HPC HW 0 Problem 2

Noah D. Brenowitz

February 9, 2015

The source code for the Gauss-Seidel method with over-relaxation is give in the file `laplace.c`:

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

#define STOP_ITER_RAT 10e-6
#define OMEGA_OVER_RELAXED 1.5


double calc_resid(int N, double* f, double *u){
  int i, j;
  double resid = 0.0;
  double ai;

  double h2 = 1.0/(N+1)/(N+1);

  ai  =  (2 * u[0] - u[1])/h2 -f[0];
  resid += ai*ai;

  ai =  (2 * u[N-1] - u[N-2])/h2 -f[N-1];
  resid += ai*ai;

  for (i = 1; i < N-1; i++) {
    ai = (- u[i+1] + 2 * u[i] - u[i-1])/h2 -f[i];
    resid += ai*ai;
  }

  return sqrt(resid);
}

void gauss_seidel_laplace(int N, double *f, double *u){
  int i;
  double h2 = 1.0/(N+1)/(N+1);

  u[0] =  (h2 * f[0] + u[1])/2.0;

  for (i = 1; i < N-1; i++) {
    u[i] = (h2 * f[i] + u[i-1] + u[i+1])/2.0;
  }

  u[N-1] = (h2* f[N-1] + u[N-2])/2.0;
}

void overrelaxed_gauss_seidel_laplace(int N, double *f, double *u, double* w){
  int i;
  double h2 = 1.0/(N+1)/(N+1);

  for (i = 0; i < N; i++)
    w[i] = u[i];


  u[0] =  (h2 * f[0] + u[1])/2.0;

  for (i = 1; i < N-1; i++) {
    u[i] = (h2 * f[i] + u[i-1] + u[i+1])/2.0;
  }

  u[N-1] = (h2* f[N-1] + u[N-2])/2.0;

  for (i = 0; i < N; i++)
    u[i] = (1-OMEGA_OVER_RELAXED) * w[i] + OMEGA_OVER_RELAXED * u[i];

}

void test_resid(){
  int N = 100;

  double *u, *f;

  // allocate arrays
  u = (double *) malloc(N*sizeof (double));
  f = (double *) malloc(N*sizeof (double));

  // initialize f and u
  int i;
  for (i = 0; i < N; i++) {
    f[i] = 2.0;
    u[i] = 0.0;
  }

  double resid;

  resid = calc_resid(N, f, u);
  printf("Resid is %f", resid);
```

```
}

int main(int argc, char *argv[])
{

  const int run_over_relaxed = 0; // if not 0 run with overrelaxed version of algorithm


  int N = atoi(argv[1]);

  double *u, *f, *w;

  // allocate arrays
  u = (double *) malloc(N*sizeof (double));
  f = (double *) malloc(N*sizeof (double));

  if (run_over_relaxed){
    w = (double *) malloc(N*sizeof (double));
  }

  // initialize f and u
  int i;
  for (i = 0; i < N; i++) {
    f[i] = 1.0;
    u[i] = 0.0;
  }

  // Begin iterations
  double resid_init, resid_cur;
  resid_init = calc_resid(N, f, u);
  resid_cur = resid_init;

  while (resid_cur / resid_init > STOP_ITER_RAT){

    if (run_over_relaxed){
      overrelaxed_gauss_seidel_laplace(N, f, u, w);
    } else {
      gauss_seidel_laplace(N, f, u);
    }

    resid_cur = calc_resid(N, f, u);
    printf("Resid is %f\n", resid_cur );
  }

  // deallocate
  free(f);
  free(u);
  return 0;
}
```

This source is compiled and timed using the following shell script (`run.sh`):

```
CC=gcc
CFLAGS=-lm

${CC} ${CFLAGS} -o laplace laplace.c
${CC} ${CFLAGS} -O0 -o laplace0 laplace.c
${CC} ${CFLAGS} -O3 -o laplace3 laplace.c


N=1000
time ./laplace $N > /dev/null
time ./laplace0 $N >/dev/null
time ./laplace3 $N > /dev/null
```

The output for N=1000 is:

```
$ zsh run.sh
./laplace $N > /dev/null  12.03s user 0.00s system 100% cpu 12.031 total
./laplace0 $N > /dev/null  12.05s user 0.00s system 100% cpu 12.045 total
./laplace3 $N > /dev/null  9.48s user 0.00s system 100% cpu 9.477 total
```

There is around a 30% speed-up for the `-O3` optimized file compared to the non-optimized version. The number of iterations required to reach convergence is

```
./laplace 1000 | wc -l

 1158227
```

For N=100, the output is:

```
zsh run.sh
./laplace $N > /dev/null  0.02s user 0.00s system 94% cpu 0.017 total
./laplace0 $N > /dev/null  0.01s user 0.00s system 98% cpu 0.016 total
./laplace3 $N > /dev/null  0.01s user 0.00s system 92% cpu 0.013 total
```

So the N=100 code runs much more quickly. The number of iterations required for convergence here is

```
./laplace 100 | wc -l

 11796
```