УДК 004.272.34+004.415

ЗАКОН АМДАЛА И ГРАНИЦЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Д.И. Черемисинов
Ведущий научный сотрудник ОИПИ НАНБ
кандидат технических наук, доцент

Объединений институт проблем информатики Национальной академии наук Беларуси, Республика Беларусь
E-mail: cher@newman.bas-net.by

Д.И. Черемисинов
Окончил Томский государственный университет, кандидат технических наук, доцент. Работает в ОИПИ НАН Беларуси в должности ведущего научного сотрудника и Белорусском государственном университете информатики и радиоэлектроники в должности доцента.
Круг научных интересов: программирование, логическое проектирование и тестирование дискретных систем управления, реализация параллельных алгоритмов управления.

Аннотация. В области программирования параллелизм является средством повышения эффективности вычислений. Эффективность параллельной программы традиционно связывают с достижением более высокой производительности по сравнению с ее последовательным вариантом. Закон Амдала часто используется в параллельных вычислениях для прогнозирования теоретического ускорения при использовании нескольких процессоров. Закон Амдала гласит: «Общая скорость параллельной системы определяется самым медленным компонентом». Обсуждается процесс решения на параллельных вычислительных системах задачи булевой выполнимости (SAT). SAT является первой проблемой, которая оказалась NP-полной. Это означает, что все проблемы в классе сложности NP, который включает в себя широкий спектр естественных решений и оптимизационных задач, имеют сложность не менее, чем SAT. NP-полные задачи представляют интерес, поскольку все они не имеют теоретических высокопараллельных решений. Было обнаружено, что сложность задачи SAT «в среднем» зависит от того, разрешим ли экземпляр задачи SAT. Если проблема разрешима, то доля «сложных» экземпляров мала, а эффективный алгоритм имеет умеренную оценку сложности. Эта проблема находится в классе задач поиска, которые по своей сути недетерминированы. Этим объясняется факт наблюдения сверхлинейного ускорения параллельной программы. И наоборот, если проблема неразрешима, доля простых примеров мала, а эффективный алгоритм имеет оценку сложности наихудшего случая. Это экземпляры задачи выполнимости имеют детерминированный характер процесса решения. В этом контексте, если накладные расходы на деление области значений оптимизированы, ускорение подчиняется закону Амдала, и возможны близкие к линейным ускорения.

Ключевые слова: закон Амдала, параллельные вычисления, SAT-задачи, суперлинейное ускорение.

Введение. Сложность алгоритмов измеряется необходимыми ресурсами, в основном это продолжительность вычислений или необходимый объём памяти. Оценка вычислительной сложности задает зависимость требований алгоритма к времени и объёму памяти от объёма входных данных.

«Эффективным» алгоритмом, считается любой полиномиальный алгоритм, т. е. алгоритм, время выполнения которого ограничено некоторым полиномом от длины записи входных данных. Задачи, разрешимые такими алгоритмами, образуют класс, который

обозначается через P. Класс NP, в который попало большинство известных дискретных задач, удачно моделирует задачи, решаемые переборными алгоритмами. Задачи, принадлежащие классу NP, можно охарактеризовать как задачи, имеющие «короткое доказательство». Например, для знаменитой проблемы «Выполнимость – SAT» (от satisfiability), заключающейся в проверке наличия для формулы, заданной в виде к.н.ф., набора значений булевых переменных, на которых формула принимает значение «1», для доказательства решения задачи достаточно предъявить такой набор.

Было установлено существование «самых трудных» задач в классе NP, названных N-полными. Это те задачи, к которым полиномиально сводится любая задача из класса NP. Упомянутые классы составляют основу теории сложности вычислений, предметом которой являются пределы вычислений. В настоящее время все согласны с тем, что наихудшие экземпляры класса N-полных задач не могут быть решены за полиномиальное время без серьезного теоретического прорыва, а накопленные в этой области знания позволяют быстро оценивать и диагностировать предлагаемые алгоритмы.

