Балансировка нагрузки на основе оценок алгоритмической сложности подзадач[1]

Бо Тянь, М.А. Посыпкин, И.Х. Сигал

Аннотация. В работе предложена новая статическая стратегия распределения вычислительной нагрузки между процессорами для параллельного метода ветвей и границ на основе оценок алгоритмической сложности подзадач, возникающих в процессе решения. Предлагаемая стратегия может быть использована на параллельных системах с низкой связностью при проблематичности динамической балансировки нагрузки. Экспериментальные результаты показали преимущество предлагаемой стратегии по сравнению с другими рассмотренными стратегиями.

Ключевые слова: метод ветвей и границ, параллельные вычислительные системы, балансировка нагрузки, оценки вычислительной сложности подзадач.

Введение

Основным источником вычислительной мощности в современных высокопроизводительных системах является большое число одновременно работающих процессоров (ядер). Суперкомпьютеры, занимающие первые позиции в списке «Top 500» [1] содержат сотни тысяч и даже миллионы ядер. Поддержка быстрой и надежной связи между таким числом ядер является трудной задачей. Наиболее мощные суперкомпьютеры нередко используют различные сопроцессоры, например графические карты общего назначения (GP GPU) [2] или процессоры нового поколения, например, Intel XeonPhi [3]. При этом взаимодействие между ядрами в пределах одной графической карты или тем более между картами, расположенными в разных модулях, становится длительной операцией, которая может существенно замедлять процесс вычислений.

В последнее время также получили распространение распределенные вычислительные системы слабой связности. В качестве примера можно привести грид-системы персональных компьютеров [4], в которых непосредственное взаимодействие между вычислительными узлами затруднено, или распределенные вычислительные системы на основе Web-сервисов [5], в которых такое взаимодействие не предполагается архитектурой системы. В этой связи приобретают актуальность методы, ориентированные на модель, сводящую к минимуму взаимодействие между параллельными потоками программы.

В прошлом в большинстве параллельных платформ устанавливалось относительно небольшое количество процессоров, что позволяло обеспечить однородные по скорости высокопроизводительные связи между ними. Поэтому абсолютное большинство параллельных реализаций метода ветвей и границы (МВГ) используют динамическую балансировку нагрузки, которая не является проблемой в высокопроизводительных системах с высокой связностью. Однако к системам с ограниченны-

[1]НИР поддержана программой Китайского Стипендиального Совета (№ 201308090004), грантами РФФИ № 13-07-00291-А, 15-07-03102-А.

ми возможностями взаимодействия необходим другой подход, основанный на статическом распределении вычислительной нагрузки: вычисления должны быть заранее разбиты на некоторое число независимых вычислительных блоков с целью их дальнейшего распределения по параллельным вычислительным устройствам.

Если все процессоры имеют одинаковую производительность, разделение задачи на расчетные блоки равной мощности обеспечивает наилучшее распределение нагрузки. Однако декомпозиция дерева ветвлений, порождаемого МВГ, на равные поддеревья является сложной задачей, так как размер и структура дерева заранее не известны.

В данной работе предлагается новая стратегия распределения нагрузки, основанная на модификации циклического («round-robin») распределения нагрузки, применяемого к списку подзадач, упорядоченных согласно убыванию сложности. Экспериментально показывается, что если удается достоверно оценить сложность подзадач, то предложенная стратегия превосходит ряд других рассматриваемых подходов. Если прогноз времени выполнения проблематичен, случайная стратегия представляется наилучшим вариантом.

Работа имеет следующую структуру: в разделе 1 приводится обзор подходов к параллельной реализации метода ветвей и границ. Подробная информация о реализации предлагаемого метода в разделе 2. Результаты экспериментов представлены в разделе 3.

1. Предпосылки и соответствующие работы

Рассматриваемые в работе методы, направлены на решение задач глобальной оптимизации:

$$f(x) \to \max \; s.t. \; x \in X. \qquad (1)$$

Одним из самых популярных подходов к решению задачи (1) является метод ветвей и границ [6], для которого необходимы две процедуры: ветвление и нахождение оценок (границ). Процедура ветвления состоит в разбиении множества допустимых значений X на подмножества меньших размеров. Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подмножествах допустимых значений параметров задачи (1).

Для различных задач глобальной оптимизации разработаны многочисленные варианты МВГ. При решении многих прикладных задач МВГ требует количество вычислительных ресурсов, превосходящее возможности однопроцессорной рабочей станции. В связи с этим для ускорения процесса решения большое внимание уделяется параллельной реализации данного метода.

