010.

Список литературы

[1]. Landau D.P., Binder K., A Guide to Monte Carlo Simulations in Statistical Physics, 4th edition, Cambridge University Press, Cambridge, 2015.

[2]. Бараш Л. Ю., Щур Л.Н. Генерация случайных чисел и параллельных потоков случайных чисел для расчетов Монте-Карло, Моделирование и анализ информационных систем, 19(2), 2012, стр. 145–162.

[3]. Intel® 64 and IA-32 architectures software developer's manual combined volumes 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[4]. L'Ecuyer P., Tables of maximally equidistributed combined LFSR generators, Mathematics of Computation 68(225), 1999, pp. 261–269.

[5]. A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, Monte Carlo simulations: Hidden errors from ''good'' random number generators, Phys. Rev. Lett. 69, 1992, pp. 3382–3384.

[6]. P. Grassberger, On correlations in "good" random number generators, Phys. Lett. A 181, 1993, pp. 43–46.

[7]. F. Schmid and N. B. Wilding, Errors in Monte Carlo simulations using shift register random number generators, Int. J. Mod. Phys. C 6, 1995, pp. 781–787.

[8]. L'Ecuyer P., Simard R. TestU01: A C library for empirical testing of random number generators, ACM Transactions on Mathematical Software 33(4), 2007, article 22.

[9]. L.Yu.Barash, L.N.Shchur, PRAND: GPU accelerated parallel random number generation library: Using most reliable algorithms and applying parallelism of modern GPUs and CPUs, Computer Physics Communications, 185(4), 2014, pp. 1343-1353.

[10]. Л.Ю. Бараш, Л.Н. Щур, О генерации параллельных потоков псевдослучайных чисел, Программная инженерия, №1, 2013, стр. 24-32.

[11]. Бараш Л. Ю., Гуськова М. С., Щур Л. Н. Использование AVX-векторизации для увеличения производительности генерации случайных чисел, Программирование, 43(3), 2017, стр. 22–40.

[12]. M. Galassi et al, GNU Scientific Library Reference Manual - Third Edition, Network Theory Ltd., 2009, ISBN: 0954612078.

[13]. M.S. Guskova, L.Yu. Barash, L.N. Shchur, RNGAVXLIB: Program library for random number generation, AVX realization, Computer Physics Communications, 200, 2016, pp. 402–405.

[14]. Barash L. Y., Shchur L. N. RNGSSELIB: Program library for random number generation. More generators, parallel streams of random numbers and Fortran compatibility, Computer Physics Communications 184(10), 2013, pp. 2367–2369.

[15]. L.Yu. Barash, L.N. Shchur, RNGSSELIB: Program library for random number generation, SSE2 realization, Comput. Phys. Commun., 182 (7), 2011, pp. 1518-1527.

[16]. P. L'Ecuyer, D. Munger, N. Kemerchou, clRNG: A library for uniform random number generation in OpenCL, 2015, http://www-labs.iro.umontreal.ca/~simul/clrng/

Applying AVX512 vectorization to improve the performance of a random number generator

[1,2] *M.S. Guskova <maria.guskova@rambler.ru>*
[1,2,3] *L.Yu. Barash <barash@itp.ac.ru>*
[1,2,3,4] *L.N. Shchur <shchur@chg.ru>*
[1] *Science Center in Chernogolovka, 142432 Chernogolovka, Russia*
[2] *National Research University Higher School of Economics, 101000 Moscow, Russia*
[3] *Landau Institute for Theoretical Physics, 142432 Chernogolovka, Russia*
[4] *Dorodnicyn Computing Centre, FRC CSC RAS, 119333 Moscow, Russia*

Abstract. The generation of uniformly distributed random numbers is necessary for computer simulation by Monte Carlo methods and molecular dynamics [1]. Generators of pseudo-random numbers (GPRS) are used to generate random numbers. GPRS uses deterministic algorithms to calculate numbers, but the sequence obtained in this way has the properties of a random sequence. For a number of problems using Monte Carlo methods, random number generation takes up a significant amount of computational time, and increasing the generation capacity is an important task. This paper describes applying SIMD instructions (Single Instruction Multiple Data) to parallelize generation of pseudorandom numbers. We review SIMD instruction set extensions such as MMX, SSE, AVX2, AVX512. The example of AVX512 implementation is given for the LFSR113 pseudorandom number generator. Performance is compared for different algorithm implementations.

