ОПТИМИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ НА ЭТАПЕ РАЗРАБОТКИ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

А.М. Теплов, М.Я. Даниленко

Морской гидрофизический институт
НАН Украины
г. Севастополь, ул. Капитанская, 2
E-mail:alex-teplov@yandex.ru

В статье приведены результаты исследования вычислительно сложных параллельных программ, разработанных на основе двух различных реализаций алгоритмов решения одной и той же задачи.

Введение. Идея создания параллельных (многопроцессорных) вычислительных систем, сформулированная в 50-х годах 20-го века, преследовала цель повышения производительности компьютеров, не прибегая к принципиальному изменению их элементной базы. Но практика показала, что написать качественную параллельную программу или приспособить уже имеющуюся последовательную программу к параллельному компьютеру чрезвычайно трудно. Связано это в первую очередь с тем, что переход к использованию многопроцессорных систем характеризуется принципиально новым содержанием работ на всех этапах разработки программы для параллельной вычислительной системы:

– при построении модели решения задачи;

– при структурном анализе и разработке алгоритмов решаемой задачи;

– при разработке самой параллельной программы (и обязательно с учетом конкретной архитектуры вычислительной системы);

– на этапе разработки архитектуры вычислительной системы (в первую очередь для сложных специальных задач, например, для прогноза погоды) [1].

Постановка проблемы. Анализ параллельных программ является одной из самых актуальных и сложных задач в области современного параллельного программирования. Эти исследования требуют наличия развитых средств отладки, таких как программы мониторинга и утилиты, позволяющие выполнить программу без ошибок, увидеть ход ее выполнения и оценить масштабируемость параллельной программы.

Масштабируемость параллельной программы – это свойство параллельной программы, характеризующее зависимость динамических характеристик (время выполнения, ускорение решения задачи за счет распараллеливания, реальная производительность и эффективность параллельной системы) параллельной системы от объема, задействованных для выполнения этой программы вычислительных ресурсов системы, и (или) от объема обрабатываемых данных. Количественно масштабируемость можно определить по зависимости величины ускорения решения задачи на данной вычислительной системе от числа задействованных процессоров.

В последние годы масштабируемость параллельных программ стала объектом большого количества исследований в области параллельного программирования. Этой теме посвящены как отдельные статьи в научных журналах, так и многие диссертации. Среди этих работ можно встретить большое количество методов, по которым исследуются те или иные задачи на масштабируемость. Эти методы во многом отличаются между собой и часто базируются на различных свойствах параллельных программ. Во многом каждый метод специфичен и предназначен для своей области задач. При выборе метода исследования необходимо учитывать большое количество факторов. У каждого из методов есть свои плюсы и минусы, выбор каждого зависит от цели исследования и акцентов при анализе результатов исследований.

Различия методов исследования сводятся, например, к способам получения данных. Разные методы могут получать результаты, которые говорят о масштабируемости в несколько различном контексте. При этом следует различать масштабируемость задачи, масштабируемость алгоритма, масштабируемость параллельной программы и масштабируемость параллельной системы (связки параллельной программы и целевой вычислительной системы фиксированной архитектуры). Как раз разный подход к

сбору информации о масштабируемости параллельной программы и обеспечивает получение более полных сведений о различных контекстах масштабируемости. При выборе метода исследования масштабируемости необходимо в первую очередь определить, какой контекст масштабируемости в нашем случае наиболее полно отражает информацию о динамических характеристиках параллельной вычислительной системы.

Анализ последних достижений. Оптимизация выбора алгоритма задачи при параллельном программировании существенно влияет на последующее время решения этой задачи и получаемые при этом погрешности вычислений.

Некоторые алгоритмы допускают для отдельных задач даже более чем линейное ускорение решения задачи при применении параллельных вычислительных систем (так называемое суперлинейное ускорение). Другие алгоритмы такого существенного увеличения скорости решения задачи не дают из-за ограниченного распараллеливания вычислений, и в результате суммарное увеличение скорости не пропорционально количеству вычислительных ядер. Третьи алгоритмы вообще не приводят к существенному увеличению скорости решения задачи по размеру набора данных, так как значительная часть операций остается общей [2].