Конечной целью распараллеливания МВГ является сокращение общего времени работы (Makespan), которое определяется как разница между моментами начала и завершения всего процесса вычислений. Главной проблемой при этом является достижение хорошего баланса между вычислительными узлами в целях минимизации времени простоя, в течение которого процессоры ожидают информацию для продолжения вычислений.

Балансировка нагрузки для многопроцессорных систем с высокой степенью связности всесторонне изучена. Большинство эффективных схем распараллеливания используют интенсивные коммуникации для достижения равномерного распределения заданий по процессорам. Реализации МВГ в многопроцессорных системах с общей памятью [7] является относительно простой задачей, так как каждый процессор имеет доступ к общей памяти и число ядер невелико. Алгоритмы балансировки, ориентированные на подобные системы [8-10], основаны на использовании одного или нескольких списков подзадач. Потоки взаимодействуют между собой, не допуская опустошения списков, тем самым, поддерживая постоянную загрузку процессоров. Процесс решения останавливается, когда становятся пустыми списки на всех процессорах.

Более сложной является эффективная балансировка на многопроцессорных системах с распределенной памятью. Для таких систем наиболее популярны динамические методы балансировки нагрузки, основанные на передаче подзадач между параллельно работающими процессорами по ходу вычислений. Динамическая балансировка предусматривает перераспределение вычислительной нагрузки на узлы во время

выполнения приложения. Динамические методы обычно применяют, если время, необходимое на балансировку, намного меньше времени выполнения задачи [9, 11]. Балансировка нагрузки может выполняться программно или аппаратно, централизованно или децентрализовано.

В случае, когда взаимодействие между процессорами нежелательно и должно быть сведено к минимуму (GP GPU, грид-системы из персональных компьютеров и т.п.) предпочтительнее статические методы распределения нагрузки. Из различных алгоритмов статического разделения нагрузки методы, наиболее близкие к рассматриваемым в данной статье, предложены в [12]. Алгоритм SelfSplit, предложенный в [12], заставляет все процессоры выполнять одинаковую последовательность на начальных этапах, после чего каждый процессор решает до конца нерешенные подзадачи, назначенные ему для обработки. Далее полученные результаты собираются на одном из процессоров, из которых выбирается минимум целевой функции в задаче (1).

В следующем разделе изложена новая статическая стратегия распределения нагрузки для параллельного МВГ, основанная на оценке сложности подзадач, генерируемых в процессе работы.

2. Предлагаемый метод статического распределения нагрузки между процессорами

2.1. Общая схема

Введем понятие *фронта дерева ветвления*, определяемого как набор подзадач, не подвергнутых ветвлению и не удаленных по правилу отсева. Подзадачи, входящие во фронт, будем называть *фронтальными*. На Рис. 1 фронтальные подзадачи помечены серым цветом.

Рассмотрим гипотетическую распределенную систему, состоящую из некоторого количества клиентских узлов (далее «клиентов») и одного выделенного узла (далее «сервер»). Под узлами здесь понимаются процессоры произвольного вида, например, ядра графической карты, вычислительного кластера, вычислительный узел в грид-системе.

Предлагаемый в работе распределенный алгоритм содержит три следующих друг за другом этапа (Рис. 2). В ходе первого этапа сервер выполняет заданное число шагов МВГ для создания фронта. Потом фронтальные подзадачи упаковываются в расчетные блоки, которые отправляются клиентам (один блок – одному клиенту, соответственно число расчетных блоков совпадает с числом клиентов). На втором этапе клиенты решают до конца подзадачи из полученных расчетных блоков. Третий этап начинается, когда все клиенты закончили свою работу. Полученные решения от всех клиентов собираются на сервере, после чего лучший результат сообщается пользователю.

Предложенная схема подходит для вычислительных сред с низкой связностью, потому что коммуникации требуются только на первом этапе и для сбора результатов. Более того, коммуникации на первом этапе могут быть пропу-

