CS5375 Computer Systems Organization and Architecture

Lecture 18

Guest Instructors:

Ghazanfar Ali, Ghazanfar.Ali@ttu.edu

Mert Side, Mert.Side@ttu.edu

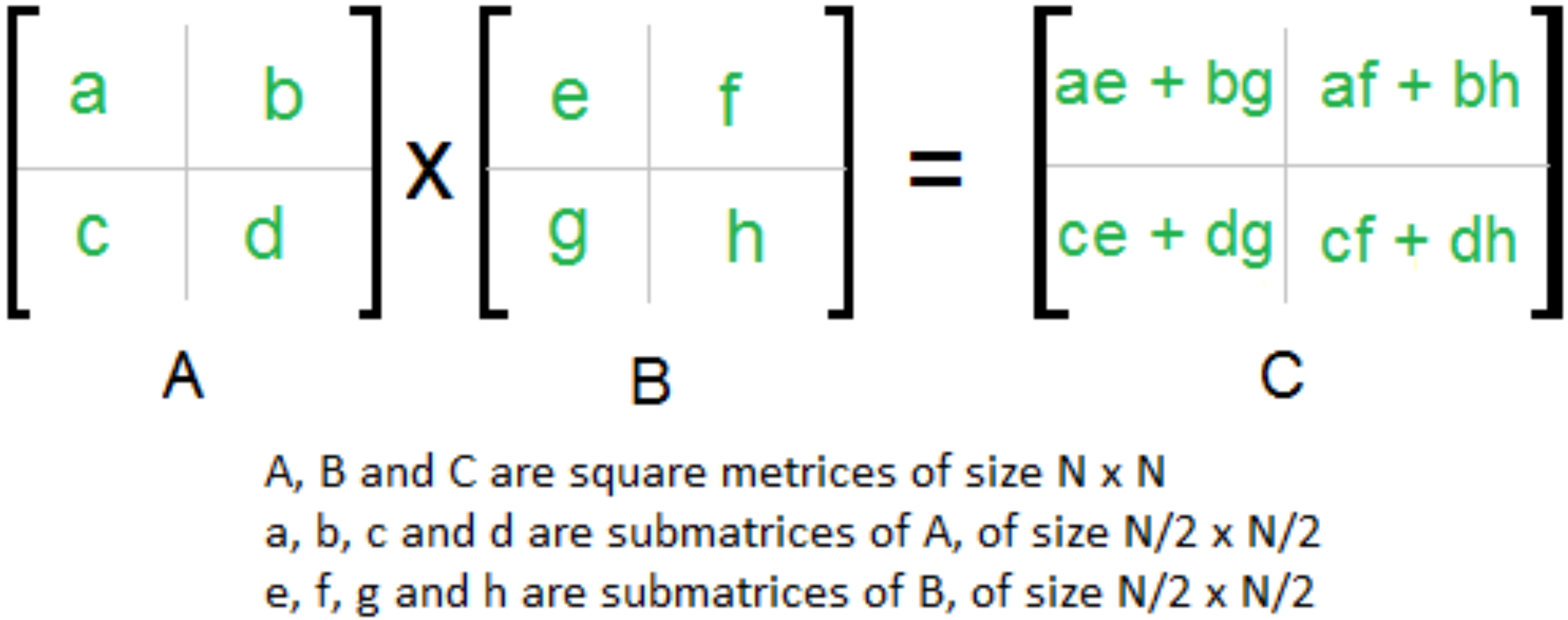
Department of Computer Science

Texas Tech University

Outline

- Programming Project #2 Walkthrough
- Midterm Exam Review

