ГЛАВА 3
ПРИНЦИПЫ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ МЕТОДОВ

Разработка алгоритмов (а в особенности методов параллельных вычислений) для решения сложных научно-технических задач часто представляет собой значительную проблему. Для снижения сложности рассматриваемой темы оставим в стороне математические аспекты разработки и доказательства сходимости алгоритмов – эти вопросы в той или иной степени изучаются в ряде «классических» математических учебных курсов. Здесь же мы будем полагать, что вычислительные схемы решения задач, рассматриваемых далее в качестве примеров, уже известны [1)]. С учетом высказанных предположений последующие действия для определения эффективных способов организации параллельных вычислений могут состоять в следующем:

- Выполнить анализ имеющихся вычислительных схем и осуществить их разделение (*декомпозицию*) на части (*подзадачи*), которые могут быть реализованы в значительной степени независимо друг от друга.
- Выделить для сформированного набора подзадач *информационные взаимодействия*, которые должны осуществляться в ходе решения исходной поставленной задачи.
- Определить необходимую (или доступную) для решения задачи *вычислительную систему* и выполнить *распределение* имеющего набора подзадач между процессорами системы.

При самом общем рассмотрении понятно, что объем вычислений для каждого используемого процессора должен быть примерно одинаков – это позволит обеспечить равномерную вычислительную загрузку (*балансировку*) процессоров. Кроме того, также понятно, что распределение подзадач между процессорами должно быть выполнено таким образом, чтобы наличие информационных связей (*коммуникационных взаимодействий*) между подзадачами было минимальным.

После выполнения всех перечисленных этапов проектирования можно оценить эффективность разрабатываемых параллельных методов – для этого обычно определяются значения показателей качества порождаемых па-

---

[1)] Несмотря на то, что для многих научно-технических задач на самом деле известны не только последовательные, но и параллельные методы решения, данное предположение является, конечно, очень сильным, поскольку для новых возникающих задач, требующих для своего решения большого объема вычислений, процесс разработки алгоритмов составляет существенную часть всех выполняемых работ.

раллельных вычислений (ускорение, эффективность, масштабируемость). По результатам проведенного анализа может оказаться необходимым повторение отдельных (в предельном случае всех) этапов разработки – следует отметить, что возврат к предшествующим этапам может происходить на любой стадии проектирования параллельных вычислительных схем.

В этом отношении часто выполняемым дополнительным действием в приведенной выше схеме проектирования является корректировка состава сформированного множества задач после определения имеющегося количества процессоров – подзадачи могу быть укрупнены (*агрегированы*) при наличии малого числа процессоров или, наоборот, *детализированы* в противном случае. В целом, данные действия могут быть определены как *масштабирование* разрабатываемого алгоритма и выделены в качестве отдельного этапа проектирования параллельных вычислений.

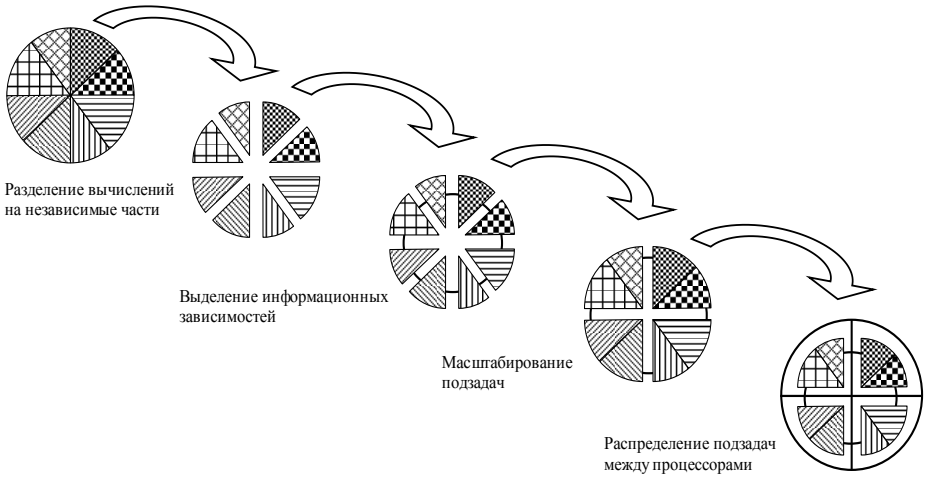

Рис. 3.1. Общая схема разработки параллельных алгоритмов

Для применения получаемого в конечном итоге параллельного метода необходимо выполнить его программную реализацию и разделить разработанный программный код по вычислительным элементам компьютерной системы в соответствии с выбранной схемой распределения подзадач. Подготовленный программный код должен обеспечивать решение сформированного набора подзадач. Практически это может быть обеспечено, например, разработкой для решения каждой подзадачи отдельной программы, однако чаще всего создается программа, которая объединяет в себе все действия, необходимые для решения всех имеющихся подзадач. Такой объединенный программный код (*метапрограмма*) разрабатывается таким образом, чтобы программа в зависимости от управляющих параметров могла настраиваться на решение требуемой подзадачи (в качестве управ-

