ISSN 1992-6502 (Print) Вестник УГАТУ ISSN 2225-2789 (Online)
2016. Т. 20, № 3 (73). С. 114–121 http://journal.ugatu.ac.ru

УДК 004.43

АСИНХРОННАЯ МОДЕЛЬ ВЫЧИСЛЕНИЙ НАД ОБЩЕЙ ПАМЯТЬЮ

С. Б. АРЫКОВ

arykov@mail.ru

ФГБУН «Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук» (ИВМиМГ СО РАН)

Поступила в редакцию 20.05.2016

Аннотация. В статье предлагается асинхронная модель вычислений, отличительной особенностью которой является управление на основе строгого частичного порядка. Показано, что такой способ задания управления обладает рядом преимуществ для параллельных вычислений, в частности, позволяет представлять алгоритмы с высокой степенью непроцедурности, а также полностью отделяет управление от вычислений. Разработан алгоритм преобразования этой модели в известную асинхронную модель с управляющими операторами. На базе предложенной модели разработана и реализована система программирования Аспект.

Ключевые слова: асинхронная модель вычислений, язык параллельного программирования Аспект, представление алгоритмов с высокой степенью непроцедурности.

Современные суперкомпьютеры имеют сложную архитектуру. Они состоят из огромного количество вычислительных узлов, на каждом из которых есть несколько процессоров, каждый из которых содержит несколько ядер. Активно внедряются гибридные системы, где в дополнение к основным процессорам используются графические ускорители, специализированные вычислители типа Cell B.E. или Intel Xeon Phi, вычислители на базе ПЛИС. Такой стремительный прогресс в использовании параллелизма аппаратной частью привел к большому разрыву между предоставляемыми ресурсами и возможностями по их эффективному использованию с помощью современных средств разработки. Задача создания эффективной параллельной программы, которая никогда не была простой, в последнее время превращается крайне трудоемкую.

Попытки повысить уровень параллельного программирования, переложив большую часть технических проблем на систему программирования, постоянно предпринимаются как в России, так и за рубежом. Из российских проектов можно отметить язык и систему НОРМА [1], систему DVM [2], язык mpC [3]. За рубежом большую известность получили такие проекты как Charm++ [4], Chapel [5] и другие. Однако, несмотря на обилие проектов, основными инструментами разработки эффективных прикладных параллельных программ по-прежнему остаются MPI и OpenMP (вместе со специализированными фреймворками для конкретных вычислителей), а проблема простой разработки параллельных программ и автоматического использования новых возможностей аппаратуры – не решенной, что связано с очень высокой сложностью задачи по повышению уровня параллельного программирования.

Преобладающий в настоящее время подход, когда человек вначале пишет последовательную программу (процедурное представление алгоритма), сознательно избавляясь от имеющегося в алгоритме параллелизма, а затем с помощью различных конструкций и инструментов пытается восстановить этот параллелизм обратно, является во многих случаях трудным при практическом использовании. С другой стороны, максимально непроцедурное представление алгоритма не позволяет конструировать эффективные параллельные программы.

Целью проводимых исследований является разработка системы параллельного программирования, в которой имеется возможность плав-

ного изменения степени непроцедурности представления алгоритмов через явное задание информационных и управляющих зависимостей. За счет этого, а также за счет специализации системы на решении задач численного моделирования, планируется добиться высокого качества автоматически конструируемых параллельных программ, в том числе, возможности адаптации к различным аппаратным архитектурам.

Одной из задач в рамках поставленной цели является разработка подходящей модели параллельных вычислений. Ее обсуждению и посвящена статья.

АСИНХРОННАЯ МОДЕЛЬ ВЫЧИСЛЕНИЙ

Простая асинхронная модель вычислений. Асинхронная модель вычислений была впервые предложена в работе [6] в результате исследований принципов программирования для вычислительных устройств, способных производить параллельную обработку информации.

Асинхронная программа (или *А-программа*) представляет собой набор $P = (M, A)$, где M – конечное множество *переменных*, $M = \{x_1, x_2, \dots, x_m\}$; A – конечное множество *А-блоков*, $A = \{A_k | k \in \overline{1,n}\}$.

Множество M называется *памятью*. Память состоит из переменных с неразрушающим чтением и записью, стирающей предыдущее содержимое переменной. Выделяется *информационная* IM и *управляющая* CM памяти: $M = IM \cup CM$.