Разработку алгоритма, как правило, начинают с выбора его модели. Модели параллельных программ различаются главным образом степенью детализации моделируемых процессов и создаются для различных целей:

– разработки оптимальных параллельных программ;

– отработки методов анализа параллельных программ;

– анализа фундаментальных свойств и законов параллельных вычислений (например, с помощью схем параллельных программ изучаются вопросы эквивалентности программ, уровня параллелизма, правильности функционирования и т.п.);

– обоснования новых программных конструкций, вводимых в языки параллельного программирования, проверке новых способов организации параллельных программ;

– расчета временных характеристик и построения расписаний выполнения и т.д.

Применяемый алгоритм должен иметь наименьший объем последовательных вычислений среди других параллельных алгоритмов и учитывать особенности применяемой вычислительной системы. Качество выбора и разработки алгоритма самым существенным образом влияет на весь ход решения задачи.

Алгоритм, составленный человеком, практически всегда создается в последовательной форме, и его распараллеливание является уже дальнейшим развитием алгоритма. Алгоритм решения задачи описывает ход ее решения из некоторого класса аналогичных задач. Вследствие этого имеется и большое разнообразие параллельных алгоритмов для решения одной и той же задачи. В результате трудоемких исследований из множества вариантов выбирается один алгоритм, обладающий вполне приемлемыми характеристиками, и именно он программируется. В программе же любой алгоритм уже всегда описывается абсолютно точно.

Выбор способа разделения вычислений на независимые (параллельные) части основывается на анализе вычислительной схемы решения исходной задачи. Требования, которым должен удовлетворять выбираемый вариант, обычно состоят в обеспечении равного объема вычислений в выделяемых подзадачах и минимума информационных зависимостей между ними. Важный вопрос при выделении подзадач состоит в выборе нужного уровня декомпозиции (разделения) вычислений. Формирование максимально возможного количества подзадач обеспечивает достижение предельно возможного уровня параллелизма решаемой задачи, но затрудняет анализ параллельных вычислений. Применение же при декомпозиции вычислений достаточно крупных подзадач приводит к ясной схеме параллельных вычислений, однако может затруднить реальное использование достаточно большого количества процессоров.

Выделение подзадач производят также с учетом возникающих информационных связей. Поэтому после анализа объема и частоты необходимых информационных обменов между подзадачами может потребоваться повторение этапа разделения вычислений.

Масштабирование разработанной вычислительной схемы параллельных вычислений проводится также в случае, если количество имеющихся подзадач больше числа планируемых к использованию процессоров. Для сокращения числа подзадач в этом случае выполняют агрегацию (укрупнение) вычислений. При этом подзадачи должны иметь одинаковую вычислительную сложность, а объем и интенсивность информационных взаимодействий между подзадачами должны оставаться на минимально возможном уровне. Первыми претендентами на объединение являются подзадачи с высокой степенью информационной взаимозависимости.

Таким образом, при выборе алгоритма вычислений необходимо обеспечить выполнение на этапе разработки параллельной программы принципов агрегации и декомпозиции подзадач и обязательно с учетом изменения количества задействованных вычислительных ресурсов и (или) объема обрабатываемых данных [3].

Цель и методика проведения экспериментов. Целью данной работы являлось отработка методики исследования зависимости масштабируемости вычислительно сложных параллельных программ от работ, проведенных на этапе *структурного анализа и выбора алгоритма решаемой задачи.* Для этого были применены две параллельные программы, разработанные на базе двух различных по структуре алгоритмов одной и той же задачи.