ляющего параметра может быть, например, номер вычислительного элемента). Для проведения вычислений подобная метапрограмма может копироваться для выполнения на все вычислительные элементы – такой подход используется, например, в технологии MPI для многопроцессорных вычислительных систем с распределенной памятью; выполняемые программы на разных вычислительных элементах обычно именуются *процессами*. Метапрограмма может использоваться также и для порождения множества отдельных командных *потоков* – так, например, происходит в случае технологии OpenMP для многопроцессорных вычислительных систем с общей разделяемой памятью.

Для проведения вычислений параллельная программа запускается на выполнение. Для реализации информационных взаимодействий параллельно выполняемые части программы (процессы или потоки) должны иметь в своем распоряжении средства обмена данными (*каналы передачи сообщений* для систем с распределенной памятью или *общие переменные* для систем с общей разделяемой памятью).

Каждый вычислительный элемент (процессор или ядро процессора) компьютерной системы обычно выделяется для решения одной единственной подзадачи, однако при наличии большого количества подзадач или использовании ограниченного числа вычислительных элементов это правило может не соблюдаться и, в результате, на вычислительных элементах может выполняться одновременно несколько параллельных частей программы (процессов или потоков). В частности, при разработке и начальной проверке параллельной программы для выполнения всех параллельных ее частей может использоваться один вычислительный элемент (при расположении на одном вычислительном элементе параллельные части программы выполняются в режиме разделения времени).

Следует отметить, что разработанная схема проектирования и реализации параллельных вычислений первоначально была предложена для вычислительных систем с распределенной памятью, когда необходимые информационные взаимодействия реализуются при помощи передачи сообщений по каналам связи между процессорами. Тем не менее, данная схема может быть использована без потери какой-либо эффективности параллельных вычислений и для разработки параллельных методов для систем с общей памятью – в этом случае механизмы передачи сообщений для обеспечения информационных взаимодействий просто заменяются операциями доступа к общим (разделяемым) переменным. Для снижения сложности излагаемого далее учебного материала *схема проектирования и реализации параллельных вычислений будет конкретизирована применительно к вычислительным системам с общей памятью*.

Важно отметить также, что для вычислительных систем с общей памятью активно пропагандируется и другой – «обратный» – способ разработки параллельных программ, когда за основу берется тот или иной последова-

тельный прототип, который постепенно преобразуется к параллельному варианту (в частности, именно так предлагается использовать технологию OpenMP на начальном этапе освоения). Безусловно, такой подход позволяет достаточно быстро получить начальные варианты параллельных программ без значительных дополнительных усилий. Однако достаточно часто такая методика приводит к получению параллельных программ со сравнительно низкой эффективностью, и достижение максимально возможных показателей ускорения вычислений можно обеспечить только при изначальном проектировании параллельных вычислений на начальных этапах разработки методов решения поставленных задач.

3.1. Моделирование параллельных программ

Рассмотренная схема проектирования и реализации параллельных вычислений дает способ понимания параллельных алгоритмов и программ. На стадии проектирования параллельный метод может быть представлен в виде *графа «подзадачи–информационные зависимости»*, который представляет собой не что иное, как укрупненное (агрегированное) представление графа информационных зависимостей (графа «операции–операнды» – см. гл. 2). Аналогично, на стадии выполнения для описания параллельной программы может быть использована модель в виде *графа «потоки–общие данные»*, в которой вместо подзадач используется понятие потоков, а информационные зависимости реализуются за счет использования общих данных. В дополнение, на этой модели может быть показано распределение потоков по вычислительным элементам компьютерной системы, если количество подзадач превышает число вычислительных элементов – см. рис. 3.2

Использование двух моделей параллельных вычислений [2)] позволяет лучше разделить проблемы, которые проявляются при разработке параллельных методов. Первая модель – граф «подзадачи–информационные зависимости» – позволяет сосредоточиться на вопросах выделения подзадач одинаковой вычислительной сложности, обеспечивая при этом низкий уровень информационной зависимости между подзадачами. Вторая модель – граф «потоки–общие данные» – концентрирует внимание на вопросах распределения подзадач по вычислительным элементам и позволяет лучше анализировать эффективность разработанного параллельного метода и обеспечивает возможность более адекватного описания процесса выполнения параллельных вычислений.

