**High Performance Computing, HOMEWORK 1**

**Solution of Part1 and 2:**

|  |  |  |
| --- | --- | --- |
| **dgemm0** | ***Operations*** | ***Number of times*** |
| *for (i=0; i<n; i++)* |  |  |
| *for (j=0; j<n; j++)* |  |  |
| *for (k=0; k<n; k++)* |  |  |
| *c[i\*n+j] += a[i\*n+k] \* b[k\*n+j];* | *2 operations(\* and +) + 4 memory access(3 read for accessing a,b,and c; and 1 store of c )* |  |

|  |  |  |
| --- | --- | --- |
| **dgemm1** | **Operations** | ***Number of times*** |
| *for (i=0; i<n; i++)* |  |  |
| *for (j=0; j<n; j++) {* |  |  |
| *register double r = c[i\*n+j] ;* | *1 memory read* |  |
| *for (k=0; k<n; k++)* |  |  |
| *r += a[i\*n+k] \* b[k\*n+j];* | *2 memory access(read a and b) and 2 operations (\* and +)* |  |
| *c[i\*n+j] = r;* | *1 memory store* |  |
| *}* |  |  |

As mentioned in question, the 4 floating point operations are executed in 1 clock cycle,

1 floating point operation will be executed in (1/4) clock cycle.

memory access to 1 operand will take 100 clock cycle.

|  |  |
| --- | --- |
| In dgemm0,  The total clock cycles needed to be executed | [2 operations\*(1/4) clock cycle]\*() + [4 memory access \*100 clock cycle] \*(  = [2\* 0.25 + 400] \*  = 400.5  = 400.5 \* for n=1000  =4.005 \*() |
| In dgemm0,  The total time needed for execution | Total clock cycles /CPU clocks per second  =4.005 \*()/2\*  = 2.0025 \*  **=200.25 seconds** |
| In dgemm1,  The total clock cycles needed to be executed | [2 operations\*(1/4) clock cycle]\*() + [2 memory access \*100 clock cycle] \*()  ] + [2 memory access \*100 clock cycle] \*()  =[2\*0.25]\*() + [200]\*[()+()]  =(0.5)\* +(200.2)\* for n=1000  =(200.7)\*  =2.007\* |
| In dgemm1,  The total time needed for execution | Total clock cycles /CPU clocks per second  = 2.007 \*()/2\*  = 1.0035 \*  =**100.35 seconds** |

The extra time wasted in accessing the operands which are not in registers is= (execution time of dgemm0 – execution time of dgemm1)= 200.25sec- 100.35 sec= **99.9 sec**

**Comparision of three algorithms dgemm0, dgemm1 and dgemm2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **dgemm0 (in ns)** | **dgemm1 (in ns)** | **dgemm2 (in ns)** | **maximum difference between all matrix elements of output matrix of dgemm0,dgemm1 and dgemm2** |
| 64 | 7136851 | 4900337 | 2216863 | 0 |
| 128 | 38424124 | 21888070 | 9324527 | 0 |
| 256 | 265270653 | 176893768 | 77361494 | 0 |
| 512 | 3022628976 | 2168284311 | 946234307 | 0 |
| 1024 | 27069849947 | 18489645925 | 8572812562 | 0 |
| 2048 | 659159459407 | 448515857179 | 176450107953 | 0 |

The GFLOPS are calculated using the formula:

Gflops=

=

|  |  |  |
| --- | --- | --- |
| ALGORITHMS | NUMBER OF OPERATIONS | GFLOP FORMULA |
| Dgemmo | 2 |  |
| Dgemm1 | 2 |  |
| Dgemm2 | ()[8 “+” and 8 “\*” operations in inner loop]  =2 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **GFLOPS** | **dgemm0** | | **dgemm1** | | **dgemm2** | |
|  | **(in ns)** | **GFLOPS** | **(in ns)** | **GFLOPS** | **(in ns)** | **GFLOPS** |
| 64 | 7136851 | 0.0734620913 | 4900337 | 0.1069901927 | 2216863 | 0.2364999551 |
| 128 | 38424124 | 0.1091580903 | 21888070 | 0.1916251181 | 9324527 | 0.4498141299 |
| 256 | 265270653 | 0.1264913084 | 176893768 | 0.1896869085 | 77361494 | 0.4337355739 |
| 512 | 3022628976 | 0.0888086027 | 2168284311 | 0.123008570 | 946234307 | 0.2836881457 |
| 1024 | 27069849947 | 0.0793311988 | 18489645925 | 0.1161452013 | 8572812562 | 0.2504993119 |
| 2048 | 659159459407 | 0.0260 | 448515857179 | 0.0383038167 | 176450107953 | 0.0973638916 |

Thus, the **maximum GFLOP for the three algorithm is found in dgemm2**

**How run the program**

1. After logging in to tardis server, A program with all the algorithms dgemm0, dgemm1 and dgemm2 were compiled on the head node .

Ex- the program can be seen using vi program\_name, then it is compiled using gcc –lrt hpc1 program\_name.c

1. Then to run the program on the nodes, the job file was created and saved in the format job\_file\_name.sub
2. Job file was submitted by qsub job\_file\_name.sub
3. To check whether the job is running or not we can use “qstat”
4. Output results can be seen using “more jobfilename.sub.o\_node\_number”
5. To see the correctness of algorithm **compare function** in main function was used. The algorithm for verifying the correctness is mentioned below.

double compare(double \*C1, double \*C2, int n)

{

Int i,j;

Int max\_diff=C1[0]-C2[0];

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

If( (C1[i\*n+j]-C2[i\*n+j]) > max\_diff)

max\_diff= ( (C1[i\*n+j]-C2[i\*n+j])

}

}

return max\_diff;

}