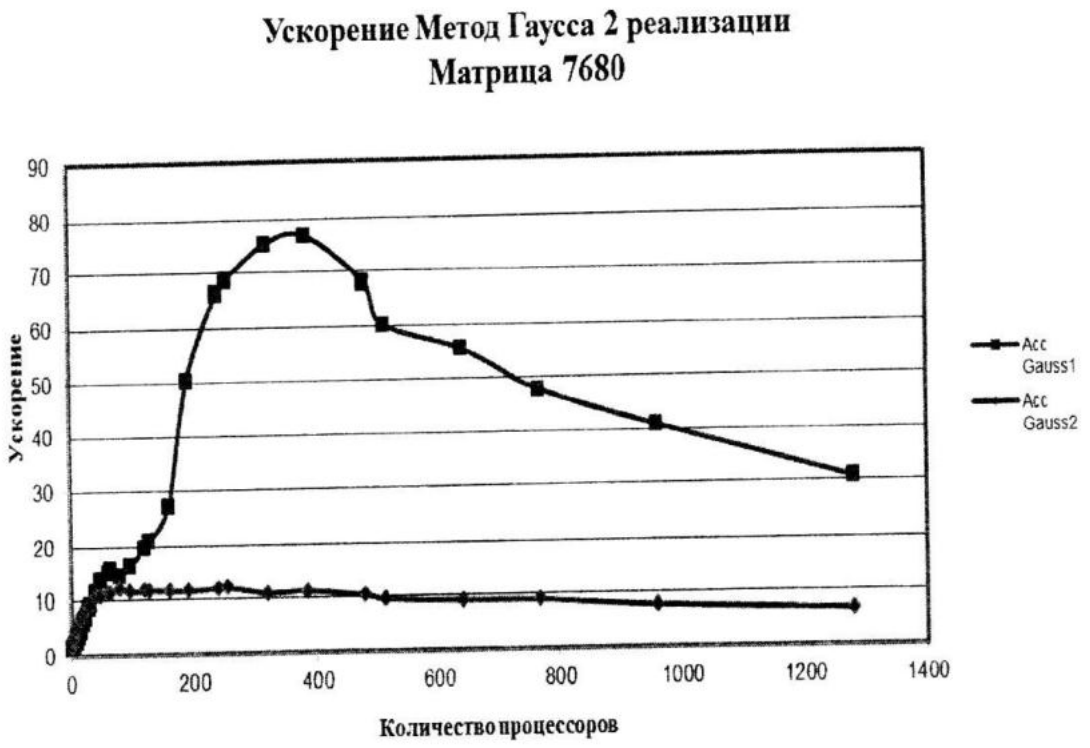
Исследования проводились эмпирическим методом, который имеет свои преимущества и недостатки. В данном исследовании он позволил получить достаточно подробную картину масштабируемости каждой из программ и провести сравнительный анализ масштабируемости этих программ. Вместе с тем эмпирический метод требует значительных вычислительных ресурсов.

Результаты работы программ определялись путем измерения времени выполнения параллельных программ на различных конфигурациях вычислительных систем. Для всех экспериментов минимальной конфигурацией вычислительной системы являлось одно вычислительное ядро. Поэтому размер задач был подобран таким образом, чтобы данные полностью помещались в оперативную память, доступную на одном узле, а также, чтобы задача решалась за разумное время.

