Name: Rohit Pardasani
Program: M.Tech
Department: CDS
Registration Number: 06-02-01-10-51-17-1-14491

Assignment – 1 (Introduction to Scalable Systems DS-221)

Objective: To design best program (with shortest execution time) for multiplying 1024 x 1024 double matrices on any machine that you normally use

Machine Hardware Configurations:

Processor	Intel Core i3-2330M CPU @2.20 GHz, 64 Bit
L1 data cache	32 KB (8 way set associative) Block Size (or Line Size) = 64 Bytes Number of Sets = 64 Blocks per set = 8
L1 instruction cache	32 KB (8 way set associative)
L2 cache	256 KB (8 way set associative)
L3 cache	3 MB (12 way set associative)
Number of Processor Cores	2
Number of Processor Threads	4
RAM	3 GB

Software:

Operating System	Ubuntu 16.04 LTS
Compiler	gcc 5.4.0

How random matrices were generated? :

```
void generateMatrix(double *mat, int rows, int cols) {
	for(int i=0; i<rows; i++) {
		for(int j=0; j<cols; j++) {
			*(mat + i*cols + j) = randomgen();
		}
	}
}
double randomgen() {
	double randval;
	int x = 5;
	randval = 2.0*((double)rand()/RAND_MAX)-1.0;
	return randval;
}
```

How program was timed? (only multiplication operation was timed):

```
start = clock();
        for(int i=0; i<M; i++) {
                for(int j=0; j<M; j++) {
                        for(int k=0; k<N; k++) {
                                C[i][j] = C[i][j] + A[i][k]*B[k][j];
                        }
                }
        }
        end = clock();
        cpu_time_used = ((double)(end - start))/CLOCKS_PER_SEC;
        printf("\n Seconds to execute : %lf \n",cpu_time_used)
```

Results:

Program Code Name	Loop Interchange	Blocking (32x32)	Vectorization (Compiler Option)	Loop Unrolling (Compiler Option)	Time 1	Time 2	Time 3	Average Time (Seconds)
one.c	X	X	X	X	18.13	18.04	18.13	18.10
two.c	X	X	✔	X	4.96	4.96	4.96	4.96
three.c	X	✔	X	X	8.92	8.88	8.88	8.89
four.c	✔	X	X	X	6.53	6.53	6.53	6.53
five.c	✔	X	✔	X	0.89	0.89	0.89	0.89
six.c	X	✔	✔	X	2.36	2.36	2.36	2.36
seven.c	✔	✔	X	X	6.64	6.64	6.64	6.64
eight.c	✔	✔	✔	X	1.02	1.02	1.02	1.02
nine.c	✔	X	✔	✔	**0.80**	**0.79**	**0.80**	**0.80**

**Note: Execution time rounded to second decimal place*

Compilation and run command of few programs:

gcc one.c –o one.out
taskset –c 0 ./one.out
**The purpose of taskset is to ensure that CPU with id 0 is only used during the execution. Because we want execution on single core*

gcc –O2 –ftree-vectorize –fopt-info-vec two.c o two.out
taskset –c 0 ./two.out
**To ensure vectorization*

gcc –O2 –funroll-loops –ftree-vectorize –fopt-info-vec nine.c –o nine.out
taskset –c 0 ./nine.out
**To ensure vectorization and loop unrolling*

Effect of Blocking:

Blocking was tried all many possible combinations of $2^n x 2^n$ i.e. 2x2, 4x4, 8x8, 16x16, 32x32, 64x64. It was found that 32x32 blocking gives best result.

```
for(int i=0; i < M; i++) {
        for(int jj=0; jj < M; jj+=BLOCK_SIZE) {
            for(int kk=0; kk < N; kk+=BLOCK_SIZE) {
                for(int j=jj; j < jj+BLOCK_SIZE; j++) {
                    for(int k=kk; k < kk+BLOCK_SIZE; k++) {
                        C[i][j] = C[i][j] + A[i][k]*B[k][j];
                    }
                }
            }
        }
    }
```

Effect of Loop Interchange:

Loop interchange optimizes code to a great extent.

Writing this:

```
for(int i=0; i<M; i++) {
        for(int k=0; k<N; k++) {
            for(int j=0; j<M; j++) {
                C[i][j] = C[i][j] + A[i][k]*B[k][j];
            }
        }
    }
```

Instead of this:

```
for(int i=0; i<M; i++) {
        for(int j=0; j<M; j++) {
            for(int k=0; k<N; k++) {
                C[i][j] = C[i][j] + A[i][k]*B[k][j];
            }
        }
    }
```

Conclusion:

nine.c when compiled with vectorization and loop unroll options gives the best time for matrix multiplication.

Attachments:

All 9 code files (.c) attached along with all nine output files (.out) .