Computational Fluid Dynamics (CFD) using Graphics Processing Units

Aaron F. Shinn

Mechanical Science and Engineering Dept., UIUC

Accelerators for Science and Engineering Applications: GPUs and Multicores

Example CFD problem: Heat Conduction in Plate

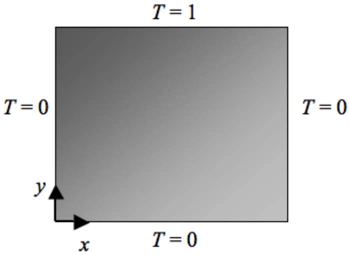


Figure: Physical domain: unit square heated from the top.

Steady-State 2D Heat Conduction (Laplace Equation)

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

Discretization/Solution of Heat Conduction Equation

- Finite-Volume Method (Temperatures are cell-centered)
- Iterative solver: Red-Black Gauss-Seidel with SOR (parallel algorithm)

Gauss-Seidel

$$T_p^{n+1} = \frac{(a_n)(T_n^n) + (a_s)(T_s^n) + (a_e)(T_e^n) + (a_w)(T_w^n)}{a_p}$$

where

$$a_p = a_n + a_s + a_e + a_w$$

and n=north, s=south, e=east, w=west

Successive-Overrelaxation (SOR)

$$T_p^{n+1}(accepted) = \omega T_p^{n+1} + (1-\omega)T_p^n$$

Red-Black Gauss-Seidel

- Color the grid like a checkboard.
- First update red cells from n to $n+1$ (only depends on black cells at n).
- Then update black cells from n to $n+1$ (only depends on the red cells at $n+1$)

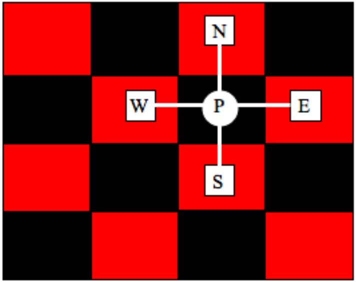


Figure: Red-Black coloring scheme on internal grid cells and stencil.

Developing the CUDA algorithm

- Experimented with various memory models
- Tried *shared* memory with "if" statements to handle B.C.s for each sub-domain → slow
- Tried *global* memory where each thread loads its nearest-neighbors → fast
- Currently using global memory
- Next step: *texture* memory

CUDA algorithm on host

Programmed CPU (host) in C and GPU (device) in CUDA

Pseudo-Code for Laplace solver on host (CPU)

- dynamic memory allocation
- set I.C. and B.C.
- setup coefficients $(a_n, a_s, a_e, a_w, a_p)$
- allocate device memory and copy all variables to device
- setup the execution configuration
- iteration loop: call red kernel, call black kernel each iteration
- copy final results from device back to host

CUDA algorithm on host: `main.cu`

Execution configuration

```
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
dim3 dimGrid(imx / BLOCK_SIZE, jmx / BLOCK_SIZE);
```

Iteration loop

```
for (iter=1; iter <= itmax; iter++) {
   // Launch kernel to update red squares
   red_kernel<<<dimGrid, dimBlock>>>
   (T_old_d,an_d,as_d,ae_d,aw_d,ap_d,imx,jmx);

   // Launch kernel to update black squares
   black_kernel<<<dimGrid, dimBlock>>>
   (T_old_d,an_d,as_d,ae_d,aw_d,ap_d,imx,jmx);
 }
```

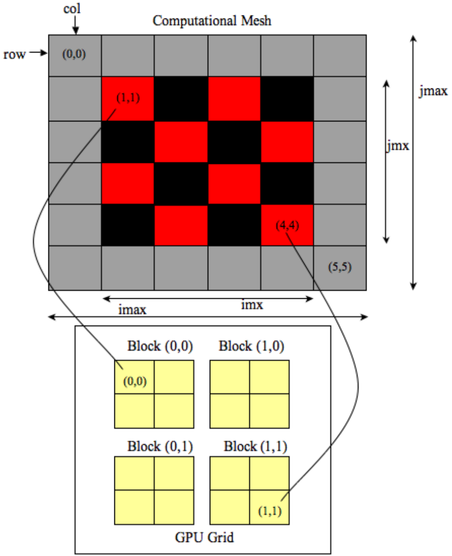
CUDA algorithm on device: `redkernel.cu`

