АЛГОРИТМ ПОИСКА ПАРАЛЛЕЛЬНОСТИ БЕЗ СИНХРОНИЗАЦИЙ В РАМКАХ ЗАДАЧИ РАСПАРАЛЛЕЛИВАНИЯ ЦИКЛОВ ДЛЯ АРХИТЕКТУРЫ NVIDIA CUDA

А.А. Новокрещенов

Нижегородский государственный технический университет им. Р.Е. Алексеева
E-mail: andrew.novokreshchenov@gmail.com

Параллельная архитектура CUDA относится к классу SIMD-архитектур [1]. Вложенности циклов представляют собой наиболее подходящий класс программ для выполнения в рамках модели вычислений SIMD, в силу того, что циклы заключают в себе большой объем параллелизма, а также большой объем вычислений, особенно в приложениях, обрабатывающих большие массивы данных [2, 3]. Как следствие, циклы представляют большой практический интерес с точки зрения распараллеливания.

Основным подходом к задаче автоматического распараллеливания циклов является поиск параллелизма на уровне инструкций, в соответствии с которым, последовательная вложенность циклов рассматривается как множество экземпляров инструкций, на котором определен лексикографический порядок и отношения зависимости между экземплярами инструкций [2-5].

Учитывая то, что архитектура CUDA допускает одновременное выполнение большого количества потоков, параллельную программу целесообразно воспринимать как набор подмножеств исходного множества экземпляров инструкций [2]. Каждое подмножество представляет собой не что иное, как отдельный поток, в рамках которого экземпляры инструкций выполняются последовательно.

С учетом такого взгляда на последовательную и параллельную программы задача распараллеливания ставится как задача поиска отображения множества экземпляров инструкций последовательной программы на множество потоков. Наиболее часто отображение представляет собой набор функций, устанавливающих соответствие между экземплярами инструкций и идентификаторами потоков. Для каждой инструкции строится своя функция отображения [2, 5].

Во вложенности циклов срабатывание экземпляра каждой инструкции определяется текущим значением индексных переменных циклов, охватывающих эту инструкцию. Поэтому функции отображения чаще всего представляют собой функции от вектора индексных переменных. Более того, в подавляющем большинстве случаев функции отображения являются аффинными [2, 4-6]. Данная работа не является исключением, и в качестве прообраза функций отображения будут использоваться функции вида:

$$\Pi_i(\vec{x}_i) = \mathbf{A}_i\vec{x}_i + \mathbf{B}_i\vec{n}_i + \vec{c}_i, \qquad (1)$$

где $\vec{x}_i$, $\vec{n}_i$ – векторы индексных переменных и структурных параметров циклов, охватывающих инструкцию i соответственно; $\mathbf{A}_i$, $\mathbf{B}_i$, $\vec{c}_i$ – матрицы и вектор, определяющие отображение всех экземпляров инструкции i в пространство потоков [2, 4-6].

Аффинность функций отображения накладывает серьезные ограничения на класс входных программ. Последние должны представлять собой вложенности циклов со следующими свойствами:

- границы каждого цикла вложенности представляют собой аффинные выражения от индексных переменных охватывающих циклов и (или) констант;
- выражение, проверяемое в каждом операторе ветвления, входящем в состав вложенности должно представлять собой аффинное выражение от индексных переменных охватывающих циклов и констант;
- все функции обращения к массивам внутри вложенности циклов должны являться аффинными функциями от индексных переменных циклов, охватывающих обращение и (или) констант;

Несмотря на указанные ограничения, данный класс программ достаточно широк и представляет интерес с практической точки зрения [7].

Прежде чем сформулировать список требований к алгоритму поиска параллельности, следует сделать некоторые замечания, относительно организации процесса параллельных вычислений в рамках CUDA. Данные замечания станут отправной точкой для составления требований к алгоритму:

- программа для CUDA в ходе выполнения представляет собой множество параллельно исполняемых потоков;
- каждый поток обладает своим уникальным идентификатором (целочисленным вектором, каждая компонента которого неотрицательна);
- архитектура CUDA предназначена для эффективного одновременного исполнения большого числа потоков, и накладные расходы на планирование и запуск потоков очень малы;
- синхронизация запущенных потоков в CUDA достаточно затруднительна и является серьезным источником снижения быстродействия.

