УДК 004.4'6

МЕТОДИКА ВЫЧИСЛЕНИЙ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

И.Н. ГУБЧИК, Н.Н. ИВАНОВ

Белорусский государственный университет информатики и радиоэлектроники, Республика Беларусь

Поступила в редакцию 21 мая 2018

Аннотация. В статье рассматривается простой и эффективный метод остановки вычислений на графическом процессоре в случае необходимости быстрого освобождения ресурсов устройства. Условия работы метода, а также подходы для обратной связи, требуемой для перезапуска ядра на другом вычислительном устройстве, описаны подробно.

Ключевые слова: графический процессор, центральное процессорное устройство, ядро, неспециализированные вычисления на графических процессорах.

Abstract. An overview of a simple and efficient method to abort execution of a running kernel on graphics processing unit when it is required for fast release of its resources is presented. Required conditions of the method and kernel feedback approaches needed for a kernel restart on another processing unit are described in details.

Keywords: graphics processing unit, central processing unit, kernel, general-purpose computing on graphics processing unit.

Doklady BGUIR. 2018, Vol. 114, No. 4, pp. 94-99
Method of computing on graphics processors
I.N. Hubchyk, N.N. Ivanov

Введение

С учетом растущего интереса к системам с гетерогенной архитектурой и системам реального времени возникает потребность в миграции задач между различными вычислительными устройствами [1, 2]. Использование таких систем с разными комбинациями центрального процессорного устройства (ЦПУ) и графического(-их) процессора (ГП) создает необходимость остановки выполнения задачи на одном вычислительном устройстве и запуск той же задачи на другом устройстве. Данная проблема особенно актуальна для программ, которые во время их выполнения используют компиляцию исходного кода ядра при помощи OpenCL/CUDA. Под ядром подразумевается функция, исполняемая на ГП. В данном случае внутренняя структура ядра неизвестна и, как следствие, неизвестно его время выполнения на вычислительном устройстве. Сигнал о приостановке ядра всегда генерируется ЦПУ и передается ГП. При принудительном освобождении ресурсов ГП происходит потеря данных программы. Метод быстрой и безопасной остановки выполнения ядра на ГП и накладные расходы при использовании рассматриваемого метода являются темой данной статьи.

Метод остановки ядра на ГП

Рассмотрим простой метод остановки выполнения ядра на ГП, требующий для его реализации минимальные изменения в исходном коде ядра и не приводящий к значительным затратам ресурсов ГП. Ядро, выполняемое на ГП, может получить отправленный ЦПУ сигнал об остановке при помощи изменения значения разделяемой переменной в памяти ЦПУ и последующего копирования данной переменной в память ГП. Прямая адресация к разделяемой переменной из ядра значительно замедляет его работу в сравнении с адресацией к памяти ГП. Каждый поток, выполняемый внутри ядра, проверяет значение данной переменной (флага),

находящейся в памяти ГП, и при выставленном флаге не производит дальнейших операций внутри ядра. Псевдокод, реализующий описанный метод, представлен ниже:

```
void Kernel(parameter1, parameter2, …, volatile bool * abortionFlag)
{
   if ((*abortionFlag) == true)
      return;

   // изначальный исходный код
}
```

Данный метод выглядит довольно простым и легко реализуем при помощи шаблонов или макросов, однако существуют определенные условия для его правильной работы. Разделяемая переменная, используемая в качестве флага для остановки ядра, должна быть создана в неподкачиваемой и некэшируемой памяти ЦПУ, чтобы копирование данных из/в нее могло производиться одновременно с выполнением ядра на ГП [3, 4]. Для выполнения данного условия ГП должен иметь как минимум одно устройство прямого доступа к памяти [5]. Кроме того, вызовы выполнения ядра и команды для асинхронного копирования памяти обязаны исполняться в отдельных очередях задач на ГП. Последним требованием является то, что количество блоков вычислительной сетки и количество потоков в блоке должны быть достаточно большими для выполнения частых проверок значения разделяемой переменной внутри ядра. Можно предположить, что копирование данных между переменными или установка значения переменной внутри памяти ГП являются лучшими подходами для выставления флага остановки, однако данные операции производятся посредством вызова определенных функций (ядер) на ГП, поэтому с учетом вышеописанных требований они не будут производиться одновременно с выполнением ядра [4, 5].