Каждый А-блок A_k образован четверкой (M_k, T_k, O_k, C_k), где M_k – подмножество памяти M, используемое А-блоком A_k, причем $M_k = T_{k_{In}} \cup O_{k_{In}} \cup O_{k_{Out}} \cup C_{k_{In}} \cup C_{k_{Out}}$, $T_{k_{In}} \subseteq M$, $O_{k_{In}} \subseteq IM$, $O_{k_{Out}} \subseteq IM$, $C_{k_{In}} \subseteq M$, $C_{k_{Out}} \subseteq CM$; T_k – *спусковая функция* (или *триггер-функция*), представляющая собой предикат $T_k(x_1, x_2, \dots, x_l)$, $x_i \in T_{k_{In}}$, $i = \overline{1,l}$; O_k – *операция*, которая по значениям *входных* переменных $x_1, x_2, \dots, x_m$ вычисляет значения *выходных* переменных $y_1, y_2, \dots, y_n$, $x_i \in O_{k_{In}}$, $y_j \in O_{k_{Out}}$, $i = \overline{1,m}$, $j = \overline{1,n}$; C_k – *управляющий опертор*, который по значениям входных переменных $x_1, x_2, \dots, x_p$ вычисляет значения выходных переменных $y_1, y_2, \dots, y_q$, $x_i \in C_{k_{In}}$, $y_j \in C_{k_{Out}}$, $i = \overline{1,p}$, $j = \overline{1,q}$. Под «вычислением» понимается вычисление одной или нескольких частично рекурсивных функций.

Асинхронный вычислительный процесс – это множество T процессов $\{t_1, t_2, \dots, t_n\}$, где каждый процесс t_i исполняет некоторый А-блок A_k, то есть вычисляет его операцию O_k, а затем – управляющий оператор C_k.

Асинхронная система (или *А-система*) есть набор правил, позволяющих произвести вычисления для заданной А-программы P:

1. В произвольный момент времени вычисляются триггер-функции А-блоков из некоторого непустого подмножества A' множества А-блоков A и формируется множество *готовых* к исполнению А-блоков A'': $A'' = \{A_k | A_k \in A' \wedge T_k = \text{ИСТИНА}, k = \overline{1,n}\}$, $A' \subseteq A$, $A'' \subseteq A'$.

2. Из множества готовых А-блоков A'' выделяется некоторое подмножество A''', А-блоки которого запускаются на исполнение. Для каждого А-блока создается соответствующий процесс. Процессы исполняются независимо друг от друга.

3. Исполнение А-программы завершается, когда ни один процесс не исполняется и значения триггер-функций всех А-блоков ложны.

Правила формирования множеств A' и A''' асинхронной системой не регламентируются.

Асинхронная модель вычислений с массовыми операциями. С точки зрения параллельных вычислений, простая асинхронная модель вычислений имеет тот недостаток, что она не позволяет явно описывать массовые вычисления. Их либо приходится программировать внутри А-блоков, что скрывает большую часть параллелизма алгоритма, либо необходимо задать фиксированное количество одинаковых А-блоков, что не позволяет обрабатывать данные неизвестной длины, а также существенно увеличивает размер программы.

Для преодоления этих недостатков в [7] предложена *асинхронная модель вычислений с массовыми операциями*, которая расширяет простую асинхронную модель вычислений следующим образом:

1. Информационная память IM теперь состоит из множества $X = \{x, y, \dots, z\}$ *простых* переменных и множества $Y = \{\overline{x}, \overline{y}, \dots, \overline{z}\}$, разбитого на конечное число счетных, непересекающихся, линейно упорядоченных подмножеств, которые называются *массивами*, или *структурными переменными*: $IM = X \cup Y$, $X \cap Y = \emptyset$. Элементы массива $\overline{x} = \{x_1, x_2, \dots, x_n\}$ называются *компонентами* $\overline{x}$; компонент $\overline{x}_i$ обозначается $\overline{x}[i]$.

2. Пусть $\mathbb{N}$ – множество натуральных чисел, $G \subseteq A \times \mathbb{N}$. Элемент $(A_k, i) \in G$ обозначается A_k^i и называется i-м экземпляром А-блока A_k. Пусть $N_{A_k} = \{i \in \mathbb{N} | (A_k, i) \in G\}$. А-блок A_k называется *простым*, если N_{A_k} – одноэлементное множе-

ство, и *массовым*, если N_{A_k} содержит более одного элемента. Множество N_{A_k} называется *областью применимости* А-блока A_k. Экземпляр А-блока A_k^i описывается четверкой $\left(M_k^i, T_k^i, O_k^i, C_k^i\right)$, при этом:

- экземпляры А-блока обязательно отличаются хотя бы одним входным параметром: $O_{k_{In}}^i \neq O_{k_{In}}^j$, причем различаться могут только компоненты одноименных массивов;
- количество входных и выходных параметров всех экземпляров одного А-блока совпадает: $\left(\left|O_{k_{In}}^i\right| = \left|O_{k_{In}}^j\right|\right) \wedge \left(\left|O_{k_{Out}}^i\right| = \left|O_{k_{Out}}^j\right|\right)$.

