УДК 681.5

Жабин В. И., канд. техн. наук

ОБЕСПЕЧЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ МУЛЬТИПРОЦЕССОРНЫХ СИСТЕМ ПРИ РЕАЛИЗАЦИИ ВЫЧИСЛЕНИЙ ПОД УПРАВЛЕНИЕМ ДЕСКРИПТОРОВ

Национальный технический университет Украины «КПИ»

Рассматриваются вопросы обеспечения отказоустойчивости мультипроцессорных систем при динамическом распределении работ между процессорами. Предлагается метод автоматического исправления ошибок, возникающих вследствие отказа процессоров.

Введение

Продолжительность решения задач в параллельных вычислительных системах во многом определяется возможностью распараллеливания процессов с целью максимальной загрузки вычислительных узлов. Для разработки параллельных программ, созданы различные средства программирования. Кроме параллельных языков, таких как *Ada*, *Java* и ряд других, которые являются самостоятельными средствами для разработки параллельных программ, находят применение различные расширения последовательных языков (например, *mpC*, *C-DVM*, *HPF*, *Fortran-DVM*), а также библиотеки параллельного программирования (*OpenMP*, *MPI*, *PVM* и др).

Большинство известных технологии относятся к средствам статического распараллеливания процессов. Основные задачи распараллеливания в этом случае решаются программистом на этапе разработки программ. Статическое распараллеливание может выполняться автоматически с помощью специальных компиляторов, хотя их возможности весьма ограничены.

Общий недостаток средств статического распараллеливания состоит в том, что при анализе алгоритмов не всегда удается выявить параллельные ветви. Это обусловлено целым рядом причин, к основным из которых можно отнести недостаток информации о динамике процессов.

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

Серьезные проблемы возникают при многозадачном режиме функционирования систем. Взаимодействие процессов может быть предусмотрено для задач, зарегистрированных предварительно в одной группе (коммуникаторе). Обеспечить это весьма проблематично, например, при решении задач управления в реальном времени, когда стратегия управления может изменяться под воздействием различных факторов.

Одним из перспективных подходов, позволяющих устранить ряд недостатков статического планирования и упростить подготовку параллельных вычислений, является разработка средства динамического распараллеливания процессов. Распределение работ между вычислительными узлами в этом случае осуществляется системой автоматически в процессе решения задач. Реализация данного подхода позволяет выявить параллельные ветви, которые возникают непосредственно в процессе вычислений.

Поскольку задачи динамического распределения заданий между вычислительными узлами решаются средствами самой системы, то это приводит к дополнительным расходам времени и ресурсов системы. Уменьшение таких расходов является одной из важнейших задач повы-

шения эффективности параллельных систем.

С целью уменьшения затрат времени на динамическое распределение процессов были предложены модели вычислений под управлением потоков данных [1, 2]. Однако с увеличением зернистости существенно возрастают объемы пересылаемых данных, что делает такую модель вычислений неэффективной.

В работах [3-6] предложены методы динамического распараллеливания процессов, позволяющие повысить эффективность реализации параллельных алгоритмов с крупнозернистой структурой. Распределение заданий между процессорами осуществляется автоматически под управлением потока дескрипторов данных и заданий.

В данной работе предлагается модификация метода, направленная на обеспечение автоматического исправления ошибок, вызванных отказом процессоров в мультипроцессорных систем.

Характеристика системы

Мультипроцессорная вычислительная система содержит вычислительные модули (ВМ), в качестве которых могут использоваться универсальные и специализированные процессоры. ВМ обмениваются данными через общее адресное пространство. При этом каждый ВМ может иметь локальную память, позволяющую выполнять собственные программы независимо от других ВМ.

Метод параллельных вычислений с исправлением ошибок

В основу метода положены следующие теоретические положения.

