УДК 004 DOI 10.25730/VSU.0536.19.017

Реализация масштабированного векторного сложения многократной точности на GPU*

К. С. Исупов[1], А. С. Куваев[2]

[1]кандидат технических наук, доцент кафедры электронных вычислительных машин, Вятский государственный университет. Россия, г. Киров. E-mail: ks_isupov@vyatsu.ru

[2]аспирант кафедры электронных вычислительных машин, Вятский государственный университет. Россия, г. Киров. E-mail: kyvaevy@gmail.com

Аннотация. Многие современные задачи, решаемые на суперкомпьютерах, требуют выполнения операций линейной алгебры с точностью, превышающей стандартные форматы IEEE 754. В статье рассматривается реализация высокоточного масштабированного векторного сложения (WAXPBY) на CUDA-совместимых графических процессорах видеокарты. Для представления многоразрядных чисел с плавающей точкой используется формат на основе системы остаточных классов, исключающий образование цепочек переносов и позволяющий вычислять все цифры мантисс одновременно. Параллельный алгоритм WAXPBY разбивается на ряд шагов, каждый из которых выполняется отдельным ядром CUDA, что позволяет добиться оптимального использования имеющихся вычислительных ресурсов. Эксперименты показали, что производительность разработанной подпрограммы выше по сравнению с аналогами для центральных процессоров. Результаты могут быть полезны в приложениях линейной алгебры, использующих графические процессоры для ускорения вычислений и критичных к ошибкам округления.

Ключевые слова: высокоточные вычисления, BLAS, система остаточных классов, GPGPU.

Введение. Для заданных двух векторов чисел с плавающей точкой **x** и **y** длины *N*, скаляров α и β, масштабированное векторное сложение (WAXPBY) состоит в вычислении вектора $\mathbf{w} \leftarrow \alpha\mathbf{x} + \beta\mathbf{y}$. Данная операция расширяет функциональность оригинальной операции AXPY и включена в обновленный набор базовых подпрограмм линейной алгебры (BLAS) [3]. В настоящее время WAXPBY реализована в библиотеке Math::BLAS языка Perl, в пакетах ATLAS, Arm Allinea Studio и XBLAS [8], а также в ряде других библиотек линейной алгебры. Большинство из этих библиотек поддерживают вычисления в формате двойной точности, IEEE 754 double precision, что соответствует длине мантиссы числа с плавающей точкой 16 десятичных цифр (исключением является XBLAS, в котором внутренние вычисления выполняются в формате double-double). Однако с ростом масштабов производимых расчетов появляется большое количество задач, для которых требуется более высокая точность [6; 2; 4].

Цель и задачи исследования. Целью работы является разработка и исследование эффективной параллельной реализации векторной операции WAXPBY многократной точности для графических процессоров (GPU), совместимых с архитектурой NVIDIA CUDA. Задачи исследования: разработка и реализация параллельного алгоритма для GPU, оценка производительности в сравнении с аналогами для центральных процессоров (CPU).

Ведущий подход. Для представления чисел с плавающей точкой произвольной длины используется система остаточных классов (СОК) [1]. Число с плавающей точкой *x* представляется следующим образом: $x = \{s, X, e, I(X/M)\}$, где *s* – знак, *X* – мантисса, *e* – порядок (экспонента) и $I(X/M)$ – интервально-позиционная характеристика (ИПХ) мантиссы. Мантисса *X* представляется в СОК остатками $(x_1, x_2, \ldots, x_n)$ относительно набора модулей $\{m_1, m_2, \ldots, m_n\}$ и интерпретируется как целое число в диапазоне [0, *M* – 1], где $M = \prod_{i=1}^{n} m_i$. Остатки $x_i = X \bmod m_i$ – целые машинные числа.

В представленном формате остатки мантиссы взаимно независимы, что исключает трудоемкую обработку цепочек переносов и обеспечивает параллелизм на уровне арифметики многократной точности: для операций сложения, вычитания и умножения, вычисления с остатками могут выполняться параллельно. ИПХ позволяет оценить величину мантиссы для эффективного сравнения, выравнивания порядков, округления и других затратных в СОК операций.