Стоит отметить разницу между сложностью алгоритма и сложностью алгоритмической задачи. При анализе по худшему случаю получение нижних оценок сложности конкретного алгоритма почти всегда оказывается посильной задачей. Для этого достаточно подобрать набор входных данных, на которых алгоритм будет работать долго. Совсем другая ситуация со сложностью задач. Для получения нижних оценок сложности задачи требуется доказать, что не существует алгоритма ее решения со сложностью не выше заданной. Ввиду необозримого множества алгоритмов эта задача находится вне пределов возможностей современной теории сложности. Теория сложности выработала и ряд прагматических рекомендаций для исследователей, занимающихся решением прикладных задач. В тех случаях, когда интересующая разработчика практических алгоритмов задача оказывается N-полной, имеет смысл попробовать построить эффективный алгоритм для какой-либо класса ее модификаций или частных случаев, приемлемых с практической точки зрения.

Параллельные вычисления. В теории сложности класс NC («класс Ника» в честь Ника Пиппенгера, который первым исследовал сложность параллельных вычислений) представляет собой набор задач, разрешимых за полилогарифмическое время на параллельном компьютере с полиномиальным числом процессоров. Другими словами, проблема принадлежит классу NC, если существуют такие константы c и k , что ее можно решить за время O (log_c n), используя O (n^k) параллельных процессоров.

Подобно тому, как класс P составляют поддающиеся решению проблемы, так и класс NC можно рассматривать как класс проблем, которые могут быть эффективно решены на параллельном компьютере. Класс NC является подмножеством P, потому что полилогарифмические параллельные вычисления могут быть смоделированы последовательными полиномиальными вычислениями. Неизвестно, является ли $NC = P$, но большинство исследователей подозревают, что это неверно, а это означает, что, вероятно, существуют некоторые проблемы, которые являются «по своей сути последовательными» и не могут быть значительно ускорены с помощью параллелизма. Поскольку любая задача в классе P может быть эффективно сведена (NC-редуцирована) к некоторой N-полной задаче, нахождение P-полной задачи внутри класса NC противоречило бы $P = NC$. Таким образом, если считать NP-полную задачу как «вероятно не решаемую за полиномиальное время», так и класс P-полных задач, при использовании NC-редуцирования, могут рассматривать как «вероятно, не распараллеливаемые» (неэффективно выполняемые параллельно) или «вероятно, последовательные». Следовательно, увеличение количества процессоров, решающих задачу, не может значительно ускорить ее решение для задач из класса -полных.

Так как целью параллельных вычислений является сокращение времени работы алгоритма, то параллельные алгоритмы можно автоматически извлекать из последовательных

программ, разрабатывая компиляторы для параллельных вычислений. Проблема поиска параллельных алгоритмов в этом случае является проблемой оптимизации последовательных программ. Это устоявшаяся область исследований, как в научных кругах, так и в промышленности. За последние 20-30 лет здесь добились заметных успехов, но ограничения такого подхода также хорошо известны. Для многих задач успех в поиске асимптотически эффективных параллельных алгоритмов (вручную или с помощью умного компилятора) решил бы основные открытые проблемы теории сложности. Для некоторых задач эффективные параллельные алгоритмы известны, но настолько отличаются от наилучших последовательных алгоритмов, что нет никакой надежды на то, что компилятор превратит последовательный алгоритм в хороший параллельный алгоритм [1]. Компиляторы не могут изобрести совершенно новые алгоритмы.

Закон Амдала. Несмотря на мощь и перспективы параллельных вычислений, существуют ограничения на возможности, достижимые при использовании компьютеров с параллельной архитектурой. Распараллеливание последовательной программы выполняется для повышения производительности вычислений. Ограничение на сокращение вычислительного времени, достижимое за счет использования множества процессоров, известно как закон Амдала [1].

Параметром оценки эффективности при решении конкретной задачи с применением суперкомпьютера является коэффициент – ускорение S (Speedup),

$$S = t(1) / t(N)$$

где $t(1)$ – время, затраченное на решение задачи последовательным алгоритмом на одном процессоре, а $t(N)$ – время решения той же задачи параллельным алгоритмом с использованием N процессоров мультипроцессорной вычислительной системы. Чем больше значение S, тем выше эффективность применения суперкомпьютера для решения задачи.

Закон Амдала является оценкой идеального ускорения, которое может произойти, когда последовательная программа модифицирована для параллельной работы. Часто этот закон трактуется как универсальный, применимый к оценке любых вычислений параллельным оборудованием. Однако чтобы эта оценка ускорения вычислений была справедливой, необходимо, выполнение определенных условий. Главное условие применимости закона Амдала состоит в том, чтобы размер вычислительной работы при распараллеливании оставался неизменным.