[2)] В работе Foster [54] рассматривается только одна модель – модель "задача-канал" для описания параллельных вычислений, которая занимает некоторое промежуточное положение по сравнению с изложенными здесь моделями. Так, в модели "задача-канал" не учитывается возможность использования одного процессора для решения нескольких подзадач одновременно.

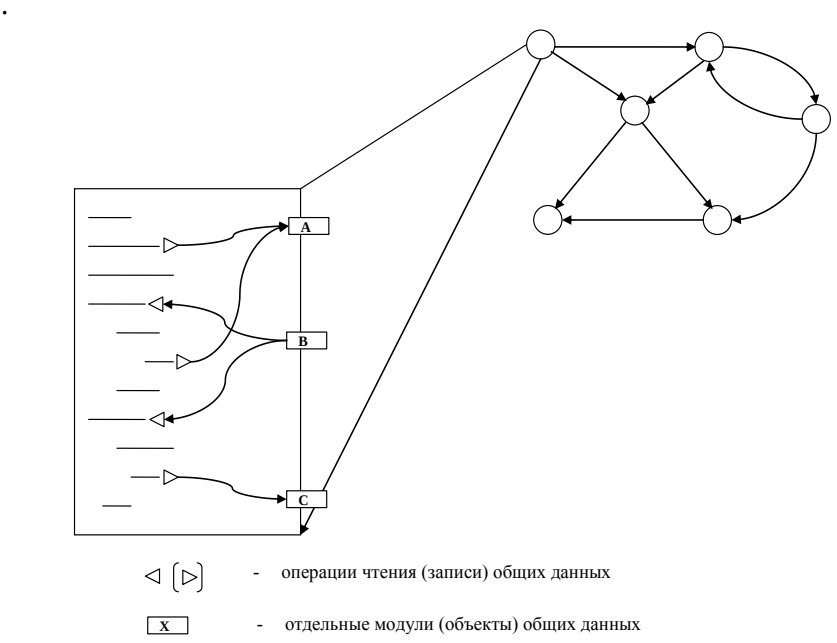

Рис. 3.2. Модель параллельной программы в виде графа «потоки–общие данные»

Дадим дополнительные пояснения для используемых понятий в модели «потоки–общие данные»:

- Под *потоком* в рамках данного учебного материала будем понимать логически выделенную с точки зрения операционной системы *последовательность команд*, которая может исполняться на одном вычислительном элементе компьютерной системы и которая содержит ряд *операций доступа* (*чтения/записи*) *к общим данным* для организации информационного взаимодействия между выполняемыми потоками параллельной программы; все потоки параллельной программы имеют общее адресное пространство; операции создания и завершения потоков – в отличие от процессов – являются менее трудоемкими;

- *Общие данные* с логической точки зрения могут рассматриваться как *общий* (*разделяемый*) *между потоками ресурс*; общие данные могут использоваться параллельно выполняемыми потоками для чтения и записи значений.

Важно отметить, что использование общих данных для обеспечения корректности должно осуществляться в соответствии с правилами *взаимного исключения* (в каждый момент времени общие данные могут исполь-

зоваться только одним потоком). При несоблюдении этих правил при использовании общих данных может иметь место ситуация *гонки потоков*, когда результат вычислений будет зависеть от взаимного соотношения темпа выполнения команд в потоках (такая ситуация возникает, например, при попытке записи значений общих данных одновременно несколькими потоками). С другой стороны, следует понимать также, что операции организации взаимоисключения могут приводить к задержкам (*блокировкам*) потоков.

Следует отметить важное достоинство рассмотренной модели «потоки–общие данные» – в этой модели проводится четкое разделение локальных (выполняемых на отдельном процессоре) вычислений и действий по организации информационного взаимодействия одновременно выполняемых потоков. Такой подход значительно снижает сложность анализа эффективности параллельных методов и существенно упрощает проблемы разработки параллельных программ.

