ЛАБОРАТОРНАЯ РАБОТА № 3

ПАРАЛЛЕЛЬНОЕ ВЫЧИСЛЕНИЕ СУММЫ ЧИСЛОВОГО РЯДА СРЕДСТВАМИ MPI

1. Цель работы

Для многопроцессорных вычислительных систем с распределенной памятью на примере задачи параллельного вычисления суммы числового ряда научиться программно реализовывать простейшие параллельные вычислительные алгоритмы и проводить анализ их эффективности.

2. Постановка задачи и описание алгоритма

В настоящее время использование многопроцессорных вычислительных систем (МВС) с распределенной памятью представляется оправданным в двух основных случаях:

1) решаемая задача настолько сложна и ресурсоемка, что существующие персональные компьютеры и рабочие станции в принципе не позволяют ее решить с требуемой точностью;
2) задача решаема на персональных компьютерах и рабочих станциях, но затрачиваемое на ее решение время неприемлемо велико.

Использование МВС требует перехода от использования общепринятых и хорошо апробированных последовательных алгоритмов к алгоритмам параллельным, что в свою очередь ведет к смене всей парадигмы программирования. Одним из негативных факторов такого перехода является, в частности, существенное (иногда на порядок) увеличение времени разработки и отладки параллельной программы по сравнению с соответствующей последовательной программой.

Из любой достаточно сложной задачи всегда можно выделить ряд существенно более простых, большинство которых решается с помощью типовых алгоритмов. Классическим примером такой типовой, с точки зрения программирования, задачи является нахождение суммы элементов числового ряда:

$$S = \sum_{n=1}^{N} a_n = a_1 + a_2 + \ldots + a_N, \quad a_n \in R, n \in N. \qquad (1.1)$$

Очевидно, что решение этой задачи на обычном персональном компьютере не представляет никакой сложности. Поэтому, с учетом сказанного вначале, решение этой задачи на МВС и разработка соответствующего параллельного алгоритма может иметь единственной целью сокращение времени вычисления. Если данная задача должна решаться многократно (например, в рамках какой-либо более общей задачи), то положительный эффект от ее распараллеливания может оказаться весьма значительным.

Для принципиальной возможности решения на МВС задача должна обладать внутренним параллелизмом, то есть допускать декомпозицию на ряд относительно независимых подзадач. Если для решения каждой подзадачи выделить отдельный процессор, то в силу независимости эти задачи могут решаться одновременно, то есть параллельно. Максимальный положительный эффект от такой декомпозиции достигается в том случае, когда все подзадачи требуют одинакового процессорного времени для своего решения.

Задача (1.1) обладает большим внутренним параллелизмом в силу ассоциативности операции сложения. Пусть в нашем распоряжении имеется МВС с количеством процессоров равном P. Будем также полагать, что число суммируемых членов ряда много больше количества процессоров, то есть $P << N$. Тогда на первом этапе задача (1.1) может быть разделена на P независимых подзадач вычисления отдельных частей суммы ряда:

$$S = \sum_{n=1}^{N_1} a_n + \sum_{n=N_1+1}^{N_2} a_n \ldots + \sum_{n=N_{P-1}+1}^{N} a_n = \sum_{p=1}^{P} S_p, \qquad (1.2)$$

где $S_p = \sum_{n=N_{p-1}+1}^{N_p} a_n$, $N_0 = 0$, $N_P = N$. Каждая сумма S_p может вычисляться своим процессором с номером $p = 1, 2, \ldots, P$.

Второй этап решения задачи заключается в вычислении окончательной суммы ряда S как суммы всех найденных на первом этапе частичных сумм S_p. Оба этапа проиллюстрированы на рис. 1.1.

