Высокопроизводительные параллельные вычисления

Лекция №10
Тема: Организация параллельных вычислений для систем с распределенной памятью

Козбакова Айнур Холдасовна– PhD, профессор,
кафедра «Информационные системы»
Факультет «Инжиниринг и информационных технологии» 01.09.2025 г.

Использование процессоров с распределенной памятью является другим общим способом построения многопроцессорных вычислительных систем. Актуальность их становится все более высокой в последнее время в связи с широким развитием высокопроизводительных кластерных вычислительных систем

Многие проблемы параллельного программирования (состязание вычислений, тупики, сериализация) являются общими для систем с общей и распределенной памятью. Основной момент, который отличает параллельные вычисления с распределенной памятью, состоит в том, что взаимодействие параллельных участков программы на разных процессорах может быть обеспечено только при помощи *передачи сообщений* (*message passing*).

Следует отметить, что вычислительный узел системы с распределенной памятью является, как правило, более сложным вычислительным устройством, чем процессор в многопроцессорной системе с общей памятью. Для учета этих различий в дальнейшем процессор с распределенной памятью будет именоваться *вычислительным сервером* (сервером может быть, в частности, многопроцессорная система с общей памятью). При проведении всех ниже рассмотренных экспериментов использовались 4 компьютера с процессорами Pentium IV, 1300 Mhz, 256 RAM, 100 Mbit Fast Ethernet.

Общие принципы распределения данных

Первая проблема, которую приходится решать при организации параллельных вычислений на *системах с распределенной памятью*, обычно состоит в выборе способа разделения обрабатываемых данных между вычислительными серверами. Успешность такого разделения определяется достигнутой степенью локализации вычислений на серверах (в силу больших временных задержек при передаче сообщений интенсивность взаимодействия серверов должна быть минимальной).

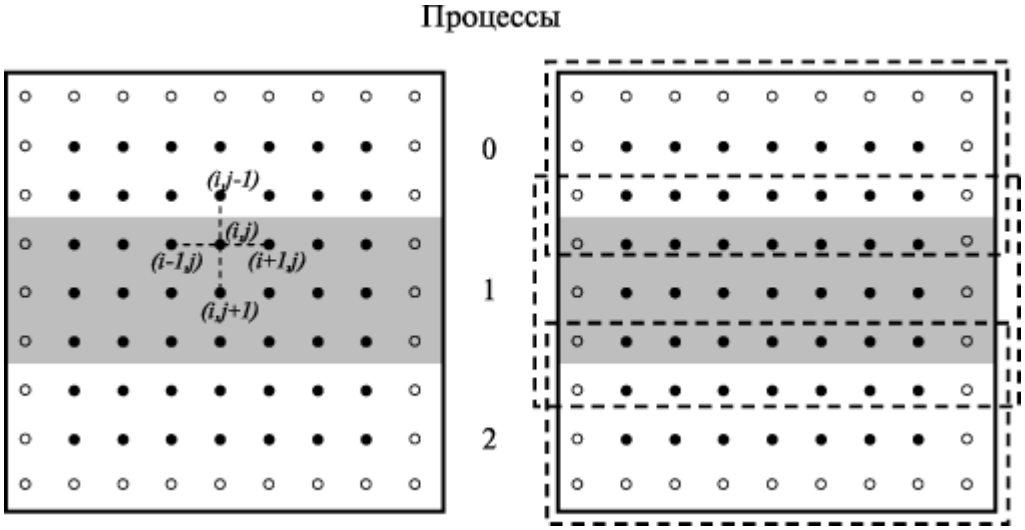

Рис. 10. Ленточное разделение области расчетов между процессорами (кружки представляют граничные узлы сетки)

В рассматриваемом учебном примере по решению *задачи Дирихле* возможны два различных способа разделения данных – *одномерная* или *ленточная* схема (см. рис. 10) и *двумерное* или *блочное* разбиение (см. рис. 9) вычислительной сетки. Дальнейшее изложение учебного материала будет проводиться на примере первого подхода; блочная схема будет рассмотрена позднее в более кратком виде.

При ленточном разбиении область расчетов делится на горизонтальные или вертикальные полосы (не уменьшая общности, далее будем рассматривать только горизонтальные полосы). Число полос определяется количеством процессоров, размер

полос обычно является одинаковым, узлы горизонтальных границ (первая и последняя строки) включаются в первую и последнюю полосы соответственно. Полосы для обработки распределяются между процессорами.

Основной момент при организации вычислений с подобным разделением данных состоит в том, что на процессор, выполняющий обработку какой-либо полосы, должны быть продублированы граничные строки предшествующей и следующей полос вычислительной сетки (получаемые в результате расширенные полосы показаны на рис. 10 справа пунктирными рамками). Продублированные граничные строки полос используются только при проведении расчетов, пересчет же этих строк происходит в полосах своего исходного месторасположения. Тем самым, дублирование граничных строк должно осуществляться перед началом выполнения каждой очередной итерации метода сеток.

