Lab 6 Parallel Programming with MPI

Speedup

1 Mục tiêu

- SV viết chương trình tuần tự và song song cho ứng dụng nhân vector với ma trận, nhân ma trận với ma trận
- SV thử nghiệm và đánh giá thời gian thực hiện của giải thuật tuần tự và song song. Nhận xét về kết quả và vẽ đồ thị minh họa.

2 Nội dung

Khái niệm về speedup liên quan đến một câu hỏi thường gặp khi phát triển chương trình song song trên hệ thống nhiều bộ xử lý (multiprocessor) là chương trình song song thực thi nhanh hơn chương trình tuần tự như thế nào?. Để thực hiện so sánh, giải pháp được đề ra là so sánh thời gian thực hiện của giải thuật tuần tự tốt nhất trên hệ thống đơn xử lý (single-processor) với giải thuật song song trên hệ thống multiprocessor.

2.1 Speedup

Hệ số speedup (S_p) là một đo đạc của hiệu suất liên quan và được định nghĩa như sau:

$$S_p = \frac{t_s}{t_p}$$

Trong đó:

t_s : Thời gian thực thi sử dụng hệ thống một bộ xử lý (giải thuật tuần tự tốt nhất)

t_p : Thời gian thực thi sử dụng hệ thống p bộ xử lý

☺ Lưu ý giá trị của S_p có thể lớn hơn p hay không ?

2.2 Efficiency

Trong một số trường hợp người phát triển ứng dụng song song cần đánh giá thời gian các bộ xử lý được sử dụng cho quá trình tính toán, đại lượng đo đạc này được gọi là độ hiệu quả (efficiency - E). E được định nghĩa như sau:

$$E = \frac{t_s}{t_p * p}$$

Hoặc:

$$E = \frac{S(p)}{p} * 100\%$$

Vd: Độ hiệu quả E = 50% có nghĩa là các bộ xử lý được sử dụng một nửa thời gian cho quá trình tính toán trên tổng thời gian thực thi của chương trình.

2.3 Chương trình minh họa

2.3.1 Chương trình nhân ma trận và vector:

```
/*   SV su dung chuong trinh hoan chinh da thuc hanh trong bai lab 5, kết
 *   hợp đo thời gian thực thi bằng hàm: MPI_Wtime()
 */

#include <mpi.h>
#include <stdio.h>

int main(int argc, char ** argv){
    int rank,size;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);

    if(rank == 0)
      master_code();
    else
      slave_code();

    MPI_Finalize();
    return 0;
}

int master(int procs){
    long matrixA[N][N], vectorC[N];
    long i,j,dotp, sender, row, numsent=0;
    MPI_Status status;

      /* Initialize data */
    for(i=0; i < N; i++)
      for(j=0; j < N; j++)
          matrixA[i][j] = 1;

    /* distribute data to slave */
    for(i=1; i < minFunc(procs, N); i++)
    {
      MPI_Send(&matrixA[i-1][0], N, MPI_LONG, i, i, MPI_COMM_WORLD );
      numsent++;
    }

    /* receive result and distribute data */
```

```
    for(i=0; i < N; i++)
    {
      MPI_Recv(&dotp, 1, MPI_LONG, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
      /* SV xac dinh process gui ket qua ve va gui tiep du lieu cho no ??? */
      sender = status.MPI_SOURCE;
      row    = status.MPI_TAG - 1;
      vectorC[row] = dotp;

      if(numsent < N) {
          MPI_Send(&matrixA[numsent][0], N, MPI_LONG, sender, numsent+1, MPI_COMM_WORLD);
          numsent++;
        }
      else {
            /* SV gui thong diep thong bao ket thuc cong viec */
          MPI_Send(MPI_BOTTOM, 0, MPI_LONG, sender, 0, MPI_COMM_WORLD);
      }
    }

    /* In ket qua de xac dinh tinh dung dan cua chuong trinh */
    for(i = 0; i < 10; i++)
      fprintf(stdout,"%ld ",vectorC[i]);
    return 0;
}
```

