ЗАКЛЮЧЕНИЕ

В работе получена характеризация хорошо k-согласованных и хорошо реберно-k-доминируемых деревьев. Для хорошо k-согласованных графов приведены свойства, обобщающие соответствующие свойства из [5]. Открытыми вопросами являются гипотеза о характеризации хорошо k-согласованных графов диаметра не менее $k + 1$ и обхвата не менее $4k + 4$ и сложность распознавания хорошо k-согласованных графов для $k \geq 3$.

Литература

1. *Plummer M.D*. Some covering concepts in graph // J. Combin. Theory. 1970. Vol. 8. P. 91–98.
2. *Chvátal V., Slater P*. A note on well-covered graphs // Ann. Discrete Math. 1993. Vol. 55. P. 179–182.
3. *Grünbaum B*. Matchings in polytopal graphs // Networks. 1974. Vol. 4. P. 175–190.
4. Lesk M., Plummer M.D., Pulleyblank W.R. Equi-matchable graphs // Graph Theory and Combinatorics. London: Academic Press. 1984. P. 239–254.
5. *Frendrup A., Hartnell B., Vestergaard P.D*. A note on equimatchable graphs // Australasian J. Combin. 2010. Vol. 46. P. 185–190.
6. *Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И.* Лекции по теории графов. М.: Наука. 1990. 384 с.
7. *Topp J., Volkmann L.* On packing and covering numbers of graphs // Discrete Math. 1991. Vol. 96. P. 229–238.
8. *Baptiste P., Kovalyov M.Y., Orlovich Y.L., Werner F., Zverovich I.E*. Graphs with maximal induced matchings of the same size // Proc. 14th IFAC Symposium on Control problems in Manufacturing. Bucharest, 2012. P. 57–62.

РЕАЛИЗАЦИЯ MIMD-ОРИЕНТИРОВАННЫХ АЛГОРИТМОВ НА CUDA

А. А. Козик

ВВЕДЕНИЕ

На сегодняшний день массивный параллелизм прочно укрепился не только в среде научно-ориентированных вычислений, но и в прикладных задачах различного уровня. Наиболее универсальной архитектурой для реализации вычислительных алгоритмов различной сложности является MIMD. Вполне очевидно, что для создания такого рода машины требуется значительное количество независимых процессоров, соединенных асинхронной коммуникативной сетью. Проблема заключается не только в физических размерах подобной машины, но и в вопросах стоимости, энергопотребления, а порой и с использования специализированных ОС и языков программирования. В качестве альтернативного решения воз-

можно использовать SIMD-машины. В силу природы и особенностей архитектуры, проблемы описанные выше, более таковыми не являются.

В общем случае любой алгоритм, в том числе и изначально ориентированный на вычисление на MIMD-машинах, можно запустить и на SIMD-машине, предварительно проведя его декомпозицию и сериализацию. Однако, в силу особенностей архитектуры SIMD-машин производительность такого алгоритма будет в разы меньше пиковой производительности, а итоговая форма исполняемого алгоритма может существенно влиять на общую производительность вычислений [1].

ПОСТАНОВКА ПРОБЛЕМЫ

CUDA – это программно-аппаратная архитектура. При этом аппаратная составляющая – это, модифицированный графический ускоритель, который в основе своей, является модифицированной SIMD-машиной. В частности, CUDA-программа выполняется в виде параллельных функций, которые именуются функциями-ядрами, которые в свою очередь исполняются множеством параллельных потоков. Эти потоки группируются по 32 потока в, так называемые, канаты (warps), которые в свою очередь организуются в блоки потоков и решетки. Каждый поток внутри блока потоков выполняет экземпляр функции-ядра на одном ПЭ.

В рамках одного каната, архитектура наиболее схожа с SIMD. Так для обеспечения наилучшей производительности, требуется чтобы все потоки (ПЭ) внутри каната, выполнялись синхронно, т.е. одни и те же инструкции [2]. В обратном случае, возникает так называемое расхождение потоков, которое приводит к тому, что часть ПЭ внутри одного каната, с отличающимися инструкциями будут простаивать и дожидаться завершения инструкций на остальных ПЭ. Что как очевидно, вызывает существенное ухудшение производительности.

Для иллюстрации примера проблемы, а также путей решения, введем некоторые условные обозначения:

- Множество **Op** – множество, содержащее n независимых операций получаемое после декомпозиции MIMD-ориентированного алгоритма.
- op_i – i-тый элемента множества **Op**, i □ [0, n].
- Множество **S** – множество, содержащее n состояний, каждое из которых соответствует элементу множества **Op**.
- *state* – переменная, содержащая следующее необходимое состояние для определенного блока потоков, *state* □ **S**.
- s_i – i-тый элемента множества **S**, такой что s_i соответствует op_i.
- c – индекс текущей операции, c □ [0, n].

После декомпозиции и сериализации MIMD-ориентированного алгоритма, его можно представить в следующем виде:

if ($state == s_0$) { op_0 }
if ($state == s_1$) { op_1 } (1)
…
if ($state == s_n$) { op_n }

Также будем считать, что операции множества **Op** – это простые программы. Под простыми программами будем понимать такую программу, которая представляет собой некую совокупность операторов присваивания, циклов и ветвлений, при этом все условия ветвлений, границы изменения параметров циклов и индексы элементов массивов описываются выражениями, как линейно, так и нелинейно зависящих как от параметров цикла, так и внешних параметров.

