Подходы к трансляции отдельных конструкций метаязыка в параллельный язык ППВС

Д.Н. Змеев, А.В. Климов, Н.Н. Левченко, А.С. Окунев

Институт проблем проектирования в микроэлектронике РАН, Москва

Аннотация: Для каждой модели вычислений параллельного программирования приходится разрабатывать алгоритм практически заново. Для решения этой проблемы предлагается новый метаязык, который позволяет описать алгоритм задачи один раз, после чего из этого описания можно будет выводить эффективные коды для разных моделей вычисления. Метаязык ориентирован на асинхронные распределенные алгоритмы и основан на принципе управления потоком данных. В статье описаны подходы к трансляции некоторых конструкций метаязыка в параллельный язык параллельной потоковой вычислительной системы (ППВС) путем трансляции произвольной программы на метаязыке (или некоторого его подмножества) в собственное базовое представление на языке ППВС.

Ключевые слова: параллельное программирование, метаязык, потоковая модель вычислений, трансляция с метаязыка.

Введение

В настоящее время существуют различные языковые парадигмы программирования высокопроизводительных вычислений на суперкомпьютерах (MPI [1], Shmem [2], CUDA [3] и др.). В каждой из них предъявляются свои требования к структуре и свойствам алгоритма, в связи с чем для каждой из этих парадигм программирования приходится создавать алгоритм решения задачи практически заново. Некоторые системы специально реконфигурируются под конкретный алгоритм [4,5], что повышает эффективность решения задачи.

Одним из решений этой проблемы является переход к новой парадигме параллельного программирования. Новый параллельный метаязык и новый подход к параллельному программированию дает возможность создавать математическое описание алгоритма, а затем систематически выводить из него эффективные программы для различных вычислительных платформ.

«Параллельность» метаязыка заключается в том, что он должен допускать такую асинхронную работу, когда разные фрагменты могут

работать независимо, а сигналом к началу выполнения некоторого вычислительного фрагмента является готовность всех его входных данных. Модель вычислений с такими свойствами называют потоковой или моделью вычислений с управлением потоком данных (dataflow) [6, 7]. Такая модель вычислений [8] реализуется в параллельной потоковой вычислительной системе (ППВС). ППВС [9] имеет свой оригинальный параллельный язык программирования высокого уровня. Трансляции отдельных конструкций метаязыка в параллельный язык ППВС и посвящена данная статья.

Ограничения базового подмножества параллельного языка ППВС

Основными для параллельного языка ППВС и метаязыка являются следующие понятия: токен, ключ, контекст, программный узел. Токеном называется структура данных, в состав которой входит передаваемый операнд (данное), ключ с контекстом и адресом программного узла, маска (позволяет «принудительно» сопоставлять токены, поля контекста которых отличаются, но маскированы), кратность (разрешенное число взаимодействий с другими токенами) и ряд других специальных признаков. Контекст определяет положение данного (операнда) в виртуальном адресном пространстве задачи.

Поскольку параллельный язык ППВС фактически образует подмножество метаязыка, то решается задача преобразования произвольной программы на метаязыке (или некоторого его широкого подмножества) в собственное базовое подмножество параллельного языка ППВС.

Базовое подмножество характеризуется следующими ограничениями:

– в базовом подмножестве отсутствуют какие-либо структуры (группировки) на уровне множеств узлов;

– если узел имеет более двух входов, то он не может принимать множественные токены (направляемые на несколько экземпляров одного

программного узла с помощью использования маскирования отдельных полей контекста) и токены с кратностью;

– суммирование (и другие свертки) на входах завершается только по задаваемому количеству или явному сигналу (токену на другой вход), то есть нет механизма автоматического определения завершения потока входящих токенов на собирающий вход.

Трансляция в базовое подмножество будет определяться через раскрытие конструкций и узлов, не попадающих в подмножество через другие средства (если последние тоже не попадают в подмножество, то трансляция продолжается рекурсивно).

