Часть 3.

Методы параллельных вычислений

6. 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.

[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.

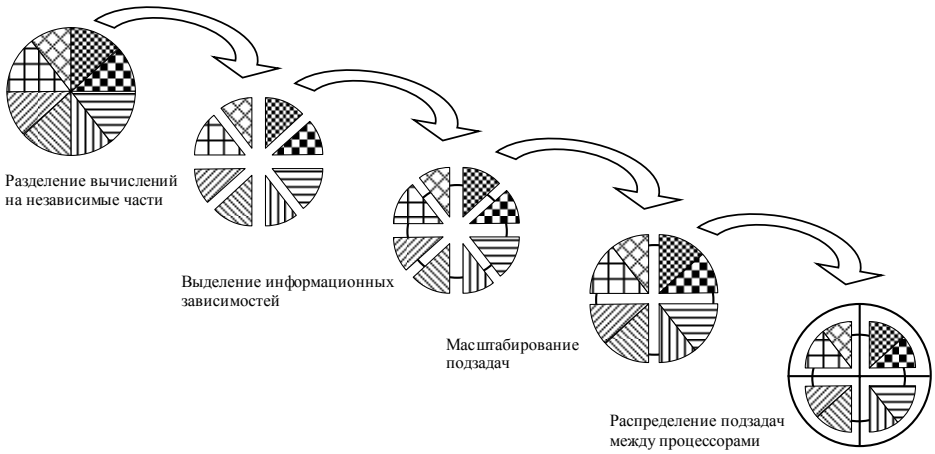


Рис. 6.1. Общая схема разработки параллельных алгоритмов

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

Для применения получаемого в конечном итоге параллельного метода необходимо выполнить разработку *программ* для решения сформированного набора подзадач и разместить разработанные программы по процессорам в соответствии с выбранной схемой распределения подзадач. Для проведения вычислений программы запускаются на выполнение (программы на стадии выполнения обычно именуются *процессами*), для реализации информационных взаимодействий программы должны иметь в своем распоряжении средства обмена данными (*каналы передачи сообщений)*.

Следует отметить, что каждый процессор обычно выделяется для решения одной единственной подзадачи, однако при наличии большого количества подзадач или использовании ограниченного числа процессоров это правило может не соблюдаться и, в результате, на процессорах может выполняться одновременно несколько программ (процессов). В частности, при разработке и начальной проверке параллельной программы для выполнения всех процессов может использоваться один процессор (при расположении на одном процессоре процессы выполняются в режиме распределения времени).

Рассмотрев внимательно разработанную схему проектирования и реализации параллельных вычислений, можно отметить, что данный подход в значительной степени ориентирован на вычислительные системы 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 должны быть заменены операциями доступа к общим (разделяемым) переменным.

6.1. 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 – сообщения"*, который представляет собой не что иное, как укрупненное (агрегированное) представление графа информационных зависимостей (графа "операции-операнды" – см. раздел 2). Аналогично на стадии выполнения для описания параллельной программы может быть использована модель в виде *графа "процессы – каналы"*, в которой вместо подзадач используется понятие процессов, а информационные зависимости заменяются каналами

передачи сообщений. В дополнение, на этой модели может быть показано распределение процессов по процессорам вычислительной системы, если количество подзадач превышает число процессоров – см. рис. 6.2.

