УДК 004(4’22+054)

С.И. Чайников[1], А.С. Солодовников[2]
[1]ХНУРЭ, г. Харьков, Украина
[2]ХНМУ, г. Харьков, Украина

К ВОПРОСУ ОРГАНИЗАЦИИ КОНТРОЛЬНЫХ ТОЧЕК ВОССТАНОВЛЕНИЯ ДАННЫХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Рассмотрен алгоритм получения структуры, необходимой для организации отката вычислительных процессов по заданным контрольным точкам и на основе пользовательских требований. Рассмотрены преобразования граф-модели предметной области в целях определения мест контрольных точек и резервирования данных вычислительных процессов.

ГРАФ-МОДЕЛЬ, ПРЕДМЕТНАЯ ОБЛАСТЬ, КОНДЕНСАЦИЯ ГРАФА, КОНТРОЛЬНАЯ ТОЧКА

Введение

Современные программные системы обладают высокой степенью сложности, а к их функционированию предъявляются высокие требования, определяющие востребованность программного обеспечения и его качество. Одной из критичных характеристик программных систем является производительность.

Средства оптимизации и тестирования программных средств, позволяющие оценить и решить проблему качества программного обеспечения, должны применяться в течение процесса разработки программы, так как это дает лучшие результаты и качество продукта по сравнению с оптимизацией на последних этапах разработки [1]. Однако существуют и другие меры повышающие производительность программного средства. Например, при решении задач с использованием архитектуры параллельных вычислительных систем предлагаются модели разбиения задач на независимые подзадачи с учетом размещение данных в локальной памяти узлов вычислительной системы [2]. Также в целях сокращения времени доступа к истории вычислений, восстановления данных и отката действий вычислительных процессов в распределенных или параллельных программах оптимизируют время создания и объема контрольных точек восстановления [3]. Для обеспечения отказоустойчивого выполнения программ предлагаются два подхода [3]: реактивный и проактивный. Первый подход является наиболее распространенным и может быть использован в специальных средствах создания контрольных точек (например, MPI-ориентированных) [4]. Причем контрольная точка включает информацию о состоянии выполняющейся параллельной программы: содержимое памяти, информацию об используемых ресурсах операционной системы, граф информационных связей и транзитные сообщения. В крупномасштабных вычислительных системах вероятность потери результатов вычислений высока, но также следует указать на необходимость восстановления данных вычислительных процессов, и в этой связи оптимизации времени восстановления, при обнаружении ошибок в вычислительных процессах и необходимости пересчета на новых исходных данных. Также при проектировании проблемно-ориентированных систем, особенно при использовании распределенной архитектуры, учитываются такие факторы, как архитектура, производительность компьютеров, скорость передачи каналов связи, разнородность подзадач по вычислительной сложности, оптимальное распределение заданий, интеграция вычислительных ресурсов компьютеров разных организаций [5]. Что указывает на актуальность проблемы оптимального отображения структуры программного средства на аппаратную часть и эффективного использования функций программных модулей, каналов передачи данных и процессорных элементов [6].

При проектировании программного обеспечения часто появляется возможность использовать различные программные объекты, которые ранее уже были использованы при решении сходных задач. К таким объектам относятся [7]:

1) повторно используемые приложения;
2) повторно используемые компоненты;
3) повторно используемые функции.

Для технологии повторного использования существует ряд недостатков, один из которых – недоступность исходного кода компонента, что может привести к увеличению расходов на сопровождение системы из-за несовместимости компонентов с изменениями программного кода. Также указывается, что большинство CASE средств, применяемых для разработки программ, не поддерживают технологию повторного использования. Поэтому выход заключается в применении программных генераторов. Такой принцип генератора заложен, например, в программном комплексе PGRAPH, который позволяет создавать программные системы, ориентированные на решение численных задач с применением параллелизма [8,9]. Генерация параллельной программы происходит на основе модели вычислительного процесса.

Также при реализации сборочного подхода к конструированию программы в CASE-средстве,

которое реализует данный метод, выделяются такие основные подсистемы как компоновщик и диспетчер [10]. При этом на вход компоновщика поступают сформированные неделимые фрагменты вычислений и алгоритм сборки, описывающий, какие фрагменты и в каком порядке собираются в единый программный код. В исполняемом коде сохраняется фрагментарная структура сборочной программы для обеспечения возможности распределять исполнение функций между процессорными элементами.

Теория графов является удобным инструментом для работы с подобными структурами, поэтому имеет смысл применять теорию для формализованного описания как самой предметной области так и взаимодействия подсистем программного средства, разрабатываемого для данной области.