Трансляция многовходовых узлов

В базовом языке на многовходовый узел могут приходить токены только с единичной кратностью и без масок. Поэтому сложные многовходовые узлы должны раскладываться на несколько узлов с меньшим числом входов.

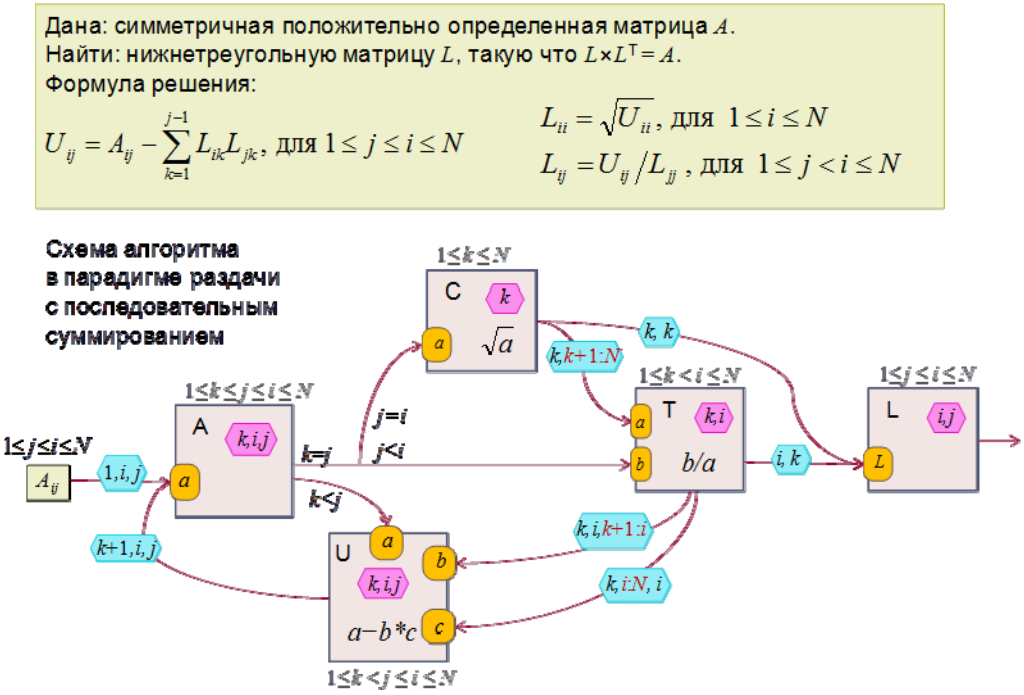

Рис. 1. – Разложение Холецкого с последовательным суммированием

Рассмотрим узел U из алгоритма разложения Холецкого в варианте с последовательным суммированием (рис. 1). Он имеет 3 входа: ***a***, ***b***, ***c*** и вычисляет формулу *a–b*c*. Его индекс (контекст) – *<k, i, j>*. При этом на вход ***a*** приходит токен с точным индексом *<k, i, j>*, на вход ***b*** – с индексом *<k, i, k+1:i>*, а на вход ***c*** – с индексом *<k, i:N, j>*.

Удобство диапазонов заключается в том, что токен с диапазоном может раскладываться сетью при рассылке на несколько ядер. Пусть, например, имеется *P=p*p* процессоров и функция распределения с двумерным массивом процессоров: *place(<k, i, j>) = (i%p, j%p)*. Токен вида *x⟶U.b<k, i, k+1:i>* может быть преобразован в набор из p токенов:

x⟶U.b<k, i, k+1:p:i>,

x⟶U.b<k, i, k+2:p:i>,

...

x⟶U.b<k, i, k+p:p:i>,

каждый из которых передается в свой процессор с номером *(i%p, (k+l)%p), l=1..p.* (средний элемент диапазона задает шаг). Аналогично, токен вида *y⟶U.c<k, i:N, j>* преобразуется в набор токенов

y⟶U.c<k, i+l:p:N, j>, l = 0..p–1,

каждый из которых передается в процессор с номером *((i+l)%p, j%p)*. В каждом процессоре по диапазону *(b:s:e)* однозначно вычисляется кратность по формуле *(e+s–b)/s*. Кроме того, при сопоставлении в ассоциативной памяти ключей должна проверяться принадлежность итогового индекса заданному диапазону.

