Homework 1

Cong Yu

February 24, 2020

1 Matrix-matrix multiplication

Processors information:

```
Architecture:  x86_64
Processor Name: 6-Core Intel Core i7
Processor Speed: 2.2 GHz
Number of processors: 1
Total number of cores: 6
L2 cache (per core): 256 KB
L3 cache: 9 MB
Hyper-Threading Technology: Enabled
Memory: 16 GB
```

Result with -O0:

Dimension	Time	Gflop/s	GB/s
20	0.002636	0.303452	4.855234
40	0.019848	0.322456	5.159292
60	0.062536	0.345401	5.526419
80	0.138181	0.370529	5.928460
100	0.260713	0.383564	6.137022
120	0.456912	0.378191	6.051062
140	0.713584	0.384538	6.152603
160	1.073854	0.381430	6.102875
180	1.516873	0.384475	6.151603
200	2.082301	0.384190	6.147046
220	2.772397	0.384072	6.145152
240	3.677722	0.375885	6.014157
260	4.555420	0.385826	6.173218
280	5.715221	0.384097	6.145555
300	6.997428	0.385856	6.173697
320	10.052299	0.325975	5.215603
340	10.199504	0.385352	6.165633
360	12.115145	0.385105	6.161676
380	14.331746	0.382870	6.125925
400	17.660181	0.362397	5.798355
420	19.414627	0.381609	6.105747
440	22.405051	0.380200	6.083200
460	25.835211	0.376757	6.028114
480	31.857209	0.347149	5.554385
500	33.760137	0.370259	5.924147
520	38.464080	0.365557	5.848906
540	42.831154	0.367639	5.882223
560	47.890686	0.366702	5.867229
580	53.257241	0.366358	5.861723

Result with -O3

Dimension	Time	Gflop/s	GB/s
20	0.000260	3.076331	49.221304
40	0.002598	2.463683	39.418925
60	0.009035	2.390723	38.251567
80	0.023668	2.163240	34.611839
100	0.042864	2.332949	37.327180
120	0.086523	1.997152	31.954430
140	0.120282	2.281310	36.500958
160	0.184476	2.220346	35.525536
180	0.289121	2.017150	32.274398
200	0.395655	2.021966	32.351454
220	0.512335	2.078327	33.253232
240	0.668782	2.067041	33.072659
260	0.855682	2.054033	32.864532
280	1.062567	2.065941	33.055059
300	1.310035	2.061014	32.976224
320	1.635527	2.003513	32.056206
340	1.920599	2.046444	32.743110
360	2.331088	2.001469	32.023502
380	2.769492	1.981302	31.700830
400	3.189466	2.006606	32.105692
420	3.740989	1.980439	31.687023
440	4.254189	2.002356	32.037696
460	4.858140	2.003565	32.057039
480	5.645376	1.958984	31.343740
500	6.288178	1.987857	31.805713
520	7.220840	1.947253	31.156041
540	7.975797	1.974273	31.588365
560	8.904532	1.972209	31.555348
580	9.880459	1.974726	31.595618

code used

```
double time = t.toc();
double bandwidth = NREPEATS*4*m*n*k*sizeof(double)/1e9/time;
double flops = NREPEATS*m*n*k*2/1e9/time;
printf("%10d %10f %10f %10f\n", p, time, flops, bandwidth);
```

2 Laplace equation in one space dimension

(a) see code in the code attachment

(b) see log in the code attachment

(c)

- Compare the number of iterations needed for the two different methods for different numbers $N = 100$ and $N = 10,000$.

Mahcine Information:

```
Architecture: x86_64
CPU op-mode (s): 32-bit, 64-bit
Byte Order: Little Endian
CPU (s): 1
On-line CPU (s) list: 0
Thread (s) per core: 1
Core (s) per socket: 1
Block: 1
NUMA node: 1
Manufacturer ID: GenuineIntel
CPU series: 6
Model: 79
Model name: Intel (R) Xeon (R) CPU E5-2682 v4 @ 2.50GHz
Step: 1
CPU MHz: 2494.222
BogoMIPS: 4988.44
Hypervisor Vendor: KVM
Virtualization Type: Full
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 40960K
NUMA node 0 CPU: 0
```

compile:

```
gcc main.c -O3 -o out -lm
```

run:

```
./out 100 -1 1 0
./out 100 -1 2 0
./out 10000 -1 1 0
./out 10000 -1 2 0
```

result:

Iteration	$N = 100$	$N = 10000$
Jacobi	28348	277899922
Gauss-Seidel	14175	138942887

- Compare the run times for $N = 10,000$ for 100 iterations using different compiler optimization flags (-O0 and -O3).