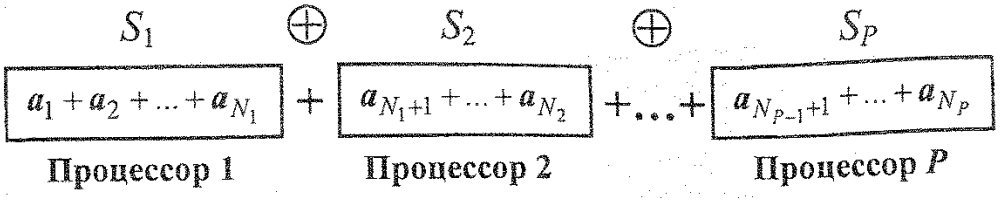

Рис.1.1. Параллельный алгоритм решения задачи нахождения суммы числового ряда

В системах с распределенной памятью, к которым относится подавляющее большинство кластерных систем, каждая частичная сумма S_p находится в локальной памяти своего вычислительного узла. Поэтому прежде чем вычислять сумму S, нужно сначала собрать все S_p на одном узле, для чего каждый узел должен выполнить вспомогательную коммуникационную операцию точечной передачи данных. Здесь необходимо сделать одно замечание: большинство современных средств параллельного программирования (в том числе MPI) предоставляют готовые программные решения в виде специальных функций, позволяющих выполнять суммирование распределенных по узлам (или, в терминах MPI, процессам) данных.

При переходе от последовательной схемы решения задачи к параллельной важнейшим является вопрос эффективности такого перехода. Для ее численной оценки традиционно используются два параметра – ускорение и эффективность.

Определение 1.1. Отношение времени выполнения последовательной программы T_1 ко времени выполнения параллельной программы на p процессорах T_p называется *ускорением* при использовании p процессоров:

$$S_p = \frac{T_1}{T_p}.$$

Отметим, что время T_1 известно не всегда, так как программно может реализовываться только параллельный алгоритм. В этом случае может использоваться оценка ускорения вида

$$S_p^* = \frac{T_1^*}{T_p}, \qquad (1.3)$$

где T_1^* – время выполнения параллельной программы на одном процессоре.

Определение 1.2. Отношение ускорения S_p к количеству процессоров p называется *эффективностью* при использовании p процессоров:

$$E_p = \frac{S_p}{p}.$$

Эффективность показывает среднюю загруженность процессоров при работе параллельной программы. Аналогично ускорению можно использовать оценку эффективности в виде

$$E_p^* = \frac{S_p^*}{p}. \qquad (1.4)$$

Время работы параллельной программы T_p состоит из двух основных частей: собственно времени вычислений T_c и времени передачи данных T_{dt}, то есть

$$T_p = T_c + T_{dt}$$

(время чтения исходных данных, сохранения и вывод результатов не учитываем).

В случае идеального распараллеливания, которое имеет место для задачи вычисления суммы ряда, имеем

$$T_c = \frac{T_1}{p}.$$

Тогда ускорение и эффективность будут тем выше, чем меньше время, затрачиваемое на коммуникацию. Поэтому при разработке или выборе параллельного алгоритма и его программной реализации необходимо уделять большое внимание организации обмена данными. Может случится так, что время передачи данных будет весьма значительным, и параллельная версия программы будет работать на несколько порядков медленнее последовательной. Время одной передачи t_{dt} можно оценить как сумму среднего времени латентности t_l коммутатора и времени собственно передачи данных t'_{dt}:

$$t_{dt} = t_l + t'_{dt}.$$

Время латентности t_l – это время случайной задержки, которую делает коммутатор между двумя последовательными передачами.

Для коммуникационной среды Fast Ethernet 100 Мбит/с время латентности t_l составляет $(100 \div 300) \cdot 10^{-6}$ с.

Общее время, затрачиваемое на коммуникации в параллельной программе, определяется как

$$T_{dt} = t_{dt} \cdot m,$$

где m – количество выполненных программой передач данных.

В решаемой задаче нахождения суммы ряда время счета T_c пропорционально количеству членов ряда N. Поэтому всегда можно выбрать N достаточно большим, чтобы условие $T_c >> T_{dt}$ было выполнено.

3. Особенности программной реализации алгоритма с использованием интерфейса MPI

При использовании MPI передача данных в описанном выше алгоритме может осуществляться несколькими способами, в частности, с использованием базовых функций и функций коллективного взаимодействия (см. прил. Б). Приведем пошаговое описание параллельного алгоритма нахождения суммы числового ряда с использованием обоих способов передачи данных:

а) нулевой процесс запрашивает у пользователя число членов ряда N, с последующей рассылкой его остальным процессам. Рассылка осуществляется с использованием функций парного взаимодействия MPI_Send и MPI_Recv. Перед рассылкой фиксируется время начала выполнения параллельного участка кода программы посредством вызова функции MPI_Wtime;