Reminder: Matrix Multiplication



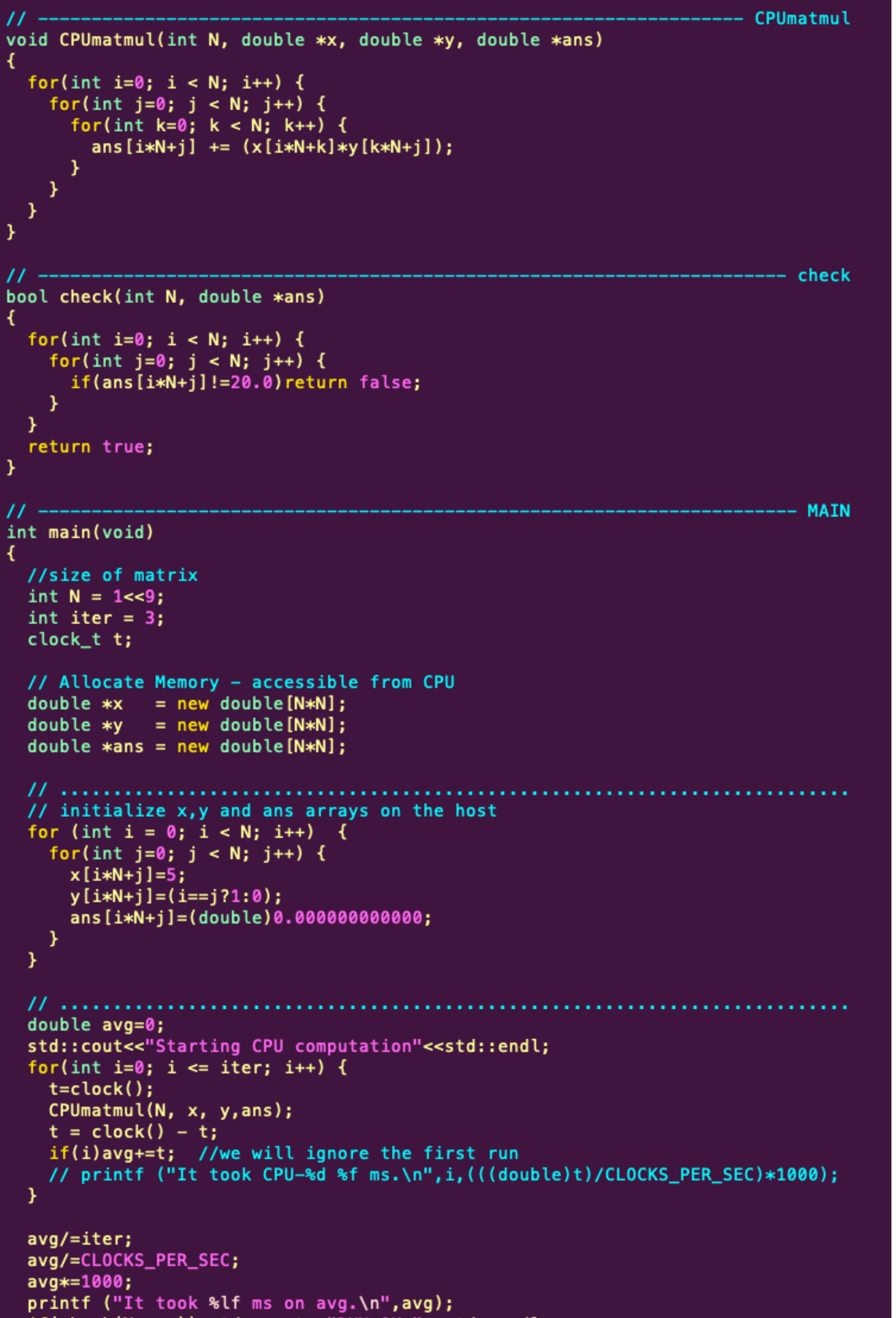
$$\underset{A}{\left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right]} \text{x} \underset{B}{\left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]} = \underset{C}{\left[\begin{array}{c|c} ae + bg & af + bh \\ \hline ce + dg & cf + dh \end{array}\right]}$$