Предположим, что последовательная программа модифицирована для параллельной работы. Далее, предположим, что объем работы, выполняемой программой, не изменяется в ее параллельной версии. При распараллеливании имеются части программы, которые распараллелить нельзя. Поэтому в параллельной программе некоторая доля вычислений выполняется последовательно. Пусть f – это доля таких вычислений в общем объеме, $0 < f < 1$. Предположим, что распараллеливание является «идеальным», то есть, при числе процессоров P, которые мы используем, ускорение параллельной части программы будет P . Максимальное ускорение S, достижимое на вычислительной системе из P процессоров, можно оценить при помощи следующей формулы (закон Амдала) [2]:

$$S = \frac{1}{(f + (1 - f)P^{(-1)})}$$

Из закона Амдала следует, что всегда при любом сколь угодно большом числе процессоров, независимо от качества реализации параллельной части кода, $S < 1/f$. Предположим, например, что мы можем распараллелить 90% последовательной программы.

В этом случае $S \leq 10$. Это говорит о том, что даже если мы проделали отличную работу по распараллеливанию 90% программы, и даже если у нас, скажем, 1000 ядер, мы никогда не получим ускорение лучше, чем 10. Ускорение равное P при числе процессоров P называется линейным. Ускорение в соответствии с законом Амдала всегда меньше линейного.

Тем не менее, суперлинейное ускорение обнаружено экспериментально при решении нескольких классов задач. Распространено мнение, что суперлинейное ускорение – это результат некорректной методики измерений времен T(1) и T(N) [3]. Считается, что одной из возможных причин служит эффект различия в алгоритмах кэширования вследствие различий оборудования и операционных систем.

Задача выполнимости КНФ. Булева задача выполнимости SAT (иногда называемая пропозициональная проблемой выполнимости) является проблемой вычисления интерпретации, которая удовлетворяет заданную булева формулу. Дано множество булевых переменных $X = \{x_1, x_2, \ldots, x_n\}$; любая переменная x_j или ее инверсия $\overline{x_j}$ называется литералом $\widetilde{x_j}$ от этой переменной; дизъюнкция литералов от различных переменных называется элементарной дизъюнкцией $d_i(X) = \widetilde{x_{j1}} \vee \widetilde{x_{j2}} \vee \ldots \vee \widetilde{x_{jk}}$; число k_i литералов в дизъюнкции называется ее рангом. Конъюнктивной нормальной формой (КНФ) называется конъюнкция некоторого множества элементарных дизъюнкций $K(X) = d_1(X) \wedge d_2(X) \wedge \ldots \wedge d_m(X)$; дизъюнкции $d_i(X)$ называются членами КНФ. Набор значений переменных, обращающий КНФ в истинное высказывание $K(X) = 1$, – это интерпретация формулы. Такой набор (значение вектора X) называется выполняющим, и если он существует, то КНФ называется выполнимой, в противном случае – невыполнимой. В частности, выполнимой является пустая КНФ (не содержащая ни одного члена), так как она тождественно истинна по определению. Пустая дизъюнкция, по определению, тождественно ложна – следовательно, невыполнима.

SAT была первой известной NP-полной проблемой. NP-полнота задачи означает экспоненциальную оценку времени выполнения наихудших случаев. Доказательство NP-полноты некоторой задачи заключается в полиномиальном сведении задачи SAT к данной задаче, возможно в несколько шагов, то есть с использованием нескольких промежуточных задач. Не известен алгоритм, который бы эффективно (т.е. с оценкой времени выполнения наихудших случаев лучшей экспоненциальной) решал каждую проблему SAT, и обычно считается, что такого алгоритма не существует. Отсюда следует, что все известные алгоритмы решения задачи SAT имеют эту оценку достижимой.

