УПРАВЛЕНИЕ ОПЕРАТИВНОЙ ПАМЯТЬЮ СУПЕРКОМПЬЮТЕРОВ С ПАРАЛЛЕЛЬНОЙ АРХИТЕКТУРОЙ ПРИ РЕШЕНИИ ЗАДАЧ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Каменщиков М.А.

Институт радиотехники и электроники РАН

Аннотация: Описана переносимая реализация библиотеки эмуляции общей памяти на основе библиотеки Message Passing Interface (MPI) версии 1.1. Она обеспечивает работу с большими распределенными массивами, части которых хранятся на разных вычислительных узлах и не требует предварительной установки дополнительного ПО на суперкомпьютере.Рассматриваемую здесь задачу можно считать типовой. Потребность в распределенных массивах возникает во многих програм. Отсутствие в языке понятия распределенного массива является серьезным препятствием для разработки и использования библиотек стандартных параллельных программ. В результате каждая такая библиотека вынуждена вводить свое понятие распределенного массива и реализовывать свой набор операций над ним. Может возникнуть необходимость согласования различных представлений о распределенных массивах (В значительной мере это относится к подходу OpenMP+MPI) [1].

Переносимость программ на суперкомпьютер

К сожалению, возможность работы с большими распределенными массивами, части которых хранятся на различных узлах, при переносе программы, написанной на языке Си, на массивно-параллельный компьютер, не может быть обеспечена автоматически. Для этого используются различные программные средства, в частности библиотеки эмуляции общей памяти. В данной работе описана одна из возможных реализаций такой библиотеки, обладающая базовой функциональностью, но в то же время обеспечивающая возможность пересылать для присваивания и запрашивать куски массива, а не делать это поэлементно с помощью большого количества запросов. Это важно с точки зрения эффективности. Библиотека реализована на языке Си, и в свою очередь использует библиотеку MPI 1.1 для обмена сообщениями, доступную на большинстве суперкомпьютеров, в том числе с общей памятью, что обеспечивает высокую степень переносимости. Библиотека не требует установления дополнительного программного обеспечения на суперкомпьютере. Она проста в изучении и использовании. Кроме того, на ее примере можно освоить некоторые приемы написания MPI-программ.

Библиотеки эмуляции общей памяти

Эмуляция общей памяти позволяет обеспечить работу с большими распределенными массивами данных, которые физически расположены на разных узлах, через единый интерфейс, и с минимальными переделками адаптировать последовательную программу к работе в суперкомпьютерной среде, позволив ей работать большими массивами данных.

На сайте http://parallel.ru в разделе “Интерфейсы программирования” перечислено около десятка библиотек для реализации общей памяти через библиотеки передачи сообщений, в частности через MPI. Но они являются достаточно объемными, их установка на доступном суперкомпьютере может оказаться невозможной. Описанную ниже библиотеку можно использовать как часть вашей MPI программы. Она обладает базовой функциональностью, но является компактной и будет работать везде, где поддерживается MPI. Кроме того, на ее примере можно освоить некоторые приемы написания MPI программ.

Использование библиотеки

При разработке библиотеки использовалась технология открытых систем [5].

Библиотека состоит из двух файлов, gar.c (от global array), и заголовочного файла gar.h.

Пример программы, использующей библиотеку, выглядит следующим образом:

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#include "gar.h"
int size; // число процессов
int me; // номер процесса
   MPI_Status status;
   int i;
int Array1;
char* dbuf;
int main(int argc, char ** argv){
  MPI_Init (&argc, &argv);        /* initialize MPI system */
   MPI_Comm_rank (MPI_COMM_WORLD, &me); /* my place in MPI system*/
   MPI_Comm_size (MPI_COMM_WORLD, &size);  /* size of MPI system */
   Gstart(me, size, 5, 5);
 // главный вычислительны узел
   if (me==0){
     Array1=GCreateArray(100,sizeof(double));
```

