Инженерный вестник Дона, №4 (2018)
ivdon.ru/ru/magazine/archive/n4y2018/5304

Элементы графического метаязыка для параллельных вычислений

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

Институт проблем проектирования в микроэлектронике РАН, Москва

Аннотация: В статье приведено описание основных конструкций графического представления метаязыка. Метаязык разрабатывается для создания параллельных алгоритмов (без учета специфических особенностей отдельных архитектур высокопроизводительных вычислительных систем), что позволит получать эффективный код для различных моделей вычислений со значительно меньшими усилиями, чем это делается сейчас для распараллеливания задач на многоядерных системах. Графическая версия метаязыка наглядно представляет алгоритм параллельной программы в отличие от её текстовой формы.
Ключевые слова: графическое представление, метаязык, распараллеливание вычислений, потоковая модель вычислений.

Введение

Распараллеливание вычислений на сегодняшний день является наиболее эффективным методом повышения производительности вычислительных систем. Данный метод состоит из программной (распараллеливание задач) и аппаратной (увеличение числа вычислительных ядер, из которых состоит высокопроизводительная вычислительная система) составляющих. Основной импульс развития распараллеливанию вычислений был дан в начале 2000-х годов, когда производительность одного вычислительного ядра приблизилась к своему технологическому пределу, и начался стремительный переход к архитектуре многоядерных кристаллов. В результате появилось множество разных параллельных моделей вычислений, и для каждой из них требуется писать свою программу [1, 2]. Это заставляет искать возможность создания такого языка программирования, на котором можно один раз записать алгоритм, а затем для каждой модели вычислений порождать эффективный код с небольшими усилиями.

Именно такой универсальный параллельный метаязык, основанный на потоковой модели вычислений с динамически формируемым контекстом, и разрабатывается авторами [3, 4]. Потоковая модель вычислений (dataflow)

особенно была популярна в конце 90-х годов прошлого века [5, 6], однако по целому ряду причин оказалась невостребованной в то время. Эта модель вычислений используется также и при создании параллельной потоковой вычислительной системы (ППВС), в которой активация вычислительных квантов осуществляется по мере готовности данных [7]. Метаязык предназначен для создания программ в парадигме «раздачи» [8]. Благодаря явному использованию индексов для различения экземпляров вычислительных узлов появляется возможность предложить языковые средства отображения, обеспечивающие компилятор важной дополнительной информацией, достаточной для построения эффективного кода для выбранной платформы параллельных вычислений [3].

В последнее время наблюдается рост интереса к моделям программирования, основанным на управлении потоком данных (dataflow). В качестве примера можно привести модель параллельного программирования Concurrent Collections [9].

Описание графической версии метаязыка

Метаязык программирования позволяет выразить на нем алгоритм один раз, а потом из этого неизменного описания можно будет порождать эффективные программы для самых разных вычислительных платформ, архитектур, моделей вычислений (MPI, CUDA и др.). В этом языке вычислительный алгоритм представляется не последовательностью действий, а направленным графом, то есть набором узлов и направленных связей между ними. И поэтому для такого языка лучше подходит графическая форма представления программы-графа, а не текстовая.

Графическая версия метаязыка обладает достоинством наглядного представления информации и гораздо лучше соответствует природе человеческого восприятия. При необходимости, графические элементы

можно дополнить возможностью вводить программный код непосредственно на метаязыке.

Метаязык основывается на потоковой модели вычислений, базовым элементом которой является токен. Токеном называется структура данных, в состав которой входит предаваемый операнд (данное), ключ с контекстом и адресом программного узла, маска, кратность (разрешенное число взаимодействий с другими токенами) и ряд других специальных признаков. Контекст определяет положение данного в виртуальном адресном пространстве задачи. Адрес программного узла указывает на программу, которая будет выполняться над данными.

На рис. 1 приведено графическое представление таких элементов метаязыка как «токен». Токен может быть локальным, нелокальным и глобальным. Локальным называется токен только в том случае, когда и узел-источник (отправитель) токена и узел-приемник (получатель) при любой функции распределения будут всегда находиться в одном ядре.

