AVX2 и эффективная генерация псевдослучайных чисел

Гуськова М.С.

НИУ ВШЭ,101000 Москва, Россия
НЦЧ РАН, 142432 Черноголовка, Россия

23 апреля 2015

Содержание

Работа выполнена совместно с Л.Ю. Барашом, Л.Н. Щуром

Случайные числа

Некоторые применения случайных чисел:

- Моделирование
- Выборочный метод
- Численный анализ
- Компьютерное программирование
- Принятие решений

Определение генераторов псевдослучайных чисел

Генератор –

это структура $g = (S, s_0, T, U, G)$

- $S-$ конечное множество состояний
- $s_0 \in S-$ начальное состояние.
- преобразование $T : S \to S-$ функция перехода.
- $U-$ конечное множество выходных символов.
- $G : S \to U-$ выходная функция генератора

$s_n = T(s_{n-1})-$ состояние генератора вычисляется на каждом шаге.
$u_n = G(s_n)-$ случайное число

Типы генераторов псевдослучайных чисел

Все генераторы принадлежат либо двум следующим классам, либо являются их модификациями и комбинациями.

Линейно-конгруэнтные генераторы (Linear Congruential Generators) самые известные и распространенные. Последовательность случайных чисел вычисляется по следующей формуле $xn + 1 = (ax_n + c) \mod m$. Простейшим примером этого класса является генератор rand из стандартной библиотеки $x_{n+1} = (1103515245x_n + 12356) \mod 2^{31}$.

Генераторы, основанные на сдвиговом регистре (Generalized Feedback Shift Register), обладают огромным периодом, если верно выбрать примитивный полином. Этот полином $P(z) = z^k - a_1 z^{k-1} - \ldots - a_k$, где коэффициенты $a_i \in \mathbb{Z}_2$, лежит в основе таких генераторов и является характеристическим для последовательности $x_n = a_1 x_{n-1} + \ldots + a_k x_{n-k}$. Тогда случайное число можно получить следующим образом $u_n = \sum_{i=1}^{L} x_{ns+i-1} 2^{-i}$

Расширения Intel

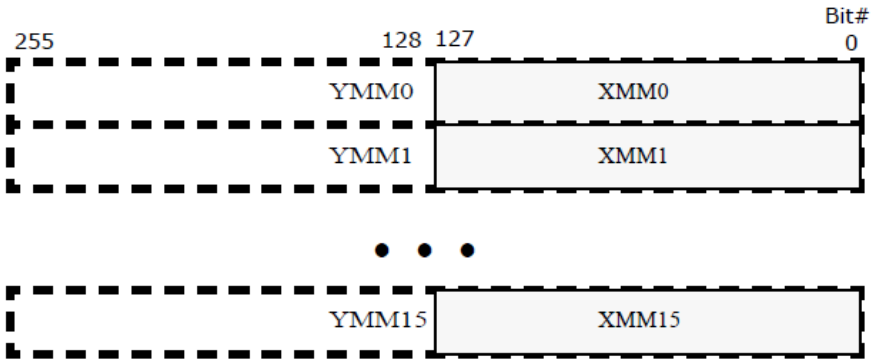
- Streaming SIMD Extensions (SSE) SIMD расшифровывается как Single Instruction - Multiple Data. Это потоковые расширения, способные обрабатывать большое количество данных одной командой.
- Intel Advanced Vector Extensions (Intel AVX) – это новый набор 256-битных команд для Intel SSE, предназначенный для приложений, интенсивно использующих операции с плавающей запятой.

Intel SSE

SSE работает с такими типами данных, как упакованные числа с плавающей запятой одинарной точности. В одном регистре одновременно могут находиться сразу 4 таких числа. Среди команд SSE различают команды:

- пересылки данных (MOVAPS,...)
- арифметические команды (ADDPS, SUBPS, ...)
- команды сравнения (CMPPS,...)
- команды преобразования типов (CVTPS2PS,...)
- логические операции (ANDPS,...)
- команды упаковки (SHUFPS,...)

Intel AVX2



AVX-расширение включает в себя 256-битные регистры (YMM0-YMM7 для 32-битных систем, YMM0-YMM15 для 64-разрядных систем). Младшие 128 бит YMM регистров называются соответственно 128-битными XMM регистрами.