Поиск с возвращением (backtracking search procedure) – универсальный метод, позволяющий исследовать пространство (экспоненциальной мощности) значений переменных задачи SAT. Он состоит в пошаговом формировании решения; на каждом шаге решение доопределяется, получая некоторое приращение (в данном случае определяется значение одной или нескольких переменных). Приращения могут быть безальтернативными (например, элементарная дизъюнкция первого ранга выполнима единственным способом) или альтернативными, когда необходимо испытать несколько различных шагов. Каждое доопределение приводит к изменению ситуации (в нашем случае делает истинными некоторые члены КНФ). В новой ситуации мы имеем ту же задачу, но меньшей размерности, и поиск продолжается. В точках ветвления поиска необходимо запомнить все остающиеся альтернативы для возвращения к ним из тупика; цепочка точек ветвления образует стек. Тупиком называется ситуация, в которой обнаруживается, что на данной ветви поиска задача не имеет решения (в нашем случае признаком невыполнимости являются противоречивые безальтернативные приращения или обнаружение пустой дизъюнкции среди членов КНФ). Выход из тупика состоит в возвращении к последней точке ветвления и выборе очередной альтернативы. Непустое множество остающихся альтернатив сохраняется в стеке, по исчерпании же альтернатов последняя запись в стеке уничтожается. Если при возвращении из тупика обнаруживается, что стек пуст, то поиск заканчивается с ответом, что решения не

существует. Когда же очередное доопределение приводит к конечному результату (в нашем случае – к пустой КНФ), поиск заканчивается с положительным ответом. Быстродействие алгоритма, реализующего поиск с возвращением, во многом зависит от удачного или неудачного способа формирования набора альтернатив в точках ветвления. На сегодняшний день наиболее эффективны так называемые «CDCL-решатели» (от Conflict Driven Clause Learning), базирующиеся на процедуре поиска с возвращением. В частности, CDCL-SAT-решатель MiniSAT, представляет собой общедоступную последовательную программу длиной около 600 строк.

Трудность наихудших случаев и большинства случаев. Из теории вычислительной сложности алгоритмов известно [5], что, хотя все NP-полные задачи эквивалентны в отношении сложности наихудших случаев, отдельные экземпляры задачи SAT могут значительно отличаться по своей сложности от сложности наихудших случаев. Сложность проблемы «в среднем» определяется следующим образом: существует эффективный алгоритм, который решает проблему с некоторой оценкой времени выполнения для всех экземпляров задачи, за исключением незначительной доли «сложных» экземпляров. В зависимости от доли часто встречающихся экземпляров задачи различают «умеренно сложный» алгоритм (т. е. такой, в котором вычисления трудны на небольшой части экземпляров) и алгоритм высокой сложности, которой не лучше алгоритма, случайно угадывающего ответ.

С практической точки зрения интересна оценка сложности задачи SAT для ее экземпляров, возникающих в «реальной жизни». Примеры задачи SAT в автоматизации проектирования СБИС включают в себя формальную проверку эквивалентности схем, автоматическую генерацию тестов, трассирование соединений в ПЛИС, планирование задач, и т. д. Известно, что генерация сложных примеров NP-полных задач требует суперполиномиальных вычислений. Трудно представить, как неодушевленный мир будет выполнять огромное количество вычислений, необходимых для создания сложного экземпляра. Криптография, таким образом, исключается как источник задач «реальной жизни».

Задача SAT является самовосстанавливаемой (self-reducible), то есть каждый алгоритм, который правильно отвечает, что экземпляр SAT является разрешимым, можно использовать для поиска выполняющего набора значений вектора X. Сложность задачи SAT «в среднем» зависит от того, является ли экземпляр SAT разрешимым. Если задача разрешима, то доля «сложных» экземпляров невелика и эффективный алгоритм имеет умеренную оценку сложности. Наоборот, если задача неразрешима, невелика доля простых экземпляров и эффективный алгоритм имеет оценку сложности наихудшего случая. Эти характеристики сложности задачи SAT «в среднем» можно обнаружить в результатах экспериментов на псевдослучайных КНФ [6]. Задача SAT «в среднем» легка на большинстве экземпляров, если КНФ выполнима, и трудна на большинстве экземпляров, когда КНФ не выполнима.