3.2. Этапы разработки параллельных алгоритмов

Рассмотрим более подробно изложенную выше методику разработки параллельных алгоритмов. В значительной степени данная методика опирается на подход, впервые разработанный в [54], и, как отмечалось ранее, включает этапы выделения подзадач, определения информационных зависимостей, масштабирования и распределения подзадач по процессорам вычислительной системы (см. рис. 3.1). Для демонстрации приводимых рекомендаций далее будет использоваться учебная задача поиска максимального значения среди элементов матрицы A (такая задача возникает, например, при численном решении систем линейных уравнений для определения ведущего элемента метода Гаусса):

$$y = \max_{1 \le i, j \le N} a_{ij}.$$

Такая задача носит полностью иллюстративный характер, и после рассмотрения этапов разработки в оставшейся части главы будет приведен более полный пример использования данной методики для разработки параллельных алгоритмов. Кроме того, данная схема разработки будет применена и при изложении всех далее рассматриваемых методов параллельных вычислений.

3.2.1. Разделение вычислений на независимые части

Выбор способа разделения вычислений на независимые части основывается на анализе вычислительной схемы решения исходной задачи. Требования, которым должен удовлетворять выбираемый подход, обычно со-

стоят в обеспечении равного объема вычислений в выделяемых подзадачах и минимума информационных зависимостей между этими подзадачами (при прочих равных условиях нужно отдавать предпочтение редким операциям передачи большего размера сообщений по сравнению с частыми пересылками данных небольшого объема). В общем случае проведение анализа и выделение задач представляет собой достаточно сложную проблему – ситуацию помогает разрешить существование двух часто встречающихся типов вычислительных схем:

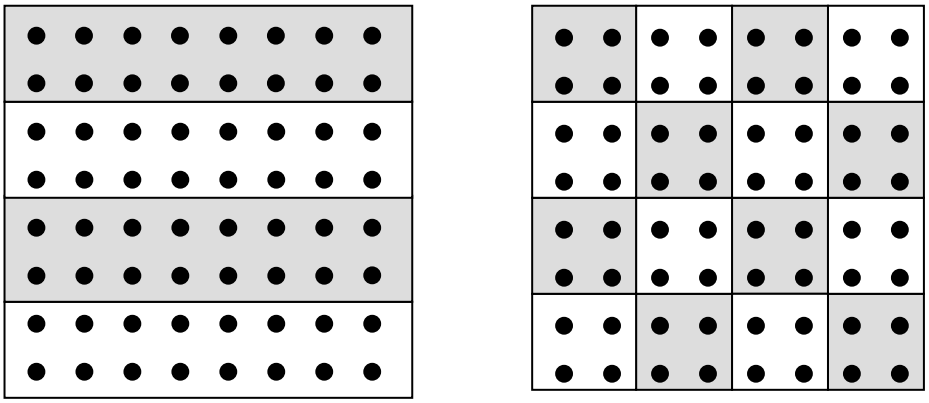

Рис. 3.3. Разделение данных для матрицы *A*:
а) ленточная схема, б) блочная схема

- Для большого класса задач вычисления сводятся к выполнению однотипной обработки большого набора данных – к такому виду задач относятся, например, матричные вычисления, численные методы решения уравнений в частных производных и др. В этом случае говорят, что существует *параллелизм по данным*, и выделение подзадач сводится к разделению имеющихся данных. Например, для нашей учебной задачи поиска максимального значения при формировании подзадач исходная матрица *A* может быть разделена на отдельные строки (или последовательные группы строк) – *ленточная схема* разделения данных (см. рис. 3.3) или на прямоугольные наборы элементов – *блочная схема* разделения данных. Для большого количества решаемых задач разделение вычислений по данным приводит к порождению одно-, двух- и трехмерных наборов подзадач, для которых информационные связи существуют лишь между ближайшими соседями (см. рис. 3.4) – такие схемы обычно именуются *сетками* или *решетками*.

- Для другой части задач вычисления могут состоять в выполнении разных операций над одним и тем же набором данных – в этом случае говорят о существовании *функционального параллелизма* (в качестве приме-

ров можно привести задачи обработки последовательности запросов к информационным базам данных, вычисления с одновременным применением разных алгоритмов расчета и т. п.). Очень часто функциональная декомпозиция может быть использована для организации конвейерной обработки данных (так, например, при выполнении каких-либо преобразований данных вычисления могут быть сведены к функциональной последовательности ввода, обработки и сохранения данных).