```
        dbuf=malloc(10*sizeof(double));
    ((double*)dbuf)[0]=105;
    ((double*)dbuf)[1]=106;((double*)dbuf)[2]=107;
     Gset(Array1, 19,3,(char*)dbuf,sizeof(double));
    ((double*)dbuf)[0]=-1;((double*)dbuf)[1]=-2;((double*)dbuf)[2]=-3;
     Gget(Array1, 19,3,(char*)dbuf,sizeof(double));
   printf("\nВывод          %f          %f          %f          %f",
((double*)dbuf)[0],((double*)dbuf)[1],((double*)dbuf)[2],((double*)dbuf)[3],(
(double*)dbuf)[4]);
     GFinalize(); //Вызываться один раз(только в главно процессе me==0)
    // для выхода сервисных узлов из GStrart()
    } MPI_Finalize();}
```

На суперкомпьютере МВС1000 программа компилируется так: **mpicc main.c, gar.c**

И запускается на выполнение так: **mpirun a.out –np 11**

Запускается на выполнение 11 экземпляров одной и той же MPI программы.

Первый процесс имеет номер 0, последний 10. Каждый узел начинает работу с того, что с помощью стандартных функций MPI MPI_Comm_rank, MPI_Comm_size определяет общее число запущенных экземпляров (переменная size) и свой номер (переменная me). Номера начинаються с 0 до size-1. Далее часть узлов выделяется библиотеке для хранения распределенных массивов. Остальные узлы, логика работы которых определяеться программистом, назовем вычислительными. Для этого каждым экземпляром запущенной программы вызывается функция GStart(), прототип которой следующий: void Gstart(int lme,int lsize,int n_Service,int n_Data);

lme - номер, вызывающего узла, lsize – число узлов на которых запущена программа, n_Service - номер первого сервисного узла, n_Data+1 – общее число сервисных узлов. То есть, последний сервисный узел имеет номер n_Service + nData. Эта функция для узлов, номера которых не лежат в диапазоне от nService до nService+nData, немедленно завершает свою работу. На сервисных же узлах она продолжает выполнение и определяет логику их работы. Функция GFinalize(), вызов которой должен производиться однократно одним из вычислительных узлов, когда планируется завершить использование библиотеки. Она посылает специальное сообщение сервисным узлам, и они заканчивают выполнение функции GSrtart(). После этого на каждом узле программы должна быть вызвана MPI_Finalize() для корректного завершения MPI программы.

Для создания массива на одном из вычислительных узлов необходимо вызвать функцию int GCreateArray(int aSize, int aeSize), передав ей в качестве параметров размер массива и размер его элемента. Функция возвращает дескриптор массива – натуральное число, которое в дальнейшем необходимо использовать для чтения/записи элементов массива. Для этих целей служат функции:

void Gset(int Array, int Abegin, int Asize, char* buf, int aeSize);

void Gget(int Array, int Abegin, int Asize, char* buf, int aeSize);

Первым параметром является дескриптор массива. Затем - номер первого записываемого/читаемого элемента массива, затем адрес буфера типа указатель на char. При работе с массивами с типами элементов отличных от char, необходимо использовать приведение типов. Библиотека работает с байтовыми массивами типа char*. Для работы с элементами других размеров четвертым параметром нужно указать размер элемента массива в байтах. Для байтовых массивов там должна быть единица. В приведенном примере показано как можно организовать хранений массивов с элементами типа double. При этом сначала создается массив со 100 байтовыми кусками. Число байтов в каждом из кусков соответствует размеру типа double. Затем создается буфер dbuf для чтения записи элементов массива. Производится инициализация трех элементов в буфере, затем их запись в кусок массива, начиная с 19 элемента. Затем значение элементов в буфере изменяется и потом происходит считывание только что записанного куска в этот же буфер.

Реализация библиотеки

Первый из сервисных узлов (номер nService) является особым. В нем не хранятся элементы массивов, но через него происходят все запросы, на создание, запись и чтение массивов. При вызове функции создания массива GCreateArray на одном из вычислительных узлов, посылается сообщение на главный сервисный узел. Он в ответ посылает вычислительному узлу дескриптор создаваемого массива, а также шлет сообщение о создании массива на остальные сервисные узлы. На остальных сервисных узлах массив размещается равными кусками. Запрос на запись элементов направляется посредством вызова функции GSet на главный сервисный узел с одного из вычислительных узлов. Главный сервисный узел копирует пересылаемый кусок массива к себе в буфер, и посылает сообщения о записи с соответствующими кусками массива на те сервисные узлы на которых они хранятся.