Здесь $i \in \mathbb{N}$, $j \in \mathbb{N}$, $i \neq j$.

3. Значение триггер-функции проверяется для всех не выполняющихся экземпляров A_k^i. Любой экземпляр A_k^i с истинным значением триггер-функции T_k^i может начать исполнение, которое состоит в исполнении операции O_k^i, а затем управляющего оператора C_k^i.

Асинхронная модель вычислений с управлением на основе строгого частичного порядка. В моделях, описанных выше, управление полностью определяется управляющими операторами и скрыто внутри них, что делает эту информацию недоступной для анализа, который необходим для автоматизированного распределения ресурсов. Кроме того, для повышения уровня параллельного программирования необходимы более явные средства задания управления.

Асинхронная модель вычислений с управлением на основе строгого частичного порядка [8] отображает информацию об управлении на уровне модели, изменяя асинхронную модель вычислений с массовыми операциями следующим образом:

1. Экземпляр А-блока A_k^i описывается четверкой $\left(M_k^i, T_k^i, O_k^i, R_k^i\right)$, где M_k^i, T_k^i и O_k^i имеют прежнее значение, а R_k^i есть строгий частичный порядок, заданный на подмножестве C_k^i экземпляров А-блоков А-программы A: $\langle C_k^i, R_k^i \rangle$, которое будем также обозначать $\langle C_k^i, < \rangle$. $R_k = \bigcup_i R_k^i$, $R = \bigcup_k R_k$.

2. Каждый экземпляр А-блока исполняется не более одного раза. Многократное исполнение простого А-блока можно представить в виде одного массового А-блока (если снять ограничение на то, что экземпляры А-блока обязательно отличаются хотя бы одним входным параметром, то есть требование $M_k^i \neq M_k^j$ исчезает), где каждому исполнению простого А-блока соответствует один экземпляр массового А-блока. Такое изменение позволяет однозначно идентифицировать исполнение любого экземпляра его именем, что необходимо для задания управления частичным порядком.

3. При запуске А-программы на исполнение формируется множество A' *готовых* к исполнению экземпляров А-блоков A_k^i: $A' = \{A_k^i | \neg\exists\left(B_q^p, A_k^i\right) \in R \wedge T_k^i = \text{ИСТИНА}\}$.

4. Исполнение экземпляра А-блока A_k^i состоит в вычислении его операции O_k^i, проверке триггер-функций всех экземпляров B_q^p, у которых после вычисления O_k^i все предшествующие А-блоки оказались вычисленными, и добавлении в множество A' тех B_q^p, у которых триггер-функции имеют значение ИСТИНА.

А-программа, определенная таким образом, позволяет задать любую частично рекурсивную функцию даже в случае, когда операции экземпляров А-блоков реализуют лишь простейшие вычислимые функции. Доказательство этого факта можно найти в [9].

Степень непроцедурности представления алгоритмов в асинхронной программе можно регулировать на двух уровнях:

- первоначальная степень непроцедурности определяется программистом при разбиение алгоритма на А-блоки;
- при конструировании параллельной программы имеется возможность уменьшить степень непроцедурности за счет группировки [10], что может существенно сократить накладные расходы на исполнение параллельной программы.

КОНСТРУИРОВАНИЕ АСИНХРОННЫХ ПРОГРАММ

Управления в асинхронной модели вычислений на основе строгого частичного порядка может быть вычислено во время исполнения программы в режиме интерпретации, однако более эффективно до начала исполнения сгенерировать специализированный код, который реализует корректное управление. Для этого необходимо формально определить содержимое управляющих операторов, а также разработать алгоритмы, позволяющие автоматически преобразовывать управление на основе строгого частичного порядка в управление на основе управляющих операторов.

Определение управляющего оператора. *Признак завершения А-блока*. Это формальное определение было предложено в работе [11]. Каждому экземпляру A_k^i в управляющей памяти

ставится в соответствие переменная логического типа, которую будем называть *признаком завершения* экземпляра A_k^i и обозначать $P(A_k^i)$. Истинность значения переменной $P(A_k^i)$ означает, что экземпляр A_k^i завершил исполнение. Если значение переменной ложно, то экземпляр может быть неготовым к исполнению, готовым к исполнению либо исполняться. В начальном состоянии памяти $P(A_k^i) =$ ЛОЖЬ для всех A_k^i, $A_k \in A$, $k = \overline{1,n}$.

Управляющий оператор C_k^i присваивает переменной $P(A_k^i)$ значение ИСТИНА, проверяет все экземпляры, которые могут начать исполнение после завершения A_k^i, и добавляет в множество готовых экземпляров те из них, у которых признак завершения всех предшествующих экземпляров имеет значение ИСТИНА и которые имеют истинное значение триггер-функции.