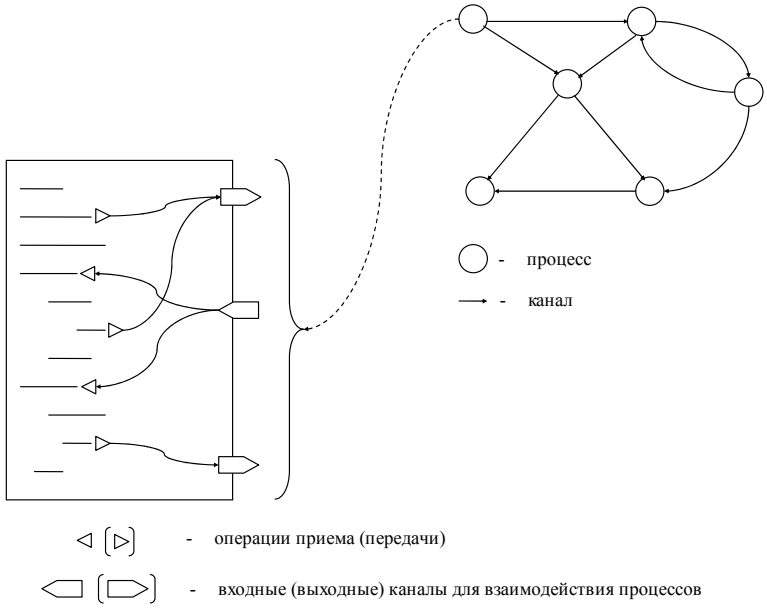


Рис. 6.2. Модель параллельной программы в виде графа "процессы-каналы"

Использование двух моделей параллельных вычислений[2)] позволяет лучше разделить проблемы, которые проявляются при разработке параллельных методов. Первая модель – граф "подзадачи - сообщения" – позволяет сосредоточиться на вопросах выделения подзадач одинаковой вычислительной сложности, обеспечивая при этом низкий уровень информационной зависимости между подзадачами. Вторая модель – граф "процессы – каналы" – концентрирует внимание на вопросах распределения подзадач по процессорам, обеспечивая еще одну возможность снижения трудоемкости информационных взаимодействий между подзадачами за счет размещения на одних и тех же процессорах интенсивно взаимодействующих процессов. Кроме того, эта модель позволяет лучше анализировать эффективность разработанного параллельного метода и обеспечивает возможность более адекватного описания процесса выполнения параллельных вычислений.

Дадим дополнительные пояснения для используемых понятий в модели "процессы-каналы":

- Под *процессом* в рамках данного учебного материала будем понимать выполняемую на процессоре *программу*, которая использует для свой работы часть локальной *памяти* процессора и которая содержит ряд *операций приема/передачи* данных для организации информационного взаимодействия между выполняемыми процессами параллельной программы,
- *Канал передачи данных* с логической точки зрения может рассматриваться как *очередь сообщений*, в которую один или несколько процессов могут отправлять пересылаемые данные и из которой процесс-адресат может извлекать сообщения, отправляемые другими процессами.

В общем случае, можно считать, что каналы возникают динамически в момент выполнения первой операции приема/передачи с каналом. По степени общности, канал может соответствовать одной или нескольким командам приема данных процесса-получателя; аналогично при передаче сообщений канал может использоваться одной или несколькими командами передачи данных одного или нескольких процессов. Для снижения сложности моделирования и анализа параллельных методов будем предполагать, что емкость каналов является неограниченной и, как результат, операции передачи данных выполняются практически без задержек простым копированием сообщений в канал. С другой стороны, операции приема сообщений могут приводить к задержкам (*блокировкам*), если запрашиваемые из канала данные еще не были отправлены процессами-источниками сообщений.

Следует отметить важное достоинство рассмотренной модели "процессы-каналы" – в этой модели проводится четкое разделение локальных (выполняемых на отдельном процессоре) вычислений и

[2)] В Foster (1995) 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.

действий по организации информационного взаимодействия одновременно выполняемых процессов. Такой подход значительно снижает сложность анализа эффективности параллельных методов и существенно упрощает проблемы разработки параллельных программ.