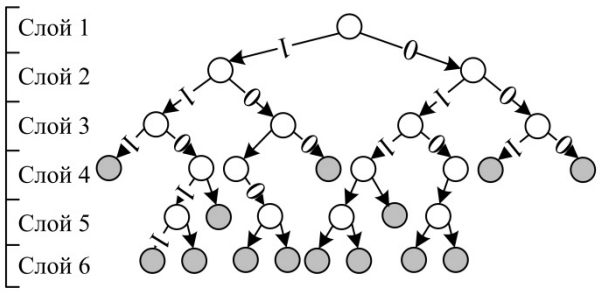

Рис. 1. Фронт в дереве ветвей и границ

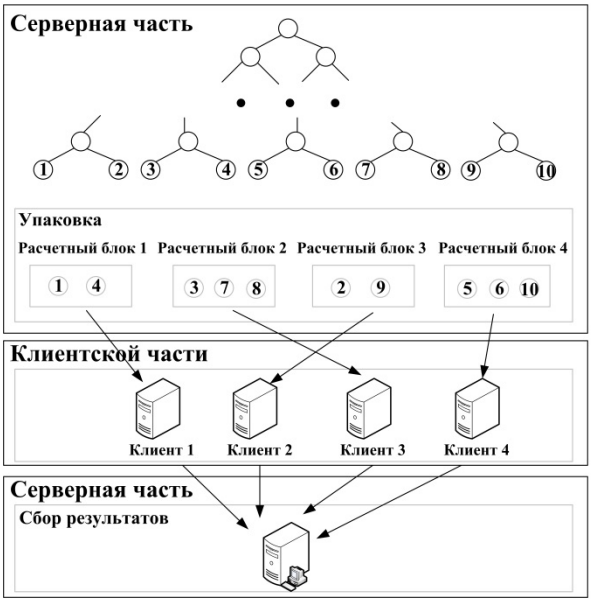

Рис. 2. Общая схема предлагаемого подхода

щены, а первая фаза просто повторена на клиентах. Такой подход экономит время в ситуациях, когда клиенты выделяются одновременно с сервером. При этом не происходит расхода лишних ресурсов, поскольку выделенные клиенты все равно простаивают на первом этапе. Такая ситуация характерна для GPGPUs и вычислительных кластеров, работающих под управлением систем пакетной обработки. В противном случае, когда клиенты включаются в вычисления с момента получения задания, отправка подзадач позволяет сэкономить вычислительные ресурсы клиентов. Примерами таких систем являются грид-системы из персональных компьютеров, где клиенты могут войти в вычислительный процесс или выйти из него в любой момент.

Цель предлагаемого подхода заключается в минимизации общей продолжительности расчетов (Makespan или сокращенно Msp). Это время может быть оценено снизу следующим образом $Msp = T_s + T_c$, где T_s представляет собой время первого этапа (серверной части) и T_c является временем второй фазы (клиентской части). В данной простой модели пренебрегаем временем, затраченным на коммуникацию. Очевидно, что T_c равно величине наибольшей продолжительности времени во всех расчетных блоках.

$$T_C = \max\ t(WU_j),\ j = 1,...,w.,$$

где $t(WU_j)$ является временем вычисления расчетного блока с номером *j*.

Несмотря на кажущуюся простоту, предлагаемая стратегия вызывает следующие важные вопросы:

1. Какие стратегии ветвления использовать на сервере и на клиентах?
2. Когда следует остановить первый этап?
3. Как распределить подзадачи по расчетным блокам?

Рассмотрим эти вопросы более подробно.

2.2. Стратегия ветвления

Рассмотрим две стратегии обхода дерева: поиск в глубину (ПГ) и поиск в ширину (ПШ). ПГ, как и следует из названия, состоит в том, чтобы идти «вглубь» дерева, насколько это возможно. При этом для ветвления выбирается наиболее удаленная от корня дерева фронтальная подзадача. ПШ, напротив всегда выбирает один из узлов, ближайших к корню дерева. ПГ является более экономичным с точки зрения памяти, а ПШ позволяет относительно быстро получить большой по объему фронт дерева.

В предлагаемом подходе на первом этапе сервер использует ПШ, потому что он позволяет генерировать больше фронтальных подзадач за относительно небольшое количество итераций. Это необходимо, чтобы обеспечить достаточное число подзадач для формирования расчетных блоков. Сервер прекращает выполнять ПШ сразу, как только число подзадач фронта достигнет верхнего порога, или после выполнения заданного числа итераций. Клиенты решают подзадачи в расчетном блоке с использованием ПГ, потому что ПГ требует значительно меньше памяти и, соответственно, работает быстрее.

2.3. Продолжительность первого этапа

Наиболее простой и естественный подход состоит в генерации на сервере количества подзадач, равного числу расчетных блоков с последующей упаковкой по одной задаче в расчетный блок. Такой подход является экономичным, так как продолжительность последовательной части (1-го этапа) при этом минимизируется. Проведенные эксперименты показали, что в условиях отсутствия информации о сложности подзадач, помещать одну подзадачу в один расчетный блок является лучшим вариантом. Если же известны оценки сложности, то размещение нескольких подзадач в расчетном блоке с учетом этих оценок может значительно повысить производительность.