С учетом указанных архитектурных особенностей алгоритм поиска параллельности должен отвечать следующим требованиям:

1. результатом работы алгоритма должен являться набор аффинных функций (компонент $\mathbf{A}$, $\mathbf{B}$, $\vec{c}$), отображающих экземпляры инструкций исходной программы в пространство потоков;
2. в качестве входа алгоритм должен принимать информацию о зависимостях по данным, присутствующих в распараллеливаемом участке кода, в форме многогранного сокращенного графа зависимостей (МСГЗ);
3. для каждой инструкции распараллеливаемого участка кода исходной программы должна быть составлена своя функция отображения;
4. функции должны быть составлены таким образом, чтобы число потоков, в рамках которых будет выполняться распараллеливаемый участок кода, было максимально возможным для конкретной программы;
5. функции должны быть составлены таким образом, чтобы все потоки выполнения были независимы;
6. каждый отдельный поток должен идентифицироваться уникальным вектором с целочисленными неотрицательными компонентами;
7. функции отображения должны быть пригодны для генерации параллельного кода уже существующими методами генерации.

МСГЗ используется потому, что является одной из распространенных форм представления зависимостей, а также обобщением таких часто используемых форм представления зависимостей, как векторы направлений, множества расстояний и уровни зависимостей [8].

В рамках данной работы проведено исследование существующих методов автоматического распараллеливания циклов для SIMD-архитектур. Исследование показало, что на сегодняшний день методов распараллеливания циклов, отвечающих всем указанным требованиям, которые можно было бы использовать «напрямую», без каких

либо модификаций, не существует. Это обстоятельство стало стимулом к настоящей работе.

Для представления предлагаемого алгоритма поиска параллельности введем следующие обозначения. Инструкции исходной вложенности циклов будем обозначать в соответствии с порядком их следования в исходном коде – 1, 2, 3 и т.д. Экземпляр инструкции *i* будем обозначать i^*. Вектор индексных переменных циклов, охватывающих инструкцию *i*, будем обозначать $\vec{x}_i$. Конкретное значение вектора $\vec{x}_i$, определяющее i^*, будем обозначать $\vec{x}_i^*$. Функцию отображения экземпляров инструкции *i* в пространство потоков будем обозначать Π_i. Разность функций отображения $\Pi_j(\vec{x}_j) - \Pi_i(\vec{x}_i)$ будем обозначать $\Delta_{i,j}$. Домен инструкции *i* будем обозначать D_i. Следует отметить, что домен D_i представляет собой параметрический многогранник от $\vec{x}_i$, вектора структурных параметров циклов и константного вектора. Многогранник зависимостей, определяющий зависимость между инструкциями *i* и *j* будем обозначать $D_{i,j}$. При этом инструкция *j* в таком обозначении, является зависимой. Многогранник зависимостей $D_{i,j}$ представляет собой параметрический многогранник от $\vec{x}_i$, $\vec{x}_j$, векторов структурных параметров циклов, охватывающих инструкции *i, j,* и константного вектора. Все используемые понятия подробно объясняются в [2, 5, 9].

Первый шаг предлагаемого алгоритма заключается в поиске размерности пространства потоков. Размерность пространства потоков непосредственно влияет на размерность искомых компонент $\mathbf{A}$, $\mathbf{B}$, $\vec{c}$.