б) каждый процесс определяет количество членов ряда, которое он суммирует. Для этого вычисляется

$$h = \left[\frac{N}{p} \right] \quad \text{и} \quad m = N \bmod p.$$

Если $m = 0$, то каждый процесс суммирует h членов ряда, номера которых лежат в диапазоне от $N_p = p \cdot h + 1$ до $N_{p+1} = (p+1) \cdot h$, где $p = 0,1,\ldots,P-1$ – номер процесса. Если

$m > 0$, то первые m процессов суммируют по $(h+1)$ членов ряда каждый, а оставшиеся $(P-m)$ процессоров – h членов;

в) каждый процесс вычисляет сумму отведенной ему части ряда и после окончания расчета пересылает ее нулевому процессу с помощью функций MPI_Send и MPI_Recv;

г) нулевой процесс находит искомую сумму S как сумму своей частичной суммы и всех присланных. Повторным вызовом функции MPI_Wtime фиксируется время окончания счета. Продолжительность вычислений определяется как разность моментов окончания и начала счета. Результаты выводятся на экран.

Обмен данными между процессами в этом алгоритме можно условно изобразить схемой (рис. 1.2).

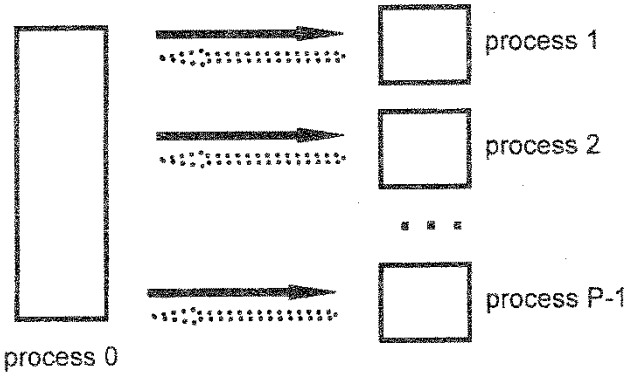

Рис.1.2. Схема обмена данными между процессами при использовании функций парного взаимодействия:
MPI_Send ——→ MPI_Recv – рассылка N
MPI_Recv ·····→ MPI_Send – прием частичных сумм S_p

Алгоритм, использующий функции коллективного взаимодействия, отличается от приведенного выше лишь тем, что на шаге (а) для рассылки N по процессам используется функция MPI_Bcast, а на шаге (в) для сборки частичных сумм S_p на нулевом процессе используется функция MPI_Reduce с оператором

суммирования MPI_SUM (см. прил. Б). Схема обмена данными показана на рис. 1.3.

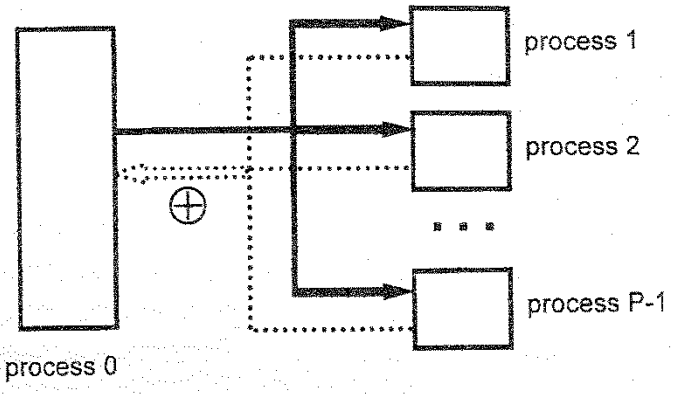

Рис. 1.3. Схема обмена данными между процессами при использовании функций коллективного взаимодействия:
MPI_Bcast ——— MPI_Bcast – рассылка N
MPI_Reduce ····· MPI_Reduce – прием и суммирование частичных сумм S_p

4. Выполнение работы

1. С использованием интерфейса MPI выполнить программную реализацию на языке С параллельного алгоритма нахождения суммы числового ряда, приведенного в п. 3.

Программно реализовать приведенный алгоритм с функциями MPI парного и коллективного взаимодействия. Предусмотреть задание количества членов ряда N пользователем с клавиатуры. В качестве суммируемого ряда взять ряд с лабораторной работы №1.