С учетом округления точность арифметических операций в битах определяется следующим образом: $p = \left\lfloor \log_2 \left\lfloor (M-1)^{1/2} \right\rfloor \right\rfloor$. Размер набора модулей может быть произвольным, что позво-

* Исследование выполнено за счет гранта Российского научного фонда (проект № 18-71-00063).

ляет добиться любой необходимой точности (разрядности) вычислений. Например, если требуются 1024-битные вычисления, то набор модулей должен быть таким, что $M > 2^{2048}$.

GPU-реализация WAXPBY. Предполагается, что исходные данные (векторы **x** и **y** длины *N*, заполненные многоразрядными числами, числа α и β) загружены в глобальную память GPU. Алгоритм выполнения операции WAXPBY состоит из следующих шагов.

1. В глобальной памяти GPU выделяется буфер dbuf длины *N*.

2. (Kernel 1) Выполняется поэлементное векторное умножение многоразрядных мантисс элементов вектора **x** и скаляра α с сохранением результатов в dbuf. Для умножения используется *bcount1* блоков по *n* потоков, где *n* – количество модулей СОК. Потоки каждого блока параллельно умножают все цифры (остатки) многоразрядных мантисс и каждый *i*-й поток назначен для вычисления цифры по модулю m_i. Таким образом, в общей сложности каждый блок потоков умножает мантиссы для *N* / *bcount1* элементов вектора **x**. Все блоки могут работать параллельно.

3. (Kernel 2) В *bcount2* блоках из *bsize2* потоков вычисляются экспоненты, знаки и ИПХ для вектора-произведения α**x** с сохранением результатов в dbuf.

4. (Kernel 3) В *bcount3* блоках выполняется проверка необходимости округления и (при необходимости) округление вектора, записанного в dbuf. Каждый многоразрядный элемент округляется *n* параллельными потоками в соответствии с алгоритмом модулярного масштабирования из [5].

5. Шаги 2–4 повторяются для **y** и β с сохранением результатов в **w**.

6. (Kernel 4) Выполняется параллельное поэлементное сложение **w** и dbuf с перезаписью результата в **w**. На этом шаге используется *bcount4* блоков по *n* потоков. Многоразрядное сложение распараллеливается по модулям СОК.

В приведенном алгоритме Kernel 1–4 – это global-функции (CUDA ядра), запускаемые с хоста и выполняемые на GPU. Для каждой такой функции задается своя конфигурация запуска, что позволяет оптимально использовать имеющиеся ресурсы GPU при заданной точности вычислений. Результатный вектор **w** хранится в глобальной памяти GPU и может быть использован для дальнейших вычислений (указатель на **w** передается в качестве аргумента вызываемой подпрограммы). При необходимости он может быть загружен в оперативную память хоста посредством вызова стандартной функции cudaMemcpy из CUDA Runtime API. Схема описанного алгоритма представлена на рис. 1.

