Е.В. ПУГИН, Р.А. СИМАКОВ

Распараллеливание математических вычислений на примере операторов линейной алгебры

УДК 004.047

Муромский институт (филиал) ФГБОУ ВПО «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых», г. Муром

В статье рассматриваются вопросы распараллеливания и оптимизации математических вычислений в распределённых системах. Приводится пример распараллеливания математического оператора линейной алгебры – ковариации.

In this paper, parallelization and optimization of mathematical calculations in the distributed systems are described. An example of paralleliza-tion of linear algebra operator - covariance is given.

В настоящее время происходит активное развитие многопроцессорных и облачных вычислительных систем, в работе которых над результатом производит действия не один процессор или ядро, а сотни и тысячи компьютеров, организованные в параллельные кластеры. В данном случае привычные последовательные алгоритмы, не использующие возможностей многопроцессорных систем, применяемые к огромным объёмам данных, не дадут никакого прироста производительности по сравнению с однопроцессорной системой и будут работать многие месяцы и даже годы. Это неприемлемо для научных исследований.

В данной ситуации существует несколько решений: увеличение производительности вычислительных единиц, распараллеливание существующих и поиск новых алгоритмов. В первом случае прирост скорости вычислений, относительно времени, в течение которого происходит обновление элементной базы компьютеров, очень небольшой. Поиск новых алгоритмов довольно затруднен, так как для

этого необходима серьёзная научная работа, и результатом не всегда будет являться более быстрый алгоритм.

Наиболее подходящим способом в данной ситуации будет распараллеливание существующих математических алгоритмов. Следует отметить, что не все из них одинаково хорошо параллелизуются, и поэтому вначале необходимо провести анализ существующих алгоритмов на предмет возможности ускорения вычислений. Хорошей оптимизацией будет являться такая, при которой зависимость между числом вычислительных потоков и приростом производительности будет линейной (Рис. 1.).

Важной целью распараллеливания является достижение максимальной загрузки имеющихся вычислительных устройств и блоков. Так как при этом достигается наилучшее соотношение полезного количества используемых ресурсов и затрат на вычисления, что даёт обоснование необходимости проведения данных работ и с экономической точки зрения.

Рис. 1. Линейный прирост производительности

Существует несколько методов распараллеливания программ [1; 2]:

1. Распараллеливание задач. Такая модель фокусируется на процессах или вычислительных потоках. Эти процессы будут отличаться друг от друга, то есть выполнять различные подзадачи. Такие процессы носят название MIMD (multiple instruction, multiple data – множественный поток команд, множественный поток данных) или

MISD (multiple instruction, single data – множественный поток команд, одиночный поток данных).

2. Распараллеливание данных. Эта модель основывается на выполнении операций над данными, которые часто представляются в виде массивов. Ряд процессов будет оперировать этими данными, но независимо друг от друга и на разных их частях. Эти вычисления носят название SIMD (single instruction, multiple data - одиночный поток команд, множественный поток данных).

Неявный параллелизм. Данный вид базируется на технологиях распараллеливания, заложенных в компилятор исходного кода и в среду времени выполнения, и не требует участия программиста или пользователя.

Это означает, что методики распараллеливания для разных алгоритмов могут различаться. Иногда для результата требуется обработать сразу все элементы исходного массива данных. Следовательно, здесь нельзя разделять переменные на составные части. В других случаях результат складывается из операций над отдельными элементами данных, что даёт нам возможность распределить вычисления по узлам сети.

Для оценки эффективность качества распараллеливания могут применяться следующие критерии [4]:

Ускорение S_p:

$$S_p = \frac{T_1}{T_p}, \quad (1)$$

где

T_p - время исполнения распараллеленной программы на p процессорах,

T_1 - время исполнения исходной программы.

В идеальном случае (отсутствие накладных расходов на организацию параллелизма) равно p.

Загруженность $\frac{S_p}{p}$:

$$\frac{S_p}{p} = \frac{T_1}{pT_p} \quad (2)$$

Она показывает долю использования процессоров. В идеальном случае равна 1, или 100%. Эта величина зачастую более наглядно характеризует эффективность параллелизма при разных значениях p.

Особую роль при распределённых вычислениях играет обмен данными. Части системы рассчитывают промежуточные результаты, которыми они в большинстве случаев должны обмениваться. При работе с огромными объёмами данных возрастает количество передаваемой информации, что увеличивает зависимость системы от коммуникационных средств сети, так как обменные операции в параллельных системах выполняются намного медленнее арифметических.

Этот случай является частью более общей проблемы синхронизации. Все вычислительные потоки (узлы) должны иметь механизмы взаимодействия между собой. Они могут реализовываться с помощью примитивов синхронизации, которые представлены в языках высокого уровня мьютексами, семафорами, условными переменными и т.п., либо с помощью операций передачи/приёма данных [3].