Вычислительный процесс представляется с помощью графа $\boldsymbol{G}=(\boldsymbol{W},\boldsymbol{D})$, где $\boldsymbol{W}$ – множество вершин, $\boldsymbol{D}$ – множество дуг. Каждой i-й вершине $\boldsymbol{W}_i$ $(i=\overline{1,m})$ соответствует задание (определенный объем работы, процесс), а каждой дуге $\boldsymbol{D}_{ji}$ – поток данных, необходимых для выполнения задания. Исходные данные для каждой задачи подготавливаются на основании графа и представляют собой набор дескрипторов заданий и данных, а также библиотеку реализации функций. В качестве компонентов библиотеки могут использоваться программные модули, функционирующие в заданной операционной среде и позволяющие осуществить необходимое для решения задачи преобразование данных.

Задание для i-й вершины графа описывается дескриптором задания

$$\boldsymbol{W}_i=\{N_i,I_i,\boldsymbol{P}_i,Q_i\},$$

где N_i – имя данного дескриптора; I_i – идентификатор задания (определяет функцию преобразования данных); $\boldsymbol{P}_i=\{p_{ij}\}$ – множество имен выходных данных (соответствуют дугам, выходящим из i-й вершины и входящим в j–ю вершину); Q_i – суммарное число входных потоков данных для i–го задания (число дуг графа, входящих в i-ю вершину). Имя потока выходных данных p_{ij} может быть представлено кортежем

$$p_{ij}=\langle N_j, n_{ij}, Q_j\rangle,$$

где n_{ij} – имя входа j-й вершины графа, соответствующего дуге D_{ij}.

Поток данных, соответствующий дуге графа, соединяющей j-ю вершину с i-й вершиной, характеризуется дескриптором данных

$$\boldsymbol{D}_{ji}=\{p_{ji}, A_{ji}\}=\{N_i, n_{ji}, Q_i, A_{ji}\},$$

где A_{ji} – элемент адресации данных, определяющий расположение данных в памяти системы.

Все вершины графа имеют уникальные имена N_i $(i=\overline{1,m})$.

Из элементов дескрипторов в соответствии с определенной процедурой F формируются заявки на выполнение i-го задания

$$F(\boldsymbol{W}_i, \boldsymbol{D}_i)\rightarrow \boldsymbol{Z}_i=\{I_i, \boldsymbol{P}_i, A_i, \boldsymbol{M}_i\},$$

где $\boldsymbol{A}_i=\{A_{ji}\mid j\in \boldsymbol{J}\}$ – множество элементов адресации данных для i-го задания; $\boldsymbol{D}_i=\{\boldsymbol{D}_{ji}\mid j\in \boldsymbol{J}\}$ – множество дескрипторов данных для задания $\boldsymbol{W}_i$; $\boldsymbol{J}$ – множество имен вершин графа, связанных выходящими дугами с i-й вершиной; $\boldsymbol{M}_i$ – маска, определяющая, какие дескрипторы вы-

ходных данных необходимо формировать при повторном выполнении задания.

Пусть, например, задание $\boldsymbol{W_a}$ в соответствии с графом задачи формирует данные для заданий $\boldsymbol{W_b}$,$\boldsymbol{W_c}$ и $\boldsymbol{W_d}$. Тогда в процессе выполнения задания $\boldsymbol{W_a}$ в строках S_b, S_c и S_d в определенной последовательности будут устанавливаться признаки δ_{ab}, δ_{ac} и δ_{ad}. Анализ признаков позволяет для задания $\boldsymbol{W_a}$ сформировать маску $\boldsymbol{M_i}=\{\delta_{ab}, \delta_{ac}, \delta_{ad}\}$, элементы (разряды) которой указывают, были ли получены соответствующие данные в управляющем ВМ.

Для формирования заявок $\boldsymbol{Z_i}$ может быть использован любой из ВМ. Такой ВМ будем называть управляющим, а ВМ, которые выполняют заявки – исполняющими.

Функции управляющего модуля могут быть возложены на специализированное устройство, которое является общим системным аппаратным ресурсом.

Сложность формирования заявок состоит в том, что данные для i–го задания формируются в разные моменты времени и, кроме того, возможно разными ВМ.