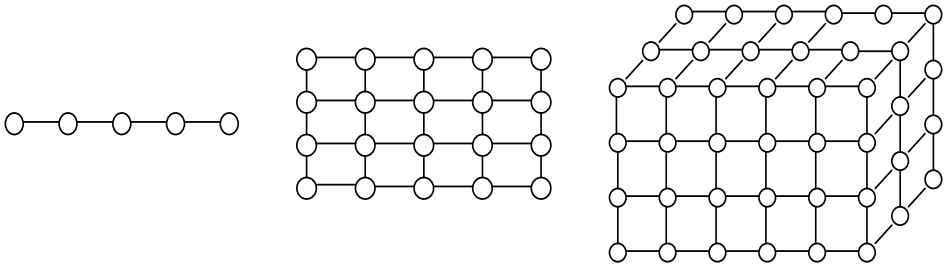

Рис. 3.4. Регулярные одно-, двух- и трехмерные структуры базовых подзадач после декомпозиции данных

Важный вопрос при выделении подзадач состоит в выборе нужного *уровня декомпозиции* вычислений. Формирование максимально возможного количества подзадач обеспечивает использование предельно достижимого уровня параллелизма решаемой задачи, однако затрудняет анализ параллельных вычислений. Использование при декомпозиции вычислений только достаточно «крупных» подзадач приводит к ясной схеме параллельных вычислений, однако может затруднить эффективное использование достаточно большого количества процессоров. Возможное разумное сочетание этих двух подходов может состоять в использовании в качестве конструктивных элементов декомпозиции только тех подзадач, для которых методы параллельных вычислений являются известными. Так, например, при анализе задачи матричного умножения в качестве подзадач можно использовать методы скалярного произведения векторов или алгоритмы матрично-векторного произведения. Подобный промежуточный способ декомпозиции вычислений позволит обеспечить и простоту представления вычислительных схем, и эффективность параллельных расчетов. Выбираемые подзадачи при таком подходе будем именовать далее *базовыми*, которые могут быть *элементарными* (неделимыми), если не допускают дальнейшего разделения, или *составными* в противном случае.

Для рассматриваемой учебной задачи достаточный уровень декомпозиции может состоять, например, в разделении матрицы A на множество отдельных строк и получении на этой основе набора подзадач поиска максимальных значений в отдельных строках; порождаемая при этом структу-

ра информационных связей соответствует линейному графу – см. ниже рис. 3.5.

Для оценки корректности этапа разделения вычислений на независимые части можно воспользоваться контрольным списком вопросов, предложенных в [54]:

- Выполненная декомпозиция не увеличивает объем вычислений и необходимый объем памяти?
- Возможна ли при выбранном способе декомпозиции равномерная загрузка всех имеющихся вычислительных элементов компьютерной системы?
- Достаточно ли выделенных частей процесса вычислений для эффективной загрузки имеющихся вычислительных элементов (с учетом возможности увеличения их количества)?

3.2.2. Выделение информационных зависимостей

При наличии вычислительной схемы решения задачи после выделения базовых подзадач определение информационных зависимостей между подзадачами обычно не вызывает больших затруднений. При этом, однако, следует отметить, что на самом деле этапы выделения подзадач и информационных зависимостей достаточно сложно поддаются разделению. Выделение подзадач должно происходить с учетом возникающих информационных связей; после анализа объема и частоты необходимых информационных обменов между подзадачами может потребоваться повторение этапа разделения вычислений.

При проведении анализа информационных зависимостей между подзадачами следует различать (предпочтительные формы информационного взаимодействия выделены подчеркиванием):

- *Локальные и глобальные схемы* информационного взаимодействия – для локальных схем в каждый момент времени информационная зависимость существует только между небольшим числом подзадач (располагаемых, как правило, на соседних вычислительных элементах), для глобальных зависимостей информационное взаимодействие имеет место для всех подзадач.
- *Структурные и произвольные способы взаимодействия* – для структурных способов организация взаимодействий приводит к формированию некоторых стандартных схем коммуникации (например, в виде кольца, прямоугольной решетки и т. д.), для произвольных структур взаимодействия схема информационных зависимостей не носит характера какой-либо однородности.
- *Статические или динамические схемы* информационной зависимости – для статических схем моменты и участники информационного взаи-

модействия фиксируются на этапах проектирования и разработки параллельных программ, для динамического варианта взаимодействия структура зависимостей определяется в ходе выполняемых вычислений.

Как уже отмечалось в предыдущем разделе, для учебной задачи поиска максимального значения при использовании в качестве базовых элементов подзадач поиска максимальных значений в отдельных строках исходной матрицы $\boldsymbol{A}$ структура информационных связей имеет вид, представленный на рис. 3.5.