6.2. Этапы разработки параллельных алгоритмов

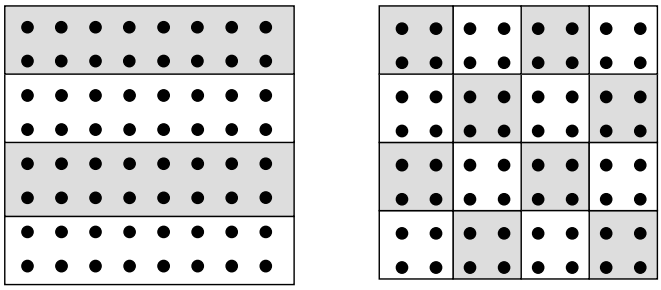
Рассмотрим более подробно изложенную выше методику разработки параллельных алгоритмов. В значительной степени данная методика опирается на подход, впервые рассмотренный в Foster (1995), и, как отмечалось ранее, включает этапы выделения подзадач, определения информационных зависимостей, масштабирования и распределения подзадач по процессорам вычислительной системы (см. рис. 6.1). Для демонстрации приводимых рекомендаций далее будет использоваться учебная задача поиска максимального значения среди элементов матрицы $\boldsymbol{A}$ (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 раздела 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.

6.2.1. 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 состоят 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:



а) б)

Рис. 6.3. Разделение данных для матрицы A: а) ленточная схема, б) блочная схема

- Для большого класса задач вычисления сводятся к выполнению однотипной обработки элемент элементов 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. Так, например, для нашей учебной задачи поиска максимального значения при формировании подзадач исходная матрица $\boldsymbol{A}$ может быть разделена на отдельные строки (или последовательные группы строк) – *ленточная схема* разделения данных (см. рис. 6.3) или на прямоугольные наборы элементов – *блочная схема* разделения данных. Для большого количества решаемых задач разделение вычислений по данным приводит к порождению одно-, двух- и трех- мерных наборов подзадач, для которых информационные связи существуют только между ближайшими соседями (такие схемы обычно именуются *сетками* или *решетками*),

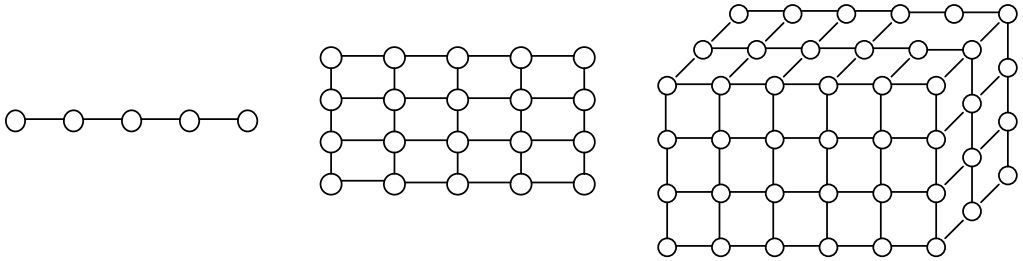


Рис. 6.4. Регулярные одно-, двух- и трех- мерные структуры базовых подзадач после декомпозиции данных

- 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 примеров 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).

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 структура информационных связей соответствует линейному графу – см. рис. 6.5.

Для оценки корректности этапа разделения вычислений на независимые части можно воспользоваться контрольным списком вопросов, предложенных в Foster (1995):

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

6.2.2. 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* передачи данных – для локальных схем передачи данных в каждый момент времени выполняются только между небольшим числом подзадач (располагаемых, как

правило, на соседних процессорах), для глобальных операций передачи данных в процессе коммуникации принимают участие все подзадачи,

- *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 выполняемых операций передач данных не носит характер однородности,
- *Статические или динамические схемы* передачи данных – для статических схем моменты и участники информационного взаимодействия фиксируются на этапах проектирования и разработки параллельных программ, для динамического варианта взаимодействия структура операции передачи данных определяется в ходе выполняемых вычислений,
- *Синхронные и асинхронные способы взаимодействия* – для синхронных способов операции передачи данных выполняются только при готовности всех участников взаимодействия и завершаются только после полного окончания всех коммуникационных действий, при асинхронном выполнении операций участники взаимодействия могут не дожидаться полного завершения действий по передаче данных. Для представленных способов взаимодействия достаточно сложно выделить предпочтительные формы организации передачи данных: синхронный вариант, как правило, более прост для использования, в то время как асинхронный способ часто позволяет существенно снизить временные задержки, вызванные операциями информационного взаимодействия.