Relations between threads and mesh cells

```
 // global thread indices (tx,ty)
int tx = blockIdx.x * BLOCK_SIZE + threadIdx.x;
int ty = blockIdx.y * BLOCK_SIZE + threadIdx.y;

// convert thread indices to mesh indices
row = (ty+1);
col = (tx+1);
```

CUDA algorithm on device: `redkernel.cu`



CUDA algorithm on device: redkernel.cu

Gauss-Seidel with SOR

```
if  ( (row + col) % 2 == 0 ) {    // red cell
   float omega = 1.85;
   float sum;
   k = row*imax + col;

  // perform SOR on red squares
  sum = aw_d[k]*T_old_d[row*imax+ (col-1)] + \
        ae_d[k]*T_old_d[row*imax+ (col+1)] + \
        as_d[k]*T_old_d[(row+1)*imax+ col] + \
        an_d[k]*T_old_d[(row-1)*imax+ col];

T_old_d[k]=T_old_d[k]*(1.0-omega)+omega*(sum/ap_d[k]);
}
```

GPU Results

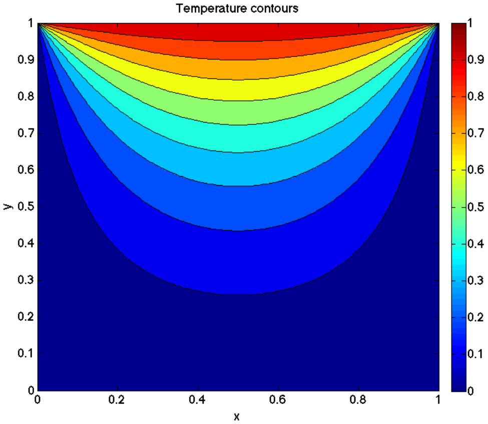


Figure: Solution of 2D heat conduction equation on unit square with T=1 as top B.C. and T=0 along left, right, and bottom

Governing Equations

Conservation of Mass

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{u} = 0$$

Conservation of Momentum

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \nabla \cdot \bar{\bar{\tau}}$$

Conservation of Energy

$$\rho C_p \frac{DT}{Dt} = \beta T \frac{Dp}{Dt} + \nabla \cdot (k \nabla T) + \Phi$$

where the viscous stress tensor is

$$\bar{\bar{\tau}} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + \delta_{ij} \lambda (\nabla \cdot \mathbf{u})$$

Discretization of Governing Equations

Fractional-Step Method

$$\rho \frac{\widehat{\mathbf{u}}^{n+1} - \mathbf{u}^n}{\Delta t} = -\mathbf{H}^n$$

$$\rho \frac{\mathbf{u}^{n+1} - \widehat{\mathbf{u}}^{n+1}}{\Delta t} = -\nabla p^{n+1}$$

$$\nabla \cdot \rho \mathbf{u}^{n+1} = 0$$

Pressure-Poisson Equation can consume 70-95% of CPU time!

Boundary Conditions

$$\mathbf{u}^{n+1} = \mathbf{u}_b$$

$$\nabla p^{n+1} \cdot \widehat{n} = 0$$

GPU research

- Developed Red-Black SOR solver for 2D heat conduction equation for GPU in CUDA
- GPU code currently 17 times faster than CPU code
- Developing CUDA code for Large-Eddy Simulations
- Collaborating with Prof. Wen-mei Hwu in ECE dept.
- Also collaborating with Jonathan Cohen from NVIDIA
- *Their guidance is greatly appreciated!*