Особенности AVX2.0

Intel AVX2.0 содержит следующие улучшения:

- Теперь возможна параллельная обработка восьми чисел типа с плавающей точкой или беззнаковых целых, четырех чисел с плавающей точкой двойной точности.
- Синтаксис инструкций является трех-операндным для увеличения гибкости и эффективности новых инструкций расширения. Теперь возможны не только операции вида $A = A + B$, но и $A = B + C$, т.к. исходные операнды остаются нетронутыми, то в некоторых случаях можно избавиться от теперь уже лишних операций копирования.

Особенности AVX2.0

Intel AVX2.0 содержит следующие улучшения:

- Теперь возможна параллельная обработка восьми чисел типа с плавающей точкой или беззнаковых целых, четырех чисел с плавающей точкой двойной точности.
- Синтаксис инструкций является трех-операндным для увеличения гибкости и эффективности новых инструкций расширения. Теперь возможны не только операции вида $A = A + B$, но и $A = B + C$, т.к. исходные операнды остаются нетронутыми, то в некоторых случаях можно избавиться от теперь уже лишних операций копирования.

Особенности AVX2.0

Intel AVX2.0 содержит следующие улучшения:

- Теперь возможна параллельная обработка восьми чисел типа с плавающей точкой или беззнаковых целых, четырех чисел с плавающей точкой двойной точности.
- Синтаксис инструкций является трех-операндным для увеличения гибкости и эффективности новых инструкций расширения. Теперь возможны не только операции вида $A = A + B$, но и $A = B + C$, т.к. исходные операнды остаются нетронутыми, то в некоторых случаях можно избавиться от теперь уже лишних операций копирования.

Продолжение

- Некоторые инструкции принимают четыре регистра в качестве операндов, делая код меньше и быстрее.
- SIMD инструкции имеют 128-битные эквиваленты в расширении AVX, поддерживающие трех-операндный синтаксис.
- Новый префикс VEX призван упростить дальнейшую переработку кода. Все инструкции SSE, SSE2, SSE3, SSE4 представлены и в AVX.
- Добавлены принципиально новые инструкции (VPERMD – Permute doublewords, VPSLLVD – Shift doublewords, и прочие)

Продолжение

- Некоторые инструкции принимают четыре регистра в качестве операндов, делая код меньше и быстрее.
- SIMD инструкции имеют 128-битные эквиваленты в расширении AVX, поддерживающие трех-операндный синтаксис.
- Новый префикс VEX призван упростить дальнейшую переработку кода. Все инструкции SSE, SSE2, SSE3, SSE4 представлены и в AVX.
- Добавлены принципиально новые инструкции (VPERMD – Permute doublewords, VPSLLVD – Shift doublewords, и прочие)

Продолжение

- Некоторые инструкции принимают четыре регистра в качестве операндов, делая код меньше и быстрее.
- SIMD инструкции имеют 128-битные эквиваленты в расширении AVX, поддерживающие трех-операндный синтаксис.
- Новый префикс VEX призван упростить дальнейшую переработку кода. Все инструкции SSE, SSE2, SSE3, SSE4 представлены и в AVX.
- Добавлены принципиально новые инструкции (VPERMD – Permute doublewords, VPSLLVD – Shift doublewords, и прочие)

Продолжение

- Некоторые инструкции принимают четыре регистра в качестве операндов, делая код меньше и быстрее.
- SIMD инструкции имеют 128-битные эквиваленты в расширении AVX, поддерживающие трех-операндный синтаксис.
- Новый префикс VEX призван упростить дальнейшую переработку кода. Все инструкции SSE, SSE2, SSE3, SSE4 представлены и в AVX.
- Добавлены принципиально новые инструкции (VPERMD – Permute doublewords, VPSLLVD – Shift doublewords, и прочие)

Описание генератора GM31

Функция перехода генератора определяется таким преобразованием.

$$\begin{pmatrix} x_i^{(n)} \\ y_i^{(n)} \end{pmatrix} = M \begin{pmatrix} x_i^{(n-1)} \\ y_i^{(n-1)} \end{pmatrix} (\mod g)$$

Выходная функция генератора:

$$u^{(n)} = \sum_{i=0}^{s-1} \lfloor 2x_i^{(n)}/g \rfloor \cdot 2^i$$

$m-$ мерсеновская экспонента, значит, $g = p = 2^m - 1$ являются простыми.

Состояние генератора состоит из s точек, лежащих на решетке $g \times g$ на торе. Решетка на торе
$L = \{0, 1, \ldots, g-1\} \times \{0, 1, \ldots, g-1\}$

Начальное состояние задается точками $\begin{pmatrix} x_i^{(0)} \\ y_i^{(0)} \end{pmatrix}$, где $x_i^{(0)}, y_i^{(0)} \in \{0, 1, \ldots, g-1\}, i = 0, 1, \ldots, s-1$. Эти точки лежат на целочисленной решетке, их координаты целые и положительные.