A, B and C are square metrices of size N x N

a, b, c and d are submatrices of A, of size N/2 x N/2

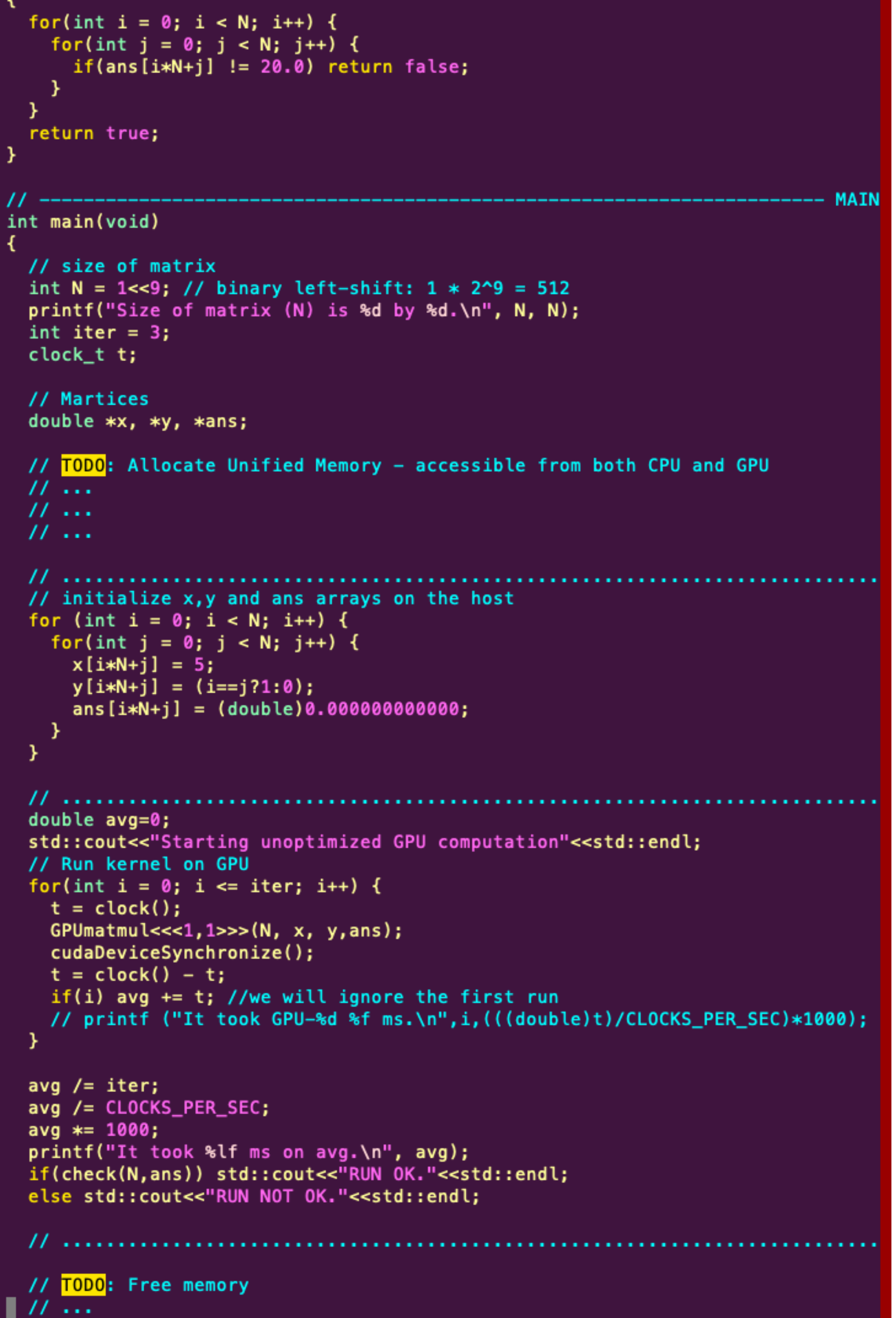
e, f, g and h are submatrices of B, of size N/2 x N/2