Machine Information:

```
Architecture:  x86_64
Processor Name: 6-Core Intel Core i7
Processor Speed: 2.2 GHz
Number of processors: 1
Total number of cores: 6
L2 cache (per core): 256 KB
L3 cache: 9 MB
Hyper-Threading Technology: Enabled
Memory: 16 GB
```

compile:

```
gcc main.c -O0 -o out0 -lm
gcc main.c -O3 -o out3 -lm
```

run:

```
./out0 10000 100 1
./out0 10000 100 2
./out3 10000 100 1
./out3 10000 100 2
```

result:

Time (s)	-O0	-O3
Jacobi	0.0093	0.0054
Gauss-Seidel	0.0148	0.0021

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

int LOG_RESIDUAL = 0;
double INIT_RESIDUAL = 0;

int N = 100;
int maxIter = 100;
double h = 1;
int method = 1;
double H; // 1/h^2
double H_inverse;

double calcRes(const double*u, const double *f) {
        double res = 0, delta = 0;
        delta = f[0] - (2*H*u[0]-1*H*u[1]);
        res += delta*delta;
        delta = f[N-1] - (-1*H*u[N-2]+2*H*u[N-1]);
        res += delta*delta;
        for (int i = 1; i < N-1; i++) {
        delta = f[i] - (-1*H*u[i-1]+2*H*u[i]-1*H*u[i+1]);
        res += delta*delta;
        }
        res = sqrt(res);
        // print
        return res;
}

void jacobi(double *u, const double *f) {
        double res;
        double lastNewVal, curNewVal;
        for (int k = 0; k < maxIter; k++) {
        lastNewVal = (H_inverse + u[1])/2;
        for (int i = 1; i < N-1; i++) {
                // (1+H*u[i-1]+H*u[i+1])/(2*H)
                curNewVal = (H_inverse + u[i-1]+u[i+1])/2;
                u[i-1] = lastNewVal;
                lastNewVal = curNewVal;
        }
        u[N-1] = (H_inverse+u[N-2])/2;
        u[N-2] = lastNewVal;
        res = calcRes(u, f);
        if (LOG_RESIDUAL) printf("iteration %d\tresidual %f\t"
                                "decreased %f\tdecreased factor %f\n",
                                k+1, res, (INIT_RESIDUAL - res),
                                INIT_RESIDUAL/res);
        if (INIT_RESIDUAL/res >= 1e6) {
                break;
        }
        }
}

void gauss(double *u, const double *f) {
        double res;
        for (int k = 0; k < maxIter; k++) {
        u[0] = (H_inverse + u[1])/2;
        for (int i = 1; i < N-1; i++) {
                u[i] = (H_inverse + u[i-1] + u[i+1])/2;
        }
        u[N-1] = (H_inverse + u[N-2])/2;
        res = calcRes(u, f);
        if (LOG_RESIDUAL) printf("iteration %d\tresidual %f\t"
                                "decreased %f\tdecreased factor %f\n",
                                k+1, res, (INIT_RESIDUAL - res),
                                INIT_RESIDUAL/res);
        if (INIT_RESIDUAL/res >= 1e6) {
                break;
        }
        }
```

```
}

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

        // read args from cmd
        /*
        * input is as follow
        * ./program N MaxIteration Method (1 -> Jacobi, 2 -> Gauss) [LOG]
        */
        if (argc == 4 || argc == 5) {
        N = (int) strtol(argv[1], NULL, 10);
        maxIter = (int) strtol(argv[2], NULL, 10);
        if (maxIter == -1) {
                maxIter = INT32_MAX;
        }
        method = (int) strtol(argv[3], NULL, 10);
        if (argc == 5) {
                LOG_RESIDUAL = (int) strtol(argv[4], NULL, 10) == 0? 0 : 1;
        }
        } else {
        printf("usage: ./program N {MaxIteration | -1 (for non stop)}"
                " {1 (for Jacobi) | 2 (for Gauss)} [Log-Residual = 0]\n");
        exit(0);
        }

        h = 1.0/(N+1);
        H = (N+1)*(N+1);
        H_inverse = 1.0 / H;

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

        memset(u, 0.0, N*sizeof(double));
        // init f
        for (int i = 0; i < N; ++i) {
        f[i] = 1;
        }

        INIT_RESIDUAL = calcRes(u, f);
        clock_t s = clock();
        if (method == 1) {
        jacobi(u, f);
        } else {
        gauss(u, f);
        }

        clock_t t = clock();

        printf("%.4lf s\n", (double) ( t-s )/CLOCKS_PER_SEC);

        // free
        free(u);
        free(f);

        return 0;
}
```