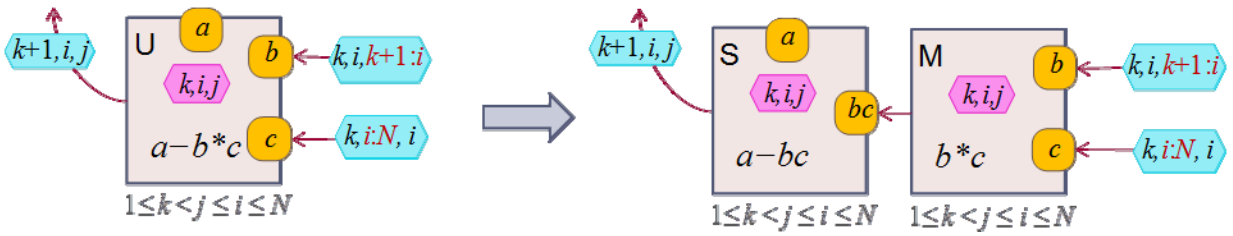

Рис. 2. – Преобразование трехвходового узла U в два двухвходовых S и M

На рис. 2 приведено преобразование трехвходового узла в схему из двухвходовых узлов. В результате получились два узла: двойной групповой узел M и обыкновенный S.

На узел M приходят токены с диапазонами. Но в базовом подмножестве языка ППВС нет диапазонов в индексах, поэтому диапазоны следует заменить звездочкой («*»), и для обоих добавить кратность *#(N–k)*. Однако, после такой замены могут появиться лишние взаимодействующие пары токенов. Чтобы их исключить, придется добавить соответствующие проверки внутрь узла M. После упрощений здесь будет нужна проверка условия $j \leq i$.

Но теперь узел M нельзя будет сделать распределенным, если мы хотим сохранить конечную кратность. Это неприемлемо, поскольку основной параллелизм получается именно при распределении узла U, а теперь это M и S. Поэтому либо придется вводить бесконечную кратность и последующие чистки, либо вводить в программу явную рассылку на несколько ядер.

Трансляция сложных структур (группы, циклы и ветви)

В метаязыке введены несколько видов группировок узлов: простая группа, цикл, изолированная группа и модуль. В эту же категорию можно отнести и узлы с ветвями.

Простая группа позволяет экономнее описать набор узлов с общей частью индекса, которая выносится в индекс группы. При трансляции (раскрытии) простой группы, надо добавить ее индекс к индексам всех узлов внутри группы и соответственно скорректировать индексы токенов.

В *группе-цикле* (или просто цикле) помимо индекса могут быть определены несколько *сдвиговых* переменных (одновходовых узлов), связывающих между собой итерации цикла. Изнутри цикла такая переменная видна как пара узлов (терминалов): один входной и один выходной. Из

выходного поступает внутрь цикла текущее значение величины, а на входной изнутри цикла подается вычисленное значение для следующей итерации. При трансляции цикла сначала индекс цикла добавляется к индексам всех внутренних узлов. Для сдвиговых переменных создается отдельный одновходовый узел, замещающий текущее значение, соответствующее левому (входному) терминалу. Он же может использоваться и для приема нового значения с той лишь поправкой, что на дуге в него появится выражение $i+1$ (вместо i) в поле индекса, соответствующем параметру цикла i. Для ясности можно завести отдельный узел для приема нового значения и уже от него передавать на первый с инкрементом i. С него же можно снимать значение для вывода наружу. Если требуется выдать только последнее значение, то следует на входе поставить условие типа $i=N$ (а для циклов с выходом по условию – с соответствующим условием).