Условие активизации заявки можно представить как конъюнкцию

$$\omega_i = \nu_i \cdot \mathop{\&}_{j} \delta_{ji},$$

где ν_i – условие получения дескриптора i–го задания; δ_{ji} – условие получения элемента адресации A_{ji}.

Учитывая уникальность имен N_i дескрипторов для каждого задания, условие активизации задания можно также представить в виде

$$\omega_i = \begin{cases} 1, \text{ если } C_i = Q_i; \\ 0, \text{ если } C_i \neq Q_i, \end{cases}$$

где C_i – число принятых в управляющий ВМ дескрипторов с именем N_i.

Второй вариант определения готовности данных для формирования заявки на выполнение задания более удобен для аппаратурной реализации, так как подсчет дескрипторов можно выполнять последовательно во времени по мере их поступления в ВМ.

В управляющем ВМ создается таблица (массив объектов), i-я строка S_i которой имеет вид

$$N_i : S_i = \langle I_i, \boldsymbol{P}_i, \boldsymbol{A}_i, \boldsymbol{L}_i, C_i, Q_i \rangle,$$

где N_i – имя строки; C_i –счетчик дескрипторов, $\boldsymbol{L}_i=\{\delta_{ji}\}$ – признаки наличия поступивших дескрипторов данных. Число признаков в строке S_i равно числу входных потоков данных Q_i.

В управляющий ВМ вводятся дескрипторы и исходные данные. При поступлении дескрипторов заданий в соответствующие позиции строк таблицы вводятся значения I_i и $\boldsymbol{P}_i$. При поступлении дескрипторов данных в соответствующие позиции строк записываются элементы множества $\boldsymbol{A}_i$ и соответствующие им признаки δ_{ji}. Позиция элементов адресации определяется значениями n_{ji}, присутствующими в дескрипторах заданий.

Полное накопление множества A_i элементов адресации данных для задания определяется с помощью счетчика C_i. При поступлении любого дескриптора сравниваются значения Q_i и C_i, затем C_i увеличивается на единицу. Равенство указанных значений является условием ω_i активизации заявки.

Заявка передается в исполняющий ВМ по его запросу. При этом обнуляется счетчик C_i, то есть система готова для повторного формирования заявки с данным именем, если это необходимо.

В системе одновременно могут решаться несколько задач. Любой ВМ может выполнять функции управляющего и исполняющего модуля (в том числе, одновременно). Свободные ВМ обращаются к управляющим ВМ за заявками. После выполнения задания управляющему ВМ возвращается дескриптор данных, который принимает участие при формировании новой заявки.

На стадии формирования заявок между модулями пересылаются только короткие сообщения (дескрипторы). Потоки непосредственно данных перемещаются Взаимодействие по данным между

таблицей формирования заявок, СГЗ и СВЗ показано на рис. 2.только на этапе выполнения заданий. В этом случае вычислительным процессом управляют стандартные средства операционной системы.

Поскольку подготовка задач в соответствии с рассматриваемым методом осуществляется без учета числа ВМ, то появляется потенциальная возможность продолжения вычислений при отказе процессоров до тех пор, пока в системе останется хотя бы один исправный ВМ. В последнем случае ВМ, конечно, должен выполнять функции управляющего и исполняющего.

Рассмотрим реализацию указанной возможности при однократном отказе ВМ (по крайней мере, на продолжении цикла восстановления системы).

Под отказом ВМ будем понимать прекращение этим модулем циклов обращения к общей памяти.

Общую память можно защитить аппаратурными методами повышения надежности, которые достаточно хорошо разработаны. Не вызывает особых затруднений и проверка правильности прохождения линейных программ в управляющем ВМ известными методами контроля.

Устранение последствий отказов исполняющих ВМ может быть возложено на управляющий ВМ. Такой подход позволяет реализовать один из основополагающих принципов обеспечения живучести систем – принцип расширяемого ядра, защищенного методами повышения надежности [7].