Обмен информацией между процессорами

Параллельный вариант метода сеток при ленточном разделении данных состоит в обработке полос на всех имеющихся серверах одновременно в соответствии со следующей схемой работы:

Алгоритм 8. Параллельный алгоритм, реализующий метод сеток при ленточном разделении данных

```
// Алгоритм 8
// схема Гаусса-Зейделя, ленточное разделение данных
// действия, выполняемые на каждом процессоре
do {
  // <обмен граничных строк полос с соседями>
  // <обработка полосы>
  // <вычисление общей погрешности вычислений dmax>}
 while ( dmax > eps ); // eps — точность решения
```

Для конкретизации представленных в алгоритме действий введем обозначения:

- ProcNum – номер процессора, на котором выполняются описываемые действия,
- PrevProc, NextProc – номера соседних процессоров, содержащих предшествующую и следующую полосы,
- NP – количество процессоров,
- M – количество строк в полосе (без учета продублированных граничных строк),
- N – количество внутренних узлов в строке сетки (т.е. всего в строке N+2 узла).

При нумерации строк полосы будем считать, что строки 0 и M+1 есть продублированные из соседних полос граничные строки, а строки собственной полосы процессора имеют номера от 1 до M.

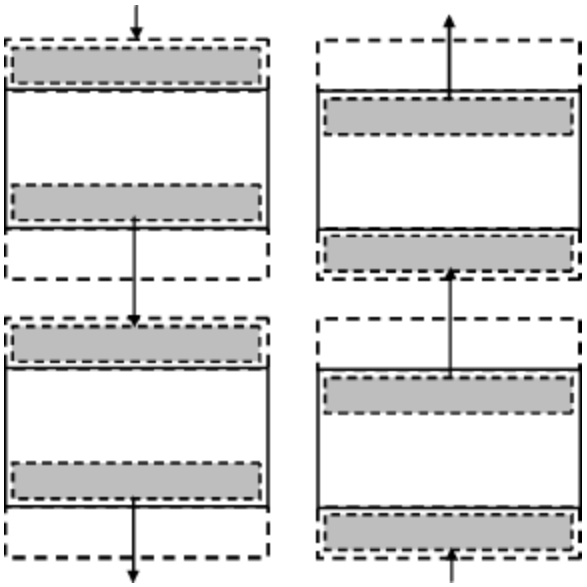

Рис. Схема передачи граничных строк между соседними процессорами

Процедура обмена граничных строк между соседними процессорами может быть разделена на две последовательные операции, во время первой из которых каждый процессор передает свою нижнюю граничную строку следующему процессору и принимает такую же строку от предыдущего процессора (см. рис. 11). Вторая часть передачи строк выполняется в обратном направлении: процессоры передают свои верхние граничные строки своим предыдущим соседям и принимают переданные строки от следующих процессоров.

Выполнение подобных *операций передачи данных* в общем виде может быть представлено следующим образом (для краткости рассмотрим только первую часть процедуры обмена):

```
// передача нижней граничной строки следующему
// процессору и прием передаваемой строки от
// предыдущего процессора
if ( ProcNum != NP-1 ) Send(u[M][*],N+2,NextProc);
if ( ProcNum != 0 ) Receive(u[0][*],N+2,PrevProc);
```

(для записи процедур приема-передачи используется близкий к стандарту *MPI* (см. "Параллельное программирование на основе MPI") формат, где первый и второй параметры представляют пересылаемые данные и их объем, а третий параметр определяет адресата (для операции Send) или источник (для операции Receive) пересылки данных).

Для передачи данных могут быть задействованы два различных механизма. При первом из них выполнение программ, инициировавших операцию передачи, приостанавливается до полного завершения всех действий по пересылке данных (т.е. до момента получения процессором-адресатом всех передаваемых ему данных). Операции приема-передачи, реализуемые подобным образом, обычно называются *синхронными* или *блокирующими*. Иной подход – *асинхронная* или *неблокирующая* передача — может состоять в том, что операции приема-передачи только инициируют процесс пересылки и на этом завершают свое выполнение. В результате программы, не дожидаясь завершения длительных коммуникационных операций, могут продолжать свои вычислительные действия, проверяя по мере необходимости готовность передаваемых данных. Оба эти варианта операций передачи широко используются при организации параллельных вычислений и имеют свои

достоинства и свои недостатки. Синхронные процедуры передачи, как правило, более просты для применения и более надежны; неблокирующие операции могут позволить совместить процессы передачи данных и вычислений, но обычно приводят к повышению сложности программирования. С учетом вышесказанного во всех последующих примерах для организации пересылки данных будут использоваться операции приема-передачи блокирующего типа.