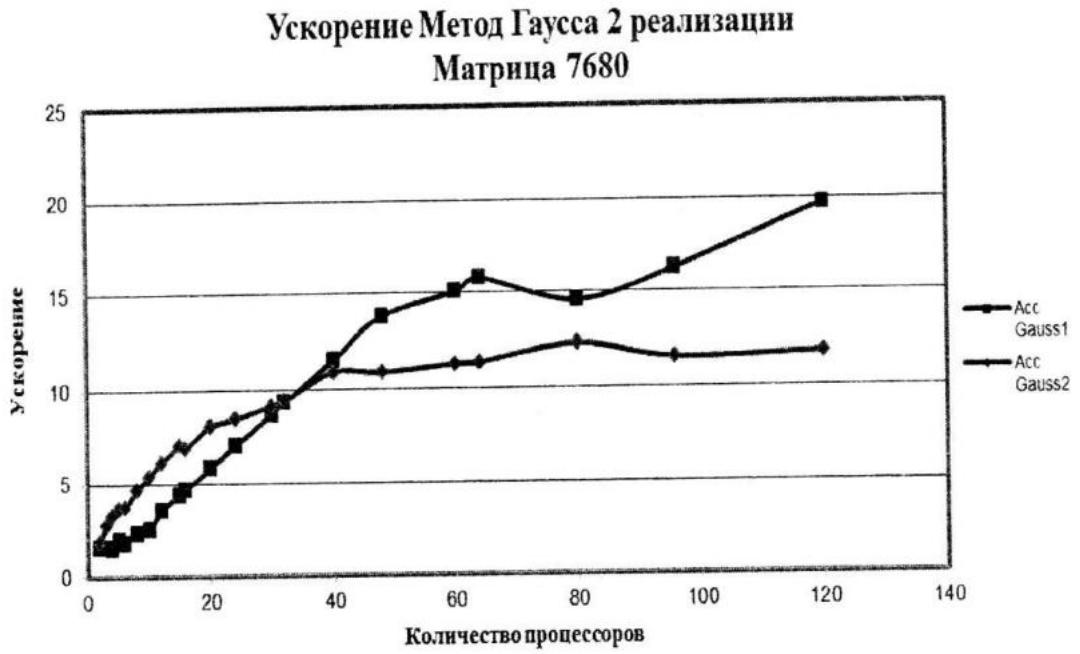
Результаты исследования. Исследование масштабируемости конкретных параллельных программ проводилось на вычислительной системе СКИФ МГУ «Ломоносов» и кластере СКИФ МГУ «Чебышев», установленных в Научно-исследовательском Вычислительном Центре Московского государственного Университета им. М.В. Ломоносова.

Были исследованы две параллельные программы решения системы линейных алгебраических уравнений методом Гаусса, реализующие разные параллельные алгоритмы одной и той же задачи: реализация 1 и реализация 2. Результаты исследований приведены на рис. 1 – 4.

Как видно из графиков (рис. 1), ускорение реализации 1 выглядит более привлекательным для больших конфигураций вычислительных систем, а ускорение реализации 2 в диапазоне процессоров от 40 процессоров и вплоть до 1024 быстро насыщается и перестает расти. Однако реализация 2 лучше масштабируется в диапазоне от 4 до 40 процессоров (рис. 2). Это объясняется тем, что если при использовании количества процессоров до 40, время накладных расходов при работе вычислительной системы покрывается выигрышем во времени за счет параллельности вычислений, то при увеличении количества процессоров они становятся уже неоправданно высокими. И связано это, в первую очередь, с наличием в реализации 2 большого количества коммуникаций. Именно поэтому на графике (рис. 1) наблюдается быстрое насыщение ускорения реализации 2 и практически линейный рост до 400 процессоров ускорения реализации 1.



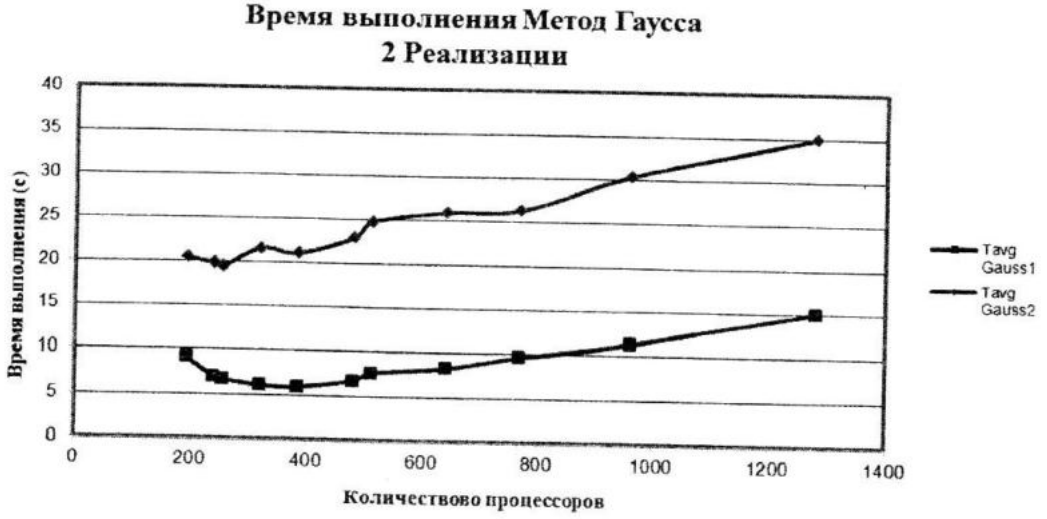
Р и с. 1. Графики ускорений двух реализаций на большом размере системы