Для определения размерности пространства, в котором будут расположены потоки, в данной работе предлагается следующий алгоритм. Для каждой инструкции *i* входной программы необходимо выполнить:

1. вход: D_i – домен инструкции *i*, все многогранники зависимостей, в которых участвует инструкция i – $D_{j,i}$, $D_{k,i}$ и т.д., а также все домены инструкций, от которых зависит i – D_j, D_k и т.д.;
2. к каждому многограннику зависимостей применить алгоритм исключений Фурье–Моцкина для удаления из них всех индексных переменных, которые входят в домены инструкций, от которых зависит инструкция *i*. Результатом данного шага являются многогранники $\overline{D}_{i,j}, \overline{D}_{k,j}, \ldots$;
3. определить множество независимых экземпляров инструкции i – $D_i^{\wedge}$ как разницу многогранников $D_i^{\wedge} = D_i - \overline{D}_{i,j} - \overline{D}_{k,j} - \ldots$;
4. определить, от какого количества индексных переменных зависит $D_i^{\wedge}$;

Размерность пространства потоков, в котором будет выполняться параллельная программа, предлагается определять как максимальную размерность многогранника $D_i^{\wedge}$ среди размерностей, полученных в ходе применения означенного алгоритма ко всем инструкциям исходной программы.

Вторым шагом предлагаемого алгоритма поиска параллельности является составление прообразов аффинных функций отображения для каждой вершины входного МСГЗ (каждой инструкции исходной программы). Данная процедура основана на аффинной форме леммы Фаркаша [4, 9]. Лемма утверждает, что функция $f(\vec{x})$ будет неотрицательна на всем многограннике $D = \mathbf{A}\vec{x} + \vec{b} \geq \vec{0}$ тогда и только тогда, когда она представляет собой положительную аффинную комбинацию вида:

$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(\mathbf{A}\vec{x} + \vec{b}), \lambda_0 \geq 0, \vec{\lambda}^T \geq \vec{0}.$$

Множители λ_i называют множителями Фаркаша. С учетом условия 6 списка требований, предъявленных к алгоритму поиска параллельности, функция отображения каждой инструкции должна быть неотрицательна на всем домене соответствующей инструкции:

$$\Pi_i(\vec{x}_i) = \lambda_0 + \vec{\lambda}^T D_i, \lambda_0, \vec{\lambda}^T \geq \vec{0}. \quad (1)$$

Таким образом, для каждой инструкции *i* исходной программы должен быть составлен прообраз функции отображения $\Pi_i(\vec{x}_i)$ в форме (1), размерность которой должна совпадать с $D_i^{\wedge}$.

После этого процедура поиска параллельности сводится к процедуре определения множителей Фаркаша, которые приводят к построению функций отображения, удовлетворяющих пунктам 4, 5, 6, 7 списка требований, предъявленных алгоритму поиска параллельности.

Третьим шагом предлагаемого алгоритма является учет зависимостей, а более точно – учет условия 5. Для этого необходимо, чтобы для каждой дуги входного МСГЗ выполнялось следующее условие. Все экземпляры инструкций *i*, *j*, принадлежащие многограннику $D_{i,j}$, помечающему дугу МСГЗ, должны выполняться в рамках потока с одним номером [2]:

$$\Delta_{i,j} = \Pi_j(\vec{x}_j^*) - \Pi_i(\vec{x}_i^*) = \vec{0}, \forall\{\vec{x}_j^*, \vec{x}_i^*\} \in D_{i,j}. \quad (2)$$

На понятийном уровне это означает, что все экземпляры инструкций, между которыми существует зависимость, должны выполняться в рамках одного потока. Выражение (2) можно сократить, выразив $\vec{x}_i$ через $\vec{x}_j$ и исключив $\vec{x}_i$ из $D_{i,j}$, используя метод исключений Фурье–Моцкина:

$$\overline{\Delta}_{i,j} = \Pi(\vec{x}_j^*) - \Pi(h(\vec{x}_j^*)) = \vec{0}, \forall\vec{x}_j^* \in \overline{D}_{i,j}, \quad (3)$$

где *h* – аффинная функция, определяющая соответствие между зависимыми экземплярами инструкций *i* и *j*. Понятие функции *h* позаимствовано из работы [4].

Результатом третьего шага является набор систем уравнений относительно искомых множителей Фаркаша, которые содержат в себе все ограничения, вытекающие из зависимостей данных. Способ получения систем уравнений из выражений вида (3) объясняется в [2]. Следует отметить, что, в силу аффинности прообразов (1), данные системы всегда представляют собой системы линейных уравнений.