Keywords: Pseudo random numbers; SIMD; AVX512 technology

DOI: 10.15514/ISPRAS-2018-30(1)-8

For citation: Guskova M.S., Barash L.Yu., Shchur L.N. Applying AVX512 vectorization to improve the performance of a random number generator. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 1, 2018, pp. 115-126. DOI: 10.15514/ISPRAS-2018-30(1)-8

References

[1]. Landau D.P., Binder K., A Guide to Monte Carlo Simulations in Statistical Physics, 4th edition, Cambridge University Press, Cambridge, 2015.
[2]. Barash L.Y., Shchur L.N., Generation of Random Numbers and Parallel Random Number Streams for Monte Carlo Simulations. Modeling and Analysis of Information Systems, 19(2), 2012, pp. 145-162 (in Russian).
[3]. Intel® 64 and IA-32 architectures software developer's manual combined volumes 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
[4]. L'Ecuyer P., Tables of maximally equidistributed combined LFSR generators, Mathematics of Computation 68(225), 1999, pp. 261–269.

[5]. A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, Monte Carlo simulations: Hidden errors from ‘‘good’’ random number generators, Phys. Rev. Lett. 69, 1992, pp. 3382–3384.

[6]. P. Grassberger, On correlations in “good” random number generators, Phys. Lett. A 181, 1993, pp. 43–46.

[7]. F. Schmid and N. B. Wilding, Errors in Monte Carlo simulations using shift register random number generators, Int. J. Mod. Phys. C 6, 1995, pp. 781–787.

[8]. L'Ecuyer P., Simard R. TestU01: A C library for empirical testing of random number generators, ACM Transactions on Mathematical Software 33(4), 2007, article 22.

[9]. L.Yu.Barash, L.N.Shchur, PRAND: GPU accelerated parallel random number generation library: Using most reliable algorithms and applying parallelism of modern GPUs and CPUs, Computer Physics Communications, 185(4), 2014, pp. 1343-1353.

[10]. L.Yu. Barash, L.N. Shchur, On the generation of parallel streams of pseudorandom numbers, Software Engineering Vol.1, 2013, pp. 24–32 (in Russian).

[11]. L.Yu. Barash, M.S. Guskova, L.N. Shchur, Employing AVX vectorization to improve the performance of random number generators, Programming and Computer Software, 43(3), 2017, pp. 145-160.

[12]. M. Galassi et al, GNU Scientific Library Reference Manual - Third Edition, Network Theory Ltd., 2009, ISBN: 0954612078.

[13]. M.S. Guskova, L.Yu. Barash, L.N. Shchur, RNGAVXLIB: Program library for random number generation, AVX realization, Computer Physics Communications, 200, 2016, pp. 402–405.

[14]. Barash L. Y., Shchur L. N. RNGSSELIB: Program library for random number generation. More generators, parallel streams of random numbers and Fortran compatibility, Computer Physics Communications 184(10), 2013, pp. 2367–2369.

[15]. L.Yu. Barash, L.N. Shchur, RNGSSELIB: Program library for random number generation, SSE2 realization, Comput. Phys. Commun., 182 (7), 2011, pp. 1518-1527.

[16]. P. L’Ecuyer, D. Munger, N. Kemerchou, clRNG: A library for uniform random number generation in OpenCL, 2015, http://www-labs.iro.umontreal.ca/~simul/clrng/