CS546

A20364639 Yung Chi Liu

The original part of Gauss elimination is

/\* Gaussian elimination \*/

for (norm = 0; norm < N - 1; norm++) { <=========loop A

for (row = norm + 1; row < N; row++) { <========loop B

multiplier = A[row][norm] / A[norm][norm];

for (col = norm; col < N; col++) <=========loop C

A[row][col] -= A[norm][col] \* multiplier;

B[row] -= B[norm] \* multiplier;

}

}

In my opinion, loop A can not be parallel because row = norm + 1 had dependency.

Both loop B and loop C could be parallel, only question is which performance is better. So, Fist I use Open MP to paralleled loop A and loop B, than came out paralleled at loop B is faster than corralled at loop C. Than I use P-thread( POSIX thread) to evaluate paralleled loop B with how many thread could get best performance.

**For Open MP :**

for (norm = 0; norm < N - 1; norm++) {

#pragma omp parallel for private ( norm, row, col ) <=== Loop2

for (row = norm + 1; row < N; row++) {

multiplier = A[row][norm] / A[norm][norm];

#pragma omp parallel for private ( norm, row, col ) <=== Loop3

for (col = norm; col < N; col++) {

A[row][col] -= A[norm][col] \* multiplier;

}

B[row] -= B[norm] \* multiplier;

}

}

In this part, I add #pragma omp parallel for private ( norm, row, col ) at location Loop2 to test Loop B; loop3 to test Loop C. And get the result Loop2 is faster than Loop3. And the result show as follow.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | N | 1st | 2nd | 3rd | Avg. | Speed Up |
| Original | 1000 | 1283.09 | 1344.61 | 1331.35 | 1319.683333 |  |
|  | 1500 | 4441.61 | 4645.39 | 4426.74 | 4504.58 |  |
|  | 2000 | 10892.1 | 10116.6 | 10263.6 | 10424.1 |  |
|  |  |  |  |  |  |  |
| OpenMP Loop2 | 1000 | 859.377 | 841.931 | 843.623 | 848.3103333 | 1.555661038 |
|  | 1500 | 2626.96 | 2637.54 | 2626.29 | 2630.263333 | 1.712596584 |
|  | 2000 | 5992.21 | 5868.37 | 5935.48 | 5932.02 | 1.757259753 |
|  |  |  |  |  |  |  |
| OpenMP Loop3 | 1000 | 1938.45 | 1914.68 | 1965.07 | 1939.4 | 0.680459592 |
|  | 1500 | 4779.78 | 4622.88 | 4582.08 | 4661.58 | 0.966320432 |
|  | 2000 | 8785.32 | 8664.99 | 8598.86 | 8683.056667 | 1.200510419 |

( N = Matrix Size, and run 3 times (ms) and Speed Up )

As the result, do parallel at Loop B is better than Loop C, and in test case N = 1000 and 1500 the running time is worst than original one.

In my opinion, I think for the Loop C the Time to create N thread is longer than simple run from 0 to N, that why Loop B is better than Loop C to parallel.

-In JAVIS the file OpenMp\_gauss.c is the program parallel on Loop B.

-And command ./OpenMp\_gauss N 3 to run program( N is Matrix Size)

**For P thread:**

In pthread, I add a function gauss\_2nd\_loop() to do the parallel for loop B.

void\* gauss\_2nd\_loop( void \*slice\_ptr )

{

float multiplier;

int row, col ;

// define valuale for paralled used

int slice = (int)slice\_ptr ;

int start\_row = g\_norm+1 ;

int from = slice\*(N-start\_row) /g\_P +start\_row;

// i\*(N-[norm+1])/P + [norm+1]

int to = (slice+1)\*(N-start\_row) /g\_P +start\_row;

// (i+1)\*(N-[norm+1])/P + [norm+1]

for (row = from; row < to; row++ )

{

multiplier = A[row][g\_norm] / A[g\_norm][g\_norm];

for (col = g\_norm; col < N; col++)

A[row][col] -= A[g\_norm][col] \* multiplier;

B[row] -= B[g\_norm] \* multiplier;

} // for

} // gauss\_2nd\_loop

In function gauss\_2nd\_loop() , use global variable g\_P to define separate Matrix to P part, global variable g\_norm to pass the int. Form gauss() to gauss\_2nd\_loop(). And I extract (g\_norm +1) to a new variable start\_row, so that I could reduce iterations( 2 iterations for each line ) to calculate (int)form and (int) to.