Приведенная выше последовательность блокирующих операций приема-передачи данных (вначале Send, затем Receive) приводит к строго последовательной схеме выполнения процесса пересылок строк, т.к. все процессоры одновременно обращаются к операции Send и переходят в режим ожидания. Первым процессором, который окажется готовым к приему пересылаемых данных, будет сервер с номером NP-1. В результате процессор NP-2 выполнит операцию передачи своей граничной строки и перейдет к приему строки от процессора NP-3 и т.д. Общее количество повторений таких операций равно NP-1. Аналогично происходит выполнение и второй части процедуры пересылки граничных строк перед началом обработки строк (см. рис. 11).

Последовательный характер рассмотренных операций пересылок данных определяется выбранным способом очередности выполнения. Изменим этот порядок очередности при помощи чередования приема и передачи для процессоров с четными и нечетными номерами.

```
// передача нижней граничной строки следующему
// процессору и прием передаваемой строки от
// предыдущего процессора
if ( ProcNum % 2 == 1 ) { //  нечетный процессор
  if ( ProcNum != NP-1 ) Send(u[M][*],N+2,NextProc);
  if ( ProcNum != 0 ) Receive(u[0][*],N+2,PrevProc);
}
else { //  процессор с четным номером
  if ( ProcNum != 0 ) Receive(u[0][*],N+2,PrevProc);
  if ( ProcNum != NP-1 ) Send(u[M][*],N+2,NextProc);
}
```

Данный прием позволяет выполнить все необходимые операции передачи всего за два последовательных шага. На первом шаге все процессоры с нечетными номерами отправляют данные, а процессоры с четными номерами осуществляют прием этих данных. На втором шаге роли процессоров меняются – четные процессоры выполняют Send, нечетные процессоры исполняют операцию приема Receive.

Рассмотренные последовательности операций приема-передачи для взаимодействия соседних процессоров широко используются в практике параллельных вычислений. Как результат, во многих базовых библиотеках параллельных программ имеются процедуры для поддержки подобных действий. Так, в стандарте *MPI* (см. *лекцию 5*) предусмотрена операция Sendrecv, с использованием которой предыдущий фрагмент программного кода может быть записан более кратко:

```
// передача нижней граничной строки следующему
// процессору и прием передаваемой строки от
// предыдущего процессора
Sendrecv(u[M][*],N+2,NextProc,u[0][*],N+2,PrevProc);
```

Реализация подобной объединенной функции Sendrecv обычно осуществляется таким образом, чтобы обеспечить и корректную работу на крайних процессорах, когда не нужно выполнять одну из операций передачи или приема, и организацию чередования процедур

передачи на процессорах для ухода от тупиковых ситуаций, и возможности параллельного выполнения всех необходимых пересылок данных.

Коллективные операции обмена информацией

Для завершения круга вопросов, связанных с параллельной реализацией метода сеток на *системах с распределенной памятью*, осталось рассмотреть способы вычисления общей для всех процессоров погрешности вычислений. Возможный очевидный подход состоит в передаче всех локальных оценок погрешности, полученных на отдельных полосах сетки, на один какой-либо процессор, вычислении на нем максимального значения и последующей рассылке полученного значения всем процессорам системы. Однако такая схема является крайне неэффективной – количество необходимых *операций передачи данных* определяется числом процессоров и выполнение этих операций может происходить только в последовательном режиме. Между тем, как показывает анализ требуемых коммуникационных действий, выполнение операций сборки и рассылки данных может быть реализовано с использованием рассмотренной в п. 2.5.2 *каскадной схемы* обработки данных. На самом деле, получение максимального значения локальных погрешностей, вычисленных на каждом процессоре, может быть обеспечено, например, путем предварительного нахождения максимальных значений для отдельных пар процессоров (такие вычисления могут выполняться параллельно), затем может быть снова осуществлен попарный поиск максимума среди полученных результатов и т.д. Всего, как полагается по каскадной схеме, необходимо выполнить $\log_2 NP$ параллельных итераций для получения конечного значения (NP – количество процессоров).

С учетом возможности применения *каскадной схемы* для выполнения коллективных *операций передачи данных* большинство базовых библиотек параллельных программ содержит процедуры для поддержки подобных действий. Так, в стандарте *MPI* (см. "Параллельное программирование на основе MPI") предусмотрены операции:

- Reduce(dm,dmax,op,proc) – процедура сборки на процессоре proc итогового результата dmax среди локальных на каждом процессоре значений dm с применением операции op ;
- Broadcast(dmax,proc) – процедура рассылки с процессора proc значения dmax всем имеющимся процессорам системы.