А-программа представляет собой набор $P = (M, A, A_0)$, где M и A имеют прежнее значение, а A_0 есть множество А-блоков, готовых к исполнению сразу после старта А-программы.

Достоинства подхода:

- не зависит от типа переменной (простая, структурная);
- не зависит от типа управления (прямое, потоковое);
- объем управляющей памяти зависит от количества А-блоков.

Недостатки подхода:

- в случае реализации экземпляра А-блока с помощью А-программы отслеживание вычисления отдельных переменных А-блока требует дополнительных усилий;
- чем больше зависимостей у экземпляра А-блока, тем сложнее вычислить триггер-функцию.

Признак готовности переменной. Каждой переменной $x_j \in M$ в управляющей памяти ставится в соответствие переменная логического типа, которую будем называть *признаком готовности* переменной x_j и обозначать $P(x_j)$. Если x_j – структурная переменная, то каждому компоненту $\overline{x}[j]$ ставится в соответствие собственный признак готовности $P(\overline{x}[j])$. Истинность переменной $P(x_j)$ означает, что переменная x_j вычислена. В начальном состоянии памяти $P(x_j) =$ $=$ ИСТИНА для переменных, содержащих исходные данные задачи и $P(x_j) =$ ЛОЖЬ для всех остальных переменных, $x_j \in M$, $j = \overline{1,m}$.

Управляющий оператор C_k^i присваивает всем переменным $P(x_q)$, где $x_q \in O_{k_{Out}}^i$ значение ИСТИНА, проверяет все экземпляры, которые могут начать исполнение после завершения A_k^i, и добавляет в множество готовых экземпляров те из них, у которых признак готовности всех входных переменных имеет значение ИСТИНА и которые имеют истинное значение триггер-функции.

Достоинства подхода:

- возможность запускать зависимые А-блоки по мере вычисления переменных.

Недостатки подхода:

- чем больше входных переменных, тем сложнее вычислить триггер-функцию (если входная переменная А-блока является структурной, необходимо проверить готовность всех ее компонент);
- объем управляющей памяти зависит от количества переменных;
- реализация прямого управления, когда информационные зависимости между А-блоками отсутствуют, требует дополнительных усилий.

Определение А-системы. Определим для подхода на основе признака завершения А-блока А-систему асинхронной модели вычислений с массовыми операциями через набор правил:

1. В пустое множество готовых экземпляров A' включаются все экземпляры из множества A_0, у которых истинно значение триггер-функции: $A' = \{A_k^i | T_k^i =$ ИСТИНА$, A_k \in A_0\}$.

2. Из множества готовых экземпляров A' выбирается некоторое подмножество A'', экземпляры которого запускаются на исполнение. Для каждого экземпляра А-блока создается процесс t_k^i, который вычисляет его операцию O_k^i, а затем – управляющий оператор C_k^i. В результате в множество A', возможно, будут добавлены новые экземпляры. Каждый экземпляр А-блока выполняется не более одного раза.

3. Выполнение А-программы завершается, когда множество A' становится пустым и не существует исполняющихся процессов.

Алгоритм конструирования управляющих операторов. Для автоматического конструирования управляющих операторов, определенных через признак завершения А-блока, разработан следующий алгоритм:

ВХОД: асинхронная программа P с управлением на основе строгого частичного порядка, экземпляр А-блока A_k^i.

ВЫХОД: код управляющего оператора C_k^i для экземпляра А-блока A_k^i.

ШАГИ:

1. Сгенерировать команду, которая установит признак завершения для A_k^i в значение ИСТИНА.

2. Сформировать множество S экземпляров А-блоков, зависящих от A_k^i:
$S = \{S_l^j | A_k^i < S_l^j, S_l^j \in A, (A_k^i, S_l^j) \in R\}$. В результате исполнения экземпляра A_k^i каждый экземпляр S_l^j потенциально может стать готовым к исполнению.

3. Для каждого экземпляра S_l^j:

- сформировать множество U экземпляров А-блоков, которые должны исполниться до запуска S_l^j:
$U = \{U_q^p | U_q^p < S_l^j, U_q^p \in A, (U_q^p, S_l^j) \in R\}$;
- сгенерировать команду создания переменной с именем $flag\_jl$ и присвоить ей значение ИСТИНА;
- для каждого элемента $U_q^p \in U$ сгенерировать команду, объединяющую значение признака завершения $P(U_q^p)$ со значением переменой $flag\_jl$ по принципу И;
- сгенерировать набор команд, который проверяет значение переменной $flag\_jl$: если оно истинно, проверяет значение триггер-функции T_l^j экземпляра S_l^j; если значение триггер-функции истинно, добавляет экземпляр S_l^j в множество готовых к исполнению экземпляров.