Принцип генерации программы на основе граф-модели предметной области [5] является эффективным решением, поскольку позволяет адаптировать программное средство к требованиям предметной области. В этом случае возможно использование ранее разработанных библиотек и включение их в структуру программы. Поскольку для обеспечения качества программного обеспечения одним из требований к интерфейсу является предоставление возможности отмены действия [7], при генерации программы требуется включить в структуру продукта функционал, позволяющий реализовать данную задачу.

1. Постановка задачи

Пусть задан граф, позволяющий описать структуру программного средства:

$$G = (V, X), \qquad (1)$$

где V – множество вершин v, соответствующих программным модулям, а X – множество дуг $x_{ij} = (v_i, v_j)$ графа G, соединяющих вершины v_i и v_j между собой, для которых задано направление ($i, j = \overline{1,n}$, где n – количество вершин графа G). Дугам соответствуют информационные связи между программными модулями.

Над графом (1) проведена операция конденсации вершин, то есть приведения к ациклической форме, что позволяет задать на данном графе множество супер вершин $Vs = \{v_i\}$, включающих в себя циклы.

Граф (1) также приведен к ярусно-параллельной форме, позволяющей установить последовательное и параллельное исполнение программных модулей процессорными элементами.

Также задан следующий набор ограничений для каждого яруса графа, необходимый при отображении структуры программного средства на аппаратную архитектуру:

1) для каждого яруса для n программных модулей, размещенных на этом ярусе, могут быть доступны m процессорных элементов, причем, $m \le n$.

2) каждый процессорный элемент характеризуется набором квот $Q = \{q_k\}$ где, например, q_1 – это квота на процессорное время, q_2 – квота на количество используемых ядер процессора, q_3 – квота на объем памяти, занимаемый программой и так далее;

3) считается, что условия обмена данными между процессорными элементами одинаковы.

Для каждой вершины графа (1) задана спецификация программного модуля, которая описывает основные требования к нему. То есть для i-го программного модуля существует спецификация $Sp_i = SPC, \forall i = \overline{1,n}$, где SPC – множество спецификаций вида [16]:

$$Sp_i = \{D^{in}, D^{out}, T^{in}, T^{out}, R^{in}, R^{out}, F\}, \qquad (2)$$

где D^{in} – множество входных массивов данных, D^{out} – множество выходных массивов данных, T^{in} множество типов входных данных, T^{out} – множество типов выходных данных, R^{in} – множество правил ввода входных данных модуля, R^{out} – множество правил вывода выходных данных модуля, F – множество управляющих воздействий i-го модуля.

Пусть также задано множество $Vd = \{v_k\}$, где v_k – множество вершин отображающих функцию организации диалога с пользователем.

Требуется определить алгоритм отката результатов вычислительных процессов по заданным контрольным точкам, а также необходимость создания избыточных копий программных модулей.

2. Метод решения задачи

Так как для реализации сходных задач допустимо применение одного программного модуля, имеет смысл различать для него семантическое и синтаксическое имя. С этой целью должна быть задана таблица с соответствующими полями, позволяющая установить соответствия между этими именами для заданной вершины.

В результате анализа структуры программного средства также выделены вершины $v_k \in Vd \setminus \{v_1, v_n\}$, соответствующие модулям, которые обеспечивают диалог с пользователем. Вершины v_1, v_n являются, по сути, фиктивными вершинами, позволяющими объединить соответственно множество входов в программу в одну корневую вершину и множество выходов программы в один выход.

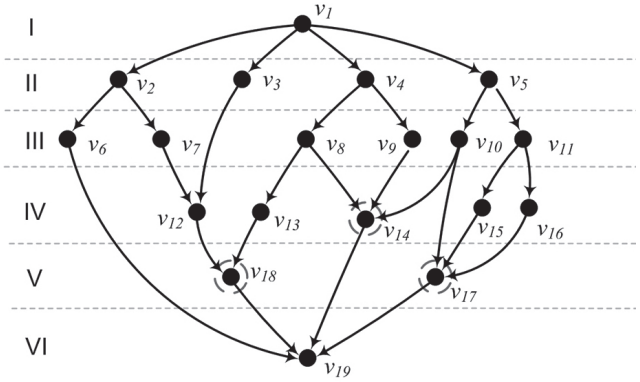
Пусть заданы вершины v_{14}, v_{17}, v_{18} (рис. 1, а), в которых пользователю необходимо осуществлять контроль над работой вычислительных процессов. Для таких вершин может быть сформировано множество подграфов $G_k \subseteq G$ (рис. 1, б), для которых заданы матрицы смежности $A_k = \|a_{ij}\|$.