Возможны случаи, когда алгоритмы имеют «узкое место», то есть такой участок вычислений, который не может быть распараллелен. При этом может теряться почти весь выигрыш от распределённых вычислений.

Важным моментом при оптимизации вычислений является попадание данных в кэш процессора [6]. Известно, что время отклика (латентность) у разных уровней составляет от 1 до 50 тактов процессора (1-5 тактов для кэша первого уровня L1, 8-20 тактов для L2 и 35-50 тактов для L3). Из-за того, что кэш третьего уровня предназначен для общего пользования всеми ядрами процессора, а также тем, что он наиболее медленный, более хорошей и быстрой оптимизацией будет являться такая, которая задействует только кэши первого и второго уровней. В настоящее время этот объём в среднем равен 256 Кб – 1 Мб, что позволяет хранить в нём до 32000 вещественных переменных двойной точности (double). Оптимизация попадания данных в кэш процессора заключается в таком построении обработки данных, при котором они будут располагаться максимально близко друг к другу в оперативной памяти, что даст воз-

можность механизму предсказаний процессора работать максимально эффективно. Примером может служить последовательная обработка одномерного массива чисел, размещённого в ОП.

Рассмотрим процесс оптимизации и распараллеливания процесса вычислений на примере оператора линейной алгебры – ковариации.

Ковариация (корреляционный момент) в теории вероятностей и математической статистике это мера линейной зависимости двух случайных величин [4; 5].

$$\operatorname{cov}(X_{(n)}, Y_{(n)}) = \frac{1}{n}\sum_{t=1}^{n}(X_t - \overline{X})(Y_t - \overline{Y}), \quad (3)$$

где $X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_n$ - выборки $X_{(n)}, Y_{(n)}$ случайных величин, определенных на одном и том же вероятностном пространстве,

$\overline{X} = \frac{1}{n}\sum_{t=1}^{n} X_t$ и $\overline{Y} = \frac{1}{n}\sum_{t=1}^{n} Y_t$ - среднее значение выборок.

Предположим, что в качестве исходных данных мы имеем не одну пару массивов $X_{(n)}$ и $Y_{(n)}$, а некоторое количество таких векторов, то есть матрицу значений размера $m \times n$.

Первым шагом к распараллеливанию вычислений должен быть анализ алгоритма. Из нашей формулы видим, что на вход оператору подаётся два массива данных, которые представляют собой выборки случайных величин. Простейшие операции производятся над каждой парой элементов исходных векторов и средним значением выборок. В конце вычислений результирующая сумма произведений делится на общее количество элементов в векторе.

Исходя из анализа алгоритма, необходимо разбить его выполнение на несколько этапов, что позволяет выделить:

1. Расчёт средних значений $\overline{X}$ и $\overline{Y}$, и определение разностей $X_t - \overline{X}$ и $Y_t - \overline{Y}$. Они должны быть вычислены в первую очередь, так как каждое дальнейшее действие оперирует с ними.

2. Расчёт суммы произведений.

3. Деление суммы на длину вектора.

До начала всех расчётов нам необходимо распределить данные между всеми узлами сети наиболее равномерно, чтобы обеспечить равную загрузку и, следовательно, максимальное ускорение вычислений. Распределение необходимо осуществлять, «не разрывая»

отдельные массивы случайных величин, которые в нашем случае расположены в столбцах матрицы. Примером системы хранения и распределения данных может служить СУБД SciDB [7].

При этом возможна оптимизация расчёта суммы произведений для кэша процессора описанным выше способом.

Распараллеливание вычислений на одном узле можно организовать с помощью циклов со счётчиком, шагом которого будет являть число доступных процессу потоков.

На следующем шаге встаёт проблема синхронизации потоков в рамках одного узла. Третий этап вычислений зависит от предыдущего, следовательно, необходимо организовать ожидание всех остальных потоков. Это можно сделать с помощью примитива синхронизации вида «барьер», который присутствует в стандарте POSIX threads.

Ещё одним препятствием на пути вычислений может стать ограниченный объём оперативной памяти. Если число узлов сети мало, а объём данных велик, то все данные одного узла не смогут поместиться в ОП компьютера. В этом случае необходимо использовать блочное вычисление результата. То есть последовательно выбираются блоки или куски данных (от англ. chunk – кусок), которые затем обрабатываются и выгружаются из памяти.

Суммарное время вычислений можно определить по формуле:

$$T = t_C + t_N + t_S, \quad (4)$$

где t_C - время обработки данных,

t_N - время обмена данными по сети,

t_S - время на системные расходы (синхронизация, работа с памятью).

Рассмотрим результат работы такого оператора ковариации на нескольких конфигурациях вычислительной системы: 1, 4, 8, 12 и 16 узлах. Исходный массив данных 12000 x 12000 (рис. 2).