Как уже отмечалось в предыдущем пункте, для учебной задачи поиска максимального значения при использовании в качестве базовых элементов подзадач поиска максимальных значений в отдельных строках исходной матрицы *A* структура информационных связей имеет вид, представленный на рис. 6.5.

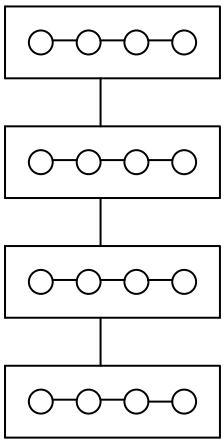


Рис. 6.5. Структура информационных связей учебной задачи

Как и ранее, для оценки правильности этапа выделения информационных зависимостей можно воспользоваться контрольным списком вопросов, предложенных в Foster (1995):

- 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?

6.2.3. Масштабирование набора подзадач

Масштабирование разработанной вычислительной схемы параллельных вычислений проводится в случае, если количество имеющихся подзадач отличается от числа планируемых к использованию процессоров. 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 процессоров 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 процессоров, 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. 6.3а), при декомпозиции подзадач строки исходной матрицы $\boldsymbol{A}$ могут разбиваться на несколько частей (блоков).

Список контрольных вопросов, предложенный в Foster (1995) 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 процессоров?
- Зависят ли параметрически правила масштабирования от количества процессоров?

6.2.4. Распределение подзадач между процессорами

Распределение подзадач между процессорами 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 процессоров, а топология сети передачи данных вычислительной системы представляет собой полный граф (т.е., все процессоры связаны между собой прямыми линиями связи).

Основной показатель успешности выполнения данного этапа – *эффективность использования процессоров*, определяемая как относительная доля времени, в течение которого процессоры 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 процессорами и минимизировать количество сообщений, передаваемых между процессорами. Точно так же, как и на предшествующих этапах проектирования, оптимальное решение проблемы распределения подзадач между процессорами основывается на анализе информационной связности графа "подзадачи - сообщения". Так, в частности, подзадачи, между которыми имеются информационные взаимодействия, целесообразно размещать на процессорах, между которыми существуют прямые линии передачи данных.

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

Для нашей учебной задачи поиска максимального значения распределение подзадач между процессорами не вызывает каких-либо затруднений – достаточно лишь обеспечить размещение подзадач, между которыми имеются информационные связи, на процессорах, для которых существуют прямые каналы передачи данных. Поскольку структура информационной связей учебной задачи имеет вид линейного графа, выполнение данного требования может быть обеспечено практически при любой топологии сети вычислительной системы.

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

[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.

процессорами уже непосредственно в процессе выполнения параллельной программы (или, как обычно говорят, придется 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 Buyya (1999) и Wilkinson and Allen (1999)).

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 процессор-менеджер, которому доступна информация обо всех имеющихся подзадачах. Остальные процессоры системы являются исполнителями, которые для получения вычислительной нагрузки обращаются к процессору-менеджеру. Порождаемые в ходе вычислений новые подзадачи передаются обратно процессору-менеджеру и могут быть получены для решения при последующих обращениях процессоров-исполнителей. Завершение вычислений происходит в момент, когда процессоры-исполнители завершили решение всех переданных им подзадач, а процессор-менеджер не имеет каких-либо вычислительных работ для выполнения.

Предложенный в Foster (1995) 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 процессор-менеджер "узким" местом при использовании схемы "менеджер-исполнитель"?

6.3. Параллельное решение гравитационной задачи N тел

Многие вычислительные 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, Andrews (2000) 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. Как результат, при таком подходе время выполнения одной итерации моделирования будет составлять[4)]

$$T_1 = \tau N(N-1)/2,$$

где τ есть время перевычисления параметров одной пары тел.