/* SV tìm hiểu mã nguồn chương trình và hoàn tất hàm slave */

```
int slave(){


    /* Cong viec cua slave */

      -  Nhận dữ liệu từ master
      -  Nhân vector dữ liệu vừa nhận với vector của nó
      -  Gửi kết quả trả về
      -  Đợi nhận thêm dữ liệu
      -  Nếu không còn dữ liệu thì kết thúc


    /* Kết thúc */


    return 0;

}
```

☺ SV thực hiện đo đạc với số lượng processor tăng dần, ghi nhận số liệu và vẽ đồ thị minh họa cho kết quả đo đạc speedup !

2.3.2 Chương trình nhân hai ma trận theo mô hình master/slave:

```
/******************************************************************************
* FILE: mpi_mm.c
* DESCRIPTION:
*   MPI Matrix Multiply - C Version
*   In this code, the master task distributes a matrix multiply
*   operation to numtasks-1 worker tasks.
*   NOTE:  C and Fortran versions of this code differ because of the way
*   arrays are stored/passed.  C arrays are row-major order but Fortran
*   arrays are column-major order.
* AUTHOR: Blaise Barney. Adapted from Ros Leibensperger, Cornell Theory
*   Center. Converted to MPI: George L. Gusciora, MHPCC (1/95)
* LAST REVISED: 04/13/05
******************************************************************************/
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define NRA 1000                 /* number of rows in matrix A */
#define NCA 1000                 /* number of columns in matrix A */
#define NCB 1000                 /* number of columns in matrix B */
#define MASTER 0                 /* taskid of first task */
#define FROM_MASTER 1            /* setting a message type */
#define FROM_WORKER 2            /* setting a message type */

double      a[NRA][NCA],           /* matrix A to be multiplied */
            b[NCA][NCB],           /* matrix B to be multiplied */
            c[NRA][NCB];           /* result matrix C */

int main (int argc, char *argv[])
{

int   numtasks,              /* number of tasks in partition */
      taskid,                /* a task identifier */
      numworkers,            /* number of worker tasks */
      source,                /* task id of message source */
      dest,                  /* task id of message destination */
      mtype,                 /* message type */
      rows,                  /* rows of matrix A sent to each worker */
      averow, extra, offset, /* used to determine rows sent to each worker */
      i, j, k, rc;           /* misc */
double  start,end;
MPI_Status status;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
if (numtasks < 2 ) {
  printf("Need at least two MPI tasks. Quitting...\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  exit(1);
  }
numworkers = numtasks-1;
```

```
/**************************** master task ************************************/
   if (taskid == MASTER)
   {
      printf("mpi_mm has started with %d tasks.\n",numtasks);
      printf("Initializing arrays...\n");
      for (i=0; i<NRA; i++)
         for (j=0; j<NCA; j++)
            a[i][j]= i+j;
      for (i=0; i<NCA; i++)
         for (j=0; j<NCB; j++)
            b[i][j]= i*j;

      start = MPI_Wtime();
      /* Send matrix data to the worker tasks */
      averow = NRA/numworkers;
      extra  = NRA%numworkers;
      offset = 0;
      mtype = FROM_MASTER;
      for (dest=1; dest<=numworkers; dest++)
      {
         rows = (dest <= extra) ? averow+1 : averow;
         printf("Sending %d rows to task %d offset=%d\n",rows,dest,offset);
         MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
         MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
         MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);
         MPI_Send(&b, NCA*NCB, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);
         offset = offset + rows;
      }

      /* Receive results from worker tasks */
      mtype = FROM_WORKER;
      for (i=1; i<=numworkers; i++)
      {
         source = i;
         MPI_Recv(&offset, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
         MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
         MPI_Recv(&c[offset][0], rows*NCB, MPI_DOUBLE, source, mtype,
                  MPI_COMM_WORLD, &status);
         printf("Received results from task %d\n",source);
      }

      end = MPI_Wtime();
      /* Print results */
      printf("Computing time = %lf \n",(end - start));
      printf("*****************************************************\n");
      printf("Result Matrix:\n");
      for (i=0; i<10; i++)
      {
         printf("\n");
         for (j=0; j<10; j++)
            printf("%6.2f   ", c[i][j]);
      }
      printf("\n*****************************************************\n");
      printf ("Done.\n");
   }
```

```
/************************** worker task ***********************************/
   if (taskid > MASTER)
   {
      mtype = FROM_MASTER;
      MPI_Recv(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
      MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
      MPI_Recv(&a, rows*NCA, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);
      MPI_Recv(&b, NCA*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD, &status);

      for (k=0; k<NCB; k++)
         for (i=0; i<rows; i++)
         {
            c[i][k] = 0.0;
            for (j=0; j<NCA; j++)
               c[i][k] = c[i][k] + a[i][j] * b[j][k];
         }
      mtype = FROM_WORKER;
      MPI_Send(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
      MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD);
   }
   MPI_Finalize();
}
```