Для данного примера можно выделить такие подграфы G_1, G_2 и G_3, для которых существуют следующие множества вершин:

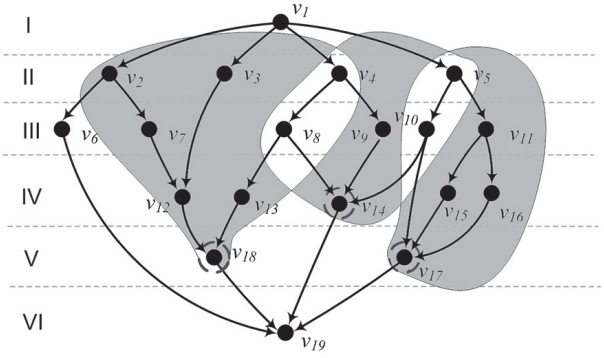
$$G_1 : V_1 = \{v_2, v_3, v_4, v_7, v_8, v_{12}, v_{13}, v_{18}\};$$
$$G_2 : V_2 = \{v_4, v_5, v_8, v_9, v_{10}, v_{14}\};$$
$$G_3 : V_3 = \{v_5, v_{10}, v_{11}, v_{15}, v_{16}, v_{17}\}.$$

Исходя из этого, получаем:

$$V_1 \cap V_2 = \{v_4, v_8\};$$
$$V_1 \cap V_3 = \varnothing;$$
$$V_2 \cap V_3 = \{v_5, v_{10}\}.$$



а



б

Рис. 1. Ориентированный граф, моделирующий структуру разрабатываемого программного средства:
а – ЯПФ графа с обозначенными вершинами, реализующими диалог;
б – ЯПФ графа с выделенными подграфами

Для вершин v_8, v_4 и для v_5, v_{10} возможны три состояния:

1) программные модули используют один и тот же выходной набор данных, который затем обрабатывается последующими программными модулями;

2) программные модули используют разный выходной набор данных для последующих модулей;

3) программные модули позволяют разделять часть своего набора выходных данных между последующими программными модулями.

То есть:

$$\begin{cases} \bigcap\limits_k Sp_k(D^{in}) = \varnothing, \forall Sp_k(D^{in}) \subset Sp_i(D^{out}) \\ \bigcap\limits_k Sp_k(D^{in}) \subseteq Sp_i(D^{out}), \end{cases} \quad (3)$$

где k – является индексом вершин, для которых существует дуга $x_{ik} = (v_i, v_k)$.

Для того, чтобы узнать, являются ли вершины графа зависимыми от общей вершины, применяется следующая логика [7].

Пусть $A = \|a_{ij}\|$ – матрица смежности графа (1), в котором исключены общая вершина входа и общая вершина выхода. Пусть дана матрица $S = A^N$, где N – это количество вершин графа. Элементы матрицы $s_{ij} > 0$ в случае, если вершина v_i предшествует вершине v_j.

Получая для матрицы $S = \|s_{ij}\|$ пересечение столбцов, можно выяснить для пары вершин *i* и *j*, зависят ли они от общей вершины. Для этого формируется следующая матрица:

$$P = \|p_{ij}\|, \text{ где } p_{ij} = \sum_{k=1}^{m} s_{ki} s_{kj}. \quad (4)$$

Используя матрицу (4), получают наборы вершин, у которых имеются только одни одинаковые предшествующие им вершины.

Далее получаем матрицу, позволяющую получить обратную характеристику:

$$Q = \|q_{ij}\|, \text{ где } q_{ij} = \sum_{k=1}^{m} s_{ik} s_{jk}. \quad (5)$$

На основании выражений (4) и (5) можно получить набор независимых и сильнозависимых вершин, то есть тех вершин v_i, для которых существуют общие вершины v_k при наличии дуги $x_{ik} = (v_i, v_k)$ и одновременно с этим существуют общие вершины v_j при наличии дуги $x_{ji} = (v_j, v_i)$. Для получения такого набора вершин необходимо выполнить следующие операции:

$$\overline{P \cup Q} \quad (6)$$

для независимых вершин и

$$P \cap Q \quad (7)$$

для сильнозависимых вершин.

Данные операции позволяют оптимально распределить нагрузку на процессорные элементы с учетом сформированных требований для отображения графа структуры программы на аппаратную архитектуру.

С учетом (3), (6) и (7) удобно использовать модель передачи сообщений или модель общей памяти для организации вычислительных процессов [11]. При этом регулирование процессов осуществляется либо посредством отправки/приема сообщений, либо путем использования семафоров.

Следующей стадией является организация отката вычислительного процесса (backtracking). Для этого для каждого из подграфов $G_k \subseteq G$ определяются транспонированные матрицы смежности A_k^T. И для них выполняется операция (6), позволяющая определить вершины графа, в которых, в соответствие с (3), должны создаваться множественные копии данных вычислительных процессов для исключения блокировок записей.

Указанные положения позволяют сформулировать алгоритм, состоящий из следующих шагов:

1) получение ациклического графа;

2) получение ярусно-параллельной формы графа;

3) формирование ограничительных условий на размещение вершин графа для каждого яруса с учетом особенностей аппаратного обеспечения;

4) определение программных модулей и соответствующих им вершин графа (1) ответственных за организацию диалога с пользователем;

5) формирование для каждого программного модуля спецификации (2);

6) получение множества вершин с учетом условия (3);

7) определение матриц (4) и (5), и получение множества вершин согласно действиям (6) и (7).

