УДК 004.3

ЖАБИНА В.В.

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ДАННЫХ НА УРОВНЕ ОПЕРАЦИЙ В ПОТОКОВЫХ СИСТЕМАХ

В статье предложен метод ускорения вычислений на уровне параллельной обработки машинных слов в системах, управляемых потоком данных. Рассмотрена структура системы, позволяющая одновременное формирование и выполнение нескольких команд. Показана возможность автоматической идентификации слов акторов и данных на основе графа задачи.

In the article the method of acceleration of calculations at the level of the simultaneous processing of computer words in the systems, guided the flow of data is offered. The structure of the system, allowing the simultaneous forming and execution of a few instructions, is considered. Possibility of automatic identification of words of actors and dates on the basis of graph of task is shown.

Введение

Время решения параллельных задач во многом определяется эффективностью распараллеливания процессов с целью максимальной загрузки ресурсов вычислительной системы.

Большинство из современных технологий параллельного программирования относятся к средствам статического распараллеливания процессов. Задачи распараллеливания в этом случае решаются на этапе разработки программ, причем, их эффективность во многом определяется квалификацией программиста [1].

К оновным недостаткам средств статического распараллеливания следует отнести следующее.

При анализе алгоритмов не всегда удается выявить параллельные ветви, то есть не всегда можно выявить скрытый параллелизм. Это обусловлено целым рядом причин, к основным из которых можно отнести недостаток информации о динамике процессов.

При разработке программ, как правило, учитывается конфигурация аппаратных средств системы, изменение которой может потребовать повторной разработки программы.

Одним из перспективных подходов, позволяющих устранить ряд недостатков статического планирования, является разработка средства динамического распараллеливания вычислений. В этом случае назначение заданий на вычислительные узлы осуществляется системой автоматически в процессе решения задач. Такой подход дает возможность достичь большей степени параллелизма, так как позволяет выявить параллельные ветви, которые возникают непосредственно в процессе вычислений.

Поскольку динамическое распределение заданий осуществляется средствами самой системы, то важной задачей является уменьшение на это расходов времени в процессе обработки информации.

Модель вычислений, управляемых потоком данных

Для решения проблемы динамического распределения заданий между вычислительными ресурсами системы предложены модели вычислений, управляемых потоком данных. К наиболее ранним можно отнести работы [2-6].

В потоковых системах отсутствует необходимость решения задачи динамического распределения заданий между вычислительными узлами на программном уровне. Распределение работ между вычислителями осуществляется автоматически на аппаратном или микропрограммном уровне.

В системах, управляемых потоком данных, команды выполняются не в заданной программой последовательности, а при наличии всех операндов, то есть определяющим в данном случае является не порядок выполнения команд, а доступность данных для команды.

Будем рассматривать алгоритмы с мелкозернистой структурой, при реализации которых распараллеливание осуществляется на уровне выполнения отдельных команд, то есть на уровне обработки машинных слов. Для определенности будем рассматривать двуместные операции, что соответствует системам команд большинства современных процессо-

ров. Одноместные операции приводятся к двуместным добавления фиктивного операнда, который не участвует в вычислениях.

Потоковая система должна содержать вычислительные модули (ВМ), среду формирования команд (СФК), устройства ввода (УВв) и вывода (Увыв) данных. Компоненты системы связаны между собой через соответствующие коммуникационные средства.

Подготовка вычислений осуществляется на основе графа, каждой i-й вершине которого соответствует операция, а каждой дуге – операнд.

Операция для i-й вершины графа описывается информационным словом, которое называют актором (actor). Актор описывается кортежем

$$A_i = \langle I_i, F_i, N_i, T_i \rangle, \qquad (1)$$

где I_i – идентификатор (уникальное имя) данного актора; F_i – функция преобразования данных (код операции); N_i – имя актора, для которого i-й актор подготавливает операнд, а T_i – совокупность признаков этого операнда.

Акторы связаны между собой только по данным. Каждой дуге графа соответствует слово данных

$$D_i = \langle I_i, Q_i, N_i, T_i \rangle, \qquad (2)$$

где Q_i – значение операнда.

Из соответствующих элементов A_i и D_i в СФК формируется команда, которая выполняется в свободном ВМ или помещается в очередь. Известны различные алгоритмы формирования команд [2-5]. Для организации СФК используется ассоциативная память или ее эмуляция с применением других технических средств.