Рис. 1. – Графическое представление элементов метаязыка (типы токенов)

Нелокальным называется токен, который может быть послан из одного вычислительного ядра в другой. Глобальный токен отличается от описанных тем, что он должен быть послан на все вычислительные ядра, причем его кратность (верхний предел числа взаимодействий) бесконечна. Контекст токена задается при помощи блока (шестиугольника) непосредственно на стрелке (знак «*» - означает, что поле контекста маскируется, то есть при сопоставлении такое поле будет считаться совпадающим с соответствующими полями других токенов, независимо от их содержимого).

На рис. 2 представлены базовые элементы графического представления метаязыка – стандартный узел и однопортовый узел. Фактически любая

программа может быть составлена из набора таких программных узлов. Каждая пара входных токенов приводит хотя бы к одной активации пакета, в ходе обработки которого генерируются новые выходные токены. Объединение таких узлов с помощью стрелок-токенов и позволяет создавать параллельный алгоритм. В зеленых прямоугольниках на рис. 2-4 приведены комментарии (пояснения) к элементам графического языка.

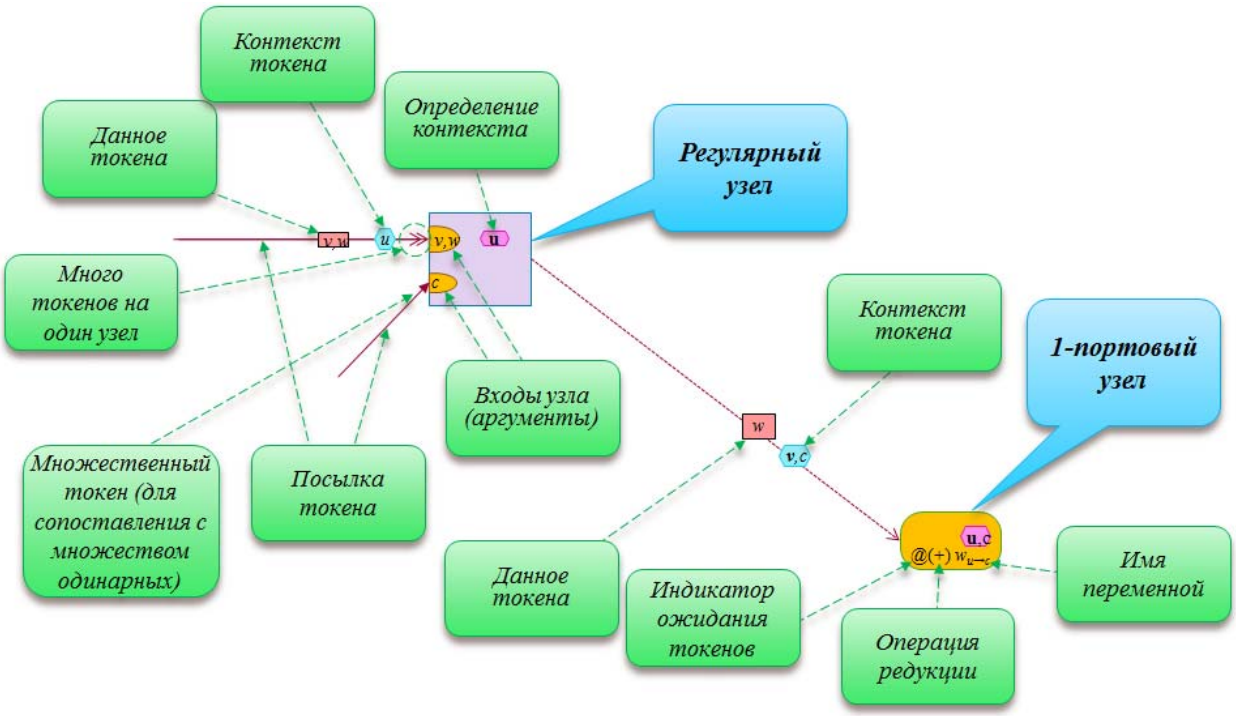

Рис. 2. – Графическое представление элементов метаязыка (регулярный и однопортовый узел)

На рис. 3 представлены такие элементы метаязыка как узел с тремя стандартными входами, групповой симметричный узел. Симметричный узел обладает свойством коммутативности входов (т.е. выполнение узла не зависит от порядка входных данных на его входах). Групповой симметричный вход на таком узле означает, что все пришедшие на такой вход токены должны образовать все возможные комбинации друг с другом.