С учетом перечисленных процедур общая схема вычислений на каждом процессоре может быть представлена в следующем виде:

```
// Алгоритм  8 – уточненный вариант
// схема Гаусса-Зейделя, ленточное разделение данных
// действия, выполняемые на каждом процессоре
do {
  // обмен граничных строк полос с соседями
  Sendrecv(u[M][*],N+2,NextProc,u[0][*],N+2,PrevProc);
  Sendrecv(u[1][*],N+2,PrevProc,u[M+1][*],N+2,NextProc);
  // <обработка полосы с оценкой погрешности dm>
  // вычисление общей погрешности вычислений dmax
  Reduce(dm,dmax,MAX,0);
  Broadcast(dmax,0);
} while ( dmax > eps ); // eps — точность решения
```

(в приведенном алгоритме переменная dm представляет собой локальную погрешность вычислений на отдельном процессоре, а параметр MAX задает операцию поиска максимального значения для операции сборки). Следует отметить, что в составе *MPI* имеется процедура Allreduce, которая совмещает действия редукции и

рассылки данных. Результаты экспериментов для данного варианта параллельных вычислений для метода Гаусса – Зейделя приведены в рис. 4

Организация волны вычислений

Представленные в пп. 3.1 – 3.3 алгоритмы определяют общую схему параллельных вычислений для метода сеток в многопроцессорных системах с распределенной памятью. Далее эта схема может быть конкретизирована реализацией практически всех вариантов методов, рассмотренных для систем с общей памятью (применение дополнительной памяти для схемы Гаусса – Якоби, чередование обработки полос и т.п.). Проработка таких вариантов не привносит каких-либо новых эффектов с точки зрения параллельных вычислений, и их разбор может использоваться как тема заданий для самостоятельных упражнений.

Таблица 4. Результаты экспериментов для систем с распределенной памятью, ленточная схема разделения данных (p=4)

Размер сетки	**Последовательный метод Гаусса - Зейделя (алгоритм 1)**		**Параллельный алгоритм 8**			**Параллельный алгоритм с волновой схемой расчета (см. п. 3.4)**		
	k	**t**	**k**	**t**	**S**	**k**	**t**	**S**
100	210	0,060	21	0,541	0,110	21	1,275	0,0
200	273	0,353	27	0,861	0,43	27	1,376	0,2
300	305	0,925	30	0,920	1,05	30	1,830	0,5
400	318	1,698	31	1,273	1,38	31	2,537	0,6
500	343	2,883	34	1,728	1,63	34	3,268	0,8
600	336	4,046	33	2,167	1,86	33	3,660	1,1
700	344	5,684	34	2,525	2,24	34	4,642	1,2
800	343	7,373	34	3,322	2,23	34	5,650	1,3
900	358	9,948	35	4,121	2,48	35	7,532	1,3

1000	351	11,87 1	35	4,43 8	2,6 1	35	8,10 6	1,4
2000	367	50,19 7	36 3	15,1 2	3,3 7	36 0	27,0 6	1,8
3000	364	113,17 4	36 6	37,9 8	2,9 4	36 6	55,7 3	2,0

(k – количество итераций, t – время (сек), S – ускорение)

В завершение рассмотрим возможность организации параллельных вычислений, при которых обеспечивалось бы нахождение таких же решений задачи Дирихле, что и при использовании исходного последовательного метода Гаусса – Зейделя. Как отмечалось ранее, такой результат может быть получен за счет организации волновой схемы расчетов. Для образования волны вычислений представим логически каждую полосу узлов области расчетов в виде набора блоков (размер блоков можно положить, в частности, равным ширине полосы) и организуем обработку полос поблочно в последовательном порядке (см. рис. 12). Тогда для полного повторения действий последовательного алгоритма вычисления могут быть начаты только для первого блока первой полосы узлов; после того как этот блок будет обработан, для вычислений будут готовы уже два блока – блок 2 первой полосы и блок 1 второй полосы (для обработки блока полосы 2 необходимо передать граничную строку узлов первого блока полосы 1). После обработки указанных блоков к вычислениям будут готовы уже 3 блока, и мы получаем знакомый уже процесс волновой обработки данных (результаты экспериментов см. в табл. 4).

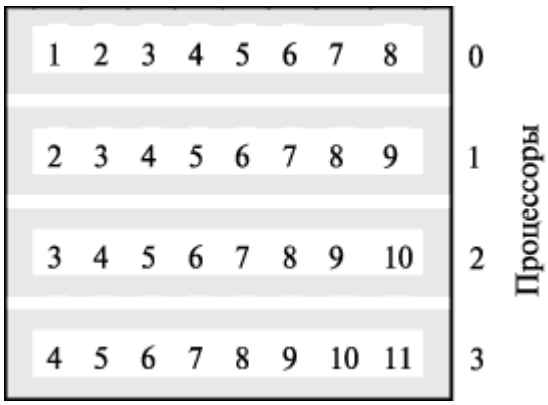

Рис. 12. Организация волны вычислений при ленточной схеме разделения данных

Интересный момент при организации подобной схемы параллельных вычислений может состоять в попытке совмещения операций пересылки граничных строк и действий по обработке блоков данных.

