Масштабируемость параллельных алгоритмов умножения матриц

Факультет компьютерных наук, Университет Миннесоты
Авторы: Anshul Gupta и Vipin Kumar
Перевод: Лямина О.В.

Аннотация

Уже разработаны некоторые алгоритмы параллельного умножения плотных матриц. Для сколь угодно большого числа процессоров, любой из этих алгоритмов или их вариантов может обеспечить примерно линейное ускорение при достаточно больших размеров матриц, и ни один из алгоритмов не может быть четко заявлен как лучший, чем другие. В этой статье мы анализируем производительность и масштабируемость ряда параллельных алгоритмов умножения матрицы, и предсказать условия, при которых каждая формулировка лучше, чем другие. Мы представляем параллельную формулировку для гиперкуба и связанных с ним архитектурой, что работает лучше, чем любая из схем, описанных в литературе до сих пор для широкого диапазона размеров матриц и числа процессоров. Хорошая производительность и масштабируемость для этого алгоритма проверена в результате экспериментов на параллельном компьютере Thinking Machines Corporation's CM-5^{TM} для 512 процессоров. Мы покажем, что специальное оборудование позволяет одновременную связь по всем портам процессоров не улучшает общую масштабируемость алгоритмов матричного умножения на топологии гиперкуб. Мы также обсудим зависимость масштабируемость по технологии зависимых факторов, таких как коммуникация и вычисление скорости и покажем, что при определенных условиях, может быть, лучше иметь параллельный компьютер с K раза больше процессоров, а не с тем же количеством процессоров, K каждой раза быстрее.

1 Введение

[...]

2 Терминология

В этом разделе мы введем терминологию, коротая будет использоваться в остальной части статьи.

Параллельная система: Мы определяем параллельную систему как сочетание параллельного алгоритма и параллельные архитектуры, на которой он реализуется.

Число процессоров, *p:* Число однородных процессоров в многопроцессорной ЭВМ, которые взаимодействуют для решения проблемы.

Размерность задачи, W*:* время, затраченное на решение последовательного алгоритма данной задачи на одном процессоре. Это также

равно сумме всех полезных работ, проделанных всеми процессорами при решении той же задачи в параллельных системах на базе p процессоров. Например, для умножения двух матриц $n \times n$, мы считаем $W = O(n^3)$.

Время параллельного выполнения, T_p**:** Время, затраченное p процессорами на решение данной задачи. Для данной параллельной системы T_p является функцией зависимости размера задачи от числа процессоров.

Параллельное ускорение, S: отношение W к T_p.

Накладные затраты на параллелизм, T_o**:** сумма всех накладных расходов всех процессоров во время параллельного выполнения алгоритма. Она включает в себя расходы на процесс передачи информации, несущественные работы и простой из-за синхронизации и серийных компонентов алгоритма. Для данной параллельной системы, T_o обычно функция зависимости размера задачи от числа процессоров и часто пишется как $T_o(W, p)$. Таким образом, $T_o(W, p) = pT_p - W$.

Эффективность, *E:* отношение S к *p.* Следовательно $E = W / pT_p = 1/(1 + \frac{T_o}{W})$.

Расходы на процесс передачи информации, t_s и t_w: На передачу сообщений в параллельном компьютере, время, необходимое для полной передачи сообщения, содержащие m слов между двумя соседними процессорами считается как $t_s + t_w m$, где t_s - это время запуска сообщение, и t_w (время передачи слова) равно $\frac{y}{B}$, где B - пропускная способность канала связи между процессорами в байт/сек и y - количество байтов в слове.

Для простоты, в этой статье мы предполагаем, что каждая основная арифметическая операция (то есть одна операция умножения с плавающей точкой и одна операция сложения с плавающей точкой в случае умножения матриц) занимает единицу времени. Поэтому t_s и t_w являются связанными показателями передачи данных, которые нормированы по времени вычисления.

3 Изоэффективная метрика масштабируемости

Хорошо известно, что с учетом параллельной архитектурой и проблемы фиксированного размера, ускорение параллельного алгоритма не продолжает возрастать с увеличением числа процессоров, но имеет тенденцию к насыщению или достижению максимума на определенном значении. Для фиксированного размера проблема, ускорение насыщения либо потому, что накладные расходы растут с увеличением числа процессоров или потому, что число процессоров в конечном итоге превышает степень параллелизма неотъемлемого в алгоритме. По целому ряду параллельных систем, для любого числа процессоров *p,* ускорение сколь угодно близко к p может быть получена просто выполнения параллельного алгоритма на достаточно большом размере задачи. Легкость, с которой параллельный алгоритм может достичь ускорений пропорциональны p на

параллельной архитектурой может служить мерой масштабируемость параллельной системы.

