Homework 9

Elizabeth Hunt

December 11, 2023

1 Question One

With a `matrix_dimension` set to 700, I consistently see about a 3x improvement in performance on my 10-thread machine. The serial implementation gives an average `0.189s` total runtime, while the below parallel implementation runs in about `0.066s` after the cpu cache has filled on the first run.

```
#include <math.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define matrix_dimension 700

int n = matrix_dimension;
float sum;

int main() {
  float A[n][n];
  float x0[n];
  float b[n];
  float x1[n];
  float res[n];

  srand((unsigned int)(time(NULL)));

  // not worth parallellization - rand() is not thread-safe
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      A[i][j] = ((float)rand() / (float)(RAND_MAX) * 5.0);
    }
    x0[i] = ((float)rand() / (float)(RAND_MAX) * 5.0);
  }

#pragma omp parallel for private(sum)
  for (int i = 0; i < n; i++) {
    sum = 0.0;
    for (int j = 0; j < n; j++) {
      sum += fabs(A[i][j]);
    }
```

```
    A[i][i] += sum;
  }

#pragma omp parallel for private(sum)
  for (int i = 0; i < n; i++) {
    sum = 0.0;
    for (int j = 0; j < n; j++) {
      sum += A[i][j];
    }
    b[i] = sum;
  }

  float tol = 0.0001;
  float error = 10.0 * tol;
  int maxiter = 100;
  int iter = 0;

  while (error > tol && iter < maxiter) {
#pragma omp parallel for
    for (int i = 0; i < n; i++) {
      float temp_sum = b[i];
      for (int j = 0; j < n; j++) {
        temp_sum -= A[i][j] * x0[j];
      }
      res[i] = temp_sum;
      x1[i] = x0[i] + res[i] / A[i][i];
    }

    sum = 0.0;
#pragma omp parallel for reduction(+ : sum)
    for (int i = 0; i < n; i++) {
      float val = x1[i] - x0[i];
      sum += val * val;
    }
    error = sqrt(sum);

#pragma omp parallel for
    for (int i = 0; i < n; i++) {
      x0[i] = x1[i];
    }

    iter++;
  }

  for (int i = 0; i < n; i++)
    printf("x[%d] = %6f \t res[%d] = %6f\n", i, x1[i], i, res[i]);

  return 0;
}
```

2 Question Two

I only see lowerings in performance (likely due to overhead) on my machine using OpenMP until `matrix_dimension` becomes quite large, about `300` in testing. At `matrix_dimension=1000`, I see another about 3x improvement in total runtime (including initialization & I/O which was untouched, so, even further improvements could be made) on my 10-thread machine; from around `0.174` seconds to `.052`.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef _OPENMP
#include <omp.h>
#else
#define omp_get_num_threads() 0
#define omp_set_num_threads(int) 0
#define omp_get_thread_num() 0
#endif

#define matrix_dimension 1000

int n = matrix_dimension;
float ynrm;

int main() {
  float A[n][n];
  float v0[n];
  float v1[n];
  float y[n];
  //
  // create a matrix
  //
  // not worth parallellization - rand() is not thread-safe
  srand((unsigned int)(time(NULL)));
  float a = 5.0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      A[i][j] = ((float)rand() / (float)(RAND_MAX)*a);
    }
    v0[i] = ((float)rand() / (float)(RAND_MAX)*a);
  }
  //
  // modify the diagonal entries for diagonal dominance
  // --------------------------------------------------
  //
  for (int i = 0; i < n; i++) {
    float sum = 0.0;
    for (int j = 0; j < n; j++) {
      sum = sum + fabs(A[i][j]);
    }
```

```
    A[i][i] = A[i][i] + sum;
  }
  //
  // generate a vector of ones
  // -------------------------
  //
  for (int j = 0; j < n; j++) {
    v0[j] = 1.0;
  }
  //
  // power iteration test
  // --------------------
  //
  float tol = 0.0000001;
  float error = 10.0 * tol;
  float lam1, lam0;
  int maxiter = 100;
  int iter = 0;

  while (error > tol && iter < maxiter) {
#pragma omp parallel for
    for (int i = 0; i < n; i++) {
      y[i] = 0;
      for (int j = 0; j < n; j++) {
        y[i] = y[i] + A[i][j] * v0[j];
      }
    }

    ynrm = 0.0;
#pragma omp parallel for reduction(+ : ynrm)
    for (int i = 0; i < n; i++) {
      ynrm += y[i] * y[i];
    }
    ynrm = sqrt(ynrm);

#pragma omp parallel for
    for (int i = 0; i < n; i++) {
      v1[i] = y[i] / ynrm;
    }

#pragma omp parallel for
    for (int i = 0; i < n; i++) {
      y[i] = 0.0;
      for (int j = 0; j < n; j++) {
        y[i] += A[i][j] * v1[j];
      }
    }

    lam1 = 0.0;
#pragma omp parallel for reduction(+ : lam1)
    for (int i = 0; i < n; i++) {
```

```
      lam1 += v1[i] * y[i];
    }

    error = fabs(lam1 - lam0);
    lam0 = lam1;

#pragma omp parallel for
    for (int i = 0; i < n; i++) {
      v0[i] = v1[i];
    }

    iter++;
  }

  printf("in %d iterations, eigenvalue = %f\n", iter, lam1);
}
```

3 Question Three

https://static.simponic.xyz/lizfcm.pdf