Блочная схема разделения данных

Ленточная схема разделения данных может быть естественным образом обобщена на блочный способ представления сетки области расчетов (см. рис. 9). При этом столь радикальное изменение способа разбиения сетки практически не потребует каких-либо существенных корректировок рассмотренной схемы параллельных вычислений. Основной новый момент при блочном представлении данных состоит в увеличении количества граничных строк на каждом процессоре (для блока их количество становится равным 4), что приводит, соответственно, к большему числу операций передачи данных при обмене

граничных строк. Сравнивая затраты на организацию передачи граничных строк, можно отметить, что при ленточной схеме для каждого процессора выполняется 4 операции приема-передачи данных, в каждой из которых пересылаются (N+2) значения. Для блочного же способа происходит 8 операций пересылки и объем каждого сообщения равен $(N/\sqrt{NP}+2)$ (N – количество внутренних узлов сетки, NP – число процессоров, размер всех блоков предполагается одинаковым). Тем самым, блочная схема представления области расчетов становится оправданной при большом количестве узлов сетки, когда увеличение количества коммуникационных операций приводит к снижению затрат на пересылку данных в силу сокращения размеров передаваемых сообщений. Результаты экспериментов при блочной схеме разделения данных приведены в табл. 5.

Таблица 5. Результаты экспериментов для систем с распределенной памятью, блочная схема разделения данных (p=4)

Размер сетки	**Последовательный метод Гаусса - Зейделя (алгоритм 1)**		**Параллельный алгоритм с блочной схемой рассчета (см. п. 3.5)**			**Параллельный алгоритм 9**		
	k	**t**	**k**	**t**	**S**	**k**	**t**	**S**
100	210	0,060	21	0,718	0,00	21	0,600	0,10
200	273	0,353	27	0,747	0,43	27	1,063	0,33
300	305	0,925	30	1,048	0,85	30	2,016	0,46
400	318	1,698	31	1,448	1,18	31	2,634	0,64
500	343	2,883	34	1,911	1,53	34	3,600	0,80
600	336	4,046	33	2,399	1,66	33	4,637	0,87
700	344	5,684	34	2,962	1,94	34	5,818	0,97
800	343	7,373	34	3,586	2,03	34	7,656	0,96
900	358	9,948	35	4,501	2,28	35	9,574	1,04
1000	351	11,871	35	4,902	2,41	356	11,166	1,06

2000	367	50,19 7	36 7	16,0 2	3,1 7	36 9	39,4 7	1,2
3000	364	113,17 4	36 5	39,2 8	2,8 4	36 2	85,7 2	1,3

(k – количество итераций, t – время (сек), S – ускорение)

При блочном представлении сетки может быть реализован также и волновой метод выполнения расчетов (см. рис. 13). Пусть процессоры образуют прямоугольную решетку размером $NB \times NB (NB = \sqrt{NP})$ и процессоры пронумерованы от 0 слева направо по строкам решетки.

Общая схема параллельных вычислений в этом случае имеет вид:

Алгоритм 9. Блочная схема разделения данных

```
// Алгоритм  9
// схема Гаусса-Зейделя, блочное разделение данных
// действия, выполняемые на каждом процессоре
do {
  // получение граничных узлов
  if ( ProcNum / NB != 0 ) { // строка не нулевая
    // получение данных от верхнего процессора
    Receive(u[0][*],M+2,TopProc);          // верхняя строка
    Receive(dmax,1,TopProc);               // погрешность
  }
  if ( ProcNum % NB != 0 ) { // столбец не нулевой
    // получение данных от левого процессора
    Receive(u[*][0],M+2,LeftProc);         // левый столбец
    Receive(dm,1,LeftProc);       // погрешность
    if ( dm > dmax ) dmax = dm;
  }
  // <обработка блока с оценкой погрешности dmax>
  // пересылка граничных узлов
  if ( ProcNum / NB != NB-1 ) { // строка решетки не последняя
    // пересылка данных нижнему процессору
    Send(u[M+1][*],M+2,DownProc);      // нижняя строка
    Send(dmax,1,DownProc);             // погрешность
  }
  if ( ProcNum % NB != NB-1 ) { // столбец решетки не последний
    // пересылка данных правому процессору
    Send(u[*][M+1],M+2,RightProc);       // правый столбец
    Send(dmax,1, RightProc);     // погрешность
  }
  // синхронизация и рассылка погрешности dmax
  Broadcast(dmax,NP-1);
} while ( dmax > eps ); // eps — точность решения
 9.
```

При реализации алгоритма необходимо обеспечить, чтобы в начальный момент времени все процессоры (кроме процессора с нулевым номером) оказались в состоянии передачи своих граничных узлов (верхней строки и левого столбца). Вычисления должен

начинать процессор с левым верхним блоком, после завершения обработки которого обновленные значения правого столбца и нижней строки блока нужно переправить правому и нижнему процессорам решетки соответственно. Данные действия обеспечат снятие блокировки процессоров второй диагонали процессорной решетки (ситуация слева на рис. 13) и т.д.