Для *изолированных групп* в метаязыке предполагается некоторый механизм порождения сигнала о прекращении всякой активности в этой группе после ее инициации. В такую группу после ее инициации не может приходить никакой токен извне, кроме тех, что связаны причинно с этим сигналом (для повторной активации). Для ее реализации следует ввести соответствующие средства в базовый язык, иначе прямая трансляция невозможна. В каких-то случаях ее можно заменить подсчетами токенов на уровне программы. А в общем случае для реализации изолированных групп необходимо ввести в базовую архитектуру многоуровневую схему «детектора тишины».

Эта же схема будет средством для реализации суммирующих входов с модификатором «@» (активация по приходу всех слагаемых). Для этого все связи (внутри одной изолированной группы) должны быть ранжированы. При этом ранг выхода обычно равен максимуму входных рангов, а для узлов с собирающим (в частности – суммирующим) входом, помеченным

модификатором @, ранг выхода на 1 больше этого максимума. Иными словами, только активация таких узлов будет повышать ранг на 1. Она будет производится по достижению (детекции) «тишины» в подгруппе узлов с выходными связями меньшего ранга.

Модули (аналог процедур в обычных языках программирования) можно транслировать двумя способами: либо как макрос (с порождением узлов с уникальными именами для всех узлов модуля), либо нужно строить один вариант схемы на все использования, но добавить дополнительный индекс **instance**, по которому различать вызовы модуля, а в выходных узлах модуля расположить операторы **case** с выбором по значению этого индекса.

Ветви – это механизм, дающий возможность описать в одном узле несколько разных реакций на разные наборы пришедших токенов. Например, если узел трехвходовый, то он может иметь три разные ветви реакций на любую пару поступивших входов без третьего. Для самого общего случая потребуется ввести аналогичный механизм (возможно, с ограничениями) на базовый уровень. Одно из возможных решений описано в статье [10]. Но для некоторых случаев может быть достаточно использовать специальные токены ППВС, такие как «косвенность», «ПП-запрос», «стирание».

Заключение

Практика экспериментального использования параллельного языка ППВС показала, что в нем удобно и компактно выражаются довольно сложные алгоритмы. Среди них разные методы сортировки, разностные методы, задачи линейной алгебры, быстрое преобразование Фурье, молекулярная динамика, SAT-задача, ряд важных алгоритмов обработки графов (поиск вширь, нахождение кратчайшего пути, построение остовного дерева, разбиения на сообщества).

Если буквально пытаться реализовать аппаратно все высокоуровневые средства метаязыка, то такая архитектура будет очень сложной. Поэтому

важно уметь раскрыть (транслировать) сложные конструкции через элементы ограниченного подмножества, которое только и будет подлежать аппаратной реализации в рамках архитектуры ППВС.

В новом метаязыке заложен потенциал эквивалентных преобразований, на основе которого могут быть построены инструменты для извлечения из заданных на этом языке алгоритмов эффективных программных кодов для различных иных аппаратно-программных платформ. Задача преобразования произвольной программы на метаязыке в собственное базовое подмножество параллельного языка ППВС в будущем поможет решить проблему трансляции метаязыка и на другие архитектурные платформы.

Литература

1. CORPORATE The MPI Forum, 1993. MPI: a message passing interface. In Proceedings of the 1993 ACM/IEEE conference on Supercomputing (Supercomputing '93), ACM, New York, pp. 878-883. DOI=doi.org/10.1145/169627.169855.

2. Hongzhang Shan and Jaswinder Pal Singh. 1999. A comparison of MPI, SHMEM and cache-coherent shared address space programming models on the SGI Origin2000. In Proceedings of the 13th international conference on Supercomputing (ICS '99), ACM, New York, NY, USA, pp. 329-338. DOI= dx.doi.org/10.1145/305138.305210.