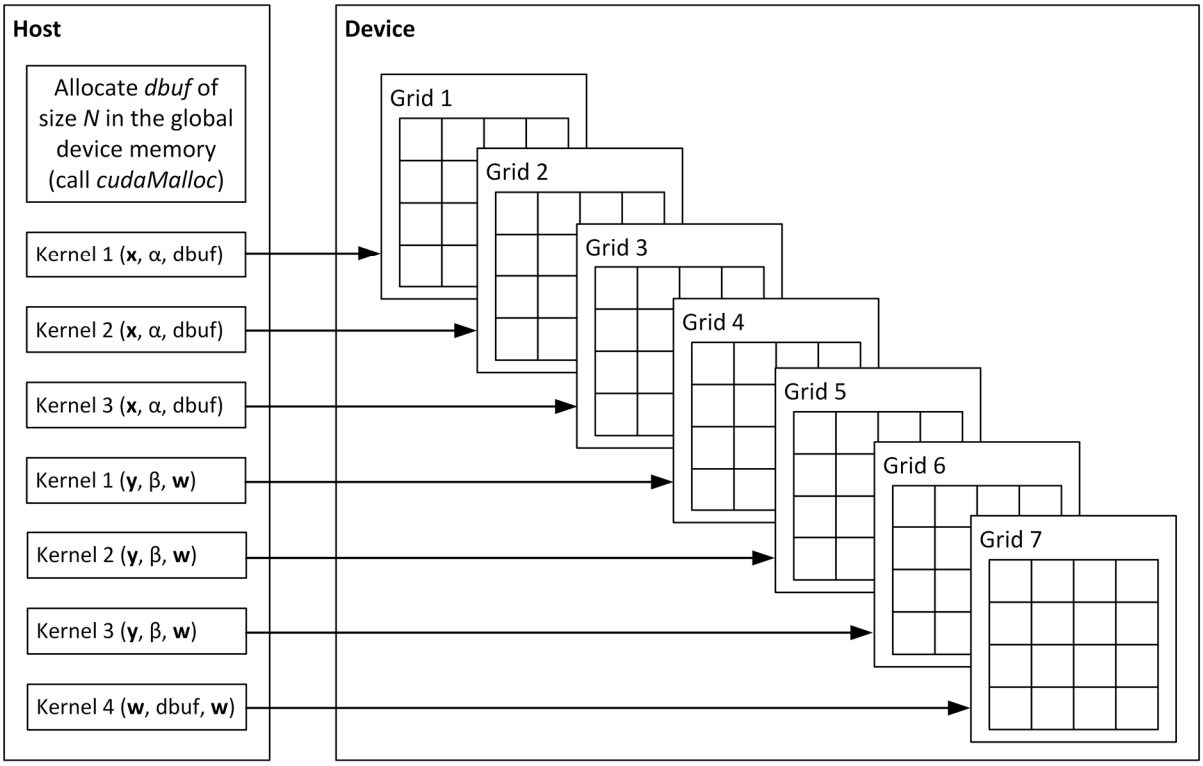

Рис. 1. Выполнение операции WAXPBY многократной точности на GPU

При наличии достаточных ресурсов GPU вычисление произведений α**x** и β**y** может перекрываться. Для этого функции на шагах 2–4 и на шаге 5 должны быть запущены в разных потоках выполнения (CUDA Stream) с синхронизацией непосредственно перед запуском функции Kernel 4 на

шаге 6. Также возможно одновременное выполнение ядер Kernel 1 и Kernel 2, так как зависимости по данным между ними отсутствуют.

На рис. 2 представлены прототипы реализованных подпрограмм. Кроме WAXPBY реализована BLAS-подпрограмма масштабированного векторного сложения с накоплением, AXPBY: $\mathbf{y} \leftarrow \alpha\mathbf{x} + \beta\mathbf{y}$. Алгоритм AXPBY аналогичен рассмотренному алгоритму для WAXPBY с обновлением **y** вместо записи в **w** на шагах 5 и 6. В подпрограмме mf_maxpby выполняется простой вызов mf_mwaxpby с передачей указателя на **y** вместо **w** и incy вместо incw.

Для индексации элементов векторов и цифр многоразрядной мантиссы отдельного элемента с учетом заданной точности вычислений (числа модулей СОК), расстояний между соседними элементами (incx, incy и incw) и шаблонных параметров, определяющих конфигурацию выполнения CUDA-ядер, разработаны специальные служебные функции.

```
/* Масштабированное векторное сложение */
template<int bcount1, int bcount2, int bsize2, int bcount3, int bcount4>
void mf_mwaxpby( int N, mf_t alpha, mf_ptr x, int incx, mf_t betta, mf_ptr y, int incy,
mf_ptr w, int incw );

/* Масштабированное векторное сложение с накоплением */
template<int bcount1, int bcount2, int bsize2, int bcount3, int bcount4>
void mf_maxpby( int N, mf_t alpha, mf_ptr x, int incx, mf_t betta, mf_ptr y, int incy );
```

Рис. 2. Прототипы разработанных высокоточных процедур

Представленные подпрограммы входят в состав разрабатываемой авторами библиотеки MPRES для параллельных вычислений многократной точности на системах с гибридными CPU-GPU узлами [7].

Результаты исследований, их обсуждение. Производительность разработанной подпрограммы сравнивалась с CPU-аналогами, реализованными с использованием последних версий известных арифметических библиотек MPFR и ARPREC, а также с подпрограммой BLAS_dwaxpby_x из пакета XBLAS. В экспериментах замерялась производительность в MFlops при выполнении операций с векторами размера N = 1 000 000. Точность вычислений, p, варьировалась от 120 до 1201 бит. Для достижения указанной точности использовалось от 16 до 160 15-битных модулей СОК. Под Flop в данном контексте понимается операция с плавающей точкой, выполняемая с точностью p бит. Эксперименты проводились на CPU Intel Core i5 4590 и GPU NVIDIA GeForce GTX 1050 Ti.

