Sreepathy Jayanand

CED17I038

High Performance Computing LAB - 5

Analysis of parallelization fraction

**Processor Specifications:**

* Intel i5 - 5350U
* Base clock speed - 1.80GHz
* Max clock speed - 2.90GHz
* Physical Cores - 2
* Threads / Core - 2
* Logical Cores - 4
* L1 cache size (per core) - 64 KB
* L2 cache size (per core) - 256 KB
* L3 cache size - 3 MB
* Main memory - 8GB

Question 1: Matrix Addition

Input size - 1000 x 1000

Segmentation fault - 1000 x 10000

Let the total processes = total,

Matrix size = r x c

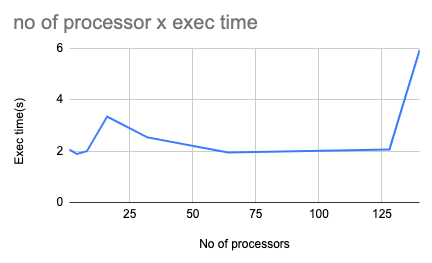
Strategy : Each process has to calculate (total / r) rows, where each row has c elements

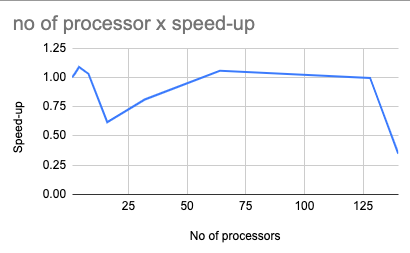
The master declares all the 3 matrices of size r x c and makes each of them row major.

It sends (total/r) \* c elements to each slave process and the slave processes compute the sum of corresponding indices of a and b matrices (now stored as an array). The slave processes send the computed result arrays back to the master process and the master receives it to the corresponding indices of the result array.

Table 1:

| No of processors | Execution time(s) |
| --- | --- |
| 1 | 2.06 |
| 2 | 2.01 |
| 4 | 1.89 |
| 8 | 2.00 |
| 16 | 3.34 |
| 32 | 2.54 |
| 64 | 1.95 |
| 128 | 2.07 |
| 140 | 5.93 |





The most efficient processor size for this program is 4

According to Amdahl’s law :

Speed up = 1/((1-p) + p/N)

Or

Parallelization factor = (N - N/S.U)/(N - 1)

So here N = 4;

Speed Up = 2.06/1.89 = 1.08 for N = 4

PF = (4 - 4/1.08)/3 = .09

Question 2: Matrix multiplication (Column major)

Input size - 1000 x 1000, 1000 x 1000

Segmentation fault - 8000 x 8000

Let the total processes = total

Matrix1 size = r x c

Matrix2 size = c x k

Result = r x k

Work per process = c / total

Strategy: Matrix1 is stored in column major order and matrix2 is stored in row major order.

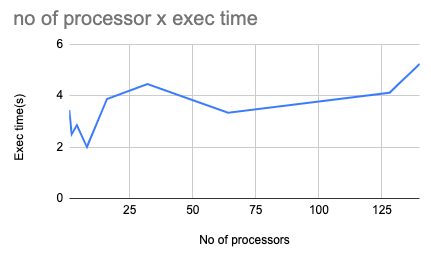
Wpp is the work per process which is the number of columns each process is responsible for.

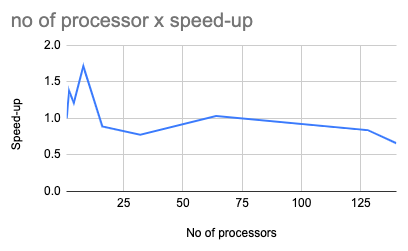
The master process sends the wpp \* r elements from matrix1 and wpp \* k elements from matrix2 to each process which is received in “aprime” and “bprime”. In each process for the column it is responsible for aprime[i] \* bprime[j] for (i from 0 to r) and (j from 0 to k) gets multiplied and added to the (i, j) th position of the “resprime” matrix(local for each process).

The “resprime” matrix is sent back which is of size r x k and received by the master and is added to the final result matrix.

Table 2:

| No of processors | Execution Time(s) |
| --- | --- |
| 1 | 3.44 |
| 2 | 2.51 |
| 4 | 2.85 |
| 8 | 2.01 |
| 16 | 3.87 |
| 32 | 4.45 |
| 64 | 3.34 |
| 128 | 4.11 |
| 140 | 5.23 |





The most efficient processor size for this program is 8

According to Amdahl’s law :

Speed up = 1/((1-p) + p/N)

Or

Parallelization factor = (N - N/S.U)/(N - 1)

So here N = 8;

Speed Up = 3.44/2.01 = 1.72 for N = 8

PF = (8 - 8/1.72)/7 = .47

Question 2: Matrix multiplication (Column major)

Input size - 1000 x 1000, 1000 x 1000

Segmentation fault - 3000 x 3000

Question 3: Matrix multiplication (Block based)

Input size - 1000 x 1000, 1000 x 1000

Segmentation fault - 10000 x 10000

Let the total processes = total

Matrix1 size = r x c

Matrix2 size = c x k

Result = r x k

Block size = 50 x 50 = bs

Strategy: Matrix1 is stored in row major order. Matrix2 is stored in column major order.

The master process assigns memory to matrix1, matrix2 and result. The master sends the complete matrices matrix1 and matrix2 to all the slave processes.

There will be (r / bs) x (k / bs) blocks in the result. Each block is given a number from 0 to (r / bs) x (k / bs) - 1 in a row major order. Block(i, j)’s number would i \* k/bs + j. Let total block = B

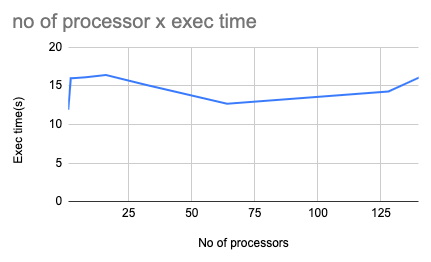
There are a total of “total” processors. We assign computation of block 0 to processor 0, block 1 to processor 1 and so on.. Till block (B - 1). Then again we assign the computation of block B to processor 0 block B+1 to processor 1 and so on. Basically block number m gets assigned to processor m % total.

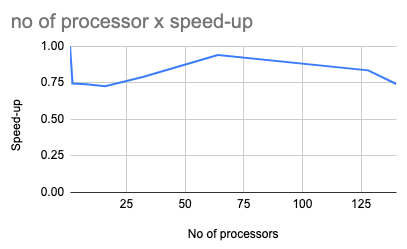
For each block the multiplication is done by the processor and the result is stored in “res\_prime” which is of size r x k and that is sent back to the master.

The master accumulates all the partial answer matrices and adds that to the result matrix.

Table 3:

| No of processors | Execution Time(s) |
| --- | --- |
| 1 | 11.89 |
| 2 | 15.96 |
| 4 | 15.99 |
| 8 | 16.08 |
| 16 | 16.39 |
| 32 | 15.08 |
| 64 | 12.66 |
| 128 | 14.25 |
| 140 | 16.04 |





The most efficient processor size for this program other than 1 is 64

According to Amdahl’s law :

Speed up = 1/((1-p) + p/N)

Or

Parallelization factor = (N - N/S.U)/(N - 1)

So here N = 64;

Speed Up = 11.89/12.66 = 0.93 for N = 64

PF = (64 - 64/0.93)/63 = -0.07