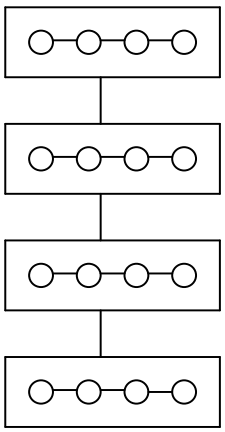

Рис. 3.5. Структура информационных связей учебной задачи

Как и ранее, для оценки правильности этапа выделения информационных зависимостей можно воспользоваться контрольным списком вопросов, предложенных в [54]:

- Соответствует ли вычислительная сложность подзадач интенсивности их информационных взаимодействий?
- Является ли одинаковой интенсивность информационных взаимодействий для разных подзадач?
- Является ли схема информационного взаимодействия локальной?
- Не препятствует ли выявленная информационная зависимость параллельному решению подзадач?

3.2.3. Масштабирование набора подзадач

Масштабирование разработанной вычислительной схемы параллельных вычислений проводится в случае, когда количество имеющихся подзадач отличается от числа планируемых к использованию вычислительных элементов. Для сокращения количества подзадач необходимо выполнить укрупнение (*агрегацию*) вычислений. Применяемые здесь

правила совпадают с рекомендациями начального этапа выделения подзадач – определяемые подзадачи, как и ранее, должны иметь одинаковую вычислительную сложность, а объем и интенсивность информационных взаимодействий между подзадачами должны оставаться на минимально-возможном уровне. Как результат, первыми претендентами на объединение являются подзадачи с высокой степенью информационной взаимозависимости.

При недостаточном количестве имеющегося набора подзадач для загрузки всех доступных к использованию вычислительных элементов необходимо выполнить детализацию (*декомпозицию*) вычислений. Как правило, проведение подобной декомпозиции не вызывает каких-либо затруднений, если для базовых задач методы параллельных вычислений являются известными.

Выполнение этапа масштабирования вычислений должно свестись, в конечном итоге, к разработке правил агрегации и декомпозиции подзадач, которые должны параметрически зависеть от числа вычислительных элементов, применяемых для вычислений.

Для рассматриваемой учебной задачи поиска максимального значения агрегация вычислений может состоять в объединении отдельных строк в группы (ленточная схема разделения матрицы – см. рис. 3.3а), при декомпозиции подзадач строки исходной матрицы A могут разбиваться на несколько частей (блоков).

Список контрольных вопросов, предложенный в [54] для оценки правильности этапа масштабирования, выглядит следующим образом:

- Не ухудшится ли локальность вычислений после масштабирования имеющегося набора подзадач?
- Имеют ли подзадачи после масштабирования одинаковую вычислительную и коммуникационную сложность?
- Соответствует ли количество задач числу имеющихся вычислительных элементов?
- Зависят ли параметрически правила масштабирования от количества вычислительных элементов?

3.2.4. Распределение подзадач между вычислительными элементами

Распределение подзадач между вычислительными элементами является завершающим этапом разработки параллельного метода. Надо отметить, что управление распределением нагрузки для процессоров возможно только для вычислительных систем с распределенной памятью; для мультипроцессоров (систем с общей памятью) распределение нагрузки обычно выполняется операционной системой автоматически. Кроме того, данный

этап распределения подзадач между процессорами является избыточным, если количество подзадач совпадает с числом имеющихся вычислительных элементов.

Основной показатель успешности выполнения данного этапа – *эффективность использования вычислительных элементов*, определяемая как относительная доля времени, в течение которого вычислительные элементы использовались для вычислений, связанных с решением исходной задачи. Пути достижения хороших результатов в этом направлении остаются прежними – как и ранее, необходимо обеспечить равномерное распределение вычислительной нагрузки между вычислительными элементами и минимизировать количество информационных взаимодействий, существующих между ними. Точно так же, как и на предшествующих этапах проектирования, оптимальное решение проблемы распределения подзадач между вычислительными элементами основывается на анализе информационной связности графа «подзадачи–информационные зависимости». Так, в частности, подзадачи, между которыми имеются интенсивные информационные взаимодействия, целесообразно размещать на вычислительных элементах (ядрах) одного и того же процессора.

Следует отметить, что требование минимизации информационных обменов между вычислительными элементами может противоречить условию равномерной загрузки процессоров вычислительной системы. Так, мы можем разместить все подзадачи на одном процессоре и полностью устранить межпроцессорную передачу данных, однако, понятно, загрузка большинства процессоров в этом случае будет минимальной.