Для разработанной GPU-подпрограммы использовались следующие параметры запуска: *bcount1* = 2048, *bcount2* = 4, bsize2 = 1024, *bcount3* = 2048, *bcount4* = 2048. Среди прочих эти параметры обеспечивают максимальную производительность на используемом GPU.

Результаты представлены на рис. 3 (разработанная GPU-реализация обозначена MPRES). Для операции AXPBY получены аналогичные результаты.

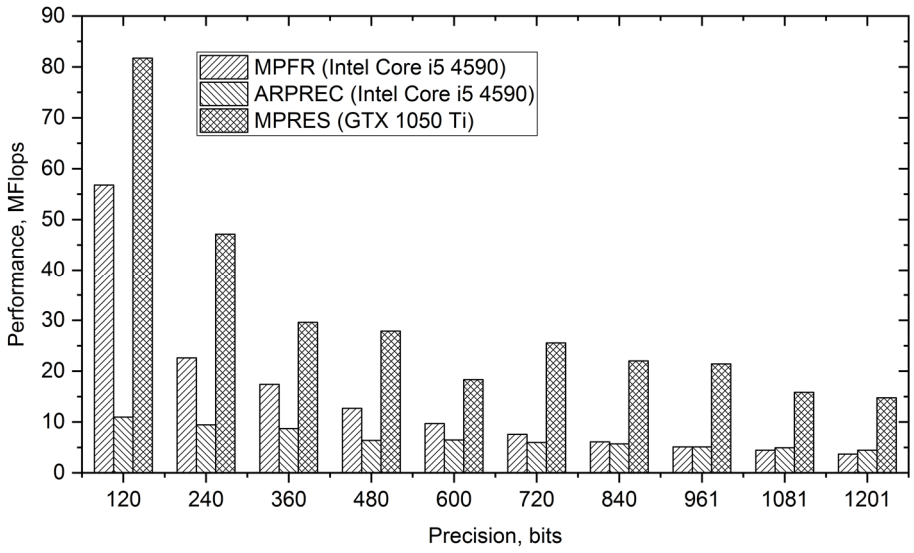

Рис. 3. Производительность высокоточных реализаций WAXPBY

При 120-битной точности вычислений производительность разработанной подпрограммы WAXPBY равна 82 MFlops, что выше приблизительно в 7,5 раза по сравнению с ARPREC и в 1,5 раза по сравнению с MPFR. В свою очередь, при точности 1201 бит производительность разработанной подпрограммы равна 14,7 MFlops, а ускорение относительно ARPREC и MPFR составляет 3,3 и 4 раза соответственно. Производительность пакета XBLAS (не представлена на графике), обеспечивающего вычисления с точностью 106 бит, составила в среднем 95 MFlops, что сопоставимо с производительностью нашей реализации при точности 120 бит.

Выводы. Разработан и реализован алгоритм выполнения операции WAXPBY многократной точности для графических процессоров, основанный на представлении чисел с плавающей точкой произвольной длины с использованием СОК, обеспечивающий эффективное использование имеющихся ресурсов посредством настройки параметров вычислительной сетки для каждого из внутренних CUDA ядер. На базе WAXPBY реализована подпрограмма масштабированного векторного сложения с накоплением (AXPBY).

Экспериментальные результаты показывают, что производительность разработанной подпрограммы WAXPBY сравнима с производительностью пакета XBLAS для CPU и значительно выше по сравнению с реализациями на основе библиотек ARPREC и MPACK. Представленные результаты могут быть улучшены с использованием современных GPU серии Tesla, спроектированных специально для высокопроизводительных вычислений общего назначения.