Для правильной работы алгоритма необходимо обеспечить корректную синхронизацию. Действительно, пусть $R = \{(A < C), (B < C)\}$, пусть триггер-функция T А-блока C всегда истина, и пусть исполнение экземпляров A и B находится на стадии исполнения их управляющих операторов, причем уже выполнены команды $P(A) =$ ИСТИНА и $P(B) =$ ИСТИНА. Тогда проверка переменной $flag$ в каждом из управляющих операторов окажется истинной и каждый управляющий оператор добавит А-блок C в множество готовых к исполнению А-блоков, что неверно. Поэтому код, выполняющий изменение признака готовности экземпляра А-блока A_k^i и проверку всех зависимых экземпляров из множества S на предмет их готовности должен быть атомарным с точки зрения остальной программы.

СИСТЕМА ПРОГРАММИРОВАНИЯ АСПЕКТ

Система программирования Аспект [12] на практике реализует идеи асинхронной модели вычислений с управлением на основе строгого частичного порядка и позволяет задать представление алгоритма с необходимой степенью непроцедурности и частичным распределением ресурсов, а затем автоматически сконструировать параллельную программу по этому представлению.

В основе архитектуры системы Аспект лежит ее разделение на два компонента: транслятор с языка Аспект и исполнительную подсистему. Транслятор осуществляет перевод Аспект-программы в асинхронную программу с массовыми А-блоками на языке С++, а исполнительная подсистема обеспечивает поддержку времени выполнения (управление потоками, памятью, очередями готовых А-блоков).

Язык программирования Аспект. Ключевым компонентом системы Аспект является язык программирования Аспект [13]. К его основным характеристикам можно отнести:

1. Явное описание зависимостей между операциями. В императивных языках на множестве операций задается линейный порядок, что ограничивает возможности параллельного исполнения программ. В отличие от них, Аспект позволяет на множестве операций задать частичный порядок.

2. Развитые средства задания управления. Аспект поддерживает различные типы статического управления (простое, сложное, массовое), которое задает статические связи между экземплярами фрагментов вычислений, а также динамическое управление, которое определяется триггер-функциями во время исполнения программы.

3. Частичное распределение ресурсов. Как и в императивных языках, в Аспект допускается повторное присваивание значения переменной. Это означает, что в одной переменной программы могут находиться (в разное время) значения различных переменных алгоритма.

4. Разделение управления и вычислений. В дополнение к обычным библиотекам, накапливающим «хорошие» решения задач, Аспект позволяет создавать библиотеки, накапливающие «хорошие» управляющие конструкции.

5. Ориентация на регулярные структуры данных. Язык программирования Аспект ориентирован на решение задач численного моделирования, поэтому основой организации вычислений являются массовые А-блоки (фрагменты вычислений), позволяющие эффективно обрабатывать регулярные структуры данных, а основой представления данных – массивы.

6. Представление вычислений внутри А-блока с помощью внешнего языка. Для этого может использоваться один из существующих язы-

ков программирования (в настоящее время поддерживается только C++). Такой подход облегчает изучение языка Аспект, позволяет легче переносить существующий код и использовать современные оптимизирующие компиляторы.

Если в императивных языках все команды по умолчанию исполняются последовательно, а участки программы, пригодные для параллельного исполнения, необходимо явно указать, то в языке Аспект используется прямо противоположный подход: по умолчанию считается, что все A-блоки могут исполняться параллельно, а если необходим порядок (например, из-за зависимости по данным), то его необходимо явно задать.

Для иллюстрации возможностей языка Аспект по заданию управления рассмотрим Аспект-программу, решающую задачу умножения матриц (рис. 1).

```
program MultMatrix
preface {
    const int M = 3;
    const int N = 3;
};
data fragments
    double Frg [M][M];
code fragments
    Mult (in Frg X, Frg Y, Frg Z; out Frg Z) {
        for (int i=0; i<M; ++i)
            for (int k=0; k<M; ++k)
                for (int j=0; j<M; ++j)
                    Z[i][j] += X[i][k]*Y[k][j];
    };
task data
    Frg A[N][N];
    Frg B[N][N];
    Frg C[N][N];
task computations
    S[i][j][k]: Mult (A[i][k], B[k][j], C[i][j], C[i][j])
        where i: 0..N -1, j: 0..N -1, k: 0..N -1;
task control
    S[i][j][k] < S[i][j][k+1];
end
```

Рис. 1. Умножение матриц (инициализация данных и вывод результатов опущены)

В программе заданы матрицы A, B и C, состоящие из $N \times N$ элементов. Каждый элемент есть матрица размера $M \times M$ (фрагмент данных Frg). Кроме того, определен фрагмент кода Mult, который получает три матрицы (фрагменты данных X, Y и Z) на вход и добавляет произведение X на Y к выходу (фрагмент данных Z). Вычисления задаются на языке C++.