Компиляторы позволяют автоматизировать подготовку акторов и данных на базе графа, причем, без учета конкретного числа ВМ в системе.

Если время формирования команд при наличии готовых ее компонентов существенно меньше времени выполнения команд в ВМ, то в разных ВМ одновременно выполняются разные команды, за счет чего и достигается распараллеливание операций. Современные технологии реализации ВМ позволяют использовать аппаратные методы ускорения операций, в результате чего зачастую интенсивность формирования команд в СФК становится недостаточной для загрузки нескольких ВМ одновременно. Покажем это на примере реализации графа, изображенного на рис. 1. Считаем, что время формирование команды в СФК равно времени выполнения команды в ВМ.

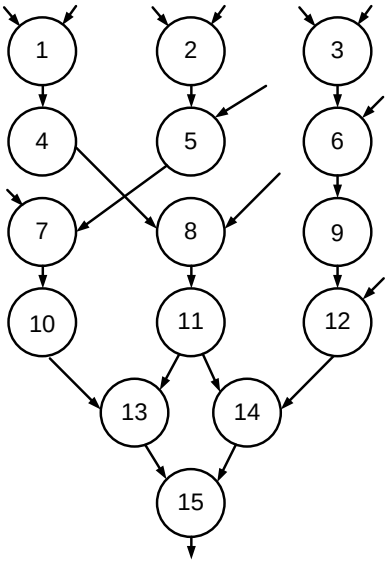

Рис. 1. Граф задачи

Возможные временные диаграммы занятости СФК и ВМ без учета времени пересылки данных показаны на рис. 2. Номера вершин графа соответствуют номерам интервалов на диаграмме. Команда может начать выполняться в ВМ только после ее формирования в СФК. Команда назначается на свободный ВМ.

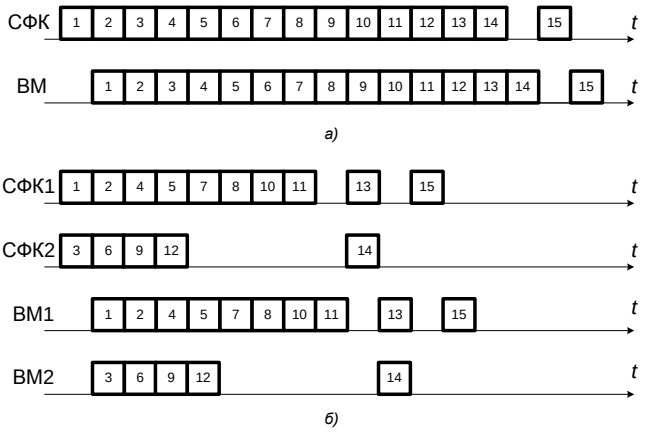

Рис.2. Временная диаграмма занятости структурных компонентов системы

Как видно из рис. 1 *а*, при наличии одной СФК достаточно иметь в системе только один ВМ, который успевает обрабатывать поток команд, формируемых в СФК. Интенсивности потока команд не хватает для загрузки второго ВМ.

Две СФК позволяют загрузить второй ВМ (рис.1 *б*). Очевидно, что время реализации вычислений при этом уменьшается.

В связи с этим важной проблемой является увеличение интенсивности потока готовых

команд с целью ускорения параллельных вычислений.

Тривиальным подходом является дублирование потоковых процессоров, но при этом возникают недостатки, присущие статическим методам подготовки параллельных программ. Программист должен предварительно разрезать граф задачи на подграфы, предопределить идентификаторы акторов и данных в разных подграфах с учетом связи по данным между ними.

Более эффективным является подход, который позволяет автоматически формировать акторы и данные при наличии нескольких СФК [6, 7]. Система содержит модули, в состав которых входит СФК и ВМ. Модули организуются в кольцевую структуру. Данные циркулируют в такой структуре в поисках своего актора. Команды формируются в разных СФК и выполняются в соответствующих ВМ.

Недостатком такого подхода являются затраты времени на пересылку данных между модулями системы. Кроме того, выход из строя любого модуля приводит к неработоспособности всей системы, поскольку информация передается последовательно от одного модуля к другому.

Ниже предлагается метод параллельного формирования команд в разных СФК, который не требует ручного вмешательства в процесс назначения идентификаторов, а также реализации последовательного обмена данными непосредственно между СФК.