And in function gauss(), create thread and join when finish.

pthread\_t thread[g\_P] ;

int pi ;

for (norm = 0; norm < N - 1; norm++) {

for ( pi = 0; pi < g\_P ; pi++ )

{

g\_norm = norm ; // copy norm for global use ;

pthread\_create(&thread[pi], NULL, gauss\_2nd\_loop,(void\*) pi );

} // for ( parallel loop )

for ( pi =0; pi <g\_P; pi++)

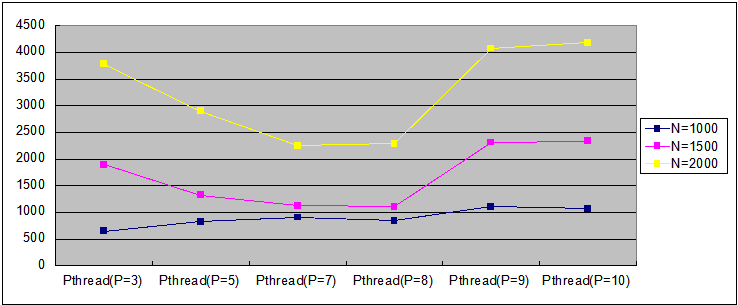
pthread\_join( thread[pi], NULL ) ;

} // for ( norm = 0 to N-1 )

Test :

For the test. g\_P should be set at top of codes manually. I test some and get difference result. P is relation with N( Matrix’s Size), the result show as a pic.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| (ms) | N | 1st | 2nd | 3rd | Avg. | Speed Up |
| Pthread(P=3) | 1000 | 782.643 | 549.022 | 619.011 | 650.2253333 | 2.029578464 |
|  | 1500 | 1936.41 | 1778.85 | 1965.64 | 1893.633333 | 2.378802655 |
|  | 2000 | 3847.26 | 3741.03 | 3742.52 | 3776.936667 | 2.75993508 |
|  |  |  |  |  |  |  |
| Pthread(P=5) | 1000 | 639.371 | 926.191 | 907.367 | 824.3096667 | 1.60095579 |
|  | 1500 | 1320.48 | 1347.67 | 1313.19 | 1327.113333 | 3.394269266 |
|  | 2000 | 2904.77 | 2892.77 | 2898.12 | 2898.553333 | 3.596311263 |
|  |  |  |  |  |  |  |
| Pthread(P=7) | 1000 | 902.772 | 912.144 | 905.216 | 906.7106667 | 1.455462455 |
|  | 1500 | 1188.26 | 1130.76 | 1046.61 | 1121.876667 | 4.015218547 |
|  | 2000 | 2251.75 | 2214.47 | 2273.64 | 2246.62 | 4.639903499 |
|  |  |  |  |  |  |  |
| Pthread(P=8) | 1000 | 890.961 | 783.11 | 848.538 | 840.8696667 | 1.569426732 |
|  | 1500 | 1115.66 | 1163.78 | 1038.18 | 1105.873333 | 4.073323648 |
|  | 2000 | 2181.88 | 2226.39 | 2455.06 | 2287.776667 | 4.556432519 |
|  |  |  |  |  |  |  |
| Pthread(P=9) | 1000 | 1097.54 | 1143.34 | 1077.38 | 1106.086667 | 1.193110245 |
|  | 1500 | 2328.36 | 2327.7 | 2277.62 | 2311.226667 | 1.948999665 |
|  | 2000 | 3976.15 | 4242.94 | 4007.29 | 4075.46 | 2.55777262 |
|  |  |  |  |  |  |  |
| Pthread(P=10) | 1000 | 1083.37 | 1081.6 | 1041.99 | 1068.986667 | 1.234518048 |
|  | 1500 | 2318.83 | 2396.43 | 2288.5 | 2334.586667 | 1.92949787 |
|  | 2000 | 4105.3 | 4403.21 | 4023.86 | 4177.456667 | 2.495322114 |



As the result when N is not big enough P’s effect a little about time; when N is 1500 and 2000, as the chart above, P’s effect could be notice.

And base on Speed Up, when P = 7 and 8, it has good performance(Sp near 4 ). When N = 2000.

And I also try N=2000 with P=100, the average time is around 7000ms.

-In JAVIS the file pthread\_gauss.c is parallel by pthread.

-And command ./pthread\_gauss N 3 to run program( N is Matrix Size)

-Set number of thread at top of codes( int g\_P), default is g\_P is 8.

-Do not Set P bigger than N !!!.

**In conclusion :**

For this project the Maximum of Matrix size is 2000, I could find the best P to fit the best Speed Up. If the N’s Size is bigger, he best value of P could be other number. In this assignment, I notice writing a parallel parallel program is no hard, But how to get the best performances is hard. It dependence on which part could be parallel, and how may thread I had to create for the Size of the data. There are various factors could effect the efficiency.