В секции «task computation» задан массовый A-блок S с областью применимости $(0\text{–}N-1) \times (0\text{–}N-1) \times (0\text{–}N-1)$, при этом имя S[i][j][k] обозначает соответствующий экземпляр S. Каждый экземпляр осуществляет вычисление фрагмента кода Mult с фрагментами данных, указанными для экземпляра в качестве фактических параметров.

В секции «task control» на множестве экземпляров A-блока S задан такой частичный порядок, что каждый k-й экземпляр A-блока должен выполниться перед соответствующим $(k+1)$-м экземпляром. Управление отражает тот факт, что для вычисления C[i][j] необходимо просуммировать все матрицы, полученные в результате вычисления произведения фрагментов A[i][k] на B[k][j] (это делается внутри фрагмента Mult добавлением результата умножения к старому значению C[i][j]), и для исключения «гонки» данных (data race) суммирование необходимо выполнять последовательно.

Все экземпляры A-блока S[i][j][k] с одинаковым индексом k могут исполняться одновременно, при этом, если выполнился экземпляр S[i][j][k], то экземпляр S[i][j][k+1] может стартовать, не дожидаясь завершения исполнения других экземпляров с индексом k, то есть нет барьерной зависимости между «слоями» вычислений.

Результаты тестовых испытаний. В табл. 1 приведены результаты тестовых испытаний Аспект-программы умножения матриц. Тестирование проводилось на машине с 4 процессорами Intel Xeon X7350 (16 ядер) под управлением Cent OS 5.3 64 bit. Размер матрицы брался равным 5040×5040, а размер фрагмента данных – 90×90.

Таблица 1

Параметр	Количество ядер				
	1	2	4	8	16
Время, сек.	148,98	74,73	37,52	18,98	9,68
Ускорение	1	1,99	3,97	7,84	15,39

Аспект-программа демонстрирует практически линейное ускорение, что объясняется блочным представлением алгоритма, за счет которого достигается эффективное использование кэш-памяти, и отсутствием «лишнего» прямого управления, часто мешающего масштабируемости параллельной программы.

Подробное обсуждение результатов тестовых испытаний системы программирования Аспект выходит за рамки статьи. В работе [12] можно найти результаты тестирования системы Аспект при решении задачи LU-разложения, в работе [11] – при решении задачи методом конечных разностей, а в работе [9] – решение прикладных задач методом Монте-Карло и методом «частицы-в-ячейках».

БЛИЗКИЕ РАБОТЫ

Наиболее близким к предложенному подходу с концептуальной точки зрения можно считать программирование с типами управления [14]. Здесь также возможно представление алгоритмов с разной степенью непроцедурности, управление также полностью отделено от вычислений, возможно конструирование сложного управления на основе простых элементов. В то же время массовое управление не поддерживается, что с точки зрения параллельных вычислений является существенным недостатком.

В работе [15] для описания структуры алгоритмов предлагается использовать граф информационных зависимостей, или граф алгоритма. По сути, граф алгоритма – это его максимально непроцедурное представление, ориентированное на описание свойств алгоритма и их анализ. Язык Аспект также позволяет представлять алгоритмы в максимально непроцедурной форме, хотя ориентирован на представление с меньшей степенью непроцедурности, так как оно лучше подходит для автоматического конструирования параллельных программ.

Из реализованных языков для параллельных вычислений наиболее близким к языку Аспект является язык Барс [16]. Как и Аспект, язык Барс имеет развитые средства конструирования управления, разделение управления и вычислений, позволяет записывать выражения в непроцедурной форме. Массовые операции можно реализовать с помощью просачивания унарной операции на данные произвольной структуры (например, массивы), а массовое управление – с помощью бинарного просачивания С-формул, однако механизм формирования структур вычислений для такого просачивания не проработан. В настоящее время проект не развивается. Работоспособная версия для современных суперкомпьютеров также отсутствует.

Весьма интересным является проект PLASMA [17] под руководством Джека Донгарры. В проекте реализуется перспективная идея представления алгоритмов линейной алгебры в виде множества фрагментов, или, в терминологии авторов, «плиток» (tiles), что близко к представлению в виде множества А-блоков. Однако реализуется она в форме библиотеки, то есть конкретной реализации конкретных алгоритмов. Поэтому, в частности, здесь не возникает достаточно сложных вопросов об условных операциях и асинхронных подпрограммах, которые имеются в языке Аспект.