Как следует из приведенного описания, вычислительная схема рассмотренного алгоритма является сравнительно простой, что позволяет использовать задачу N тел в качестве еще одной наглядной демонстрации применения методики разработки параллельных алгоритмов.

[4)] Следует отметить, что для решения задачи N тел существует и более эффективные последовательные алгоритмы, однако их изучение может потребовать достаточно больших усилий. С учетом данного обстоятельства для дальнейшего рассмотрения выбирается именно данный "очевидный" (но не самый быстрый) метод, хотя, в общем случае, безусловно, для распараллеливания следует выбирать наилучшие схемы выполнения расчетов.

6.3.1. 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.

6.3.2. 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 передачи данных, как отмечалось в разделе 3, именуется *операцией сбора данных* (*single-node gather*). В рассматриваемом алгоритме данная операция должна быть выполнена для каждой подзадачи – такой вариант передачи данных обычно именуется как *операция обобщенного сбора данных* (*multi-node gather* or *all gather*).

Определение требований к необходимым результатам информационного обмена не приводит к однозначному установлению нужного информационного обмена между подзадачами – достижение требуемых результатов может быть обеспечено при помощи разных алгоритмов выполнения операции обобщенного сбора данных.

Наиболее простой способ выполнения необходимого информационного обмена состоит в реализации последовательности шагов, на каждом из которых все имеющиеся подзадачи разбиваются попарно и обмен данными осуществляется между подзадачами образовавшихся пар. При надлежащей организации попарного разделения подзадач (*N-1*)-кратное повторение описанных действий приведет к полной реализации требуемой операции сбора данных.

Рассмотренный выше метод организации информационного обмена является достаточно трудоемким – для сбора всех необходимых данных требуется (*N-1*) итераций, на каждой из которых выполняется одновременно (*N/2*) операций передачи данных. Для сокращения требуемого количества итераций можно обратить внимание на факт, что после выполнения первого шага операции сбора данных подзадачи будут уже содержать не только свои данные, но и данные подзадач, с которыми они образовывали пары. Как результат, на второй итерации сбора данных можно будет образовывать пары подзадач для обмена данными сразу о двух телах физической системы – тем самым, после завершения второй итерации каждая подзадача будет содержать сведения о четырех телах системы и т.д. Как можно заметить, данный способ реализации обменов позволяет завершить необходимую процедуру за $log_2 N$ итераций. Следует отметить, что при этом объем пересылаемых данных в каждой операции обмена удваивается от итерации к итерации – на первой итерации между подзадачами пересылаются данные об одном теле системы, на второй итерации – о двух телах и т.д.

Использование рассмотренного способа реализации операции обобщенного сбора данных приводит к определению структуры информационных связей между подзадачами в виде *N*-мерного гиперкуба.

6.3.3. Масштабирование и распределение подзадач по процессорам

Как правило, число тел физической системы *N* значительно превышает количество процессоров *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 процессоров будет совпадать, и при распределении подзадач между процессорами останется лишь обеспечить наличие прямых коммуникационных линий между процессорами с подзадачами, между которыми имеются информационные обмены при выполнении операции сбора данных.

6.3.4. Анализ эффективности параллельных вычислений

Оценим эффективность разработанных способов параллельных вычислений для решения задачи *N* тел. Поскольку предложенные варианты отличаются только методами выполнения информационных обменов, для сравнения подходов достаточно определить длительность операции обобщенного сбора данных. Используем для оценки времени передачи сообщений модель, предложенную Хокни (см. раздел 3), тогда длительность выполнения операции сбора данных для первого варианта параллельных вычислений может быть выражена как

$$T_p^1(comm) = (p-1)(\alpha + m(N/p)/\beta),$$

где α, β есть параметры модели Хокни (латентность и пропускная способность сети передачи данных), а *m* задает объем пересылаемых данных для одного тела физической системы.