Четвертым шагом предлагаемого метода является учет условия 4. Для того, чтобы количество параллельных потоков выполнения было максимальным, необходимо, чтобы каждый независимый экземпляр каждой инструкции выполнялся в отдельном потоке. Из геометрических соображений очевидно, что последнее будет справедливо, если для каждой инструкции *i* исходной программы все точки многогранника $D_i^{\wedge}$ будут взаимнооднозначно отображаться в пространство потоков. Поскольку пространства итераций и пространство потоков являются целочисленными, то взаимнооднозначное отображение возможно лишь в том случае, если матрица, составленная из коэффициентов $\Pi_i(\vec{x}_i)$ при индексных переменных, оставшихся в $D_i^{\wedge}$, является унимодулярной.

Всякая элементарная матрица унимодулярна [10]. Основываясь на данном утверждении, можно получить линейные уравнения, учитывающие требование 4. Данные уравнения следует добавить к соответствующим системам, полученным на третьем шаге. Так же к этим системам добавляются неравенства, требующие положительности множителей Фаркаша.

Пятым шагом предлагаемого метода является отыскание решения на полученных системах уравнений/неравенств. Решение данных систем предлагается находить следующим способом. Если с каждым неизвестным множителем Фаркаша связать одномерное пространство, то системы ограничений, полученные на предыдущем шаге, превращаются в многогранники в Z^d, где d – это количество искомых множителей Фаркаша. Конечное решение предлагается определять как точку лексикографического минимума данного многогранника. Для определения лексикографического минимума на многограннике следует ввести отношение порядка, т.е. упорядочить координатные оси пространства, в котором определен многогранник. Упорядочить их следует таким образом, чтобы коэффициенты при индексных переменных оказались, в результате, минимальными. Требование минимальности коэффициентов не является необходимым. Однако данное требование позволяет снизить степень «разреженности» пространства потоков. Наличие «разреженности» в пространстве потоков приводит к дополнительным неудобствам при генерации кода.

Результатом данного шага является вектор искомых множителей Фаркаша, по которому можно построить компоненты $\mathbf{A}$, $\mathbf{B}$, $\vec{c}$ для каждой функции исходной программы.

На основе найденных функций отображения может быть сгенерирован код функции-ядра CUDA. Для этого с некоторыми модификациями может быть использован алгоритм, подробно описанный в [2].

Литература

1. NVIDIA CUDA C programming guide. http://developer.nvidia.com/cuda-downloads.
2. Компиляторы: принципы, технологии и инструментарий / Под ред. И.В. Красиковой. М.: Вильямс, 2008. 1184 с.
3. Lamport L. The parallel execution of DO loops // Communications of ACM: ACM New York, 1974. Vol. 17, No. 2. P. 83-93.
4. Feautrier P. Some efficient solutions to the affine scheduling problem. Part 1: One dimensional time // International Journal of Parallel Programming. Norwell: Kluwer Academic Publishers, 1992. Vol. 21. P. 313-347.
5. Lim A., Lam M. Maximizing parallelism and minimizing synchronization with affine transform // Proceedings of the 24th ACM SIGPLAN-SIGACT: ACM New York, 1997. P. 215-227.
6. Lengauer C. Loop parallesation in polytope model // Proceedings of the 4th International Conference of Concurrency Theory CONCUR'93. Hildesheim, Germany, 1993. London: Springer-Verlag, 1993. P. 398-416.
7. Gautam G., Radjopadhye S. The Z-polyhedra model // Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. San Jose, USA: ACM, 2007. P. 237-248.
8. Касьянов В.Н., Мирзуитова И.Л. Реструктурирующие преобразования: алгоритмы распараллеливания циклов // Программные средства и математические основы информатики. Новосибирск: Институт систем информатики им. А.П. Ершова СО РАН, 2004. С. 142-188.
9. Pouchet L., Bastoul C. Iterative optimization in the polyhedral model: part 1 // Proceedings of the International Symposium on Code Generation and Optimization, CGO'07: IEEE Computer Society, Washington, 2007. P. 144-156.
10. Курош А.Г. Курс высшей алгебры. СПб.: Лань, 2004. 432 с.