Matrix Multiplication on the CPU



```
// ------------------------------------------------------------------ CPUmatmul
void CPUmatmul(int N, double *x, double *y, double *ans)
{
  for(int i=0; i < N; i++) {
    for(int j=0; j < N; j++) {
      for(int k=0; k < N; k++) {
        ans[i*N+j] += (x[i*N+k]*y[k*N+j]);
      }
    }
  }
}

// ---------------------------------------------------------------------- check
bool check(int N, double *ans)
{
  for(int i=0; i < N; i++) {
    for(int j=0; j < N; j++) {
      if(ans[i*N+j]!=20.0)return false;
    }
  }
  return true;
}

// ----------------------------------------------------------------------- MAIN
int main(void)
{
  //size of matrix
  int N = 1<<9;
  int iter = 3;
  clock_t t;

  // Allocate Memory - accessible from CPU
  double *x   = new double[N*N];
  double *y   = new double[N*N];
  double *ans = new double[N*N];

  // .........................................................................
  // initialize x,y and ans arrays on the host
  for (int i = 0; i < N; i++)  {
    for(int j=0; j < N; j++) {
      x[i*N+j]=5;
      y[i*N+j]=(i==j?1:0);
      ans[i*N+j]=(double)0.000000000000;
    }
  }

  // .........................................................................
  double avg=0;
  std::cout<<"Starting CPU computation"<<std::endl;
  for(int i=0; i <= iter; i++) {
    t=clock();
    CPUmatmul(N, x, y,ans);
    t = clock() - t;
    if(i)avg+=t;  //we will ignore the first run
    // printf ("It took CPU-%d %f ms.\n",i,(((double)t)/CLOCKS_PER_SEC)*1000);
  }

  avg/=iter;
  avg/=CLOCKS_PER_SEC;
  avg*=1000;
  printf ("It took %lf ms on avg.\n",avg);
```

- Sequential Matrix Multiplication
  - Here is a code for matrix multiplication using C++.
  - It is the standard O(N^3) procedure.
- Here x, y, and ans are three N^2 size matrices
  - N^2 sized 1D array.
  - We are using 1D arrays as of it were 2D.



```
{
  for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
      if(ans[i*N+j] != 20.0) return false;
    }
  }
  return true;
}

// ---------------------------------------------------------------------- MAIN
int main(void)
{
  // size of matrix
  int N = 1<<9; // binary left-shift: 1 * 2^9 = 512
  printf("Size of matrix (N) is %d by %d.\n", N, N);
  int iter = 3;
  clock_t t;

  // Martices
  double *x, *y, *ans;

  // TODO: Allocate Unified Memory - accessible from both CPU and GPU
  // ...
  // ...
  // ...

  // .........................................................................
  // initialize x,y and ans arrays on the host
  for (int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
      x[i*N+j] = 5;
      y[i*N+j] = (i==j?1:0);
      ans[i*N+j] = (double)0.000000000000;
    }
  }

  // .........................................................................
  double avg=0;
  std::cout<<"Starting unoptimized GPU computation"<<std::endl;
  // Run kernel on GPU
  for(int i = 0; i <= iter; i++) {
    t = clock();
    GPUmatmul<<<1,1>>>(N, x, y,ans);
    cudaDeviceSynchronize();
    t = clock() - t;
    if(i) avg += t; //we will ignore the first run
    // printf ("It took GPU-%d %f ms.\n",i,(((double)t)/CLOCKS_PER_SEC)*1000);
  }

  avg /= iter;
  avg /= CLOCKS_PER_SEC;
  avg *= 1000;
  printf("It took %lf ms on avg.\n", avg);
  if(check(N,ans)) std::cout<<"RUN OK."<<std::endl;
  else std::cout<<"RUN NOT OK."<<std::endl;

  // .........................................................................

  // TODO: Free memory
  // ...
```

Matrix Multiplication on the GPU

- This is the code to run on the GPU.
  - But it only uses one GPU thread.
  - And it is still sequential.
- You are tasked to use a simple stride pattern to make this parallel on the GPU.

Source Code

- Code from the lecture and project:

https://github.com/mertside/CS5375_GPU_Lecture

Readings

- How to CUDA? GPU Accelerated Computing with C and C++:
  - https://developer.nvidia.com/how-to-cuda-c-cpp
- Introduction to CUDA:
  - https://developer.nvidia.com/blog/even-easier-introduction-cuda/
- Unified Memory with CUDA:
  - https://developer.nvidia.com/blog/unified-memory-cuda-beginners/
- How to Optimize Data Transfers in CUDA C/C++:
  - https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/
- An Efficient Matrix Transpose in CUDA using Shared Memory:
  - https://developer.nvidia.com/blog/efficient-matrix-transpose-cuda-cc/