Дополнительным элементом метаязыка являются следующие программные структуры: группа, цикл и модуль.

Группа представляет собой объединение нескольких узлов и связей между ними. Отображается в виде прямоугольника с тонкой линией границы, внутри которой расположены объединенные узлы. Каждая группа имеет уникальное имя и индекс. Группа предназначена для группового перемещения и копирования объектов. В качестве атрибутов содержит: имя группы (ставится слева вверху группы) и индекс, который визуализируется в правом верхнем углу в шестиугольнике (если список его полей не пуст). Этот индекс семантически приписывается слева к индексам всех узлов группы.

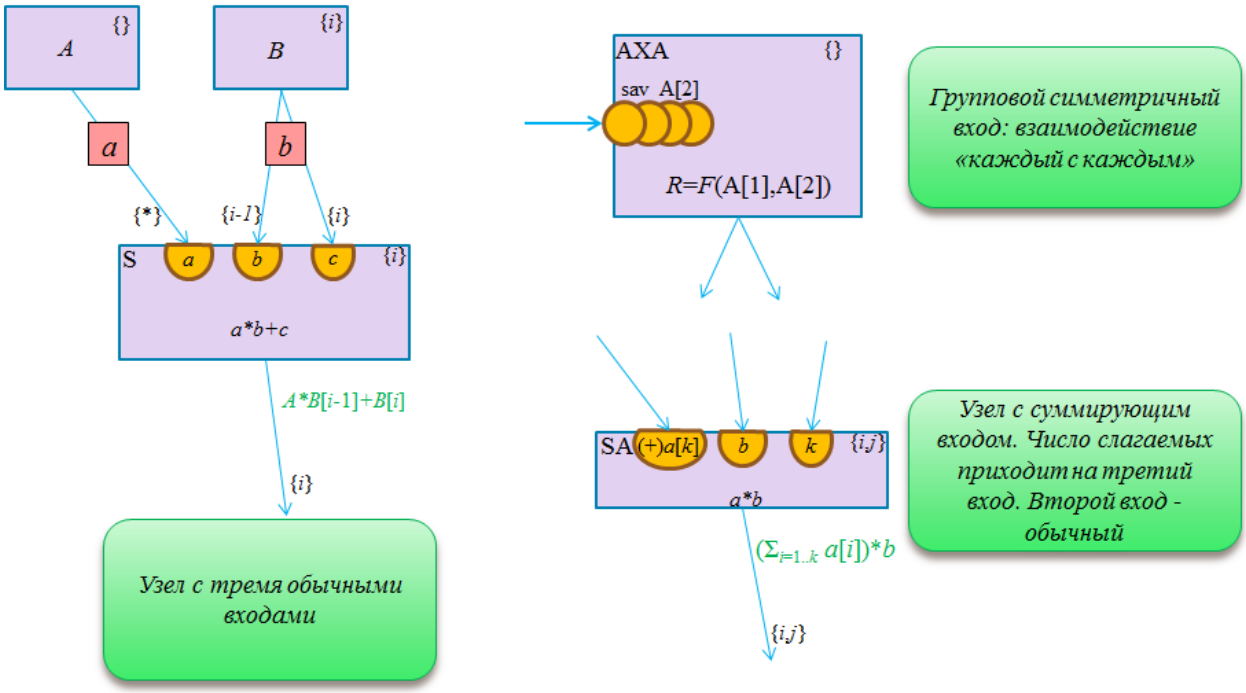

Рис. 3. – Графическое представление элементов метаязыка (узлы с суммирующим, симметричным входом, а также с тремя обычными входами)

Модуль – группа, образующая отдельную программную компоненту для многократного повторного использования. Отображается в виде прямоугольника с жирной линией границы.

Цикл (рис. 4) – группа узлов, которые составляют тело цикла, обводятся двойной сплошной линией. В свойствах группы-цикла задаются переменная цикла и условия выхода из него. Цикл может быть один из двух видов по способу окончания: по числу повторений и по условию (WHILE и UNTILL).