Стоит отметить, что если в данный момент времени ГП используется для отрисовки, асинхронное копирование памяти будет произведено только после завершения выполнения ядра на ГП, даже если используются отдельные очереди задач. В системе с несколькими графическими процессорами конкретный ГП, используемый для вычислений, не должен быть задействован для отрисовки. В системах с одним ГП довольно сложно избежать данной проблемы и рекомендуется минимизировать отрисовку на ГП (данная проблема выходит за рамки статьи) [5, 6].

Обратная связь на ядре

Информация об уже выполненных ГП вычислениях внутри ядра до выставления флага остановки весьма полезна в случае, когда в гетерогенной системе более быстрое вычислительное устройство становится доступным для использования. При наличии данной информации другому устройству не нужно выполнять повторно те же вычисления, которые были выполнены на изначальном ГП, таким образом достигается ускорение работы системы при миграции вычислений между устройствами. Существуют несколько подходов для решения данной проблемы. Первый подход основан на использовании атомарных операций над переменной, которая выступает как счетчик потоков в ядре. Такой подход дает точное значение числа потоков, которые закончили вычисления внутри ядра, до выставления флага остановки. Псевдокод подхода приведен ниже:

```
void Kernel(parameter1, parameter2, …, volatile bool * abortionFlag, unsigned int * counter)
{
   if ((*abortionFlag) == true)
      return;

   // изначальный исходный код

   atomicAdd(counter);
}
```

Недостатком такого подхода является вышеуказанное использование атомарных операций над переменной, что замедляет работу ядра, поскольку в определенный момент времени только один поток может изменить состояние переменной. Чтобы избежать этого, можно использовать только информацию с первого (или любого другого) потока в блоке (второй подход). Поток сохраняет номер текущего блока (по сути, количество блоков, которые были обработаны) в переменную. Псевдокод представлен ниже:

```
void Kernel(parameter1, parameter2, …, volatile bool * abortionFlag, volatile int * startedBlockID)
{
   if ((*abortionFlag) == true)
      return;

   // изначальный исходный код

   if (threadIDInBlock == 0)
      atomicSet(startedBlockID, blockID);
}
```

К сожалению, данный подход не может правильно работать на всех ГП, поскольку не каждый ГП гарантирует последовательную обработку блоков в вычислительной сетке. Стоит также учесть, что несколько блоков в вычислительной сетке могут одновременно обрабатываться одним ГП при наличии нескольких потоковых процессоров [4, 5]. Точка синхронизации разрешает проблему состояния гонки между потоками. Третий подход основывается на идее, что первый (или любой другой) поток записывает номер блока в массив только после точки синхронизации. Размер массива равен количеству блоков в вычислительной сетке. Псевдокод приведен ниже:

```
void Kernel(parameter1, parameter2, …, volatile bool * abortionFlag, int * completedBlockID)
{
   if ((*abortionFlag) == true)
      return;

   // изначальный исходный код

   synchronise();
   if (threadIDInBlock == 0)
      completedBlockID[blockID] = 1;
}
```

Точка синхронизации в конце тела ядра не должна значительно замедлить его работу, поскольку потоковый процессор всегда выполняет синхронизацию потоков внутри одного блока до того, как приступить к выполнению следующего блока.

Результаты

Для получения достоверных данных измерения для рассмотренного метода были проведены на различных вычислительных системах. Для этих целей было создано два отдельных ядра: «оригинальное ядро», содержащее набор простых математических операций, и «ядро с проверкой флага» с тем же самым набором операций и с дополнительным кодом для проверки значения разделяемой переменной (флага). Если флаг был выставлен, дальнейший код внутри ядра не выполнялся. Размер массива данных для обработки равнялся 1048576 (1024×1024) элементов, размер блока был установлен в 1024 потока. Запуск каждого ядра с ожиданием его завершения был выполнен 1000 раз, и для каждой такой итерации измерялось время. Затем были вычислены среднее значение и дисперсия времени выполнения ядра. Все значения находятся в интервале +/– 3 сигма, что подтверждает стабильность измерений. Результаты представлены в табл. 1.