Функция изоэффективности является одним из таких метрики масштабируемости, которая является мерой алгоритмической способности эффективно использовать большее число процессоров на параллельной архитектуре. Функция изоэффективности сочетания параллельного алгоритма и параллельные архитектуры зависит от размерности задачи и числа процессоров, необходимых для поддержания фиксированной эффективности или достижения ускорения, увеличивающейся пропорционально с ростом числа процессоров. Эффективность параллельной системы считается, как $E = \frac{W}{W + T_o(W, p)}$. Если параллельная система используется для решения задачи с фиксированный размер W, то эффективность уменьшается по мере увеличения *p*. Причина в том, что общие накладных расходы $T_o(W, p)$ возрастают с *p*. Для многих параллельных систем, для фиксированного числа *p,* если размерность задачи *W* увеличивается, то эффективность возрастает из-за числа *p*, $T_o(W, p)$ растет медленнее, чем $O(W)$. Для этих параллельных систем, эффективность может поддерживаться на нужном значение (от 0 до 1) для повышения числа *p,* при условии, что *W* также увеличилось. Мы называем такие системы масштабируемые параллельные системы. Обратите внимание, что для данного параллельного алгоритма в условии различных параллельных архитектур, *W* может увеличиваться с различной скоростью относительно *p* в целях поддержания фиксированного эффективности. Например, в некоторых случаях, *W,* возможно, необходимо увеличить в геометрической прогрессии относительно числа *p,* для сохранения эффективности от падения при увеличении числа *p*. Такая параллельная система плохо масштабируема, потому что будет трудно получить хорошее ускорение для большого числа процессоров, если только размер решаемой задачи не является чрезвычайно большим. С другой стороны, если W нужно увеличить только линейно относительно *p,* то параллельная система является хорошо масштабируемой и может легко поддерживать ускорение увеличиваясь линейно по отношению к числу процессоров для разумно растущей размерности задачи. Функции изоэффективности некоторых общих параллельных систем являются полиномиальными функциями *p*: то есть они являются $O(p^x)$, где $x \geq 1$. Малая мощность *p* в функции изоэффективности указывает на высокую масштабируемость.

Если параллельная система берет на себя общие накладные расходы $T_o(W, p)$, где *p* число процессоров в параллельной системе и W является размерностью задачи, то эффективность системы определяется как $E = \frac{1}{1 + \frac{T_o(W, p)}{W}}$.В целях поддержания постоянной эффективности *W* должна быть пропорциональна $T_o(W, p)$ или должно быть выполнено соотношение:

$$W = KT_o(W, p) \qquad (1)$$

Здесь $K = \frac{E}{1-E}$ является постоянно зависящая от эффективности. Уравнение (1) является центральным отношением, которое используется для определения функции изоэффективности. Это достигается путем абстрагирования W как функции от *p* путем алгебраических преобразований уравнения (1). Если размерность задачи нужно увеличить так быстро, как $f_E(p)$ для поддержания эффективности , тогда $f_E(p)$ определяется как функция изоэффективности сочетания параллельного алгоритма и архитектуры для эффективности *E*.

Изоэффективный анализа был изобретен для очень полезной характеризации масштабируемости различных параллельных систем. Важной особенностью изоэффективного анализа является то, что в одно выражение, он лаконично фиксирует последствия характеристики параллельного алгоритма, а также параллельной архитектурой, на которых она осуществляется. Проведя изоэффективный анализ, можно протестировать производительность параллельной программы на нескольких процессорах, а затем прогнозировать свою деятельность на большее число процессоров. Но полезность изоэффективного анализа не ограничивается для предсказания воздействия на выполнение большего числа процессоров. Также может быть использована для исследования поведения параллельной системы относительно изменениям в другом оборудовании связанных с такими параметрами, как скорость процессоров и каналов связи данных.

4 Параллельные алгоритмы умножения матриц

В этом разделе мы опишем некоторые известные параллельные алгоритмы умножения матриц и дадим их параллельное время выполнения.

4.1 Простой алгоритм

Рассмотрим логическую двумерную сеть из *p* процессоров (с $\sqrt{p}$ строк и $\sqrt{p}$ столбцов), на которой две $n \times n$ матрицы *A* и *B* должны быть умножены для получения на выходе матрицы *C*. Пусть $n \geq \sqrt{p}$. Матрицы делятся на подблоки, каждая из которых размером $\frac{n}{\times}\sqrt{p} \times \frac{n}{\times}\sqrt{p}$, которые естественно отражаются на массиве процессора. Алгоритм может быть реализован на гиперкуб путем внедрения этой сетки процессора в него. На первом шаге алгоритма, каждый процессор получает все эти элементы двух матриц, которые необходимы для создания $\frac{n^2}{p}$ элементов выходной матрицы, которая должна находиться в этом процессоре. Это включает в себя все трансляции из $\frac{n^2}{p}$ элементов матрицы *A* среди $\sqrt{p}$ процессоров каждого ряда

процессоров и того же размера блоков матрицы B среди $\sqrt{p}$ процессоров каждого столбца, которые могут быть достигнуты $2t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}$ время. После, каждый процессор получает все необходимые ему данные, он умножает $\sqrt{p}$ пар подблоков двух матриц для вычисления $\frac{n^2}{p}$ элементов выходной матрицы. Предполагая, что операции сложения и умножения занимают единицу времени. умножения фаза может быть завершена в - единицы времени:. Таким образом, общее время выполнения параллельного алгоритма задается уравнением

$$T_p = \frac{n^3}{p} + 2t_s \log p + 2t_w \frac{n^2}{\sqrt{p}} \qquad (2)$$

Этот алгоритм неэффективный, с точки зрения использования памяти. Требования к памяти для каждого процессора равна $O(\frac{n^2}{\sqrt{p}})$ и, таким образом общее требование к памяти равно $O(n^2\sqrt{p})$ слова, в то время как требования для последовательного алгоритма является $O(n^2)$.