Концепция формирования параллельных потоков команд

Подготовку задачи для случая параллельного формирования различных потоков команд рассмотрим на примере структуры системы, показанной на рис. 3, где КС – коммуникационная среда.

Система может иметь любое число Увв, Увыв и ВМ. Количество СФК должно быть равно $k = 2^j$, где $j = 1,2,3,...$

Модифицируем объекты (1) и (2) следующим образом:

$$A_i = \langle M_i, I_i, F_i, N_i, T_i \rangle, \quad (3)$$

$$D_i = \langle M_i, I_i, Q_i, N_i, T_i \rangle, \quad (4)$$

где M_i – идентификатор СФК, который назначается компилятором автоматически с учетом связей акторов по данным.

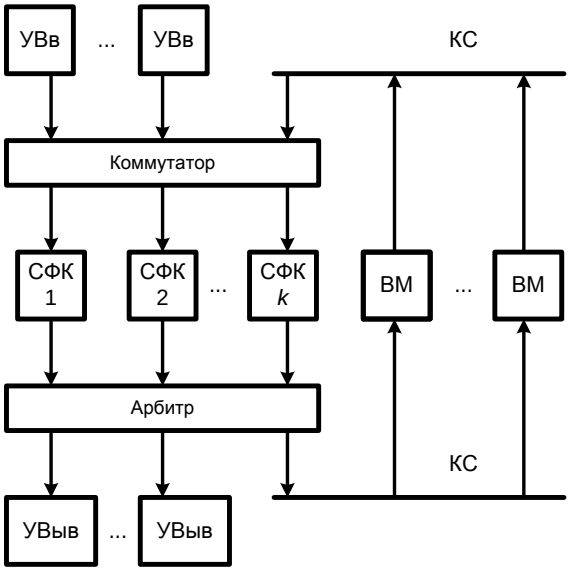

Рис.3. Структура потоковой системы

Алгоритм нахождения M_i представлен ниже.

1. В графе задачи вершинам, которые соответствуют одноместным операциям, добавить входную дугу с фиктивным операндом γ.

2. Используя известные в теории компиляторов алгоритмы, получить префиксную (постфиксную) бесскобочную форму записи вычисляемого результата в виде строки.

3. Просматривая полученную строку выделить все пары имен операндов вместе с именем соответствующего актора (выполняемой операции). В результате прохода выписать имена всех акторов, которые могут выполняться параллельно во времени. Далее рассматривать эти тройки объектов в качестве операндов для следующего прохода. Повторять аналогичные проходы до полного сворачивания строки, выписывая после каждого прохода имена акторов, которые могут выполняться одновременно.

4. Составить цепочки имен акторов следующим образом. В первую цепочку входят все акторы, полученные последними при каждом проходе (см. п.3), во вторую цепочку – предпоследними и т.д.

5. Первой цепочке назначить 0-й номер СФК, второй – 1-й номер и т.д. Полученные номера СФК присваивают акторам (3) и данным (4) в качестве имен M_i.

Команды, соответствующие разным цепочкам, формируются в разных СФК, за счет чего

создаются параллельные потоки команд в системе.

Некоторые вопросы, связанные, например, с реализацией циклических алгоритмов, а также разветвлением данных выходят за рамки данной статьи.

Заметим, что в теории компиляторов бесскобочные формы применяют для формирования последовательности операций с использованием стека. В рассматриваемом случае на основе такой записи определяются операции, которые могут выполняться параллельно во времени.

Пример автоматического назначения идентификаторов

Рассмотрим граф (рис. 4), в котором номера вершин соответствуют именам акторов, то есть определяют выполняемую операцию. Исходные данные заданы буквами.

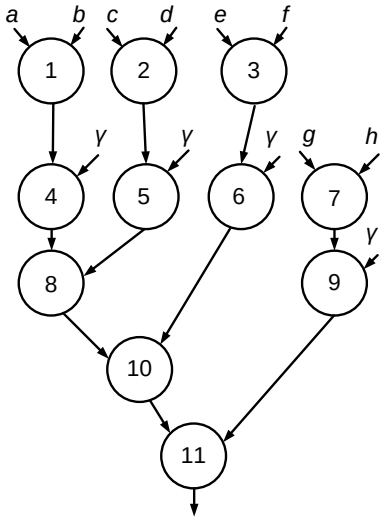

Рис. 4. Граф задачи

В соответствии с п.1 приведенного алгоритма вершинам 4, 5, 6 и 9 с одноместными операциями приписаны фиктивные операнды γ.