В цикле по числу повторений основной атрибут – число повторений, записывается в левом верхнем углу в форме квадратика с переменной (N), в которую извне поступает значение (если оно не константа). Аналогичный квадратик внутри с именем переменной цикла (например, i) представляет генератор номера итерации.

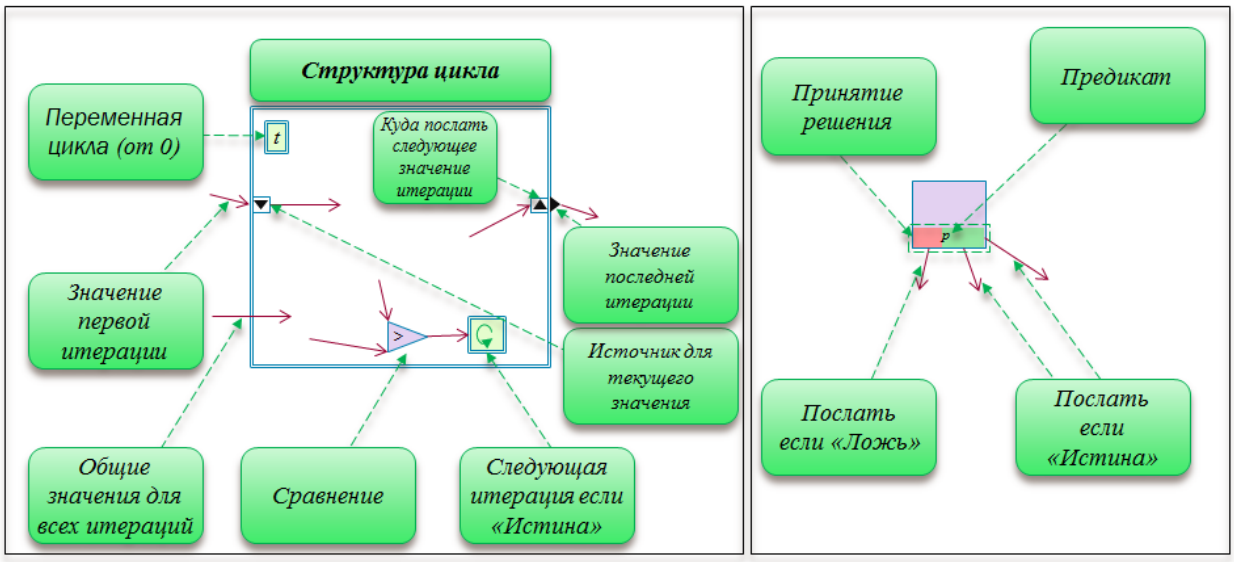

Рис. 4. – Графическое представление элементов метаязыка («цикл» и «принятие решения»)

В цикле по условию может быть генератор номера итерации и особого вида терминал в виде кружка красного цвета (условие выхода) или с кольцевой стрелкой зеленого цвета (условие продолжения), в который входит стрелка от узла, вычисляющего условие. Терминал стоит в левом верхнем углу, если это цикл WHILE (проверка в начале тела), и в правом нижнем, если это цикл UNTIL. Мнемоники частично заимствованы из среды разработки LabVIEW [10]. Цикл WHILE отличается от цикла UNTIL тем, что выходные значения сдвиговых переменных берутся из предыдущей итерации, а для UNTILL – из текущей.

Заключение

Графическая версия метаязыка позволяет наглядно представить создаваемый алгоритм параллельной программы. Концепция графического

программирования заключается в составлении программы из набора определенных объектов и соединении их между собой по особым правилам. При этом каждый из объектов и связь между ними в виде направленной линии имеют набор свойств и атрибутов, позволяющие при компиляции преобразовать их в объект с необходимой логической нагрузкой, соответствующей создаваемому алгоритму программы.