Начальными точками на единичном двумерном торе $(0, 1] \times (0, 1]$ являются $\begin{pmatrix} x_i^{(0)}/g \\ y_i^{(0)}/g \end{pmatrix}, i = 0, 1, \ldots, s-1$

В данной реализации вычисляются параллельно 32 рекуррентных соотношения.

$$x^{(n)} = kx^{n-1} - qx * n - 2 \mod g$$

$$y^{(n)} = ky^{n-1} - qy * n - 2 \mod g$$

$k = Tr(M), q = det(M)$

SSE

```
"movaps(%1), %%xmm0\n" \
"movaps%%xmm0, %%xmm7\n" \
"movaps(%2), %%xmm6\n" \
"pmuludq16(%3), %%xmm0\n" \
"pmuludq32(%3), %%xmm6\n" \
"paddq(%3), %%xmm0\n" \
"psubq%%xmm6, %%xmm0\n" \
"movaps%%xmm0, %%xmm6\n" \
"psrlq$31, %%xmm6\n" \
"andps%%xmm5, %%xmm0\n" \
"paddq%%xmm6, %%xmm0\n" \
"movaps%%xmm0, (%1)\n" \
"movaps%%xmm7, (%2)\n" \
```

AVX2

```
"vmovaps(%1), %%ymm0\n" \
"vmovaps(%2), %%ymm6\n" \
"vmovaps%%ymm0, (%2)\n" \
"vpmuludq32(%3), %%ymm0, %%ymm0\n" \
"vpmuludq64(%3), %%ymm6, %%ymm6\n" \
"vpaddq(%3), %%ymm0, %%ymm0\n" \
"vpsubq%%ymm6, %%ymm0, %%ymm0\n" \
"vmovaps%%ymm0, %%ymm6\n" \
"vpsrlq$31, %%ymm6, %%ymm6\n" \
"vandps%%ymm5, %%ymm0, %%ymm0\n" \
"vpaddq%%ymm6, %%ymm0, %%ymm0\n" \
"vmovaps%%ymm0, (%1)\n" \
```

SSE

```
"shufps $136, %%xmm2, %%xmm1\n" \
"shufps $136, %%xmm4, %%xmm3\n" \
"psrld $30, %%xmm1\n" \
"psrld $30, %%xmm3\n" \
"packssdw %%xmm3, %%xmm1\n" \
"\n" \
"packsswb %%xmm1, %%xmm0\n" \
"psllw $7, %%xmm0\n" \
"pmovmskb %%xmm0, %0\n" \
"shll $16, %0\n" \
```

AVX2

```
"vshufps $136, %%ymm1, %%ymm0, %%ymm0\n" \
"vpermpd $216, %%ymm0, %%ymm0\n" \
"vshufps $136, %%ymm3, %%ymm2, %%ymm2\n" \
"vpermpd $216, %%ymm2, %%ymm2\n" \
"vpsrld $30, %%ymm0, %%ymm0\n" \
"vpsrld $30, %%ymm2, %%ymm2\n" \
"vpackssdw %%ymm2, %%ymm0, %%ymm0\n" \
"vpermpd $216, %%ymm0, %%ymm0\n" \
"vpacksswb %%ymm0, %%ymm7, %%ymm0\n" \
"vpermpd $216, %%ymm0, %%ymm0\n" \
"vpsllw $7, %%ymm0, %%ymm0\n" \
"vpmovmskb %%ymm0, %0\n" \
```

Тестирование проводилось на машине со следующими характеристиками

Таблица: Информация о процессоре

Name	Intel Core i5-5200U
Codename	Broadwell-U
Specification	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
Instructions sets	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3

$N = 10^8$ количество генерируемых случайных чисел. Используется компилятор gcc (GNU Compiler Collection). Цифра после -o определяет уровень оптимизации. Время указано в секундах.

h	AVX2	SSE	%
gcc -o0	2,1332	3,5406	40
gcc -o1	2,0896	3,453	39
gcc -o2	2,086	3,3322	37
gcc -o3	2,1012	3,327	37

Заключение

Использование расширения AVX2 позволило получить идентичную последовательность случайных чисел. Причем время уменьшилось на 40%, а строк кода стало в два раза меньше (264 - 129).

Спасибо за внимание!