Анализ эффективности организации волновых вычислений в системах с распределенной памятью (см. табл. 5) показывает значительное снижение полезной вычислительной нагрузки для процессоров, которые занимаются обработкой данных только в моменты, когда их блоки попадают во фронт волны вычислений. При этом балансировка (перераспределение) нагрузки является крайне затруднительной, поскольку связана с пересылкой между процессорами блоков данных большого объема. Возможный интересный способ улучшения ситуации состоит в организации *множественной волны вычислений*, в соответствии с которой процессоры после отработки волны текущей итерации расчетов могут приступить к выполнению волны следующей итерации метода сеток. Так, например, процессор 0 (см. рис. 13), передав после обработки своего блока граничные данные и запустив тем самым вычисления на процессорах 1 и 4, оказывается готовым к исполнению следующей итерации метода Гаусса – Зейделя. После обработки блоков первой (процессоры 1 и 4) и второй (процессор 0) волн к вычислениям окажутся готовыми следующие группы процессоров (для первой волны — процессоры 2, 5 и 8, для второй — процессоры 1 и 4). Кроме того, процессор 0 опять окажется готовым к запуску очередной волны обработки данных. После выполнения NB подобных шагов в обработке будет находиться одновременно NB итераций и все процессоры окажутся задействованными. Подобная схема организации расчетов позволяет рассматривать имеющуюся процессорную решетку как *вычислительный конвейер* поэтапного выполнения итераций метода сеток. Остановка конвейера может осуществляться, как и ранее, по максимальной погрешности вычислений (проверку условия остановки следует начинать только при достижении полной загрузки конвейера после запуска NB итераций расчетов). Необходимо отметить также, что получаемое после выполнения условия остановки решение задачи Дирихле будет содержать значения узлов сетки от разных итераций метода и не будет, тем самым, совпадать с решением, получаемым при помощи исходного последовательного алгоритма.

Рис. 13. Организация волны вычислений при блочной схеме разделения данных

Оценка трудоемкости операций передачи данных

Время выполнения коммуникационных операций значительно превышает длительность вычислительных команд. Оценка трудоемкости операций приема-передачи может быть осуществлена с использованием двух основных характеристик сети передачи: *латентности* (*latency*), определяющей время подготовки данных к передаче по сети, и *пропускной способности сети* (*bandwidth*), задающей объем передаваемых по сети за 1 секунду данных, – более полное изложение вопроса содержится в "Оценка коммуникационной трудоемкости параллельных алгоритмов" .

Пропускная способность наиболее распространенной на данный момент сети Fast Ethernet – 100 Мбит/с, для более современной сети Gigabit Ethernet – 1000 Мбит/с. В то же время скорость передачи данных в системах с общей памятью обычно составляет сотни и тысячи миллионов байт в секунду. Тем самым, использование систем с распределенной памятью приводит к снижению скорости передачи данных не менее чем в 100 раз.

Еще хуже дело обстоит с латентностью. Для сети Fast Ethernet эта характеристика имеет значение порядка 150 мкс, для сети Gigabit Ethernet – около 100 мкс. Для современных компьютеров с тактовой частотой свыше 2 ГГц различие в производительности достигает не менее чем 10000 – 100000 раз. При указанных характеристиках вычислительной системы для достижения 90% эффективности в рассматриваемом примере решения задачи Дирихле (т.е. чтобы в ходе расчетов обработка данных занимала не менее 90% времени от общей длительности вычислений и только 10% времени тратилось бы на операции передачи данных) размер блоков вычислительной сетки должен быть не менее $N=7500$ узлов по вертикали и горизонтали (объем вычислений в блоке составляет $5N^2$ операций с плавающей запятой).

Как результат, можно заключить, что эффективность параллельных вычислений при использовании распределенной памяти определяется в основном интенсивностью и видом выполняемых коммуникационных операций при взаимодействии процессоров. Необходимый при этом анализ параллельных методов и программ может быть выполнен значительно быстрее за счет выделения типовых операций передачи данных – см. "Оценка коммуникационной трудоемкости параллельных алгоритмов". Так, например, в рассматриваемом учебном примере решения задачи Дирихле практически все пересылки значений сводятся к стандартным коммуникационным действиям, имеющим адекватную поддержку в стандарте MPI (см. рис. 14):