В случае отказа исполняющего ВМ во время выполнения задания управляющий ВМ должен обеспечить инициализацию повторного выполнения задания в работоспособном ВМ. Следует учитывать, что до отказа ВМ часть дескрипторов данных могла быть передана в управляющий ВМ. В этом случае повторное выполнение задания продублирует уже полученные дескрипторы, что приведет к неправильной модификации счетчиков C_i в некоторых строках таблицы формирования заявок, то есть к ошибке вычислений.

Для обеспечения отказоустойчивости ВМ должны обеспечивать ряд специальных функций.

Специальные функции в режиме исполняющего ВМ. В процессе выполнения задания исполняющий ВМ возвращает управляющему ВМ в общем случае несколько дескрипторов данных D_{ji} согласно маске $\boldsymbol{M}_i$ и числу элементов в множестве $\boldsymbol{P}_i$. После успешного завершения задания исполняющий ВМ должен переслать управляющему ВМ признак E_i окончания задания.

Специальные функции в режиме управляющего ВМ. Управляющий ВМ должен выявлять факт неисправности исполняющего ВМ и формировать маску для повторного выполнения задания.

Заявка снабжается маской $\boldsymbol{M}_i$, которая формируется процедурой

$$G(\boldsymbol{P}_i, \delta_{ri} \mid r \in \boldsymbol{R}) \rightarrow \boldsymbol{M}_i = \{ \overline{\delta_{ri}} \},$$

где $\boldsymbol{R}$ – множество имен вершин графа, связанных входящими дугами с i–й вершиной.

Разряды маски определяют, какие выходные потоки данных необходимо формировать при выполнении заданий. При первом выполнении задания формируются все выходные данные, а при повторном (вследствие отказов ВМ) – выходные потоки данных, которые еще не были переданы в управляющий ВМ.

Механизм исправления ошибки заключается в следующем.

После получения полной информации для формирования очередной заявки $\boldsymbol{Z}_i$ ее имя N_i^* заносится в *список имен готовых заявок* (СГЗ). Заявки передаются исполняющим ВМ только из указанного списка.

Имя выданной на выполнение заявки переписывается из СГЗ в *список имен выполняемых заявок* (СВЗ). При поступлении признака E_i заявка Z_i считается выполненной, при этом ее имя N_i^{**} удаляется из СВЗ.

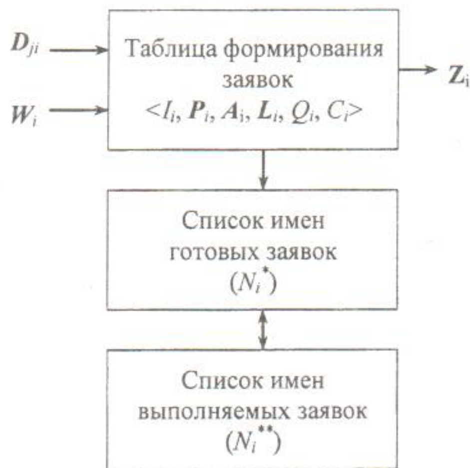

Рис. 1. Схема взаимодействия объектов

При решении задачи пустой список готовых заявок может указывать на различные ситуации в системе. Первая ситуация заключается в том, что выполнение заявок в исполняющих ВМ не закончено и дескрипторы данных для формирования очередных заявок еще не поступили, но будут получены через определенный промежуток времени. Вторая ситуация связана с отказом исполняющего ВМ. В этом случае вычислительный процесс не может продолжаться.

Возможным подходом к распознаванию указанных вариантов может служить длительность интервала ожидания поступления дескрипторов из исполняющего ВМ. Для этого можно использовать таймер, запрограммированный на интервал времени срабатывания, длительность которого связана с максимальным временем выполнения заданий.