ПРЕДЛАГАЕМОЕ РЕШЕНИЕ ПРОБЛЕМЫ

В качестве первичного решения этой проблемы в Алгоритме 1, можно модифицировать внешний цикл следующим образом:

while (!*complete*) {
  if ($state == s_0$) { op_0 }
  if ($state == s_1$) { op_1 } (2)
  …
  if ($state == s_n$) { op_n }
}

Переменная *complete* – принимает значение *true*, после завершения последней необходимой операции *op*. Это обеспечит постоянную нагрузку машины, в контексте CUDA, по возможности исключая простой в работе системы, а также дублирования операций. Однако, стоит также обратить внимание на то, что при не оптимальном расположении проверок на определение текущего состояния возникают избыточные проверки. Таким образом, для минимизации времени выполнения итерации внешнего цикла, особое внимание следует обратить на порядок расположения проверок.

Будем считать, что операции можно классифицировать по признаку предполагаемой сложности (в смысле затраченного времени на выполнение) на: легкие, сложные, переменной сложности и др. Кроме этого, операции должны иметь возможность выполняться как в независимом порядке, так и в строгой последовательности, которая может зависеть от контекста алгоритма и изменяться в процессе его исполнения. В таких

случаях может потребоваться добавление дополнительных проверок внутри условного цикла, либо может возрасти количество лишних проверок, когда порядок операций для данного случая предопределен.

Предлагаемое решение заключается в составление ориентированного взвешенного графа **G**, такого что вершинами этого графа будут являться элементы множества **S**, ребра графа будут указывать на зависимости, в смысле последовательности исполнения, состояний (операций), а вес каждого ребра будет задаваться динамически при помощи функции **F(***a, b, ...***)**. Таким образом мы получим некоторый модифицированный подграф графа зависимостей точек пространства итерации [3]. В нашем случае точками пространства итерации будут выступать состояния из множества **S**, так как именно они определяют выполняемую операцию, ориентация ребер будет соответствовать ориентации ребер графа зависимостей, так как их смысловая нагрузка остается той же. Дуги зависимостей могут быть использованы как один из дополнительных параметров функции **F(***a, b, ...***)**, кроме этого вес ребра будет определять следующее состояние, которое необходимо проверить.

Таким образом Алгоритм 2 можно свести к следующему:

```
while (true)
{
  i = M(c);  Op[i];                                    (3)
}
```

Добавим новый s_{n+1} соответствующий дополнительной операции o_{n+1}, которая прерывает выполнение внешнего цикла **while** необходимыми средствами языка программирования. Функция **M(***c***)**, которая в качестве параметра принимает индекс текущего состояния, а возвращает индекс следующей возможной операции.

Функция **M(***c***)** представляет собой поиск кратчайшего пути по графу **G** от вершины s_c и до вершины равной по значению новому значению переменной *state*. Алгоритмом поиска может быть любой подходящий алгоритм учитывающий ориентированность и вес ребер. При этом вес ребер определяется на этапе поиска при помощи функции **F(***a, b, ...***)**, где параметрами являются: *a* – начальная вершина, *b* – конечная вершина, а также дополнительный набор параметров, который может быть необходим для определения веса конкретного ребра. Очевидно, что функция **M(***c***)** должна учитывать, что при наличии всего одного исходящего ребра из s_c к следующей вершины, либо петли, выполнение **F(***a, b, ...***)** должно быть пропущено.

ЗАКЛЮЧЕНИЕ

Даже при простейшей реализации, предложенное решение ограничивает необходимый объем проверок. Оно может быть адаптировано под алгоритмы различной сложности и условия декомпозиции. Таким образом, предложенное решение позволяет контролировать и адаптировать под текущую ситуацию, поток исполняемых инструкций на уровне одного каната, минимизировать количество выполнений лишних проверок, определять и контролировать ход выполнения сложных алгоритмов на SIMD-машине и в частности на CUDA, обеспечивая полезной нагрузкой большее количество ПЭ и снижая степень расхождение потоков.

Литература

1. *Dietz, H. G., Young B. D.* MIMD Interpretation on a GPU. Kentucky, 2009.
2. NVIDIA CUDA compute unified device architecture programming guide version 2.0. Santa Clara, 2008.
3. *Воеводин, В. В.* Массивный параллелизм и декомпозиция алгоритмов // Журнал Вычислительной Математики и Математической Физики. Т. 35. С. 988 – 996, 1995.
4. *Sanders, P.* Efficient Emulation of MIMD Behavior on SIMD Machines. Karlsruhe, Germany, 1996.
5. NVIDIA’s Next Generation CUDA Compute Architecture: Fermi. Santa Clara, 2009.
6. NVIDIA’s Next Generation CUDA Compute Architecture: Kepler GK110. Santa Clara, 2012.

ПОСТРОЕНИЕ КОМПЛЕКСА ИНФОРМАТИВНЫХ ДЕРМАТОГЛИФИЧЕСКИХ ПАРАМЕТРОВ В ЗАДАЧАХ БИОМЕТРИЧЕСКОГО АНАЛИЗА

В. А. Кулинкович

В последнее время цифровая обработка изображений находит все большее применение в различных областях биомедицины. В медицине изучаются и используются для диагностики различных заболеваний отклонения в пальцевой дерматоглифике. Визуальная обработка дерматоглифических отпечатков весьма трудоемкая, часто бывает необъективной и уменьшает достоверность результатов. Автоматизация этого процесса, применение методов цифровой обработки изображений и дискриминантного анализа способно существенно повысить скорость и эффективность проведения диагностики.

Дерматоглифика – это наука, изучающая рисунки кожи. Наиболее характерные рисунки кожи человека находятся на подушечках пальцев, хотя можно найти рисунки кожи на всех фалангах пальцев, на ладонях и на ногах. Современная наука не имеет общепринятой теории о происхо-