|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Work -load** | **Concurrency** | **CPUBench Measured Ops/Sec (GigaOPS)** | **MatrixBench Measured Ops/Sec (GigaOPS)** | **HPL Measured Ops/Sec (GigaOPS)** | **Theoretical Ops/Sec (GigaOPS)** | **CPUBench Efficiency** | **MatrixBench Efficiency (%)** | **HPL Efficiency (%)** |
| **SP** | 1 | **26.852673 Gflops** | **0.229702 GFLOPS** | **2.805 GFOPS** | **146.56 GFlops** | **26.852673\*100=18.33%**  **----------------------**  **146.46** | **0.22\*100 =22 22/146.56=0.15%** | **2.805\*100 = 1.91%**  **146.56** |
| **SP** | 2 | **25.219150 Gflops** | **0.223601 GFLOPS** | **5.44 GFLOPS** | **146.56 GFlops** | **25.219150\*100=17.21%**  **----------------------**  **146.46** | **0.22\*100 = 22/146.56=0.15%** | **5.44\*100 = 3.71%**  **146.56** |
| **SP** | 4 | **18.560569 Gflops** | **0.212103 GFLOPS** | **10.60GFLOPS** | **146.56 GFlops** | **18.560569\*100=12.67%**  **----------------------**  **146.46** | **0.21\*100= 21**  **21/100 = 0.14%** | **10.60\*100 = 1060**  **1060/146.56 = 10%** |
| **SP** | 8 | **11.073043 Gflops** | **0.119171 GFLOPS** | **9.24 GFLOPS** | **146.56 GFlops** | **11.073043\*100=7.56%**  **----------------------**  **146.46** | **0.11\*100 = 11**  **11/100 = 0.11** | **9.24\*100 = 6.30%**  **146.56** |
| **DP** | 1 | **25.407668 Gflops** | **0.229702 GFLOPS** | **2.307GFLOPS** | **73.28 GFlops** | **25.407668\*100=34.67%**  **----------------------**  **73.28** | **0.22\*100 =22 22/146.56=0.15%** | **2.307\*100 =230.7**  **230.7/73.28 = 3.14%** |
| **DP** | 2 | **25.619958 Gflops** | **0.223601 GFLOPS** | **2.68 GFLOPS** | **73.28 GFlops** | **25.619958\*100=34.96%**  **----------------------**  **73.28** | **0.22\*100 = 22/146.56=0.15%** | **2.68\*100 =268**  **268/73.28= 3.65%** |
| **DP** | 4 | **19.671067 Gflops** | **0.212103 GFLOPS** | **8.41 GFLOPS** | **73.28 GFlops** | **19.671067\*100=26.84%**  **----------------------**  **73.28** | **0.21\*100= 21**  **21/100 = 0.14%** | **8.41\*100 =11.47 %**  **------------**  **73.28** |
| **DP** | **8** | **11.03 1430 Gflops** | **0.119171 GFLOPS** | **8.99 GFLOPS** | **73.28 GFlops** | **11.031430\*100=15.05%**  **----------------------**  **73.28** | **0.11\*100 = 11**  **11/100 = 0.11** | **8.99\*100 =12.26 %**  **------------**  **73.28** |

The table and proof of generating the table has been given in doc hw3.

Design CPUBench.c

The program takes input from user namely number of threads to use and which kind of mathematics to follow , that is single precision(SP) and double precision (DP).

Once this is gathered through argv array , a call to function ThreadCall() is made. This function is responsible for generating the Pthreads (Threads) and calculating the entire time

taken to execute 1000000000000 arithmetic operations, eventually reporting average

FLOPS completed in this function in Gflops. This function is also responsible to

distribute the threads into different cores of the machine.

ThreadCall() function calls ThreadFunction() which is the workhorse of this program.

This function does the heavy lifting of doing a trillion calculation or 100 calculation divided

Into 1000000000000/100 loops.

**Improvement**: Different thread distribution into different core does not work properly

All the time , meaning more than one thread can run into single cpu core , which defeats

The purpose of the code. These routines or may be some other routines can be investigated further to see if there is any way to make the threads stick into one core for entire execution cycle.

Design Matmul4.c

Matmul also follows the same design principle as discussed above from the perspective

Of responsibility distribution among the functions.

The only difference is that in the above case the function which does the actual work is called ThreadFunction(), but in this case it is called Matmul().

In Matmul() function threads has been distributed along the rows of the matrix, no distribution of thread has been done along the column , meaning if there are 1000 rows,

And 10 threads then 1 st thread will do the work for 1-100 2nd thread from 100-200 and so on. Once all the calculations are done , that is matrix multiplication is completed, then

We calculate time spend in doing that and GFLOPS. For calculating GFLOPS we assume that

Matrix has O(n3) operations , which may be less than the actual number of FLOPS.

**Trade-off**: SP and DP has not been coded separately as is the case for CPUBench.c

Random number generation routines in c are not capable enough to restrict floating

Point numbers up-to some predetermined precision. So execution time or FLOPS

cannot be separately checked for SP and DP. Right now all the numbers lies between

-1 to +1 but they look like 0.000001 or -0.00001 or as many value as float or double

Supports after decimal point.

**Improvements**: we can try to use fork-join method of matrix multiplication for better parallelism .

C++ can be used for doing SP and DP separately .