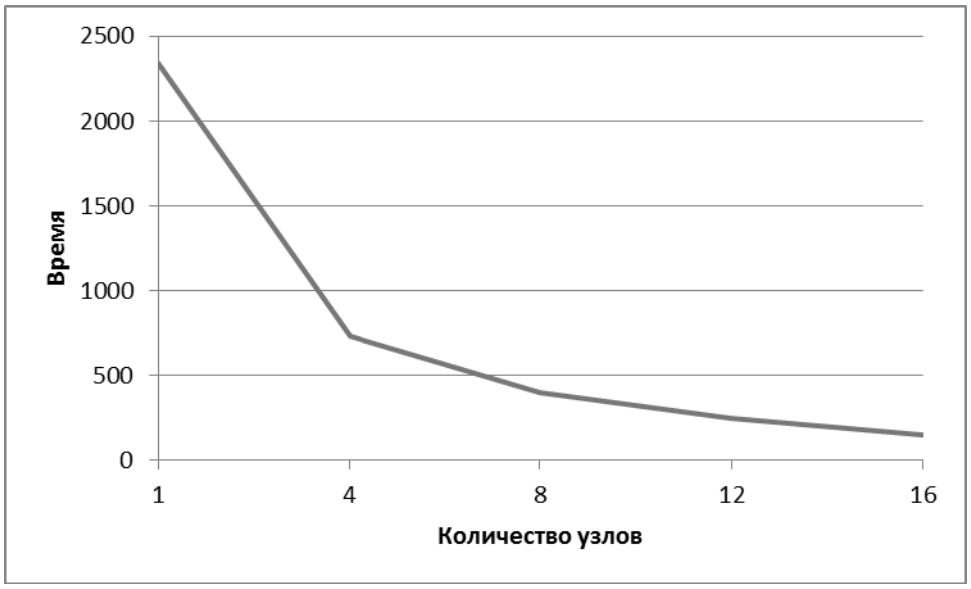

Рис. 2. Зависимость времени вычислений от количества узлов

Точные значения времени сведены в таблицу 1:

Таблица 1

Количество узлов, p	Время работы, T_p
1	2343,712
4	732,41
8	401,8
12	243,2
16	179,5

По данным значениям рассчитаем показатели эффективности качества распараллеливания:

Таблица 2

Количество узлов, p	Ускорение, S_p	Загруженность, $\frac{S_p}{p}$
1	1	1
4	3,2	0,8
8	5,8	0,725
12	9,63	0,8025
16	13,05	0,816

Из таблицы видим, что зависимость между ускорением и количеством узлов не является полностью линейной, но показывает довольно хорошие результаты при наращивании сети. Исходя из пока-

зателей загруженности, можно сделать вывод, что часть времени тратится на обмен данными между узлами, синхронизацию потоков и другие системные нужды. Можно сделать вывод о том, что для массива данных 12000x12000 большее число узлов в сети не даст большого прироста в скорости вычислений. Следовательно, большие кластеры актуальны по соотношению затраченных средств ко времени работы только для больших объёмов данных.

Таким образом, рассмотренные механизмы распараллеливания и оптимизации алгоритмов показывают хорошие результаты при вычислениях математических операторов. Данные подходы подтверждают возможность и необходимость использования распределённых вычислений в таких науках, как физика, астрономия, медицина; при обработке больших объёмов статистических данных, показателей различных датчиков, снимаемых с ускорителей элементарных частиц или других источников, которые оперируют с огромными массивами данных вплоть до 1 Петабайта.

Литература

1. *H. Shan* and *J. Pal Singh*. Comparison of Three Programming Models for Adaptive Applications on the Origin 2000. Journal of Parallel and Distributed Computing, 62:241–266, 2002.

2. *Leslie G. Valiant*, A bridging model for parallel computation, Commun. ACM, volume 33, issue 8, August, 1990, pages 103—111

3. *Daan Leijen, Wolfram Schulte, and Sebastian Burkhardt* The Design of a Task Parallel Library Proceedings of the 24th ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. 2009

4. *Pébay, Philippe* (2008), "Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments", Technical Report SAND2008-6212, Sandia National Laboratories

5. *Ling, Robert F.* (1974). Comparison of Several Algorithms for Computing Sample Means and Variances. Journal of the American Statistical Association, Vol. 69, No. 348, 859-866.

6. *Liang Chen, Deepankar Bairagi and Yuan Lin*. MCFX: A New Parallel Programming Framework for multicore systems. Proceedings of International Supercomputing Conference. 2009

7. *Смирнов А.В., Андрианов Д.Е.* Особенности распределения данных в многомерной СУБД SciDB // Алгоритмы, методы и системы обработки данных: сборник научных статей; Выпуск 14 / Под ред. С.С.Садыкова, Д.Е.Андрианова – М.: «Центр информационных технологий в природопользовании», 2009, - 208 с.: илл.

ПУГИН Е.В.
E-MAIL: EGOR.PUGIN@GMAIL.COM