Assignment 1

Arpit Prasad
COL718 - Architecture for High Performance Computers

August 14, 2025

1 Overview

In this assignment we perform analysis of different variants of computer hardware benchmarking on the matrix multiplication algorithm

2 Code for MM

For this assignment we assume the following code for MM. The fundamental principle for the given code is:

1. **Pattern of Memory Storage**: Main Memory stores information linearly however when we access (i, j) we might encounter misses due to two calls of (i, j) being in completly different blocks

Code for Matrix Multiplication

```
#include <x86intrin.h>
#include <stdio.h>

#define UNROLL (4)
#define BLOCKSIZE 32
#define N 150

// void do_block (int n, int si, int sj, int sk, double *A, double *B,
    double *C) {
//     for ( int i = si; i < si+BLOCKSIZE; i+=UNROLL*4 ) {
//         for ( int j = sj; j < sj+BLOCKSIZE; j++ ) {
//             /*using 4 bytes for double storage*/
//             __m256d c[4];
//             for ( int x = 0; x < UNROLL; x++ ){
//                 /*accessing the in the linear array*/
//                 c[x] = _mm256_load_pd(C+i+x*4+j*n);
//             }
//             for( int k = sk; k < sk+BLOCKSIZE; k++ ) {
//                 /*accessing b in the linear array*/
//                 __m256d b = _mm256_broadcast_sd(B+k+j*n);
//                 for (int x = 0; x < UNROLL; x++){
```

```
//                      /* performing matrix multiplication */
//                      c[x] = _mm256_add_pd(c[x], /* c[x]+=A[i][k]*b */
    _mm256_mul_pd(_mm256_load_pd(A+n*k+x*4+i), b));
//                  }
//              }

//              for ( int x = 0; x < UNROLL; x++ ){
//                  _mm256_store_pd(C+i+x*4+j*n, c[x]);
//              }
//          }
//      }
// }

// void dgemm (int n, double* A, double* B, double* C) {
//     for ( int sj = 0; sj < n; sj += BLOCKSIZE )
//         for ( int si = 0; si < n; si += BLOCKSIZE )
//             for ( int sk = 0; sk < n; sk += BLOCKSIZE )
//                 do_block(n, si, sj, sk, A, B, C);
// }

int main() {
    double A[N][N], B[N][N], C[N][N];
    double *a_ptr=&A[0][0], *b_ptr=&B[0][0], *c_ptr=&C[0][0];
    printf("performing matrix multiplication");


    for ( int sj = 0; sj < N; sj += BLOCKSIZE )
        for ( int si = 0; si < N; si += BLOCKSIZE )
            for ( int sk = 0; sk < N; sk += BLOCKSIZE )
                for ( int i = si; i < si+BLOCKSIZE; i+=UNROLL*4 ) {
                    for ( int j = sj; j < sj+BLOCKSIZE; j++ ) {
                        /*using 4 bytes for double storage*/
                        __m256d c[4];
                        for ( int x = 0; x < UNROLL; x++ ){
                            /*accessing the in the linear array*/
                            c[x] = _mm256_load_pd(C+i+x*4+j*N);
                        }
                        for( int k = sk; k < sk+BLOCKSIZE; k++ ) {
                            /*accessing b in the linear array*/
                            __m256d b = _mm256_broadcast_sd(B+k+j*N);
                            for (int x = 0; x < UNROLL; x++){
                                /* performing matrix multiplication */
                                c[x] = _mm256_add_pd(c[x], /* c[x]+=A[i][k
    ]*b */_mm256_mul_pd(_mm256_load_pd(A+N*k+x*4+i), b));
                            }
                        }

                        for ( int x = 0; x < UNROLL; x++ ){
                            _mm256_store_pd(C+i+x*4+j*N, c[x]);
                        }
                    }
                }
}
```

3 Performance Metrics

The following is the overview of the performance metrics that are used to analyse the performance of a code with respect to a Model

1. Raw speed of the processor
2. Latency
3. Period

4 Performance vs CPU Models

Here we vary the CPU Models and check performance in the metrics specified above