Д.М. Наширванов, С.В. Востокин

АКТОРНАЯ МОДЕЛЬ ДЛЯ СТАТИЧЕСКИХ АЛГОРИТМОВ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА ПЕРЕДАЧИ СООБЩЕНИЙ MPI

(Самарский национальный исследовательский университет
имени академика С.П. Королёва)

Идея модели акторов состоит в том, что процессы разделены по отдельным задачам, выполняются параллельно и передают информацию с помощью сообщений. MPI (Message Passing Interface) – программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Стандарт MPI ориентирован на системы с распределенной памятью, когда затраты на передачу данных велики [1]. MPI существенно упрощает реализацию акторной модели на распределенных системах, т.к. позволяет отправлять сообщения процессам, запущенным на другом вычислительном устройстве.

Реализация данной модели предназначена для применения в составе инструмента быстрой разработки параллельных алгоритмов Templet Web [2]. Он предусматривает автоматизацию написания кода статических алгоритмов, в которых сеть акторов, соединённых каналами передачи сообщений, не изменяется во время вычислений, т.е. для акторных сетей с неизменной топологией. Реализация выполнена на языке C++, в качестве механизма передачи сообщений используется библиотека MPI из пакета Intel Parallel Studio XE 2016 и библиотека MPICH2.

Реализация модели основана на разделении вычислений в программе на следующие этапы:

1. создание акторов в виде объектов/структур языка C++;
2. построение коммуникационной топологии путём связывания акторов каналами передачи сообщений;
3. определение привязки конкретных акторов к процессам MPI;
4. ввод исходных данных в программу;
5. выполнение вычислений;
6. вывод/сохранение результатов работы.

Этапы (4) и (6) выполняются исключительно на мастер-процессе MPI, этап выполнения вычислений (5) обычно выполняется на рабочих процессах, но так же можно использовать и мастер-процесс. Остальные же этапы протекают для всех процессов.

Алгоритм управления вычислениями в рассматриваемой реализации основан на технике потокового пула в разделяемой памяти. В предлагаемой распределённой реализации алгоритма используется сериализация акторов и сообщений, а также алгоритм Хуанга (Huang) для определения момента остановки вычислений в распределённой системе.

В качестве тестового примера реализации акторной модели с использованием MPI рассматривается распределенный алгоритм голосования – алгоритм забияки (Bully algorithm) [3]. Алгоритм позволяет выявлять отказавший процесс координатор, отказ моделируется программно на основе генератора случайных чисел. Каждый процесс – это актор нашей модели. Все процессы имеют собственный номер (приоритет), а процесс-координатор – это процесс с наивысшим приоритетом среди рабочих процессов.

Когда один из процессов замечает, что координатор перестал отвечать, он инициирует голосование. Голосование проводится следующим образом.

1. Некоторый процесс P посылает всем процессам с большими, чем у него номерами специальное сообщение «Голосование».
2. Если никто не отвечает, то процесс P становится процессом-координатором.
3. Если один из процессов с большим номером отвечает, то процесс P заканчивает свою работу и передается под управление новому процессу-координатору, который будет выбран дальнейшим голосованием процессов с большими номерами.

В любой момент процесс может получить специальное сообщение «Голосование» от одного из процессов с меньшим номером. При получении данного сообщения, он должен немедленно послать отправителю ответ «ОК», показывая, что он работает и готов стать координатором. Затем получатель сам организует голосование. В конце концов, все процессы, кроме одного, отпадут, и этот последний будет новым процессом-координатором. Он должен уведомить все остальные процессы об этом и приступить к выполнению задачи координатора.

Если процесс, который находился в нерабочем состоянии, начинает работать, он запускает новое голосование, и, если у него окажется самый высокий приоритет, то он снова станет координатором.

Данный тестовый алгоритм включает парные двухсторонние взаимодействия, коллективную рассылку, асинхронное взаимодействие, недетерминированное выполнение. Корректное выполнение тестового алгоритма позволит с высокой степени вероятности утверждать о корректности реализации модели акторов.

Литература

1. Меньшикова Л. В. Способы реализации параллельных вычислений [Электронный ресурс] – http://www.tssonline.ru/articles2/fix-corp/sposoby-realizatsii-parallelnyh-vychisleniy

2. Востокин С.В. Templet: язык разметки для параллельного программирования. СГАУ им. академика С.П. Королёва, Самара, 2014.