☺ SV hãy phát triển chương trình nhân vector với ma trận trong phần 2.3.1 thành chương trình nhân hai ma trận. Nhận xét về kết quả và speedup của cả hai chương trình.

2.2.2 Chương trình nhân hai ma trận theo mô hình workpool:

```
#include <mpi.h>
#include <stdio.h>

int master(int procs){
    long matrixA[N][N], vectorC[N];
    long i,j,dotp, sender, row, numsent=0;
    MPI_Status status;

      /* Initialize data */
    for(i=0; i < N; i++)
      for(j=0; j < N; j++)
          matrixA[i][j] = 1;

    /* distribute data to slave */
    for(i=1; i < minFunc(procs, N); i++)
    {
      MPI_Send(&matrixA[i-1][0], N, MPI_LONG, i, i, MPI_COMM_WORLD );
      numsent++;
    }

    /* receive result and distribute data */
    for(i=0; i < N; i++)
    {
      MPI_Recv(&dotp, 1, MPI_LONG, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
```

```
      /* SV xac dinh process gui ket qua ve va gui tiep du lieu cho no ??? */
      sender = status.MPI_SOURCE;
      row    = status.MPI_TAG - 1;
      vectorC[row] = dotp;

      if(numsent < N) {
          MPI_Send(&matrixA[numsent][0], N, MPI_LONG, sender, numsent+1, MPI_COMM_WORLD);
          numsent++;
        }
      else {
            /* SV gui thong diep thong bao ket thuc cong viec */
          MPI_Send(MPI_BOTTOM, 0, MPI_LONG, sender, 0, MPI_COMM_WORLD);
      }
    }

    /* In ket qua de xac dinh tinh dung dan cua chuong trinh */
    for(i = 0; i < 10; i++)
      fprintf(stdout,"%ld ",vectorC[i]);
    return 0;
}
```

/* SV tìm hiểu mã nguồn chương trình và hoàn tất hàm slave */

```
int slave(){


    /* Cong viec cua slave */

      -  Nhận dữ liệu từ master
      -  Nhân vector dữ liệu vừa nhận với matran của nó
      -  Gửi kết quả trả về
      -  Đợi nhận thêm dữ liệu
      -  Nếu không còn dữ liệu thì kết thúc


    /* Kết thúc */


    return 0;

}
```

☺ SV phát triển bài trên thành bài toán nhân hai ma trận theo mô hình workpool !

3 Bài tập

SV kết hợp đo đạc thời gian và tính speedup cho mỗi chương trình sau:

3.1 Viết chương trình tính số π theo mô hình master/slave

3.2 Viết chương trình nhân hai ma trận theo mô hình workpool

3.3 SV tìm hiểu về hình Mandelbrot Set, viết chương trình MPI minh họa.

3.4 Viết chương trình song song giải hệ phương trình tuyến tính theo phương pháp lặp Jacobi.

3.5 Viết chương trình song song giải hệ phương trình tuyến tính theo phương pháp lặp Gauss-Seidel.

3.6 SV tìm hiểu về bài toán N-body, viết chương trình MPI minh họa.