ЗАКЛЮЧЕНИЕ

В статье предложена асинхронная модель вычислений с управлением на основе строгого частичного порядка над общей памятью. Разработан алгоритм преобразования этой модели в известную асинхронную модель вычислений с управляющими операторами. На базе предложенной модели разработана и реализована система программирования Аспект.

Дальнейшее направление исследований связано с разработкой системы синтеза параллельных программ на базе вычислительных моделей, выходным языком которой будет являться язык программирования Аспект.

СПИСОК ЛИТЕРАТУРЫ

1. **Андрианов А. Н., Бугеря А. Б., Ефимкин К. Н., Колударов П. И.** Модульная архитектура компилятора языка Норма+. Препринты ИПМ им. М. В. Келдыша. 2011. № 064. 16 с. [A. N. Andrianov, A. B. Bugerya, K. N. Efimkin, P. I. Koludarov “Modular architecture of Norma+ language compiler”, (in Russian). Preprinty IPM im. M. V. Keldysha, no. 064, 2011.]

2. **Бахтин В. А., Клинов М. С., Крюков В. А., Поддерюгина Н. В., Притула М. Н., Сазанов Ю. Л.** Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Вестник Южно-Уральского государственного университета: серия «Математическое моделирование и программирование». 2012. № 18 (277). Выпуск 12. С. 82–92. [V. A. Bakhtin, M. S. Klinov, V. A. Kryukov, N. V. Podderyugina, M. N. Pritula, Yu. L. Sazanov. “Extension of DVM parallel programming model for clusters with heterogeneous nodes”, (in Russian), Vestnik Yuzhno-Uralskogo gosudarstvennogo universiteta: seriya «Matematicheskoe modelirovanie i programmirovanie», no. 18 (277), issue 12, pp. 82-92, 2012.]

3. **Lastovetsky A.** Adaptive parallel computing on heterogeneous networks with mpC // Parallel Computing. 2002. Vol. 28. No. 10. Pp. 1369–1407. [A. Lastovetsky. Adaptive parallel computing on heterogeneous networks with mpC // Parallel Computing, vol. 28, no. 10, pp. 1369-1407, 2002.]

4. **Kale L. V., Bhatele A.** Parallel Science and Engineering Applications: The Charm++ Approach. CRC Press, 2013. 123 P. [L. V. Kale, A. Bhatele. Parallel Science and Engineering Applications: The Charm++ Approach. CRC Press, 2013.]

5. **Chamberlain B. L., Callahan D., Zima H. P.** Parallel Programmability and the Chapel Language // International Journal of High Performance Computing Applications. 2007. Vol. 21. No. 3. Pp. 291–312. [B. L. Chamberlain, D. Callahan, H. P. Zima Parallel Programmability and the Chapel Language // International Journal of High Performance Computing Applications, vol. 21, no. 3, pp. 291-312, 2007.]

6. **Котов В. Е, Нариньяни А. С.** Асинхронные вычислительные процессы над памятью // Кибернетика. 1966. № 3. С. 64–71. [V. E. Kotov, A. S. “Asynchronous computational precesses on the memory”, (in Russian), Kibernetika, no. 3, pp. 64-71, 1966.]

7. **Вальковский В. А., Малышкин В. Э.** Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука, 1988. 129 с. [V. A. Valkovskiy, V. E. Malyshkin. “Parallel programs and systems synthesis on

the basis of computational models", (in Russian). Novosibirsk: Nauka, 1988.]

8. **Арыков С. Б.** Асинхронная модель вычислений с управлением на основе строгого частичного порядка // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (Архангельск, 28 марта – 1 апреля 2016 г.). Челябинск: Издательский центр ЮУрГУ, 2016. С. 54–67. [S. B. Arykov. "Asynchronous model of computation controlled by strict partial order", (in Russian), in Proc. of Parallelnye vychislitelnye tekhnologii (PaVT'2016), Arkhangelsk, Russia, Mar. 28 – Apr. 1, Publishing of the South Ural State University, pp. 54-67, 2016.]

9. **Арыков С. Б.** Язык и система фрагментированного параллельного программирования задач численного моделирования: Диссертация на соискание ученой степени кандидата физико-математических наук: 05.13.11 / Институт систем информатики им. А.П. Ершова СО РАН. Новосибирск, 2010. 195 с. [S. B. Arykov. Parallel fragmented programming language and system for numerical simulation tasks: PhD Thesis. A.P. Ershov Institute of Informatics Systems SB RAS, Novosibirsk, 2010.]