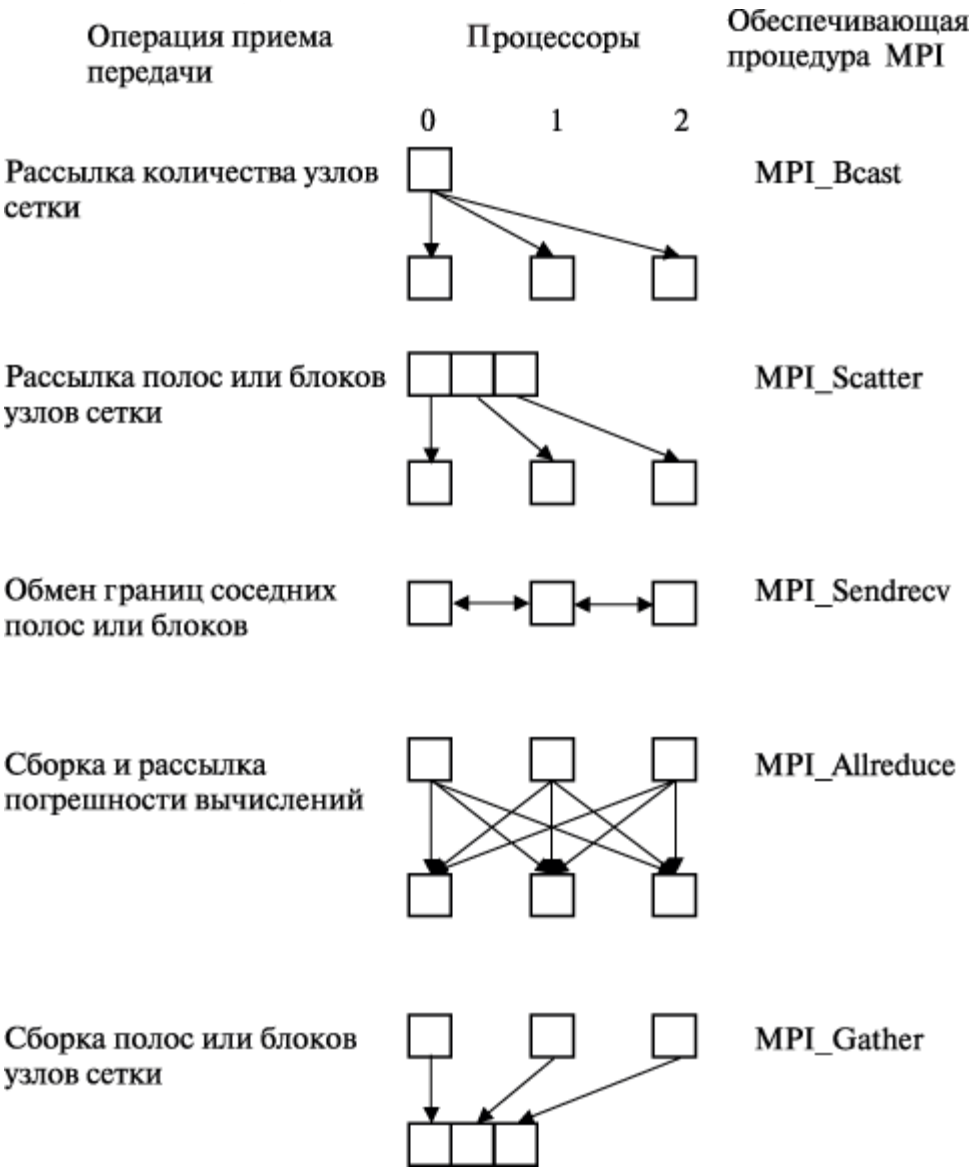

Рис. 14. Операции передачи данных при выполнении метода сеток с распределенной памятью

- рассылка количества узлов сетки всем процессорам – типовая операция передачи данных от одного процессора всем процессорам сети (функция MPI_Bcast);
- рассылка полос или блоков узлов сетки всем процессорам – типовая операция передачи разных данных от одного процессора всем процессорам сети (функция MPI_Scatter);
- обмен граничных строк или столбцов сетки между соседними процессорами – типовая операция передачи данных между соседними процессорами сети (функция MPI_Sendrecv);
- сборка и рассылка погрешности вычислений всем процессорам – типовая операция передачи данных от всех процессоров всем процессорам сети (функция MPI_Allreduce);
- сборка на одном процессоре решения задачи (всех полос или блоков сетки) – типовая операция передачи данных от всех процессоров сети одному процессору (функция MPI_Gather).

В лекции рассматриваются вопросы организации параллельных вычислений для решения задач, в которых при математическом моделировании используются дифференциальные уравнения в частных производных. Для численного решения подобных задач обычно применяется *метод конечных разностей* (*метод сеток*), обладающий высокой вычислительной трудоемкостью. В лекции последовательно разбираются возможные способы распараллеливания сеточных методов на многопроцессорных вычислительных системах с общей и распределенной памятью. При этом большое внимание уделяется проблемам, возникающим при организации параллельных вычислений, анализу причин появления таких проблем и нахождению путей их преодоления. Для наглядной демонстрации излагаемого материала в качестве учебного примера рассматривается проблема численного решения задачи Дирихле для уравнения Пуассона.

Приводится краткое описание сеточных методов на примере решения задачи Дирихле.

Даются возможные способы организации параллельных вычислений при численном решении дифференциальных уравнений в частных производных для вычислительных систем с общей памятью. В основе излагаемого подхода – технология OpenMP, широко применяемая в настоящее время для разработки параллельных программ. В рамках этой технологии параллельный программный код формируется программистом посредством добавления специальных директив или комментариев в существующие последовательные программы. Как результат, программный код является единым для последовательных и параллельных программ, что делает более простым развитие и сопровождение программного обеспечения.