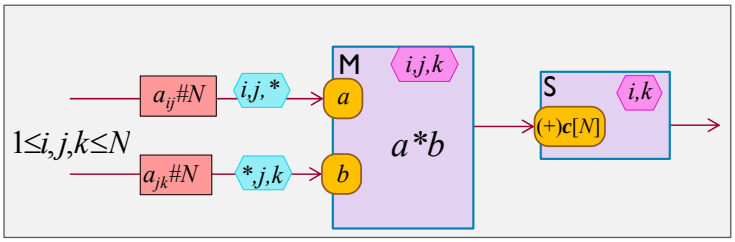

Рис. 5. – Графическое форма представления алгоритма «перемножение матриц»

С использованием графической версии метаязыка был создан целый ряд программ (перемножение матриц (рис. 5), задача N-тел, поиск кратчайших путей из заданной вершины – SSSP, Лувенский алгоритм разбиения графа на сообщества и др.).

Литература

1. Баклагин В.Н. Реализация распараллеливания алгоритмических структур, моделирующих экосистему озерных объектов, на многоядерные процессоры // Инженерный вестник Дона, 2013, №3. URL: ivdon.ru/ru/magazine/archive/n3y2013/1750/.

2. Аль-Хулайди А.А., Чернышев Ю.О. Разработка параллельного алгоритма нахождения оптимального решения транспортной задачи на кластере // Инженерный вестник Дона, 2011, №2. URL: ivdon.ru/ru/magazine/archive/n2y2011/445/.

3. Климов А.В. О парадигме универсального языка параллельного программирования // PLC–2017. Труды конференции. Ростов-на-Дону: Изд-во ЮФУ, 2017. С. 141–146. URL: plc.sfedu.ru/files/PLC-2017-proceedings.pdf.

4. Климов А.В., Окунев А.С. Графический потоковый метаязык для асинхронного распределенного программирования // МЭС–2016, сборник трудов, часть II, 3-7 октября 2016 года. М.: ИППМ РАН, 2016. С. 151-158. URL: mes-conference.ru/data/year2016/pdf/D149.pdf.

5. Arvind, D.E. Culler and K. Ekanadham, 1988. The Price of Fine-Grain Asynchronous Parallelism: An Analysis of Dataflow Methods. CONPAR 88, pp: 541-555.

6. Ben, L. and A.R. Hurson, 1994. Dataflow Architectures and Multithreading. Computer, 8(Aug. V), pp: 27-39.

7. 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.

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. Burke, M.G., K. Knobe, R. Newton and V. Sarkar. Concurrent Collections Programming Model. Encyclopedia of Parallel Computing, ed. D. Padua. Springer, Boston, MA, 2011, pp: 364–371.

10. Bress, Thomas J., 2013. Effective LabVIEW Programming. [S.l.]: NTS Press, 720 pages. ISBN 1-934891-08-8.

References

1. Baklagin V.N. Inženernyj vestnik Dona (Rus), 2013, №3. URL: ivdon.ru/ru/magazine/archive/n3y2013/1750/.

2. Al'-Khulaydi A.A., Chernyshev Yu.O. Inženernyj vestnik Dona (Rus), 2011, №2. URL: ivdon.ru/ru/magazine/archive/n2y2011/445/.

3. Klimov A.V. PLC–2017: trudy konferentsii (Proc. PLC–2017 conference). Rostov-on-Don, 2017. pp. 141–146. URL: plc.sfedu.ru/files/PLC-2017-proceedings.pdf.

4. Klimov A.V., Okunev A.S. MES–2016, sbornik trudov, chast' II, 3-7 oktyabrya 2016 goda (Proc. All-Russia Science&Technology Conference “Problems of Advanced Micro- and Nanoelectronic Systems Development”). Moscow, 2016. Part II. pp. 151-158. URL: mes-conference.ru/data/year2016/pdf/D149.pdf.

5. Arvind, D.E. Culler and K. Ekanadham, 1988. The Price of Fine-Grain Asynchronous Parallelism: An Analysis of Dataflow Methods. CONPAR 88, pp. 541-555.

6. Ben, L. and A.R. Hurson, 1994. Dataflow Architectures and Multithreading. Computer, 8(Aug. V), pp. 27-39.

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

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. Burke, M.G., K. Knobe, R. Newton and V. Sarkar. Concurrent Collections Programming Model. Encyclopedia of Parallel Computing, ed. D. Padua. Springer, Boston, MA, 2011, pp. 364–371.

10. Bress, Thomas J., 2013. Effective LabVIEW Programming. [S.l.]: NTS Press, 720 pages. ISBN 1-934891-08-8.