Для нашей учебной задачи поиска максимального значения распределение подзадач между вычислительными элементами не вызывает каких-либо затруднений – размещение подзадач может быть выполнено непосредственно операционной системой.

Решение вопросов балансировки вычислительной нагрузки значительно усложняется, если схема вычислений может изменяться в ходе решения задачи. Причиной этого могут быть, например, неоднородные сетки при решении уравнений в частных производных, разреженность матриц и т. п. [3)]. Кроме того, используемые на этапах проектирования оценки вычислительной сложности решения подзадач могут иметь приближенный характер, и, наконец, количество подзадач может изменяться в ходе вычислений. В таких ситуациях может потребоваться перераспределение базовых подзадач между процессорами уже непосредственно в процессе выполнения параллельной программы (или, как обычно говорят, придется

[3)] Можно отметить, что даже для нашей простой учебной задачи может наблюдаться различная вычислительная сложность сформированных базовых задач. Так, например, количество операций при поиске максимального значения для строки, в которой максимальное значение имеет первый элемент, и строки, в которой значения являются упорядоченными по возрастанию, будет различаться в два раза.

выполнить *динамическую балансировку* вычислительной нагрузки). Данные вопросы являются одними из наиболее сложных (и наиболее интересных) в области параллельных вычислений – к сожалению, рассмотрение данных вопросов выходит за рамки данного учебного материала (дополнительная информация может быть получена, например, в [47,99]).

В качестве примера дадим краткую характеристику широко используемого способа динамического управления распределением вычислительной нагрузки, обычно именуемого *схемой «менеджер–исполнитель»* (*manager–worker scheme*). При использовании данного подхода предполагается, что подзадачи могут возникать и завершаться в ходе вычислений, при этом информационные взаимодействия между подзадачами либо полностью отсутствует, либо минимальны. В соответствии с рассматриваемой схемой для управления распределением нагрузки в системе выделяется отдельный вычислительный элемент-менеджер, которому доступна информация о всех имеющихся подзадачах. Остальные вычислительные элементы системы являются исполнителями, которые для получения вычислительной нагрузки обращаются к менеджеру. Порождаемые в ходе вычислений новые подзадачи передаются обратно менеджеру и могут быть получены для решения при последующих обращениях исполнителей. Завершение вычислений происходит в момент, когда исполнители завершили решение всех переданных им подзадач, а менеджер не имеет каких-либо других вычислительных работ для выполнения.

Предложенный в [54] перечень контрольных вопросов для проверки этапа распределения подзадач состоит в следующем:

- Не приводит ли распределение нескольких задач на один процессор к росту дополнительных вычислительных затрат?
- Существует ли необходимость динамической балансировки вычислений?
- Не является ли вычислительный элемент-менеджер «узким» местом при использовании схемы «менеджер–исполнитель»?

3.3. Параллельное решение гравитационной задачи N тел

Рассмотрим более сложную задачу для демонстрации приведенной выше схемы проектирования и реализации параллельных вычислений.

Многие задачи в области физики сводятся к операциям обработки данных для каждой пары объектов имеющейся физической системы. Такой задачей является, в частности, проблема, широко известная в литературе как *гравитационная задача N тел* (или просто *задача N тел*) – см., например, [39] В самом общем виде, задача может быть описана следующим образом.

Пусть дано большое количество тел (планет, звезд и т. д.), для каждого из которых известна масса, начальное положение и скорость. Под действием гравитации положение тел меняется, и требуемое решение задачи состоит в моделировании динамики изменения системы N тел на протяжении некоторого задаваемого интервала времени. Для проведения такого моделирования заданный интервал времени обычно разбивается на временные отрезки небольшой длительности и далее на каждом шаге моделирования вычисляются силы, действующие на каждое тело, а затем обновляются скорости и положения тел.

Очевидный алгоритм решения задачи N тел состоит в рассмотрении на каждом шаге моделирования всех пар объектов физической системы и выполнении для каждой получаемой пары всех необходимых расчетов.

3.3.1. Разделение вычислений на независимые части

Выбор способа разделения вычислений не вызывает каких-либо затруднений – очевидный подход состоит в выборе в качестве базовой подзадачи всего набора вычислений, связанных с обработкой данных одного какого-либо тела физической системы.