2.4. Стратегия упаковки

Целью упаковки является минимизация длительности второго этапа T_c. Так как T_c определяется суммарной продолжительностью вычислений в самом сложном расчетном блоке, то цель состоит в том, чтобы сделать расчетные блоки максимально близкими по сложности. Очевидно, нижняя граница $\hat{T}_C$ для T_c достигается, когда все расчетные блоки равны (по оценке трудоемкости). Эта нижняя граница может быть вычислена как отношение сово-

купного времени решения всех подзадач во фронте и количества клиентов. Если фронт дерева ветвлений, полученного на управляющем процессоре, содержит s подзадач, упакованных в w расчетных блоков, то:

$$\hat{T}_C = \left(\sum_{i=1}^{s} t_i \right) / w,$$

где t_i является временем вычисления i-й подзадачи во фронте.

Для проведения экспериментального сравнения были реализованы *четыре различные стратегии упаковки*. Во всех стратегиях, каждый расчетный блок содержит $\left\lfloor \frac{s}{w} \right\rfloor$ или $\left\lfloor \frac{s}{w} \right\rfloor + 1$ подзадач: при этом несколько ($s \bmod w$) первых расчетных блоков имеют $\left\lfloor \frac{s}{w} \right\rfloor + 1$, а остальные расчетные блоки – $\left\lfloor \frac{s}{w} \right\rfloor$ подзадач в каждом блоке.

Наиболее естественной и простой в реализации является *плотная* стратегия упаковки, в рамках которой в один расчетный блок помещаются смежные фронтальные подзадачи. Данная стратегия весьма естественно реализуется на базе очереди FIFO. Плотная стратегия создает расчетные блоки, последовательно выбирая подзадачи с конца очереди. Слабым местом этой стратегии является то, что расчетный блок хранит соседние узлы дерева, имеющие, как правило, близкую сложность. Это означает, что сложные подзадачи сгруппированы вместе, образуя значительный дисбаланс нагрузки. Псевдокод плотной стратегии приведен в алгоритме 1.

Случайная стратегия использует случайное перемешивание фронта по алгоритму Fisher Yates [12] для создания очереди со случайной перестановкой подзадач. К перемешанной очереди применяется плотная стратегия. Предполагается, что перемешивание позволяет добиться более равномерного распределения сложности подзадач. Псевдокод Fisher Yates Shuffle показан в алгоритме 2.

Оставшиеся две стратегии основаны на оценках сложности. Предполагается, что существует возможность каким-либо образом оценить сложность подзадач [16]. Полученные

Алгоритм 1: Плотная стратегия

Ввод: S: количество подзадач
W: количество расчетных блоков
L_s[s]: список для хранения подзадачи
Выход: L_w[w]: список для хранения расчетных блоков

```
1. Sw ← S/W
2. Swl ← S mod W
3. Order ← 0
4. while Order <= Swl do
5.     for i ← 0 to Sw+1 do
6.         push Ls[i] into Lw[order]
7.     order++
8.  while Swl <= Order <= Sw do
5.     for i ← 0 to Sw do
6.         push Ls[i] into Lw[order]
7.     order++
8. return order;
```

Алгоритм 2: Перемешивание Fisher Yates

Ввод: S: Количество подзадач
Выход: N[s]: Массив идентификатора последовательности

```
1. for i ← 1 to S do
2.    N[i] = i
2. for i ← S-1 to 1 do
3.    j ← random integer from 0<=j<=i
4.    swap N[j] and N[i]
```

оценки используются для формирования расчетных блоков. Для простоты предположим, что число работ равномерно делится на количество расчетных блоков: $s = m \cdot w$. Обе стратегии относятся к классу *циклических распределений* и начинаются с сортировки подзадач в порядке убывания оценок сложности. Конкретная стратегия из этого класса определяется набором n перестановок $\sigma_1, ..., \sigma_n$. Расчетный блок i получает подзадачи с номерами $\sigma_1(i), w + \sigma_2(i), ..., (m-1)w + \sigma_n(i)$ Циклическое распределение может быть представлено в виде таблицы, в которой столбцы представляют расчетные блоки (Рис. 3).

Время второго этапа вычисляется следующим образом:

$$T_c = \max_{i=1,..,w} \sum_{j=1}^{m} t_{(j-1)w+\sigma_j(i)},$$

где t_k является временем работы подзадачи с номером k.