Список литературы

1. *Акушский И. Я., Юдицкий Д. И.* Машинная арифметика в остаточных классах. М. : Сов. радио, 1968. 440 с.
2. *Alperovich A., Druinsky A., Toledo S.* Experiences with a Lanczos Eigensolver in High-Precision Arithmetic // Lecture Notes in Computer Science. 2014. Vol. 8384. Pp. 36–46.
3. An Updated Set of Basic Linear Algebra Subprograms (BLAS) / L. S. Blackford et al. // ACM Transactions on Mathematical Software. 2002. Vol. 28. № 2. Pp. 135–151.
4. *Bailey D. H., Borwein J. M.* High-Precision Arithmetic in Mathematical Physics // Mathematics. 2015. Vol. 3. Pp. 337–367.
5. *Isupov K., Knyazkov V., Kuvaev A.* Fast Power-of-Two RNS Scaling Algorithm for Large Dynamic Ranges // IVth International Conference on Engineering and Telecommunication, 29–30 Nov. 2017. URL: https://ieeexplore.ieee.org/abstract/document/8241272 (date of access 01.02.2018).
6. *Mittelmann H. D., Vallentin F.* High-accuracy semidefinite programming bounds for kissing numbers // Experimental Mathematics. 2010. Vol. 19. № 2. Pp. 175–179.
7. Multiple-Precision Residue-Based Arithmetic Library for Parallel CPU-GPU Architectures: Data Types and Features / Isupov K. et al. // Lecture Notes in Computer Science. 2017. Vol. 10421. Pp. 196–204.
8. XBLAS – Extra Precise Basic Linear Algebra Subroutines. URL: https://www.netlib.org/xblas (date of access 21.11.2019).

Implementation of scaled vector addition of multiple precision on GPU

K. S. Isupov[1], A. S. Kuvaev[2]

[1]PhD of technical sciences, associate professor of the Department of electronic computing machines, Vyatka State University. Russia, Kirov. E-mail: ks_isupov@vyatsu.ru.

[2]post-graduate student of the Department of electronic computing machines, Vyatka State University. Russia, Kirov. E-mail: kyvaevy@gmail.com

Abstract. Many modern problems solved on supercomputers require linear algebra operations to be performed with a precision exceeding the IEEE 754 formats. This article discusses the implementation of high-precision scaled vector addition (WAXPBY) on CUDA-compatible graphics processing units (GPUs). To represent multiple-precision floating-point numbers, a format based on a residue number system is used, which eliminates the need for carry propagation and allows to compute all significand digits simultaneously. The parallel WAXPBY algorithm is divided into a number of steps, each of which is performed as a separate CUDA kernel, which allows for efficient use of available computing resources. Experiments have shown that the performance of the developed routine is higher compared to solutions for central processing units. The results can be useful in linear algebra applications that use GPUs to speed up calculations and are critical to rounding errors.

Keywords: high-precision calculations, BLAS, system of residual classes, GPGPU.

References

1. *Akushskij I. Ya., Yudickij D. I. Mashinnaya arifmetika v ostatochnyh klassah* [Machine arithmetic in residual classes]. M. Sov. radio. 1968. 440 p.
2. *Alperovich A., Druinsky A., Toledo S.* Experiences with a Lanczos Eigensolver in High-Precision Arithmetic //

Lecture Notes in Computer Science. 2014. Vol. 8384. Pp. 36–46.

3. An Updated Set of Basic Linear Algebra Subprograms (BLAS) / L. S. Blackford et al. // ACM Transactions on Mathematical Software. 2002. Vol. 28. № 2. Pp. 135–151.

4. *Bailey D. H., Borwein J. M.* High-Precision Arithmetic in Mathematical Physics // Mathematics. 2015. Vol. 3. Pp. 337–367.

5. *Isupov K., Knyazkov V., Kuvaev A.* Fast Power-of-Two RNS Scaling Algorithm for Large Dynamic Ranges // IVth International Conference on Engineering and Telecommunication, 29–30 Nov. 2017. Available at: https://ieeexplore. ieee.org/abstract/document/8241272 (date accessed: 01.02.2018).

6. *Mittelmann H. D., Vallentin F.* High-accuracy semidefinite programming bounds for kissing numbers // Experimental Mathematics. 2010. Vol. 19. № 2. Pp. 175–179.

7. Multiple-Precision Residue-Based Arithmetic Library for Parallel CPU-GPU Architectures: Data Types and Features / Isupov K. et al. // Lecture Notes in Computer Science. 2017. Vol. 10421. Pp. 196–204.

8. XBLAS – Extra Precise Basic Linear Algebra Subroutines. Available at: https://www.netlib.org/xblas (date accessed: 21.11.2019).