Методом обхода графа, начиная с вершины 11, формируем в соответствии с п.2 строку

$$11\,9\,\gamma\,7\,h\,g\,10\,6\,\gamma 3\,f\,e\,8\,5\gamma\,2\,d\,c\,4\gamma 1\,b\,a$$

Согласно п.3 выполняем пять проходов.

1-й проход.

Операции: $7hg$; $3fe$; $2dc$; $1ba$.

Акторы: 7, 3, 2, 1.

2-й проход.

Операции: $9\gamma-$; $6\gamma-$; $5\gamma-$; $4\gamma-$.

Акторы: 9, 6, 5, 4.

3-й проход.

Операция $8--$.

Актор 8.

4-й проход.

Операция $10--$.

Актор 10.

5-й проход.

Операция $11--$.

Актор 11.

Прочерками обозначены операнды, которые соответствуют результатам операций предыдущих проходов.

В соответствии с п.4 и п.5 составляем цепочки операций и присваиваем им номера M_i, начиная с нулевого.

Номер M_i=0 назначается для команд 1, 4, 8, 10, 11, M_i=1 – для команд 2 и 5, M_i=2 – для команд 3 и 6, M_i=3 – для команд 7 и 9.

При наличие в системе четырех ВМ параллельно могут выполняться четыре потока команд. Если в системе два ВМ, то команды с четными номерами будут выполняться в одном ВМ, а с нечетными – в другом. Функцию распределения данных между разными СФК выполняет коммутатор. Таким образом, при подготовке задачи нет необходимости учитывать число ВМ в системе.

Выводы

Предложенный метод формирования параллельных потоков команд в системах, управляемых потоком данных, имеет ряд преимуществ как по сравнению с классическими параллельными системами, так и по сравнению с кольцевыми потоковыми структурами.

По сравнению с традиционной моделью вычислений существенно упрощается процесс подготовки задач. Граф задачи может быть составлен весьма просто с использованием графического редактора. Нет необходимости учитывать число вычислителей в системе и длительность выполнения команд. Управляющие слова и данные могут быть сформированы автоматически компилятором на основе графа. Динамическое распределение операций позволяет выявить непосредственно в процессе вычислений скрытый параллелизм, связанный с различными длительностями об-

работки данных в различных ветвях алгоритмов.

Вычислительные модули могут выполнять операции, относящиеся к различным задачам, в любой последовательности. В связи с этим в системе могут одновременно решаться независимые задачи, причем, нет необходимости синхронизации задач, что позволяет начинать решение новой задачи в любой момент времени.

По сравнению с кольцевой организацией потоковых систем предложенный метод позволяет ускорить вычисления за счет сокращения непроизводительных затрат времени на последовательную пересылку данных между модулями системы. Кроме того, это способствует повышению надежности систем.

Все это повышается эффективность обработки данных в потоковых системах.

Список литературы

1. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2004. – 608 с.
2. Dennis J. B., Missunas D. P. A preliminary architecture for basic data flow processor// Proc. 2nd Annual Symp. Comput. Stockholm, May 1975. N. Y. IEEE. – 1975. – P. 126 – 132.
3. Silva J.G.D., Wood J.V. Design of processing subsystems for Manchester data flow computer // IEEE Proc. N.Y. – 1981. – Vol. 128, N 5. – P. 218 – 224.
4. Watson R., Guard J. A practical data flow computer // Computer. – 1982. – Vol. 15, N 2.– P. 51 – 57.
5. Hogenauer E.B., Newbold R.F. Inn Y.T. DDSP – a data flow computer for signal processing/ Proc. Int. Conf. Parall. Process. Ohio, August 1982. N.Y. // IEEE. – 1982. – P. 126 – 133.
6. Johnson D. Data flow machines threaten the program counter// Electronic Design. – 1980. – N 22. – P. 255 – 258.
7. Функционально ориентированные процессоры / Водяхо А.Н., Смолов В.Б., Плюснин В.У., Пузанков Д.В. / Под ред. В.Б.Смолова. – Л.: Машиностроение. Ленингр. отд-ние, 1988. – 224 с.