Ниже рассмотрим два варианта стратегии циклического распределения: нормальное Normal

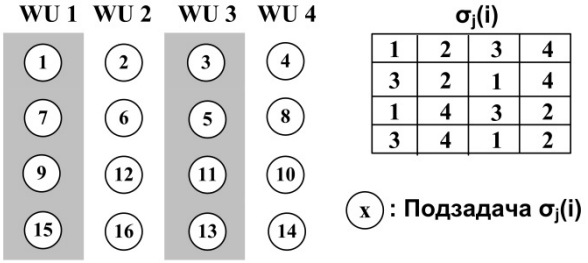

Рис. 3. Стратегия циклического распределения (4 расчетных блока и 16 подзадач)

Round Robin (NRR) и обратное Reverse Round Robin (RRR). В стратегии NRR все перестановки являются идентичными отображениями:

$$\sigma_j(i) = i,\ i = 1,...,w,\ j = 1,...,n$$

Данная стратегия очень популярна и используется по умолчанию во многих системах, см. например [13]. Однако эта стратегия имеет худшие показатели из всех стратегий циклического класса. Действительно, в последнем расчетном блоке (с номером w) содержатся наиболее сложные подзадачи из каждой строки, таким образом, достигая верхней границы *Makespan* (*Msp*).

Стратегия RRR обращает в обратный порядок каждую вторую строку. Перестановки задаются следующей формулой:

$$\sigma_j(i) = \begin{cases} i, & j \bmod 2 = 1, \\ n - i, & j \bmod 2 = 0. \end{cases}$$

Рис. 4 сравнивает обе стратегии для гипотетического примера с 16 подзадачами, упакованными в 4 расчетных блока. Продолжительность решения подзадачи указана числом в соответствующем прямоугольнике. Продолжительность расчетов для стратегии NRR составляет 49 условных единицы, а для стратегии RRR – 42, что очень близко к теоретическому пределу $\hat{T}_c = 41$. Очевидно, что не только в данном примере, но и вообще, стратегия RRR всегда не хуже, чем NRR, однако она не всегда является лучшей из возможных.

Можно выделить два конкретных случая, когда стратегия RRR дает наименьшее время работы второго этапа в классе циклических распределений: при наличии двух строк ($n = 2$) и, когда сложность подзадач линейно возрастает и имеет чётное количество строк. Доказа-

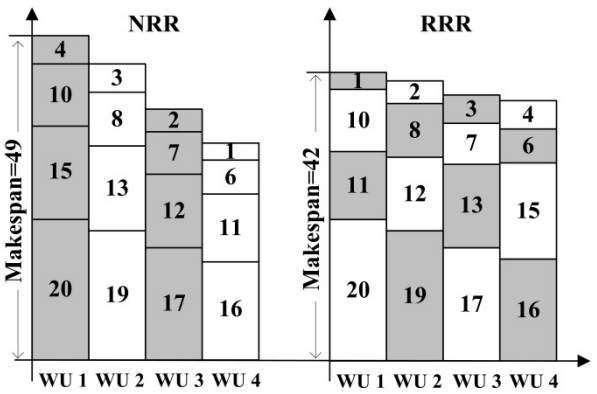

Рис. 4. Пример NRR и RRR с 16 подзадачами в 4-х расчетных блоках

тельства этих фактов достаточно просты. Хотя для общего случая RRR не обязательно дает наиболее выгодное циклическое распределение, экспериментальная оценка показывает, что RRR обеспечивает разумную производительность и значительно превосходит NRR.

3. Вычислительный эксперимент

Для экспериментальной проверки предложенного подхода была выбрана задача о сумме подмножеств (ЗСП) [15], определяемая следующим образом.

Дано множество n элементов с весами $w_1,...,w_n$ и ранец с вместимостью C. Требуется выбрать подмножество элементов, общий вес которых находится ближе всего к вместимости, не превышая ее, т.е.:

$$\text{maximize } C = \sum_{j=1}^{n} w_j x_j$$

$$\text{subject to } \sum_{j=1}^{n} w_j x_j \le z \qquad (2)$$

$$x_j = 0 \text{ or } 1,\ \text{j} \in \text{N} = \{1,...,\text{n}\}$$

Получение точной формулы для числа шагов в решении задачи (2) является проблематичным. Оценки сложности, полученные в [14], позволяют предположить, что максимальной сложностью обладают задачи, в которых вместимость C близка к половине суммы всего веса.

