LAPORAN PRAKTIKUM ALGORITMA PARALEL

APLIKASI VAGRANT

Laporan Praktikum ini digunakan untuk Memenuhi Nilai pada Matakuliah Algoritma Paralel



DISUSUN OLEH:

BELLA SITY NURHASANAH

2014470086

TEKNIK INFORMATIKA

UNIVERSITAS MUHAMMADIYAH JAKARTA

2017

FINAL TASK

1. Start from small number element and number of processor (100x100 matrices, 2 processor)
2. Compare computational time for each iteration.
3. Is the computational time gradually faster or slower? Explain!
4. Mention and explain the communication method used on your implementation!

**Answer**

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <mpi.h>

int \*createMatrix (int nrows, int ncols) {

int \*matrix;

int h, i, j;

if (( matrix = malloc(nrows\*ncols\*sizeof(int))) == NULL) {

printf("Malloc error");

exit(1);

}

for (h=0; h<nrows\*ncols; h++) {

matrix[h] = h+1;

}

return matrix;

}

void printArray (int \*row, int nElements) {

int i;

for (i=0; i<nElements; i++) {

printf("%d ", row[i]);

}

printf("\n");

}

int main (int argc, char \*\*argv) {

if (MPI\_Init(&argc, &argv) != MPI\_SUCCESS) {

perror("Error initializing MPI");

exit(1);

}

int p, id;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &p); // Get number of processes

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id); // Get own ID

int \*matrix;

if (id == 0) {

matrix = createMatrix(p, p); // Master process creates matrix

printf("Initial matrix:\n");

printArray(matrix, p\*p);

}

int \*procRow = malloc(sizeof(int) \* p); // received row will contain p integers

if (procRow == NULL) {

perror("Error in malloc 3");

exit(1);

}

if (MPI\_Scatter(matrix, p, MPI\_INT, // send one row, which contains p integers

procRow, p, MPI\_INT, // receive one row, which contains p integers

0, MPI\_COMM\_WORLD) != MPI\_SUCCESS) {

perror("Scatter error");

exit(1);

}

printf("Process %d received elements: ", id);

printArray(procRow, p);

MPI\_Finalize();

return 0;

}

**Result:**

1. Start from small number element and number of processor (100x100 matrices, 2 processor)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tabel 1. Indication of time (ms) per processor per matrix | | | | |
|  |  |  |  |  |
|  | 40 x 40 | 60 x 60 | 80 x 80 | 100 x 100 |
| 2 processors | 88,733 | 826,578 | 1280,438 | 5515,629 |
| 4 processors | 70,086 | 315,925 | 379,082 | 3591,148 |

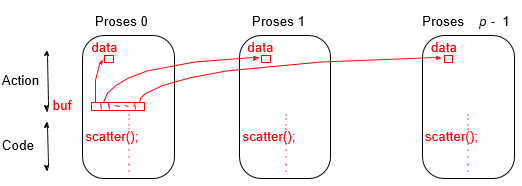
1. Compare computational time for each iteration
2. Is the computational time gradually faster or slower? Explain!

Keterkaitan antara jumlah prosesor dan jumlah matrix (sebagai beban pekerjaan) adalah berbanding lurus terhadap waktu penyelesaian. Semakin besar jumlah prosesor, semakin cepat suatu data selesai dikerjakan. Ketika jumlah prosesor yang digunakan sedikit, sedangkan beban pekerjaan banyak, maka akan mengakibatkan waktu penyelesaian pekerjaan akan semakin lama.

Terlihat pada Tabel 1 yang menggambarkan antara jumlah prosesor, jumlah matrix yang akan dikerjakan (sebagai beban pekerjaan), dan waktu penyelesaiannya. Ketika jumlah prosesor hanya 2, memiliki beban matriks 100x100 maka waktu penyelesainnya hingga 5515,629ms. Waktu penyelesaian akan lebih cepat ketika jumlah prosesor 4, memiliki beban matriks yang sama yaitu 100x100, membutuhkan waktu penyelesaian 3591,148ms.

1. Mention and explain the communication method used on your implementation!

Pada tahapan ini saya menggunakan MPI\_Scatter yang digunakan untuk mengirimkan setiap elemen array dari suatu data (pada root) ke proses yang terpisah. Isi lokasi ke-i dikirim ke proses ke-i.



Gambar 1. Operasi Scatter

Tahapan eksekusi scatter yaitu:

* Identifikasi grup proses termasuk proses rootnya
* Berdasarkan Gambar 1, proses root (proses 0) juga menerima data
* Setiap proses (di dalam grup) mengeksekusi rutin scatter() yang sama
* Hal yang perlu diperhatikan dalam rutin scatter() yaitu pada model SPMD atau Single Proses Multiple Data akan lebih mudah.