Таблица 1**. Результаты измерения выполнения ядер**

ГП	Время выполнения оригинального ядра, мс	Время выполнения ядра с проверкой флага, мс	Разница во времени между ядрами, мс	Разница во времени между ядрами, %	Отклик на выставление флага остановки, мс
GeForce GT 620	88,789 +/– 0,0468	89,195 +/– 0,0464	0,406	0,46	1,098 +/– 0,0461
GeForce 940M	34,271 +/– 0,0194	34,345 +/– 0,0198	0,074	0,22	0,161 +/– 0,0269
GeForce GTX 960	16,041 +/– 0,0277	16,052 +/– 0,0190	0,011	0,069	0,130 +/– 0,0215

Логично предположить, что ядро с дополнительным кодом для проверки флага медленнее оригинального, однако накладные расходы являются незначительными (< 0,5 %). Для получения точных результатов использовался дополнительный таймер, применение которого позволяло генерировать сигнал для остановки ядра на ГП через фиксированный промежуток времени (в

данном случае 10 мс). Флаг остановки ядра выставлялся при помощи асинхронного копирования памяти с ЦПУ на ГП. Время отклика ядра на выставление флага остановки занесено в последний столбец таблицы.

Результаты, представленные в табл. 1, выглядят многообещающе, но они были получены для ядер с большим размером вычислительной сетки, которая была установлена в 1024 блока. Теоретически, изменение размера вычислительной сетки должно влиять на время отклика ядра на выставление флага остановки. Табл. 2 отображает зависимость между размером вычислительной сетки и временем отклика. На основе данных в табл. 1 был выбран самый медленный ГП – GeForce GT 620 для получения потенциально наихудших результатов. Изначальный размер массива данных равнялся 4194304 (4096×1024) элементам, размер блока составил 1024 потока. Количество обработанных элементов в потоке зависит от вычислительной сетки ядра: при меньшем его размере больше элементов должны быть выполнены в одном потоке. Для каждого конкретного размера вычислительной сетки ядра были разработаны тесты с различным временем ожидания перед выставлением флага остановки (от 2 до 50 мс с шагом 1 мс). Это требовалось для получения наибольшего времени отклика ядра, так как возможна ситуация, при которой флаг остановки может быть изменен перед самым началом вызова варпа (группа из 32-х потоков) [4, 5].

Таблица 2. **Время отклика на выставление флага остановки ядра при различных размерах вычислительной сетки на GeForce GT 620**

Размер вычислительной сетки, блоки	Время выполнения ядра без выставления флага, мс	Время отклика на выставление флага остановки, мс	Время отклика в сравнении со временем выполнения оригинального ядра, %
4096	106,126	1,691	1,59
2048	102,736	1,844	1,80
1024	100,361	1,942	1,94
512	98,8313	1,990	2,01
256	97,7749	2,012	2,06
128	96,9064	2,596	2,68
64	96,1945	3,053	3,17
32	95,8686	3,920	4,09
16	95,6339	7,477	7,82
8	95,4486	13,391	14,03
4	95,329	24,937	26,16
2	95,241	48,081	50,48
1	95,199	79,260	83,26

Зависимость между размером вычислительной сетки и временем отклика ядра на выставление флага остановки легко описывается линейной функцией. Данная зависимость наблюдается до момента, когда размер вычислительной сетки не превышает 64 блоков. После этого уменьшение времени отклика на выставление флага остановки не является существенным: при увеличении вычислительной сетки в 64 раза, время отклика уменьшается только в 2 раза. Стоит отметить, что резкие скачки в значениях времени отклика, более заметные при малых размерах вычислительной сетки ядра, вызваны тем, что ЦПУ и ГП не работают в полной синхронизации и флаг остановки невозможно установить сразу же после запуска очередного блока или варпа. Увеличение времени работы ядра без выставления флага остановки с ростом размера вычислительной сетки связано с тем, что для случаев с меньшим размером сетки один поток выполняет вычисления на нескольких элементах входного массива, тем самым не тратится дополнительное время на перезапуск потока. Данная зависимость не является верной для всех ядер, выполняемых на ГП.

Все ядра, результаты которых представлены в табл. 1 и 2, не содержали кода для обратной связи. Авторы выполнили измерения для трех подходов, описанных в разделе «Обратная связь на ядре», на GeForce GT 620, результаты которых представлены в табл. 3. Условия запуска ядер и измерения времени их выполнения такие же, как и для измерений, результаты которых представленных в табл. 1.