10. **Арыков С. Б., Малышкин В. Э.** Алгоритмы конструирования асинхронных программ заданной степени непроцедурности методом группировки // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2009. Т. 7. № 1. С. 3–15. [S. B. Arykov, V. E. Malyshkin. "Algorithms of asynchronous programs construction with predefined level of non-procedurality based on grouping method", (in Russian), Vestn. Novosib. gos. un-ta. Seriya: Informatsionnye tekhnologii, vol. 7, no. 1, pp. 3-15, 2009.]

11. **Арыков С. Б.** Асинхронное программирование численных задач // Параллельные вычислительные технологии (ПаВТ'2010): труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.). Челябинск: Издательский центр ЮУрГУ, 2010. С. 28–39. [S. B. Arykov. "Asynchronous programming of numerical tasks", (in Russian), in Proc. of Parallelnye vychislitelnye tekhnologii (PaVT'2010), Ufa, Russia, March 29 – April 2, Publishing of the South Ural State University, pp. 28-39, 2010.]

12. **Arykov S. B., Malyshkin V. E.** Asynchronous Language and System of Numerical Algorithms Fragmented Programming // Proc. of the 10th International Conference on Parallel Computing Technologies (PaCT'2009), (Novosibirsk, Aug. 31-Sep. 4). Berlin: Springer-Verlag, 2009. LNCS 5698. Pp. 1–7. [S. B. Arykov, V. E. Malyshkin. Asynchronous Language and System of Numerical Algorithms Fragmented Programming, in Proc. of the 10th International Conference on Parallel Computing Technologies (PaCT'2009), Novosibirsk, Russia, Aug. 31-Sep. 4, LNCS 5698, pp. 1-7, 2009.]

13. **Арыков С. Б.** Язык программирования Аспект // Известия Томского политехнического университета. 2008. Т. 313. № 5. С. 89–92. [S. B. Arykov. "Programming Language Aspect", (in Russian), Izvestiya Tomskogo politekhnicheskogo universiteta, vol. 313, no. 5, pp. 89-92, 2008.]

14. **Котов В. Е.** Параллельное программирование с типами управления // Кибернетика. 1979. № 3. С. 1–13. [V. E. Kotov. "Parallel programming with control types", (in Russian), Kibernetika, no. 3, pp. 1-13, 1979.]

15. **Воеводин В. В., Воеводин Вл. В.** Параллельные вычисления. СПб.: БХВПетербург, 2002. 608 с. [V. V. Voevodin, Vl. V. Voevodin. "Parallel computing", (in Russian). St. Petersburg: BKhVPeterburg, 2002.]

16. **Быстров А. В., Дудоров Н. Н., Котов В. Е.** О базовом языке // В кн. Языки и системы программирования. Новосибирск: ВЦ СО АН СССР, 1979. С. 85–106. [A. V. Bystrov, N. N. Dudorov, V. E. Kotov. "About the core language", (in Russian), V kn. Yazyki i sistemy programmirovaniya. Novosibirsk: CC SB AS USSR, pp. 85-106, 1979.]

17. **Buttari A., Langou J., Kurzak J., Dongarra J.** A class of parallel tiled linear algebra algorithms for multicore architectures // Parallel Computing. 2009. Vol. 35. No. 1. Pp. 38–53. [A. Buttari, J. Langou, J. Kurzak, J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Computing, vol. 35, no. 1, pp. 38-53, 2009.]

ОБ АВТОРЕ

АРЫКОВ Сергей Борисович, мл. науч. сотр. Дипл. инж.-программист (Новосиб. гос. ун-т, 2004). Канд. физ.-мат. наук по спец. 05.13.11 (ИСИ СО РАН, 2010). Иссл. в обл. параллельных вычислений.

METADATA

Title: Asynchronous model of computation on shared memory.

Authors: S. B. Arykov

Affiliation:
Institute of Computational Mathematics and Mathematical Geophysics Siberian Branch of the Russian Academy of Sciences (ICMMG SB RAS), Russia.

Email: arykov@mail.ru.

Language: Russian.

Source: Vestnik UGATU (scientific journal of Ufa State Aviation Technical University), vol. 20, no. 3 (73), pp. 114-121, 2016. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: The article proposes an asynchronous model of computation, the distinguishing feature of which is computation control based on strict partial order. It's shown that such approach to define control in the program provides several advantages for parallel computing, in particular, allows to represent algorithms with high level of non-procedurality and completely separates control from computations. An algorithm of convertation of that model to well-known asynchronous model with control operators is developed. Aspect programming system was designed and developed on the basis of the proposed model.

Key words: asynchronous model of computations, parallel programming language Aspect, algorithms representation with high non-procedurality.

About author:

ARYKOV, Sergey Borisovich, Junior Researcher. Dipl. Software Engineer (Novosibirsk State University, 2004). PhD in Computer Science (IIS SB RAS, 2010).