Выводы

Сформированный граф структуры программной системы является основой для разворачивания программных компонент на аппаратном обеспечении и необходим для эффективного восстановления состояний вычислительных процессов в связи с необходимостью отмены действий посредством пользовательского интерфейса. Данный функционал включается в программный продукт при генерации программы. И позволяет повысить производительность программы за счет сокращения времени обработки запросов пользователей на восстановление данных указанных вычислительных процессов. Для обеспечения возможности отката вычислительного процесса предложен алгоритм приведения исходного граф-представления программного продукта к структуре, требуемой для выполнения поставленной задачи. Метод имеет практическое значение, прежде всего, при проектировании компьютерных систем, предназначенных для автоматизации вычислительных процессов крупномасштабных объектов, таких как проектные организации, производственные компании.

Список литературы: 1. *Гербер Р.* Оптимизация ПО. Сборник рецептов / Р. Гербер, А. Бик, К. Смит, К. Тиан. – Спб.: Питер –2010. – 352 с. **2.** *Стиренко С.Г.* Модель организации вычислений в распределённой системе / С.Г. Стиренко, А.И. Зиненко, Д.В. Грибенко // Вісник НТУУ «КПІ»: Інформатика, управління та обчислювальна техніка. – К.: Век+. – 2012. – № 57. – С. 101-109. **3.** *Поляков А.Ю.* Оптимизация времени создания и объема контрольных точек восстановления параллельных программ / А.Ю. Поляков, А.А. Данекина // Вестник СибГУТИ. – 2010. – № 2. – С. 87-100. **4.** *Elnozahy E.N.* A survey of rollback-recovery protocols in message-passing systems [Text] / E.N. Elnozahy, L. Alvisi, Y.M. Wang, D.B. Johnson // ACM Computing Surveys. – 2002. – № 3 (34) – P. 375-408. **5.** *Чайников С.И.* Принципы организации вычислений на базе граф-модели предметной области / С.И. Чайников, А.С. Солодовников // Бионика интеллекта: науч.-техн. журнал. – 2012. – № 2 (79). – С. 72-75. **6.** *Алекперов Р.К.* Организация распределенных вычислений на базе GRID-технологии / Р.К. Алекперов // «Искусственный интеллект», № 1. – 2011. – С. 6-14. **7.** *Байцер Б.* Архитектура вычислительных комплексов. Т.2: пер. с англ. / Б. Байцер. – М.: Мир, 1974. – 566 с. **8.** *Жидченко В.В.* Комплекс программ моделирования параллельных синхронных вычислительных процессов / В.В. Жидченко // Методы и средства обработки информации: труды второй Всероссийской научной конференции. – М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова. – 2005. – С. 465-471. **9.** *Коварцев А.Н.* Программный комплекс моделирования параллельных вычислительных процессов PGRAPH 1.0 / А.Н. Коварцев, В.В. Жидченко // Инновации в науке и образовании. – 2006. – № 6. – С. 7. **10.** *Малышкин В.Э., Корнеев В.Д.* Параллельное программирование мультикомпьютеров / В.Э. Малышкин, В.Д. Корнеев – Новосибирск: Новосибирский государственный технический университет, 2006. – 452 с. **11.** *Воеводин В.В.* Параллельные вычисления / В.В. Воеводин, Вл. В. Воеводин. – СПб.: БВХ-Петербург. – 2002. – 608 с.

Поступила в редколлегию 17.09.2014

УДК 004(4’22+054)

До питання організації контрольних точок відновлення даних обчислювальних процесів / С. І. Чайніков, А. С. Солодовніков // Біоніка інтелекту: наук.-техн. журнал. – 2014. – № 2 (83). – С. 128–131.

Пропонується алгоритм отримання структури, необхідної для організації відкату обчислювальних процесів по заданих контрольних точках і на основі вимог користувачів. Розглянуто принцип оптимізації копій даних обчислювальних процесів, необхідних для коректного відновлення для заданого моменту часу за множинними запитами користувачів.

Іл. 1. Бібліогр.: 17 найм.

UDC 004(4’22+054)

To the question of checkpoints organization for the data of computational processes / S. I. Chaynikov, A. S. Solodovnikov // Bionics of Intelligence: Sci. Mag. – 2014. – № 2 (83). – P. 128–131.

The authors propose an algorithm to obtain the structure that is necessary for organizing backtracking of computational processes data using checkpoints, and based on user requirements. The authors consider the principle of optimization of data copies for computational processes required for proper data recovery.

Fig. 1. Ref.: 17 items.