2. Выполнить предварительную отладку параллельных программ на персональном компьютере в режиме эмуляции параллельного режима исполнения. Для этого необходимо в терминале Linux откомпилировать программу, выполнив команду

```
mpicc -lm -o name file_name.c
```

Если компиляция прошла без ошибок, то в текущей директории появится исполняемый файл `name`, который можно запустить на исполнение командой

```
mpirun -np 4 name
```

Здесь цифра, указанная после ключа, `-np`, показывает количество требуемых процессоров. В режиме эмуляции максимально возможное значение этого параметра зависит от настроек MPI.

3. Для каждой программы последовательно просчитать варианты запуска на 1, 2, 4, 6, 8 процессорах, выбирая N так, чтобы

время работы на одном процессоре T_1 в первом случае составляло порядка 15 с ($N = N_1$), во втором случае – 30-40 с ($N = N_2$). Предварительно предусмотреть и предотвратить возможность переполнения данных.

5. Полученные результаты по продолжительности выполнения параллельного участка кода каждой программы занести в табл. 1.1.

Таблица 1.1

p	T_p, с	
	N_1	N_2
1		
2		
4		
6		
8		

6. Вычислить ускорение и эффективность по формулам (1.3) и (1.4), полученные данные занести в табл. 1.2 для каждой программы.

7. По данным табл. 1.2 построить графики зависимостей ускорения и эффективности от числа процессоров при различных N. Объяснить полученные результаты.

Таблица 1.2

p	S_p^*		E_p^*	
	N_1	N_2	N_1	N_2
1				
2				
4				
6				
8				

Приложение А
(обязательное)

Индивидуальные задания к лабораторным работам

А.1. ***Индивидуальные задания к лабораторной работе №1***

№	Ряд	№	Ряд	№	Ряд
1.	$\sum_{n=1}^{N}\frac{(-1)^n}{(3n-1)^2}$	10.	$\sum_{n=1}^{N}\frac{(-1)^n}{(3n-2)(3n+1)}$	19.	$\sum_{n=1}^{N}\frac{(-1)^{n-1}}{\sqrt{n}}$
2.	$\sum_{n=1}^{N}\frac{\sqrt[3]{n}}{(n+1)\sqrt{n}}$	11.	$\sum_{n=1}^{N}(-1)^{n-1}\frac{2n+1}{n(n+1)}$	20.	$\sum_{n=1}^{N}\frac{(-1)^n}{(2n+1)^3-1}$
3.	$\sum_{n=1}^{N}\frac{(-1)^n}{(n+1)^2-1}$	12.	$\sum_{n=1}^{N}(-1)^n\frac{n+1}{(n+1)\sqrt{n+1}-1}$	21.	$\sum_{n=1}^{N}\frac{(-1)^{n+1}}{2n-\sqrt{n}}$
4.	$\sum_{n=1}^{N}\frac{(-1)^{n-1}}{n^2}$	13.	$\sum_{n=1}^{N}\frac{\sin(2n+1)}{(n+1)^2(n+2)^2}$	22.	$\sum_{n=1}^{N}\frac{(-1)^{n-1}}{\ln(n+1)}$
5.	$\sum_{n=1}^{N}(-1)^n\frac{\ln n}{n}$	14.	$\sum_{n=1}^{N}(-1)^{n-1}\operatorname{tg}\left(\frac{1}{n\sqrt{n}}\right)$	23.	$\sum_{n=2}^{N}\frac{(-1)^n}{n\sqrt[3]{n}-\sqrt{n}}$
6.	$\sum_{n=1}^{N}\frac{\sin(1/n^2)}{(5n-1)^2}$	15.	$\sum_{n=1}^{N}\frac{\sin(5n+1)}{(6n+4)^2(7n-1)^3}$	24.	$\sum_{n=1}^{N}\frac{\sin(2n-1)}{(2n-1)^2}$
7.	$\sum_{n=1}^{N}\frac{(-1)^{n-1}}{(2n-1)^2}$	16.	$\sum_{n=1}^{N}(-1)^n\ln\left(\frac{n^2+1}{n^2}\right)$	25.	$\sum_{n=1}^{N}\frac{(-1)^n}{n-\ln n}$
8.	$\sum_{n=2}^{N}\frac{1}{n\ln^2 n}$	17.	$\sum_{n=1}^{N}\frac{(-1)^n}{n(n+1)(n+2)}$		
9.	$\sum_{n=2}^{N}\frac{(-1)^{n-1}}{n^2-n}$	18.	$\sum_{n=1}^{N}\left(1-\cos\left(\frac{\pi}{n}\right)\right)$		