Сложность параллельных вычислений SAT. Используя точки ветвления в поиске с возвращением можно получить декомпозицию исходной SAT-задач на несколько SAT-задач меньшей размерности. Выберем некоторую переменную $x_{i1} \in X$ и расщепим исходную КНФ $K(X) = \mathrm{d}_1(\mathrm{X}) \wedge \mathrm{d}_2(\mathrm{X}) \wedge \ldots \wedge \mathrm{d}_\mathrm{m}(\mathrm{X})$ на две – $K|_{xi} = 0$ и $K|_{xi} = 1$. Здесь через $K|_{xi} = \alpha$ обозначена КНФ, полученная в результате подстановки в K значения $\alpha \in \{0,1\}$ переменной x (с последующим выполнением возможных элементарных булевых операций). КНФ $K|_{xi} = 0$ и $K|_{xi} = 1$ могут быть точно так же расщеплены по некоторым переменным из $X \backslash \{x_{i1}\}$. В результате некоторого числа шагов данного процесса будет построено двоичное дерево. Решив все SAT-задачи, соответствующие листьям данного дерева (часть из них могут оказаться тривиальными), мы решим и исходную SAT-задачу. Действительно, по выполняющему набору КНФ, соответствующей некоторому листу, эффективно строится набор, выполняющий исходную КНФ K, а если все КНФ, соответствующие листьям,

невыполнимы, то невыполнима и K. Очевидно, что SAT-задачи, соответствующие листьям описанного дерева, можно решать на независимых узлах параллельной вычислительной среды.

Если КНФ K невыполнима, то время вычислений определяется временем работы узла, выполняющего наибольший объем работы. Если имеется несбалансированность загрузки процессоров, то часть процессоров вынуждена простаивать, пока остальные заканчивают свою вычислительную работу. Время решения задачи определяется временем «самого медленного» (последнего из закончивших работу) процессора, а общее время зависит от равномерности распределения нагрузки между процессорами при декомпозиции исходной задачи на частные подзадачи. Объем вычислительной работы параллельного и последовательного алгоритмов одинаков, и ускорение при идеальной сбалансированности определяется законом Амдала. Несбалансированность нагрузки процессоров делает ускорение по Амдалу недостижимой верхней оценкой. В этом случае (класс J в [7]) исходная задача решается узлами параллельной вычислительной среды совместно (jointly).

Если КНФ K выполнима, то решение любой частной задачи, полученное любым из узлов в ходе соревнования (competition) по времени с другими, является решением общей задачи (класс C в [7]). Когда какой-либо узел находит решение, все остальные должны прекратить работу, т.е. время решения задачи определяется «самым быстрым» узлом. Объем выполняемой работы зависит от сбалансированности нагрузки. При идеальной балансировке объем вычислительной работы параллельного и последовательного алгоритмов одинаков. Чем хуже балансировка, тем меньший объем вычислительной работы выполняет параллельный алгоритм. В этом случае работа параллельной системы характеризуется суперлинейным ускорением (рис. 1) , его величина случайна и может быть очень большой. Суперлинейное ускорение в SAT задачах наблюдалось, например, в конкурсах параллельных SAT-решателей в решениях параллельной программы ManySAT [8].

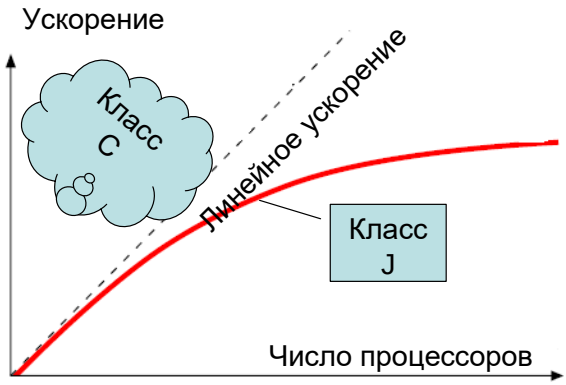

Рисунок *1*. – Поведение ускорения для выполнимой и невыполнимой КНФ

Если выполняющий вектор отсутствует, то процесс решения из класса C переходит в класс J, так как для доказательства отсутствия решений каждый из подчиненных процессоров должен обследовать все свое подпространство до конца.