Был применен классический метод ветвей и границ для решения тестовых задач ЗСП. Было рассмотрено 10 задач со случайно выбранными весами предметов. Для каждой из задач слу-

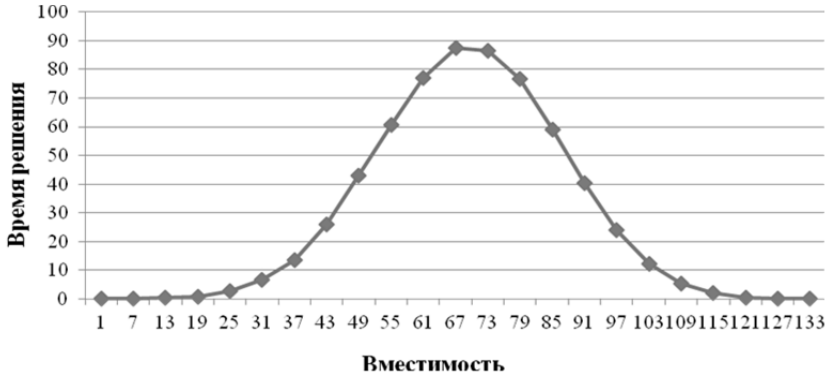

Рис. 5. Вычислительное время в другой вместимости

чайно выбиралась вместимость в диапазоне от 1 до суммарного веса предметов десять раз для каждого примера. Если веса элементов зафиксированы, то вычислительная сложность является функцией, зависящей от емкости. Типичный характер этой зависимости представлен на Рис. 5, который показывает соотношение между вычисляемым временем и вместимостью (при фиксированных коэффициентах) в случайно выбранном примере. В этом примере: $k = 2, n = 20$. Ось X соответствует вместимости C, изменяемой в диапазоне от минимальной 0 до максимальной $\sum_{i=1}^{n} w_i$. По оси Y отложено время вычисления в секундах.

Анализ достаточно большой совокупности примеров позволил выделить следующие закономерности:

- выбор $C = \hat{C} = \frac{1}{2}\sum_{i=1}^{n} w_i$, обеспечивает наибольшую сложность подзадачи; чем дальше C от $\widehat{C}$, тем меньше сложность подзадачи;
- кривая времени вычисления является приблизительной симметричной кривой распределения;
- существуют некоторые случаи, нарушающие данное наблюдение, но в целом введенное правило соблюдается.

Цель дальнейших экспериментов состояла в: оценке эффективности четырех предложенных стратегий упаковки и выяснении возможности достижении максимальной производительности для каждой из них. Сравнивались следующие стратегии:

1) Плотная стратегия – DS.
2) Случайная стратегия – RS.
3) Стратегия нормального циклического распределения Normal Round-robin – NRR.
4) Стратегия обратного циклического распределения Reverse Round-robin – RRR.

Эксперименты проводились в рамках эмуляции параллельного выполнения. Для этого создавались расчетные блоки, которые решались последовательно, а затем выбиралось максимальное время работы. Так как физическое время обработки расчетного блока является величиной в значительной степени подверженной случайным факторам, которые сложно учесть, использовалось теоретическое время, равное суммарной фактической сложности решенных подзадач. Сложность вычислялась как число шагов МВГ.

3.1. Эксперимент 1

В первом эксперименте участвуют 10 примеров (задач) со случайными параметрами. В дальнейшем используются следующие обозначения:

- λ означает количество подзадач в расчетных блоках;
- T_c, T_s означают сложность на стороне клиента и сервера соответственно;
- Msp ($Makespan$) означает суммарное время работы приложения, вычисляемое по формуле $Msp = T_s + T_c$.

Сравнивались T_c, T_s и Msp по 10-ти различным примерам. В Табл. 1 представлены значения Msp для четырех стратегий. Здесь фиксируется $\lambda = 4$. Результаты, приведенные в Табл. 1 и на Рис. 6, показывают, что примене-

Табл. 1. Сложность 10 примеров ЗСП

Instance	λ=4											
	RRR			NRR			RS			DS		
	T_c	T_s	Msp	T_c	T_s	Msp	T_c	T_s	Msp	T_c	T_s	Msp
1	15258	3953	19211	18306	3953	22259	29218	3953	33171	31592	3953	35545
2	11975	3937	15912	15532	3937	19469	28901	3937	32838	32226	3937	36163
3	13901	3844	17745	17647	3844	21491	27009	3844	30853	34061	3844	37905
4	15783	3971	19754	19071	3971	23042	29846	3971	33817	37291	3971	41262
5	13067	3939	17006	15683	3939	19622	26028	3939	29967	37453	3939	41392
6	14086	3901	17987	16094	3901	19995	26897	3901	30798	36829	3901	40730
7	14335	3884	18219	18297	3884	22181	23049	3884	26933	34929	3884	38813
8	16462	3947	20409	17478	3947	21425	25799	3947	29746	34221	3947	38168
9	14785	3918	18703	17436	3918	21354	27886	3918	31804	32578	3918	36496
10	15734	3905	19639	16865	3905	20770	28325	3905	32230	35332	3905	39237