3. Таненбаум Э. Распределенные системы. Принципы и парадигмы. «Питер», Санкт-Петербург, 2003

Ю.В. Орлов

ОСНОВНЫЕ ПРОБЛЕМЫ РЕАЛИЗАЦИИ СРЕДЫ КОМПЛЕКСНОГО АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ОПТИМИЗАЦИИ[4]

(Вычислительный центр им. А.А. Дородницына
Федерального исследовательского центра «Информатика и управление» Российской академии наук)

Многие задачи глобальной оптимизации относятся к классу NP и их решение требует значительных вычислительных ресурсов. Поэтому представляется целесообразным применение методов параллельных [1,2] и распределенных вычислений [3]. Метод ветвей и границ (МВГ) является одним из наиболее распространенных алгоритмов решения задач дискретной оптимизации. В его основе лежит идея декомпозиции, которая делает естественным применение параллельных вычислений. Обзор различных подходов можно найти в работах [4-6]. Основной проблемой при параллельной реализации методов типа ветвей и границ является адекватная балансировка вычислительной нагрузки между параллельными процессорами. Так как информационный граф алгоритма [7], в данном случае представляющий собой дерево, заранее не известен, особую важность получают методы динамической балансировки, перераспределяющие вычисления в процессе работы в зависимости от загрузки процессоров.

В ближайшее время ожидается, что суперкомпьютеры обретут производительность порядка экзафлопс (около 10^{18} операций с плавающей точкой в секунду). Основным способом увеличения производительности является наращивание количества ядер, число которых в некоторых современных системах уже превосходит 10^{16}. В такой ситуации балансировка нагрузки становится весьма нетривиальной задачей. Следовательно, необходим развитый инструментарий для анализа производительности алгоритмов балансировки. Часто без визуализации, используя одну лишь трассу, выявить причины потери производительности практически невозможно, так как трассы выполнения параллельных программ обычно очень велики по объему и слабо поддаются визуальному анализу.

[4] Работа выполнена при поддержке РФФИ (проекты № 16-07-00873 А и № 16-07-00458 А).

Основные принципы работы среды

Для того, чтобы проводить исследование алгоритмов балансировки без проведения ресурсоемких вычислений, был разработан симулятор многопроцессорной системы BNB-Simulator[8] на базе компонентов библиотеки BNB-Solver[9,10]. Данная библиотека предлагает набор модулей для разработки параллельных приложений. Симулятор представляет собой приложение, управляемое через файл настроек settings.json. Он имитирует выполнение реального параллельного приложения. Результатом работы симулятора является трасса, которая имеет тот же формат, что и трасса параллельного приложения, разработанного на основе BNB-Solver.

Среда комплексного анализа позволяет анализировать трассу, загруженную из файла с расширением «.trc». Таким образом, визуальная среда дает возможность работать с трассой установленного формата, независимо от источника ее получения – реального приложения или симулятора (Рис. 1).

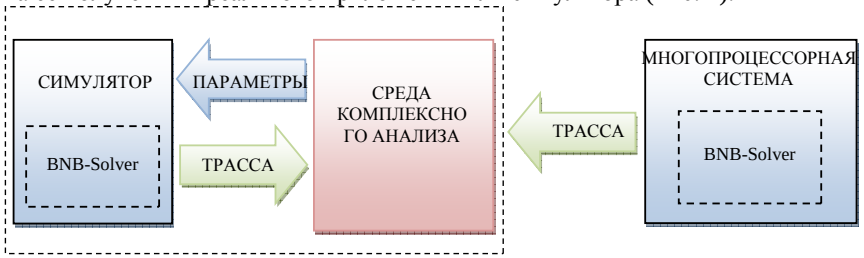

Рис. 1. – Общая схема работы визуальной среды

Схема отображения трассы показана на рис. 2.

Изначально предполагалось, что среда комплексного анализа в процессе своей работы будет хранить данные трассы внутри двумерного массива процессоров размерности n*m, где n – количество процессоров, а m – количество проработанного времени, измеряемого специальными метками. Однако такой подход оказался неприемлемым при обработке трасс большого размера, так как при превышении лимита оперативной памяти, программа начинает задействовать пространство жесткого диска машины, что в свою очередь значительно замедляло ее работу, а в ряде случаев приводило к аварийному завершению программы.

Таким образом, понадобился более гибкий способ обработки трассы. Было принято решение делить в процессе обработки трассу на множество файлов, состоящих из результирующей и содержательной частей. Содержательная часть описывает работу подмножества процессоров на некотором промежутке времени. Результирующая часть содержит итоговые статистические данные о результате работы подмножества процессоров на установленном промежутке времени. Для обеспечения быстрого доступа к таким данным в процессе счета программы результирующая часть помещается в начале файла.