Заключение. В работе [9] сообщается о суперлинейном ускорении решения задачи SAT и делается попытка поставить под сомнение методику измерения ускорения. Используя компьютер с несколькими ядрами было достигнуто ускорение по сравнению с последовательной программой в ~200 раз. Утверждается, что ускорение было измерено некорректно: сравнивались эвристический приближенный (suboptimal sequential) последовательный алгоритм с эвристическим приближенным параллельным алгоритмом. Суперлинейность объяснялась разницей в эвристиках. В этой работе описана класс

экземпляров задачи SAT, в которых суперлинейность объясняется разницей в работе (объеме исследованного пространства решений) параллельной и последовательной программы, использующих точные алгоритмы. Использование программ для параллельных вычислений действительно позволяют превратить кремний (предварительно разработанные процессорные ядра) в суперкомпьютер без увеличения тактовой частоты и мощности рассеивание на ядро. СБИС этого суперкомпьютера стоят на вес дороже, чем золота. Это оставляет далеко позади мечты древних алхимиков превратить свинец в золото, так как из песка делается что-то намного дороже золота.

Список литературы

[1] Greenlaw R., Hoover H. J., Ruzzo W. L. Limits to Parallel Computation: P-completeness Theory In URL: https://homes.cs.washinaton.edu/~ruzzo/papers/limits.pdf (access date: 4.11.2019).

[2] Amdahl G.M. Validity of the single processor approach to achieving large-scale computing capabilities. AFIPS Conf Proc. 1967.– pp. 483–485.

[3] Черемисинов Д.И. О суперлинейном ускорении вычислений на кластерном компьютере // Третья Международная конференция «Суперкомпьютерные системы и их применение» (SSA'2010): доклады конференции (25-27 мая 2010 года, Минск) : в двух томах. – Минск: ОИПИ НАН Беларуси, 2010. –Т.1. – С. 165-170.

[4] Schreiner, W. Superlinear Speedup / W. Schreiner // [Electronic resource]. – Mode of access: http://www.risc.uni-linz.ac.at/education/courses/ws96/intropar/theory/index_5.html. – Date of access: 01.02.2010.

[5] Arora S., Barak B. Computational Complexity. A Modern Approach – Cambridge University Press, 2009. – 605 pp.

[6] Уткин, А.А. Экспериментальное исследование задачи о выполнимости / А.А Уткин // Препринт Ин-т техн.кибернетики АН БССР – Минск, 1988.– 30 с.

[7] Параллельные логико-комбинаторные вычисления в среде MPI / Н.Р. Торопов // Информатика. – 2005. – № 3. – С. 82–90.

[8] Hamadi Y. ManySAT: a Parallel SAT Solver / Y. Hamadi, S. Jabbour, L. Sais // Journal on Satisfiability, Boolean Modeling and Computation. – 2009. – Vol. 6. – P. 245-262.

[9] Markov I. L. Know your limits (review of "limits to parallel computation: p-completeness theory"; greenlaw, r., et al; 1995) [book review] // IEEE Design & Test – Volume: 30 , Issue: 1 , Feb. 2013. – P. 78-83.

AMDAHL'S LAW AND BOUNDS ON SPEEDUP

D.I. Cheremisinov
Leading researcher of UIIP of NAS of Belarus,
candidate of technical sciences, associate professor

United Institute of Informatics Problems of National Academy of Sciences of Belarus, Republic of Belarus
cher@newman.bas-net.by

Abstract. Amdahl's law is often used in parallel computing to predict the theoretical speedup when using multiple processors. Amdahl's law states, "Overall parallel system speed is governed by the slowest component". The solving process for examples of the Boolean satisfiability problem (SAT) on parallel computing systems is discussed. SAT is the first problem that was proven to be NP-complete. This means that all problems in the complexity class NP, which includes a wide range of natural decision and optimization problems, are at most as difficult to solve as SAT. NP-complete problems are of interest because they all appear to lack theoretical highly parallel solutions. It was found that the complexity of the SAT task "on average" depends on whether the SAT instance is solvable. If the problem is solvable, then the proportion of "complex" instances is small and an effective algorithm has a moderate complexity estimate. This problem is in a class of search problems what are intrinsically nondeterministic. They encountered this fact in the form of observing superlinear speed-ups. Conversely, if the problem is unsolvable, the proportion of simple instances is small and an effective algorithm has an estimate of the complexity of the worst case. This computational problems solved on a computer have a deterministic nature. In that context, if the overhead of dividing is well controlled, speedups are on Amdahl's law and linear or close to linear speedups are possible.

Key words: Amdahl's law, High performance computing, Boolean satisfiability problem (SAT), Superlinear speedup.