3.3.2. Выделение информационных зависимостей

Выполнение вычислений, связанных с каждой подзадачей, становится возможным только в случае, когда в подзадачах имеются данные (положение и скорости передвижения) обо всех телах имеющейся физической системы. Как результат, перед началом каждой итерации моделирования каждая подзадача должна получить все необходимые сведения от всех других подзадач системы. Такая процедура информационного взаимодействия обычно именуется *операцией сбора данных* (*single-node gather*). В рассматриваемом алгоритме данная операция должна быть выполнена для каждой подзадачи – такой вариант информационного взаимодействия часто именуется как *операция обобщенного сбора данных* (*multi-node gather* или *all gather*).

3.3.3. Масштабирование и распределение подзадач по процессорам

Как правило, число тел физической системы N значительно превышает количество вычислительных элементов p. Как результат, рассмотренные ранее подзадачи следует укрупнить, объединив в рамках одной подзадачи вычисления для группы (N/p) тел – после проведения подобной агрегации число подзадач и количество вычислительных элементов будет совпадать.

Учитывая характер имеющихся информационных зависимостей распределение подзадач между вычислительными элементами может быть непосредственно операционной системой.

3.4. Краткий обзор главы

В главе была рассмотрена методика разработки параллельных алгоритмов, предложенная в [54]. Данная методика включает этапы выделения подзадач, определения информационных зависимостей, масштабирования и распределения подзадач по вычислительным элементам компьютерной системы. При применении методики предполагается, что вычислительная схема решения рассматриваемой задачи уже является известной. Основные требования, которые должны быть обеспечены при разработке параллельных алгоритмов, состоят в обеспечении равномерной загрузки вычислительных элементов при низком информационном взаимодействии сформированного множества подзадач.

Для описания получаемых в ходе разработки вычислительных параллельных схем рассмотрены две модели. Первая из них – модель «подзадачи–информационные зависимости» может быть использована на стадии проектирования параллельных алгоритмов; вторая модель «потоки–общие данные» может быть применена на стадии реализации методов в виде параллельных программ.

В завершение главы показывается применение рассмотренной методики разработки параллельных алгоритмов на примере решения гравитационной задачи N тел.

3.5. Обзор литературы

Рассмотренная в разделе методика разработки параллельных алгоритмов впервые была предложена в [54]. В этой работе изложение методики проводится более детально; кроме того, в работе содержится несколько примеров использования методики для разработки параллельных методов для решения ряда вычислительных задач.

Полезной при рассмотрении вопросов проектирования и разработки параллельных алгоритмов может оказаться также работа [85].

Гравитационная задача N тел более подробно рассматривается в [39].

3.6. Контрольные вопросы

1. В чем состоят исходные предположения для возможности применения рассмотренной в данной главе методики разработки параллельных алгоритмов?

2. Каковы основные этапы проектирования и разработки методов параллельных вычислений?

3. Как определяется модель «подзадачи–информационные зависимости»?

4. Как определяется модель «потоки–общие данные»?

5. Какие основные требования должны быть обеспечены при разработке параллельных алгоритмов?

6. В чем состоят основные действия на этапе выделения подзадач?

7. Каковы основные действия на этапе определения информационных зависимостей?

8. В чем состоят основные действия на этапе масштабирования имеющегося набора подзадач?

9. В чем состоят основные действия на этапе распределения подзадач по вычислительным элементам компьютерной системы?

10. Как происходит динамическое управление распределением вычислительной нагрузки при помощи схемы «менеджер–исполнитель»?

11. Какой метод параллельных вычислений был разработан для решения гравитационной задачи N тел?

3.7. Задачи и упражнения

1. Разработайте схему параллельных вычислений, используя рассмотренную в разделе методику проектирования и разработки параллельных методов:

- для задачи поиска максимального значения среди минимальных элементов строк матрицы (такая задача имеет место для решения матричных игр)

$$y = \max_{1 \le i \le N} \min_{1 \le j \le N} a_{ij},$$

(обратите особое внимание на ситуацию, когда число процессоров превышает порядок матрицы, т. е. $p > N$),

- для задачи вычисления определенного интеграла с использованием метода прямоугольников

$$y = \int_a^b f(x)dx \approx h\sum_{i=0}^{N-1} f_i,\ f_i = f(x_i),\ x_i = i\,h,\ h = (b-a)/N.$$

(описание методов интегрирования дано, например, в [68]),

- Разработайте схему параллельных вычислений для задачи умножения матрицы на вектор, используя рассмотренную в разделе методику проектирования и разработки параллельных методов.