Для второго способа информационного обмена, как уже отмечалось ранее, объем пересылаемых данных на разных итерациях операции сбора данных различается. На первой итерации объем пересылаемых сообщений составляет (mN/p), на второй итерации этот объем увеличивается вдвое и оказывается равным $2(mN/p)$ и т.д. В общем случае, для итерации с номером i объем сообщений оценивается как $2^{i-1}(mN/p)$. Как результат, длительность выполнения операции сбора данных в этом случае может быть определена при помощи следующего выражения

$$T_p^2(comm) = \sum_{i=1}^{\log p}(\alpha + 2^{i-1} m(N/p)/\beta) = \alpha \log p + m\ (N/p)(p-1)/\beta\ .$$

Сравнение полученных выражений показывает, что второй разработанный способ параллельных вычислений имеет существенно более высокую эффективность, несет меньшие коммуникационные затраты и допускает лучшую масштабируемость при увеличении количества используемых процессоров.

6.4. Краткий обзор раздела

В разделе была рассмотрена методика разработки параллельных алгоритмов, предложенная в Foster (1995). Данная методика включает этапы выделения подзадач, определения информационных зависимостей, масштабирования и распределения подзадач по процессорам вычислительной 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 процессоров при низком информационном взаимодействии сформированного множества подзадач.

Для описания получаемых в ходе разработки вычислительных параллельных схем рассмотрены две модели. Первая из них – модель "подзадачи-сообщения" может быть использована на стадии проектирования параллельных алгоритмов, вторая модель "процессы-каналы" может быть применена на стадии реализации методов в виде параллельных программ.

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

6.5. Обзор литературы

Рассмотренная в разделе методика разработки параллельных алгоритмов впервые была предложена в Foster (1995). 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 Quinn (2004).

Гравитационная задача N тел более подробно рассматривается в Andrews (2000).

6.6. Контрольные вопросы

1. В чем состоят исходные предположения для возможности применения рассмотренной в разделе методики разработки параллельных алгоритмов?
2. Каковы основные этапы проектирования и разработки методов параллельных вычислений?
3. Как определяется модель "подзадачи-сообщения"?
4. Как определяется модель "процессы-каналы"?
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 процессорам вычислительной 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?

12. Какой способ выполнения операции обобщенного сбора данных является более эффективным?

6.7. Задачи и упражнения

1. Выполните реализацию каскадной схемы вычисления суммы последовательности числовых значений (см. раздел 2) и сравните время выполнения выполненной реализации и функции *MPI_Bcast* библиотеки MPI.

2. Выполните реализацию рассмотренных способов выполнения обобщенной операции сбора данных и сравните время их выполнения. Сопоставьте получаемые временные характеристики с имеющими теоретическими оценками. Выполните сравнение со временем выполнения функции *MPI_Allgather* библиотеки MPI.

3. 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 Kahaner, Moler and Nash (1988))

4. Выполните реализацию разработанных параллельных методов для задач п. 3.

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

Литература

Andrews, G. R. (2000). Foundations of Multithreaded, Parallel, and Distributed Programming.. – Reading, MA: Addison-Wesley (русский перевод Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. – М.: Издательский дом "Вильямс", 2003)

Bertsekas, **D.P., Tsitsiklis, J.N.** (1989) Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.

Buyya, R. (Ed.) (1999). High Performance Cluster Computing. Volume1: Architectures and Systems. Volume 2: Programming and Applications. - Prentice Hall PTR, Prentice-Hall Inc.

Kahaner, D., Moler, C., Nash, S. (1988). Numerical Methods and Software. – Prentice Hall (русский перевод Каханер Д., Моулер Л., Нэш С. Численные методы и программное обеспечение. М.: Мир, 2001)

Foster, I. (1995). Designing and Building Parallel Programs: Concepts and Tools for Software Engineering. Reading, MA: Addison-Wesley.

Quinn, M. J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.

Wilkinson, B., Allen, M. (1999). Parallel programming. – Prenrice Hall.