В ситуации, показанной на рисунке 1, запросы посылаются на 2,3,4 и 5. узлы. Аналогичным образом производится и чтение, главный сервисный узел заказывает необходимые куски массива на узлах хранения в свой буфер и пересылает собранный кусок заказавшему его вычислительному узлу.

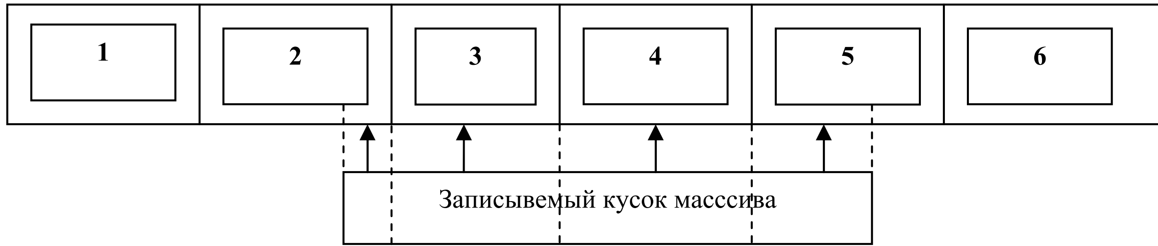

Рисунок 1 - Запись куска распределенного массива через узлы хранения

Отметим одну особенность реализации операции записи куска массива, после ее выполнения главный сервисный узел должен переслать сообщение о том, что операция выполнена на вычислительный узел заказавший запись, после того как он получит такие сообщение от узлов хранения, которые он просил произвести запись. Только после этого должна завершиться функция записи GSet. Если этого не сделать, то может оказаться, что последующие операции чтения опередят окончание операции записи. Так оно и случалось в отлаживаемой автором программе. Ключевой в работе библиотеке является функция void Gstart(int lme,int lsize,int n_Service,int n_Data). Блоксхема ее работа приведена на на рисунке 2

Вход: void Gstart(int lme,int lsize,int n_Service,int n_Data) вызывается каждым улом после инициализации MPI.

да

(Me<=n_Serice) // Не сервисный узел ➔ Выхо дим из функции

Нет

(Me==n_Serice) //Главный сервисный

да

Не

(me>nService) && (me<=nService+nData) // сервисные узлы хранения Ждем запросов msg от главного сервисного узла на создание маассива AllocateTag, Присвайвание его куска SetTag и получение куска GetTag и отвечаем на них. В случае SetTag сообщаем о завершении операции. После получения FinishTag заканчиваем работу

конец

Ждать сообщения msg от одного из вычислительных узлов from=Status.MPI_SOURCE

msg[0]== FinishTag Посылаем такое же сообщение сервисным узлам хранения Завершаем работу

msg[0]==AllocateTag Вычисляем новый дескриптор массива CurArray, шлем запрос на размещение его кусков сервисным узлам i nService+1<i<=nService+nData

msg[0]==SetTag ➔ Ждем следующего сообщения с самим массивов buf от узла from. Отсылаем buf кусками по сервисным узлам хранения

msg[0]==GetTag ➔ Запрашиваем куски массива с сервисных узлов хранения в нужные части массива buf, и отсылаем массив buf заказавшему его узлу from

Рисунок 2 - Реализация библиотеки Блок схема функции Gstart()

Отметим также, что в нашей библиотеке реализовано чтение запись массивов кусками, а не поэлементно (что несколько проще). Это важно с точки зрения производительности. Так как если передавать массивы поэлементно, то резко возрастают накладные расходы на передачу служебной информации в сообщениях и снижается скорость выполнения программы (рисунок 3).

На этом рисунке приведены результаты работы программы, которая пересылает большой массив в 100000 элементов типа double с главного узла на другие десять вспомогательных. По оси Y - отложено

время работы программы в секундах, по оси X - число элементов N в массива в пересылаемом пакете. На графике видно, что при слишком маленьком размере пакета "накладные расходы" на пересылку служебной информации в пакетах чрезмерно велики, что обуславливает чрезмерно большое время работы программы. Это необходимо учитывать при разработке программ вычислительного эксперимента. Гораздо выгоднее организовать программу таким образом, чтобы информация между процессами передавалась достаточно большими пакетами. Отметим, что при некоторых размерах пакетов (эти значения свои для МВС-1000 и кластера ИРЭ-РАН) зафиксировано существенное замедление работы программы и эти результаты устойчивы – повторяются от запуску к запуску. Объяснить эти выбросы пока затруднительно.

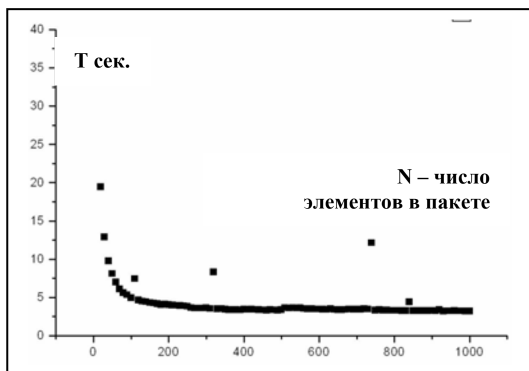

Рисунок 3. График зависимости времени работы программы от размера пакета

Литература

1. В.А.Крюков Разработка параллельных программ для вычислительных кластеров и сетей // журнал "Информационные технологии и вычислительные системы",1-2, 2003
2. Параллельные вычисления Воеводин В.В. Воеводин Вл. В - БХВ-Петербург" 2002
3. www.mpi-forum.org – портал посвященный MPI
4. С.Немнюгин, Ольга Стесик Параллельное программирование для многопроцессорных вычислительных систем - "БХВ-Петербург" 2002
5. А. А. Букатов, В. Н. Дацюк, А. И. Жегуло Программирование многопроцессорных вычислительных систем - Ростов-на-Дону 2003