Следует отметить, что принятая в лекции последовательность представления учебного материала может быть рассмотрена как наглядная демонстрация поэтапной методики разработки программного обеспечения. Такой подход позволяет достаточно быстро получать начальные варианты параллельных программ, которые далее могут совершенствоваться для достижения максимально возможной эффективности параллельных вычислений. В ходе изложения учебного материала в лекции проводится последовательное развитие параллельной программы для решения задачи Дирихле; для каждого очередного варианта программы проводится анализ порождаемых программой параллельных вычислений, определяются причины имеющихся потерь эффективности расчетов и обосновываются пути дальнейшего совершенствования вычислений. Подобный порядок расположения материала позволяет последовательно показать ряд типовых проблем параллельного программирования – *излишней синхронизации* (*serialization*), *состязания потоков* (*race condition*), *тупиков* (*deadlock*) и др. Особое внимание уделяется проблеме возможной неоднозначности результатов последовательных и параллельных вычислений. Для достижения однозначности получаемых результатов расчетов в приводимом учебном материале оценивается возможность применения нескольких различных подходов, последовательный анализ которых приводит к определению *методов волновой обработки данных* (*wavefront or hyperplane methods*). На примере реализации волновых схем вычислений дается блочная схема представления данных для эффективного использования быстрой кэш- памяти компьютера. В завершение лекции излагается методика организации очередей заданий для равномерной балансировки вычислительной нагрузки процессоров.

В пункте 3 вопросы организации параллельных вычислений при численном решении дифференциальных уравнений в частных производных рассматриваются применительно к вычислительным системам с распределенной памятью. Прежде всего отмечается, что многие проблемы параллельного программирования (состязание вычислений, тупики, сериализация) являются общими для систем с общей и распределенной памятью. Основное

отличие параллельных вычислений с распределенной памятью состоит в том, что взаимодействие параллельных участков программы на разных процессорах может быть обеспечено только при помощи *передачи сообщений* (*message passing*). При этом эффективность параллельных вычислений во многом определяется равномерностью распределения обрабатываемых данных между процессорами и достигаемой степенью локализации вычислений.

Изложение учебного материала данного раздела лекции начинается с обсуждения общих принципов распределения данных между процессорами, которые применительно к рассматриваемой учебной задаче Дирихле сводятся к *одномерной* (*ленточной*) схеме или *двумерному* (*блочному*) разбиению области расчетов. Последующее рассмотрение вопросов организации параллельных вычислений проводится в основном на примере ленточной схемы; блочный метод разделения данных представлен в более кратком виде. Среди основных тем, выбранных для обсуждения при изложении ленточной схемы: возможные способы выполнения парных и коллективных операций передачи данных между процессорами, особенности реализации волновых схем вычислений в системах с распределенной памятью, возможность совмещения выполняемых вычислений и операций передачи данных. В завершение лекции проводится сравнительная оценка трудоемкости коммуникационных действий и длительности выполнения вычислительных операций.

Контрольные вопросы

1. Как определяется задача Дирихле для уравнения Пуассона?
2. В чем состоят основные положения метода конечных разностей?
3. Какие способы распараллеливания сеточных методов могут быть использованы для многопроцессорных вычислительных систем с общей памятью?
4. В каких ситуациях необходима синхронизация параллельных вычислений?
5. Как характеризуется поведение параллельных участков программы при наличии условий состязания потоков?
6. В чем состоит проблема взаимоблокировки?
7. Какие методы могут быть использованы для достижения однозначности результатов параллельных вычислений для сеточных методов?
8. Как изменяется объем вычислений при применении методов волновой обработки данных?
9. Как повысить эффективность методов волновой обработки данных?
10. Как очередь заданий позволяет улучшить балансировку вычислительной нагрузки процессоров?
11. Какие проблемы приходится решать при организации параллельных вычислений на системах с распределенной памятью?
12. Какие основные схемы распределения данных между процессорами могут быть использованы для сеточных методов?
13. Какие основные операции передачи данных используются в параллельных методах решения задачи Дирихле?
14. Каким образом организация множественной волны вычислений позволяет повысить эффективность волновых вычислений в системах с распределенной памятью?

Задачи и упражнения

- Выполните реализацию первого и второго вариантов параллельного алгоритма Гаусса – Зейделя для систем с общей памятью. Проведите вычислительные эксперименты и сравните время выполнения разработанных программ.
- Выполните реализации параллельного алгоритма Гаусса – Зейделя при волновой схеме организации вычислений и блочном представлении обрабатываемых данных.

Проведите вычислительные эксперименты при разном размере блоков и сравните получаемые характеристики эффективности параллельных вычислений.

- Выполните реализацию очереди заданий для параллельного алгоритма Гаусса – Зейделя. Подготовьте несколько разных правил выделения заданий из очереди и проведите оценку эффективности для каждого использованного правила.