3. Sanders, J. and E. Kandrot, 2010. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley, 312 pages. ISBN: 0131387685.

4. Андреев А.А. Методика выбора базовой архитектуры реконфигурируемой вычислительной системы на основе методов теоретико-игровой оптимизации // Инженерный вестник Дона, 2013, №1. URL: ivdon.ru/ru/magazine/archive/n1y2013/1569/.

5. Строцев А.А., Андреев А.А. Оценка нахождения реконфигурируемой вычислительной системы в состояниях эффективного функционирования // Инженерный вестник Дона, 2012, №4 (часть 1). URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1212/.

6. Silc, J., B. Robic and T. Ungerer, 1998. Asynchrony in parallel computing: From dataflow to multithreading. Parallel and Distributed Computing Practices, vol. 1, no. 1, pp. 3-30.

7. Nowatzki, T., V. Gangadhar and K. Sankaralingam, 2015. Exploring the potential of heterogeneous von neumann/dataflow execution models. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15), ACM, New York, NY, USA, pp. 298-310.

8. Змеев Д.Н., Климов А.В., Левченко Н.Н., Окунев А.С., Стемпковский А.Л. Потоковая модель вычислений как парадигма программирования будущего // Информатика и её применения. 2015. Т. 9. Вып. 4. С. 29-36.

9. Стемпковский А.Л., Левченко Н.Н., Окунев А.С., Цветков В.В. Параллельная потоковая вычислительная система – дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // Журнал «Информационные технологии», 2008. №10. С. 2-7.

10. Климов А.В., Левченко Н.Н. Механизм ветвей в потоковом метаязыке UPL (METAL) и методы его реализации в ППВС «Буран» // Проблемы разработки перспективных микро- и наноэлектронных систем. 2018. Выпуск 3. С. 31-37.

References

1. CORPORATE The MPI Forum, 1993. MPI: a message passing interface. In Proceedings of the 1993 ACM/IEEE conference on Supercomputing

(Supercomputing '93), ACM, New-York, NY, USA, pp. 878-883. DOI= doi.org/10.1145/169627.169855.

2. Hongzhang Shan and Jaswinder Pal Singh. 1999. A comparison of MPI, SHMEM and cache-coherent shared address space programming models on the SGI Origin2000. In Proceedings of the 13th international conference on Supercomputing (ICS '99), ACM, New York, NY, USA, pp. 329-338. DOI= dx.doi.org/10.1145/305138.305210.

3. Sanders, J. and E. Kandrot, 2010. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley, 312 pages. ISBN: 0131387685.

4. Andreev A.A. Inženernyj vestnik Dona (Rus), 2013, №1. URL: ivdon.ru/ru/magazine/archive/n1y2013/1569/.

5. Strotsev A.A., Andreev A.A. Inženernyj vestnik Dona (Rus), 2012, №4 (1). URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1212/.

6. Silc, J., B. Robic and T. Ungerer, 1998. Parallel and Distributed Computing Practices, vol. 1, no. 1, pp. 3-30.

7. Nowatzki, T., V. Gangadhar and K. Sankaralingam, 2015. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15), ACM, New York, NY, USA, pp. 298-310.

8. Zmeev D.N., Klimov A.V., Levchenko N.N., Okunev A.S., Stempkovskiy A.L. Informatika i ee primeneniya. 2015. Vol. 9. No. 4. pp. 29-36.

9. Stempkovskiy A.L., Levchenko N.N., Okunev A.S., Tsvetkov V.V. Zhurnal «Informatsionnye tekhnologii». 2008. No. 10. pp. 2-7.

10. Klimov A.V., Levchenko N.N. Problemy razrabotki perspektivnykh mikro- i nanoelektronnykh sistem. 2018. Issue 3. pp. 31-37.