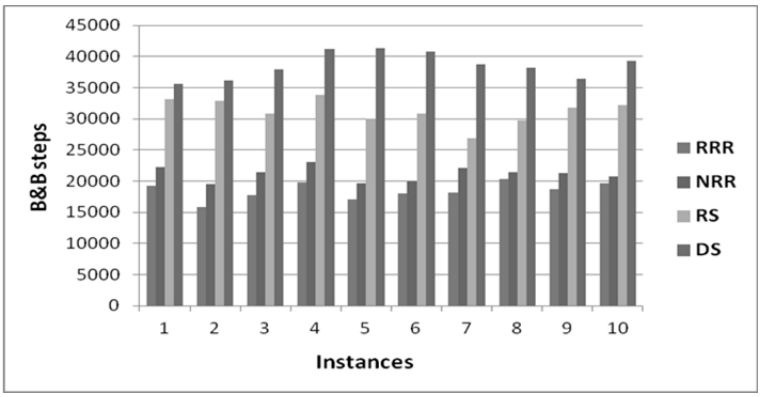

Рис. 6. Общее время работы Msp для сравниваемых стратегий

ние стратегии RRR уменьшает Msp примерно вдвое по сравнению с использованием стратегии DS, и заметно опережает стратегию NRR.

3.2. Эксперимент 2

Из результатов эксперимента 1 следует, что если время работы подзадач может быть надежно оценено, циклические распределения значительно превосходят стратегии DS и RS. Однако не ясно, сколько подзадач необходимо генерировать на сервере. Интуиция подсказывает, что генерация большего количества подзадач расширяет возможности для балансировки нагрузки, но может замедлить расчеты из-за длительного последовательного первого этапа. Таким образом, генерацию подзадач нужно останавливать, как только получено достаточное их количество для эффективной балансировки нагрузки.

В Табл. 2 представлены значения метрики распределения нагрузки $L_b = t_l / t_a$ для различного числа подзадач в расчетном блоке. Здесь t_l является временем работы в самом сложном расчетном блоке, и t_a является средним временем работы всех расчетных блоков. Чем меньше значение метрики, тем больше степень сбалансированности нагрузки. Идеальная балансировка соответствует $L_b = 1$. Полученные результаты показывают, что стратегии DS и RS не дают значимого улучшения при увеличении числа подзадач в расчетном блоке. Это означает, что для таких стратегий сервер должен прекратить вычисления, как только он произведёт количество подзадач, равное числу расчетных блоков.

Табл. 2. Баланс нагрузки при разных λ

Workunits	λ	L_b			
		RRR	NRR	RS	DS
16	1	1.31483514	1.31483514	**1.31483514**	**1.31483514**
	2	1.16998602	1.27823437	1.34955023	1.38845466
	4	1.07203548	1.16692388	1.38283754	1.48273533
	8	**1.00992259**	**1.09653621**	1.34983729	1.58121552
64	1	1.48367634	1.48367634	**1.48367634**	**1.48367634**
	2	1.16805444	1.36313744	1.5831098	1.58556033
	4	1.03367658	1.21452538	1.62135578	1.70261504
	8	**1.01660665**	1.08353156	1.70056863	1.84284239
256	1	1.70289081	1.70289081	**1.70289081**	**1.70289081**
	2	1.1082803	1.47086962	1.80354769	1.89609691
	4	1.02942558	1.18794537	1.94981824	2.03367382
	8	**1.02614737**	**1.07200816**	2.4028646	2.31120395
1024	1	1.89231094	1.89231094	**1.89231094**	**1.89231094**
	2	1.12800522	1.59749536	2.10938697	2.0129846
	4	1.03237205	1.31666541	2.3605455	2.42495702
	8	**1.03028772**	**1.16701499**	2.76982747	2.98157864

В циклических стратегиях NRR и RRR наилучший результат достигается при использовании от 4 до 8 подзадач в расчетном блоке. При этом сбалансированность близка к идеальной. Видно, что помещение четырех подзадач в расчетный блок уже дает достаточно хорошую балансировку и, следовательно, нет необходимости увеличивать это значение далее.

Заключение

В работе предложена новая статическая стратегия распределения нагрузки для параллельного метода ветвей и границ на основе оценок вычислительной сложности подзадач. Эта стратегия может быть использована в параллельных системах с ограниченными возможностями коммуникаций между вычислительными ядрами, где динамическое распределение нагрузки является проблематичным. Проведенные эксперименты показали преимущество предлагаемой стратегии (Reverse-Round-robin) по отношению к другим статическим стратегиям распределения нагрузки.

