Matrix Multiplication Optimisation Assignment

Sourabh Aggarwal

111601025@smail.iitpkd.ac.in

---

1. Optimisations

1. I started with basic blocking. Following are the results which I obtained which was expected as `LEVEL1_DCACHE_LINESIZE = 64`, this can be obtained by executing from terminal:

   ```
   >> getconf LEVEL1_DCACHE_LINESIZE
   ```

   And since size of double is 8, optimality is obtained at block size of 8 (= 64/8).

   ```
   ------------ Results -------------
   Iteration: 1
   1, 6445.611429
   Iteration: 2
   2, 2272.318962
   Iteration: 3
   4, 897.756441
   Iteration: 4
   8, 725.708793
   Iteration: 5
   16, 731.699237
   Iteration: 6
   32, 906.470085
   Iteration: 7
   64, 999.502892
   Iteration: 8
   128, 975.474192
   Iteration: 9
   256, 1356.794308
   Iteration: 10
   512, 2576.451370
   Iteration: 11
   1024, 3304.505859
   Iteration: 12
   2048, 3478.298679
   Iteration: 13
   ```

```
4096, 3459.928579
---------- END ----------
```

2. After this I tried loop re-ordering and found the best ordering to be `i-k-j` (both for outer 3 loops and inner 3 loops).
3. Thirdly I tried to save repeated pointer arithmetic by storing the common calculation in a variable, significant time was saved doing this.
4. Lastly loop unrolling helped a lot. I started with unrolling the inner most loop which improved the time, then the second inner most which again improved the time but further unrolling increased the time taken.

2. Final Code

Following code (final implementation) took around 170 seconds.

```
#include <stdio.h>
#include <assert.h>
#include <time.h>

#define N 5120

double arr1[N][N], arr2[N][N], res[N][N];

int main() {
  // int B = CLS / sizeof(double); // CLS (Cache line size) = 64 in this
  machine and can be obtained using "getconf LEVEL1_DCACHE_LINESIZE".
  int B = 8;
  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++) {
      arr1[i][j] = i + j + 0.5;
      arr2[i][j] = i + j + 0.5;
    }
  double *rres, *rmul1, *rmul2, tmp;
  int i, j, k, ii;
  double st = (double)clock();
  for (i = 0; i < N; i += B)
    for (k = 0; k < N; k += B)
      for (j = 0; j < N; j += B)
        for (ii = 0, rres = &res[i][j], rmul1 = &arr1[i][k]; ii < B; ii++,
        rres += N, rmul1 += N) {
          tmp = rmul1[0];
          rmul2 = &arr2[k][j];
          rres[0] += tmp * rmul2[0];
```

```
rres[1] += tmp * rmul2[1];
rres[2] += tmp * rmul2[2];
rres[3] += tmp * rmul2[3];
rres[4] += tmp * rmul2[4];
rres[5] += tmp * rmul2[5];
rres[6] += tmp * rmul2[6];
rres[7] += tmp * rmul2[7];
rmul2 += N;
tmp = rmul1[1];
rres[0] += tmp * rmul2[0];
rres[1] += tmp * rmul2[1];
rres[2] += tmp * rmul2[2];
rres[3] += tmp * rmul2[3];
rres[4] += tmp * rmul2[4];
rres[5] += tmp * rmul2[5];
rres[6] += tmp * rmul2[6];
rres[7] += tmp * rmul2[7];
rmul2 += N;
tmp = rmul1[2];
rres[0] += tmp * rmul2[0];
rres[1] += tmp * rmul2[1];
rres[2] += tmp * rmul2[2];
rres[3] += tmp * rmul2[3];
rres[4] += tmp * rmul2[4];
rres[5] += tmp * rmul2[5];
rres[6] += tmp * rmul2[6];
rres[7] += tmp * rmul2[7];
rmul2 += N;
tmp = rmul1[3];
rres[0] += tmp * rmul2[0];
rres[1] += tmp * rmul2[1];
rres[2] += tmp * rmul2[2];
rres[3] += tmp * rmul2[3];
rres[4] += tmp * rmul2[4];
rres[5] += tmp * rmul2[5];
rres[6] += tmp * rmul2[6];
rres[7] += tmp * rmul2[7];
rmul2 += N;
tmp = rmul1[4];
rres[0] += tmp * rmul2[0];
rres[1] += tmp * rmul2[1];
rres[2] += tmp * rmul2[2];
rres[3] += tmp * rmul2[3];
```

```
            rres[4] += tmp * rmul2[4];
            rres[5] += tmp * rmul2[5];
            rres[6] += tmp * rmul2[6];
            rres[7] += tmp * rmul2[7];
            rmul2 += N;
            tmp = rmul1[5];
            rres[0] += tmp * rmul2[0];
            rres[1] += tmp * rmul2[1];
            rres[2] += tmp * rmul2[2];
            rres[3] += tmp * rmul2[3];
            rres[4] += tmp * rmul2[4];
            rres[5] += tmp * rmul2[5];
            rres[6] += tmp * rmul2[6];
            rres[7] += tmp * rmul2[7];
            rmul2 += N;
            tmp = rmul1[6];
            rres[0] += tmp * rmul2[0];
            rres[1] += tmp * rmul2[1];
            rres[2] += tmp * rmul2[2];
            rres[3] += tmp * rmul2[3];
            rres[4] += tmp * rmul2[4];
            rres[5] += tmp * rmul2[5];
            rres[6] += tmp * rmul2[6];
            rres[7] += tmp * rmul2[7];
            rmul2 += N;
            tmp = rmul1[7];
            rres[0] += tmp * rmul2[0];
            rres[1] += tmp * rmul2[1];
            rres[2] += tmp * rmul2[2];
            rres[3] += tmp * rmul2[3];
            rres[4] += tmp * rmul2[4];
            rres[5] += tmp * rmul2[5];
            rres[6] += tmp * rmul2[6];
            rres[7] += tmp * rmul2[7];
        }
  double en = (double)clock();
  printf("time taken: %f", (en - st)/CLOCKS_PER_SEC);
  return 0;
}
```

3. Reference

Besides basic material, I referred to this text.