Таблица 3**. Время выполнения ядра на GeForce GT 620**

Ядро	Время выполнения, мс	Время выполнения в сравнении с оригинальным ядром, %
Оригинальное ядро	88,195 +/– 0,0464	100,00
Ядро с проверкой флага	88,537 +/– 0,0475	100,39
Ядро с проверкой флага и счетчиком	90,745 +/– 0,0475	102,89

Ядро с проверкой флага и записью номера блока	88,676 +/– 0,0468	100,54
Ядро с проверкой флага, точкой синхронизации и массивом	88,796 +/– 0,0479	100,68

Результаты для первого, второго и третьего подхода для обратной связи на ядре представлены в строках «Ядро с проверкой флага и счетчиком», «Ядро с проверкой флага и записи номера блока» и «Ядро с проверкой флага, точкой синхронизации и массивом» соответственно. Первый подход значительно увеличивает время выполнения ядра (более чем на 2 %), в то же время второй и третий подход не вносят значительных замедлений в оригинальное ядро с кодом проверки флага остановки.

Заключение

Рассмотренный метод довольно гибок с точки зрения программной разработки и требует незначительных изменений в исходном коде. Результаты показывают, что метод лучше работает для ядер с большим размером вычислительных сеток, нежели для ядер с малым размером, однако он совершенно незначительно увеличивает время выполнения ядра на ГП и гарантирует быстрый отклик на выставление флага остановки, даже при условии использования дополнительного кода для обратной связи. Накладные расходы для реализации метода значительно меньшие для более быстрых ГП.

Список литературы

1. Dynamic Checkpointing Policy in Heterogeneous Real-Time Standby Systems / G. Levitin [et al.] // IEEE Transactions on Computers. 2017. Vol. 66, iss. 8. P. 1449–1456.
2. Sousa Dynamic Load Balancing for Real-Time Video Encoding on Heterogeneous CPU+GPU Systems / S. Momcilovic [et al.] // IEEE Transactions on Multimedia. 2014. Vol. 16, iss. P. 108–121.
3. Fu C., Wang Z., Zhai Y. A CPU-GPU Data Transfer Optimization Approach Based on Code Migration and Merging // 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES). 13–16 October 2017.
4. Cheng J., Grossman M., McKercher T. Professional CUDA C Programming. John Wiley & Sons, Inc. 2014. 499 p.
5. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах. Введение в программирование графических процессоров. М.: ДМК Пресс, 2013. 232 с.
6. Shevtsov M. OpenCL™ and OpenGL* Interoperability Tutorial [Electronic resource]. URL: https://software.intel.com/en-us/articles/opencl-and-opengl-interoperability-tutorial (Date of access: 07.05.2018).

References

1. Dynamic Checkpointing Policy in Heterogeneous Real-Time Standby Systems / G. Levitin [et al.] // IEEE Transactions on Computers. 2017. Vol. 66, iss. 8. P. 1449–1456.
2. Sousa Dynamic Load Balancing for Real-Time Video Encoding on Heterogeneous CPU+GPU Systems / S. Momcilovic [et al.] // IEEE Transactions on Multimedia. 2014. Vol. 16, iss. P. 108–121.
3. Fu C., Wang Z., Zhai Y. A CPU-GPU Data Transfer Optimization Approach Based on Code Migration and Merging // 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES). 13–16 October 2017.
4. Cheng J., Grossman M., McKercher T. Professional CUDA C Programming. John Wiley & Sons, Inc. 2014. 499 p.
5. Sanders Dzh., Kjendrot Je. Tehnologija CUDA v primerah. Vvedenie v programmirovanie graficheskih processorov. M.: DMK Press, 2013. 232 s. (in Russ.)
6. Shevtsov M. OpenCL™ and OpenGL* Interoperability Tutorial [Electronic resource]. URL: https://software.intel.com/en-us/articles/opencl-and-opengl-interoperability-tutorial (Date of access: 07.05.2018).

Сведения об авторах

Губчик И.Н., инженер-программист «Blackmagic Design Technology Pte Ltd».

Иванов Н.Н., к.ф.-м.н., доцент, доцент кафедры электронных вычислительных машин Белорусского

Information about the authors

Hubchyk I.N., engineer-programmer of «Blackmagic Design Technology Pte Ltd».

Ivanov N.N., PhD, associate professor, associate professor of electronic computing machines

государственного университета информатики и радиоэлектроники.

Адрес для корреспонденции

220133, Республика Беларусь,
г. Минск, ул. Гамарника, 24–5
тел. +375-29-180-55-89;
e-mail: ivanovnn@gmail.com
Иванов Николай Николаевич

department of Belarusian state university of informatics and radioelectronics.

Address for correspondence

220133, Republic of Belarus,
Minsk, Gamarnika st., 24–5
tel.: +375-29-180-55-89;
e-mail: ivanovnn@gmail.com
Ivanov Nickolai Nickolaevich