Литература

1. Dongarra J., Meuer H., and Strohmaier E., Top 500 supercomputers. website, November 2014.
2. John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Tim Purcell: A Survey of General-Purpose Computation on Graphics Hardware, Computer Graphics Forum, 26(1):80–113, March 2007.
3. Radek (June 18, 2012). "Chip Shot: Intel Names the Technology to Revolutionize the Future of HPC - Intel® Xeon® Phi™ Product Family". Intel. Retrieved December 12, 2012.
4. Заикин О.С., Посыпкин М.А., Семёнов А.А., Храпов Н.П., Организация добровольных вычислений на платформе BOINC на примере проектов OPTIMA@home и SAT@home // CAD/CAM/CAE Observer # 3 (71) / 2012. С. 87-92.
5. Afanasiev A., Sukhoroslov O., Voloshinov V. MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services // Parallel Computing Technologies. – Springer Berlin Heidelberg, 2013. – S. 394-408.
6. Сигал И. Х., Иванова А. П. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. – 2-е изд. испр. и доп. – М.: ФИЗМАТЛИТ, 2007. –304 с.
7. Casado L.G., Martínez J.A., Inmaculada García, Eligius M.T. Hendrix: Branch-and-Bound interval global optimization on shared memory multiprocessors. Optimization Methods and Software 23(5): 689-701 (2008).
8. Посыпкин М. А. Решение задач глобальной оптимизации в среде распределенных вычислений // Программные продукты и системы. № 1. 2010. С. 23-29.
9. Catalyurek U., Boman E., Devine K., Bozdag D., Heaphy R., and Riesen L., Hypergraph-based dynamic load balancing for adaptive scientific computations. In Proc. of 21st International Parallel and Distributed Processing Symposium (IPDPS'07), pages 1–11. IEEE, 2007. Best Algorithms Paper Award.
10. Chevalier C., Pellegrini F., Futurs I., and U. B. I. Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework. In Proceedings of Euro-Par 2006, LNCS, pages 243–252, 2006.
11. Немнюгин С.А. Средства программирования для многопроцессорных вычислительных систем //СПб.: СПбГУ. – 2007.
12. Fischetti M., Monaci M., Salvagnin D., Self-splitting of workload in parallel computation, Integration of AI and OR Techniques in Constraint Programming (CPAIOR), Lecture Notes in Computer Science, Volume 8451, pp 394-404, 2014.
13. Pisinger D.: Where are the hard knapsack problems? Computers & Operations Research, vol. 32, pp. 2271–2284, 2005.
14. Колпаков Р.М., Посыпкин М.А., Об эффективных верхних оценках сложности решения задачи о булевом ранце методом ветвей и границ, Материалы XVII Международной школы-семинара "Синтез и сложность управляющих систем" им. академика О.Б. Лупанова, Издательство Института математики, Новосибирск 2008, С. 60-63.
15. Schnorr C. P. and Euchner M., Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems, in Proceedings of Fundamentals of Computation Theory '91, L. Budach, ed., Lecture Notes in Computer Science 529, Springer-Verlag, New York, 1991, 68–85.
16. Колпаков Р.М., Посыпкин М.А., Сигал И.Х. О нижней оценке вычислительной сложности одной параллельной реализации метода ветвей и границ// Автоматика и телемеханика, 2010. № 10, С. 156-166.

Тянь Бо. Аспирант ВМК МГУ им. М.В. Ломоносова. Область научных интересов: параллельные вычислительные системы, глобальная оптимизация. E-mail: yesyestian@gmail.com

Посыпкин Михаил Анатольевич. Ведущий научный сотрудник ИППИ РАН, ведущий научный сотрудник ВЦ им. А.А.Дородницына РАН. Окончил МГУ имени М.В. Ломоносова в 1996 году. Кандидат физико-математических наук, доцент. Автор 142 печатных работ. Область научных интересов: методы оптимизации, распределенные вычисления, параллельные вычисления. E-mail: mposypkin@gmail.com

Сигал Израиль Хаимович. Главный научный сотрудник ВЦ им. А.А. Дородницына РАН. Окончил Одесский государственный университет им. И.И. Мечникова в 1960 году. Доктор технических наук, профессор. Автор 155 печатных работ. Область научных интересов: методы оптимизации и исследование операций, дискретная оптимизация, параллельные вычисления. E-mail: sigal_ikh@mail.ru