Р и с. 2. Графики ускорений двух реализаций на малом размере системы

Также стоит отметить, что кроме ускорения необходимо учитывать и время выполнения программы, потому что ускорение – относительная величина. В этом плане на небольших конфигурациях вычислительной системы более выигрышной будет также реализация 2, что видно из диаграмм времени выполнения (рис. 3). Также на них видно очевидное преимущество реализации 1 на больших размерах вычислительной системы (рис. 4).

Р и с. 3. График времени выполнения двух реализаций на малом размере системы



Р и с. 4. Графики времени выполнения двух реализаций на большом размере системы

Разница между рассмотренными реализациями алгоритма заключается в принципе распределения данных по процессам.

В алгоритме реализации 2 исходная матрица коэффициентов **A** и вектор правых частей **F** разрезаны горизонтальными полосами. Каждая полоса загружается в соответствующий вычислительный процесс: нулевая полоса – в нулевой процесс, первая полоса – в первый процесс, и т.д. В программе предполагается, что матрица коэффициентов **A** и вектор правых частей **F** разрезаны на части заранее, и каждая ветвь считывает свои части из памяти.

Особенностью этого алгоритма является то, что как при прямом, так и при обратном ходе, вычислительные процессы, завершившие свою часть работы, переходят в состояние ожидания, пока не завершат эту работу другие процессы. Таким образом, вычислительная нагрузка распределяется по процессам неравномерно, несмотря на то, что *данные* из-

начально распределяются по процессам приблизительно одинаково.

Простои процессов работы значительно уменьшаются при распределении матрицы горизонтальными циклическими полосами. В алгоритме реализации 1, исходная матрица коэффициентов **A** и вектор правых частей **F** разрезаны горизонтальными циклическими полосами. Особенностью этого алгоритма является то, что как при прямом, так и при обратном ходе процессы будут более равномерно загруженными, чем в алгоритме реализации 2. Значит, и вычислительная нагрузка распределяется по вычислительным процессам более равномерно. Например, нулевой процесс, завершив обработку своих строк при прямом ходе, ожидает, пока другие процессы обработают только по одной, оставшейся у них необработанной строке, а не полностью обработают полосы, как в алгоритме реализации 2.

Из диаграмм времени выполнения также становится понятно, что, хотя у реализации 1 масштабируемость лучше, чем у реализации 2, но время ее выполнения на малых конфигурациях значительно больше (рис. 3).

Выводы. Результаты эксперимента показывают, что исследуя различные реализации алгоритма решения одной и той же задач, можно определить какую из них предпочтительнее использовать в зависимости от приоритетов использования программы, а также целевых конфигураций вычислительной системы.

Влияние на время выполнения решения задач и производительность вычислительных систем каждого из факторов (различных составляющих архитектуры вычислительной системы, системного программного обеспечения, используемых библиотек и т.п.) является мало изученным и может быть целью дальнейших исследований в этой области.

СПИСОК ЛИТЕРАТУРЫ

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления (http://www.bookshunt.ru/b41281_parallelnie_vichisleniya)
2. *Патил Рахул В., Джордж Бобби* Инструменты и методики для выявления проблем, связанных с параллельным выполнением кода (http://www.microsoft.com/Rus/Msdn/Magaine/2008/07/rasparallelivanie.mspx)
3. *Бухановский А.В., Иванов С.В.* Проектирование прикладного математического обеспечения параллельной обработки данных.// Учебные материалы Зимней школы-практикума «Технологии параллельного программирования, 2006» (http://window.edu.ru/window_catalog/files/r62365/book-2006.pdf)