Если после срабатывания таймера дескриптор данных не поступил, то произошел отказ ВМ, то есть необходимо повторное выполнение задания, которое формирует ожидаемый дескриптор данных. Имя задания, которое не завершено, в данном случае находится в списке исполняемых заявок. Для повторного выполнения задания достаточно переписать его имя из таблицы исполняемых заявок в таблицу готовых заявок. По запросу исправного ВМ заявка будет выдана на повторное исполнение с модифицированной маской. В соответствии с единичными разрядами маски будут формироваться только те данные, которые ранее не были переданы в управляющий ВМ.

Пример исправления ошибки. Пусть задан граф умножения матриц X и Y, показанный на рис. 2 (обозначения дуг и входов вершин показаны выборочно, чтобы не затенять рисунок). В качестве имен вершин (N1 – N9) воспользуемся их номерами от 1 до 9.

Функция *read* производит чтение данных из памяти. С помощью функции *split* осуществляется разделение матрицы по строкам на фрагменты. Перемножение фрагментов матриц выполняется функцией *mul*. Функция *comb* объединяет результаты умножения и формирует окончательный результат, который с помощью функции *write* записывается в память.

Дескрипторы данных и заданий представлены соответственно в табл. 1 и 2.

Таблица 1

Дескрипторы данных

D_{ji}	N_i, n_{ji}, Q_i	A_{ji}
D_{01}	1,1,1	A_{01}
D_{03}	3,1,1	A_{03}

Пусть в результате отказа ВМ задание W_3 выполнено не полностью. Для определенности будем считать, что не сформированы данные D_{35} и D_{57}. В результате этого вычислительный процесс продолжаться не может. Все данные, которые на данный момент не поступили в управляющий ВМ, отмечены на рис. 2 звездочками. Состояние процесса формирования заявок иллюстрируется табл. 3, список имен готовых заявок пуст, а список имен выполняемых заявок содержит имя N_3, которое соответствует невыполненной заявке W_3.

Таблица 2

Дескрипторы заданий

W_i	N_i	I_i	P_i				Q_i
W_1	1:	*read*	2,1,1				1
W_2	2:	*split*	4,1,2	5,1,2	6,1,2	7,1,2	1
W_3	3:	*read*	4,2,2	5,2,2	6,2,2	7,2,2	1
W_4	4:	*mul*	8,1,4				2
W_5	5:	*mul*	8,1,4				2
W_6	6:	*mul*	8,1,4				2
W_7	7:	*mul*	8,1,4				2
W_8	8:	*comb*	9,1,1				4
W_9	9:	*write*	–				1

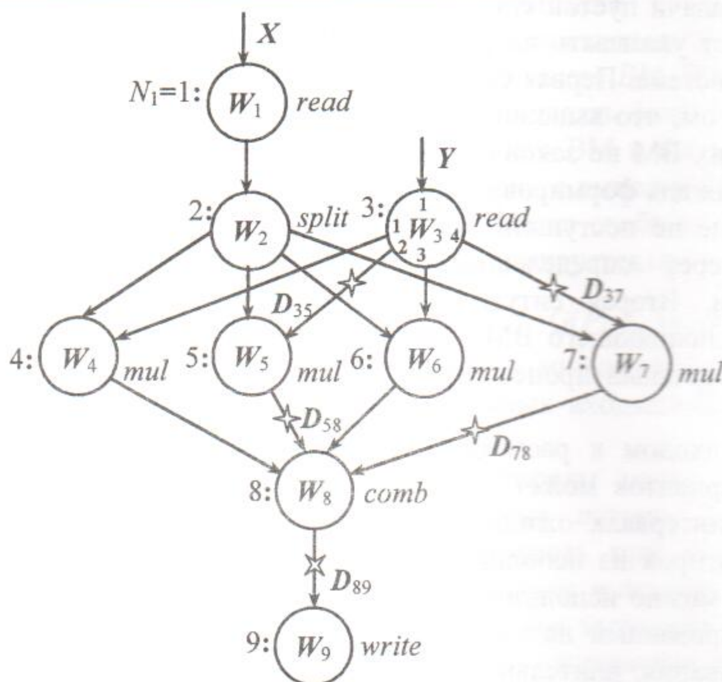

Рис. 2. Граф задачи

После срабатывания таймера в строке S_3 табл. 3 просматриваются адреса рассылки результатов (выделены курсивом), по которым в колонке $L_i=\{\delta_{ji}\}$ определяется маска $M_3=\{\bar{1},\bar{0},\bar{1},\bar{0}\}=\{0,1,0,1\}$. Далее имя невыполненной заявки переписывается из СВЗ в СГЗ. После этого заявка передается на повторное выполнение с полученной новой маской, то есть вычислительный процесс будет продолжен.

