**CS 211: High Performance Computing**

Project 1

Performance Optimization via Register Reuse

Due date: 11:59pm, Oct 9 th, 2015

Note: You need to upload a pdf report for the project into the iLearn system. Please also upload all your source codes and makefile as a tar file into iLearn system so that our TA can verify what you achieved

**Part #1**. (50 points) Assume your computer is able to complete 4 double floating-point operations per cycle when operands are in registers and it takes an additional delay of 100 cycles to access any operands that are not in registers. The clock frequency of your computer is 2 Ghz.

1. How long it will take for your computer to finish the following algorithm dgemm0 and dgemm1 respectively for n= 1000?
2. How much time is wasted on accessing operands that are not in registers?
3. Implement the algorithm dgemm0 and dgemm1 and test them on TARDIS with n= 64, 128, 256, 512, 1024, 2048. Measure the time spend in the triple loop for each algorithm. Calculate the performance (in Gflops) of each algorithm.

Performance is often defined as the number of floating-point operations performed per second. A performance of 1 Gflops means 1 billion of floating-point operations per second.

You must use the system default compiler to compile your program.

Your test matrices have to be 64 bit double floating point random numbers.

1. Report the maximum difference of all matrix elements between the two results obtained from the two algorithms. This maximum difference can be used as a way to check the correctness of your implementation.

|  |
| --- |
| ***/\*dgemm0: simple ijk version triple loop algorithm\*/*** |
| ***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];*** |

|  |
| --- |
| ***/\*dgemm1: simple ijk version triple loop algorithm with register reuse\*/*** |
| ***for (i=0; i<n; i++)*** |
| ***for (j=0; j<n; j++) {*** |
| ***register double r = c[i\*n+j] ;*** |
| ***for (k=0; k<n; k++)*** |
| ***r += a[i\*n+k] \* b[k\*n+j];*** |
| ***c[i\*n+j] = r;*** |
| ***}*** |

**Part #2. (50 points)** Let’s use ***dgemm2*** to denote the algorithm in the following ppt slide from our class. Implement ***dgemm2*** and test it on TARDIS with n= 64, 128, 256, 512, 1024, 2048. Measure the time spend in the algorithm. Calculate the performance (in Gflops) of the algorithm. You must use the system default compiler to compile your program. Your test matrices have to be 64 bit double floating point random numbers. Do not forget to check the correctness of your computation results.

**Part #3 (Optional bonus part: 5 bonus points).** Explore other possible ways to further improve the register reuse and compare your performance with ***dgemm0, dgemm1, and dgemm2***.

**Solution of Part1:**

|  |  |  |
| --- | --- | --- |
| 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 )*** | ***n^3*** |

|  |  |  |
| --- | --- | --- |
| 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*** | ***n^2*** |
| ***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 +)*** | ***n^3*** |
| ***c[i\*n+j] = r;*** | ***1 memory store*** | ***n^2*** |
| ***}*** |  |  |

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]\*(n^3) + [2 memory access \*100 clock cycle] \*(n^3)  ] + [2 memory access \*100 clock cycle] \*(n^2)  =[2\*0.25]\*(n^3) + [200]\*[(n^3)+(n^2)]  =(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 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **GFLOPS** | **dgemm0** | | **dgemm1** | | **dgemm2** | |
|  | **(in ns)** | **GFLOPS** | **(in ns)** | **GFLOPS** | **(in ns)** | **GFLOPS** |
| 64 | 7136851 |  | 4900337 |  | 2216863 |  |
| 128 | 38424124 |  | 21888070 |  | 9324527 |  |
| 256 | 265270653 |  | 176893768 |  | 77361494 |  |
| 512 | 3022628976 |  | 2168284311 |  | 946234307 |  |
| 1024 | 27069849947 |  | 18489645925 |  | 8572812562 |  |
| 2048 | 659159459407 |  | 448515857179 |  | 176450107953 |  |

**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 programname, then it is compiled using gcc –lrt hpc1 programname.c

1. Then to run the program on the nodes, the job file was created and saved in the format jobfilename.sub
2. Job file was submitted by qsub jobfilename.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.onodenumber”
5. To see the correctness of algorithm **compare function** in main function was used. The algorithm for verifying the correctnesss 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])

}

}

if (fabs(a[i] - b[i]) > diff) diff = fabs(a[i] - b[i]);

return diff;

}