Заключение

Предложенный метод реализации вычислений в параллельных системах позволяет устранить ряд проблем, связанных с традиционным планированием вычислений.

Существенно упрощается процесс подготовки задачи. Нет необходимости учитывать длительность выполнения заданий, выявлять параллельные ветви на основе статического анализа.

Подготовка задач не зависит от числа вычислительных модулей в системе. Благодаря этому реконфигурация системы не приводит к необходимости повторной подготовки задачи.

Не требуется обязательная регистрация всех задач перед началом счета, что позволяет начинать решение новой задачи в любой момент времени, независимо от состояния других задач.

Таблица 3

Таблица формирования заявок

S_i	N_i	I_i	$P_i=\{p_{ij}\}$				$A_i=\{A_{ji}\}$	$L_i=\{\delta_{ji}\}$	Q_i	C_i
S_1	1:	*read*	2,1,1				A_{01}	1	1	1
S_2	2:	*split*	4,1,2	5,1,2	6,1,2	7,1,2	A_{12}	1	1	1
S_3	3:	*read*	4,2,2	5,2,2	6,2,2	7,2,2	A_{03}	1	1	1
S_4	4:	*mul*	8,1,4				A_{24} , A_{34}	1 1	2	2
S_5	5:	*mul*	8,1,4				A_{25}, –	1 0	2	1
S_6	6:	*mul*	8,1,4				A_{26} , A_{36}	1 1	2	2
S_7	7:	*mul*	8,1,4				A_{27}, –	1 0	2	1
S_8	8:	*comb*	9,1,1				A_{48}, –, A_{68}, –	1 0 1 0	4	2
S_9	9:	*write*	–				–	0	1	0

Для неоднородных систем становится тривиальной задача поиска паросочетаний (соответствия задания функциональным возможностям вычислительных модулей), поскольку исполняющий модуль может игнорировать задание, которое не в состоянии выполнить.

Исправление ошибок, связанных с отказом процессоров, осуществляется автоматически. Для этого достаточно на этапе разработки библиотеки функций предусмотреть выдачу результатов согласно значению разрядов маски.

Все это позволяет уменьшить непроизводительные затраты времени, что создает предпосылки для ускорения вычислительных процессов, то есть повышения эффективности использования параллельных систем.

Список литературы

1. *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.

2. *Watson R., Guard J.* A practical data flow computer// Computer. – 1982. – Vol. 15, N 2.– P. 51-57.

3. *Жабин В.И.* Метод распараллеливания процессов в вычислительных системах// Вісник Національного технічного університету України «Київський політехнічний інститут». Інформатика, управління та обчислювальна техніка. – 2000. – № 34. – С. 136-142.

4. *Жабин В. И.* Реализация параллельных процессов в вычислительных системах// Искусственный интеллект. – 2002. – №3. – С. 235 – 241.

5. *Жабин В. И.* Организация параллельных вычислений в распределенных системах// Вісник Національного технічного університету України «Київський політехнічний інститут», «Інформатика, управління та обчислювальна техніка». – 2002. – № 38. – С. 10-15.

6. *Жабин В. И.* Реализация вычислений под управлением потока дескрипторов данных в мультипроцессорных системах // Электронное моделирование. – 2003. – Т.25, №1. – С. 35-47.

7. *Авиженис А.* Отказоустойчивость – свойство, обеспечивающее постоянную работоспособность цифровых систем // ТИИЭР (пер. с англ.). – 1978. – Т.66, №10. – С. 5-25.