Final Report

Pierce Hunter, Nick Kuckuck, Haoran Wang

6-07-2020

1 1D Approach

We solved the 1D equation

$$\frac{\partial^2 u}{\partial z^2} = -1; \quad 0 \leq z \leq 1 \tag{1}$$

with boundary conditions

$$\frac{\partial u}{\partial z}(1) = 0; \quad u(0) = 0 \tag{2}$$

in the four following ways:

- Direct solve on the CPU
- CG on the CPU
- Direct solve on the GPU
- CG on the GPU.

2 2D Approach

We now seek to expand the problem into 2D and solve using the same techniques. The 2D version of the problem is

$$\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = -1; \quad \begin{cases} 0 \leq y \leq 1 \\ 0 \leq z \leq 1 \end{cases} \tag{3}$$

and we expand upon the boundary conditions which become

$$\frac{\partial u}{\partial y} = 0 \text{ at } y = 1 \tag{4}$$

$$\frac{\partial u}{\partial z} = 0 \text{ at } z = 1 \tag{5}$$

$$u = 0 \text{ at } y = 0 \tag{6}$$

$$u = 0 \text{ at } z = 0. \tag{7}$$

2.1 Discretization

We discretize in space in both y and z using centered difference as

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta y^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta z^2} = -1 \qquad (8)$$

which simplifies when $\Delta y = \Delta z$ to

$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = -\Delta y^2. \qquad (9)$$

This discretization works when $2 \leq i \leq N$ and $2 \leq j \leq N$, but we need to solve on the boundaries

- $i = 2$
  - AND $j = 2$
    $$\begin{aligned} u_{1,2} + u_{2,1} - 4u_{2,2} + u_{3,2} + u_{2,3} &= -\Delta y^2 \\ -4u_{2,2} + u_{3,2} + u_{2,3} &= -\Delta y^2 \end{aligned}$$
  - AND $j = N$
    $$\begin{aligned} u_{1,N} + u_{2,N-1} - 4u_{2,N} + u_{3,N} + u_{2,N+1} &= -\Delta y^2 \\ u_{2,N-1} - 3u_{2,N} + u_{3,N} &= -\Delta y^2 \end{aligned}$$
  - OTHERWISE
    $$\begin{aligned} u_{1,j} + u_{2,j-1} - 4u_{2,j} + u_{3,j} + u_{2,j+1} &= -\Delta y^2 \\ u_{2,j-1} - 4u_{2,j} + u_{3,j} + u_{2,j+1} &= -\Delta y^2 \end{aligned}$$
- $i = N$
  - AND $j = 2$
    $$\begin{aligned} u_{N-1,2} + u_{2,1} - 4u_{N,2} + u_{N+1,2} + u_{N,3} &= -\Delta y^2 \\ u_{N-1,2} - 3u_{N,2} + u_{N,3} &= -\Delta y^2 \end{aligned}$$
  - AND $j = N$
    $$\begin{aligned} u_{N-1,N} + u_{N,N-1} - 4u_{N,N} + u_{N+1,N} + u_{N,N+1} &= -\Delta y^2 \\ u_{N-1,N} + u_{N,N-1} - 2u_{N,N} &= -\Delta y^2 \end{aligned}$$
  - OTHERWISE
    $$\begin{aligned} u_{N-1,j} + u_{N,j-1} - 4u_{N,j} + u_{N+1,j} + u_{N,j+1} &= -\Delta y^2 \\ u_{N-1,j} + u_{N,j-1} - 3u_{N,j} + u_{N,j+1} &= -\Delta y^2 \end{aligned}$$
- $j = 2$ AND $3 \leq i \leq N-1$
  $$\begin{aligned} u_{i-1,2} + u_{i,1} - 4u_{i,2} + u_{i+1,2} + u_{i,3} &= -\Delta y^2 \\ u_{i-1,2} - 4u_{i,2} + u_{i+1,2} + u_{i,3} &= -\Delta y^2 \end{aligned}$$
- $j = N$ AND $3 \leq i \leq N-1$
  $$\begin{aligned} u_{i-1,N} + u_{i,N-1} - 4u_{i,N} + u_{i+1,N} + u_{i,N+1} &= -\Delta y^2 \\ u_{i-1,N} + u_{i,N-1} - 3u_{i,N} + u_{i+1,N} &= -\Delta y^2 \end{aligned}$$

3 Convert to a Matrix

In order to convert this discretization to a matrix that can be used for a direct solve we need to define a new indexing convention. For this we calculate a global index k as

$$k = (i-2)(N-1) + (j-1). \tag{10}$$

We can then translate our discretization into this new system, starting with the corner (2,2):

- $(2,2)$

$$-4u_1 + u_N + u_2 = -\Delta y^2$$

- $(2,j)$ with $3 \leq j \leq N-1$

$$u_{j-2} - 4u_{j-1} + u_{N-2+j} + u_j = -\Delta y^2$$

- $(2,N)$

$$u_{N-2} - 3u_{N-1} + u_{2N-2} = -\Delta y^2$$

- $(i,2)$ with $3 \leq i \leq N-1$

$$\begin{aligned} &u_{(i-3)(N-1)+1} - 4u_{(i-2)(N-1)+1} \\ &+u_{(i-1)(N-1)+1} + u_{(i-2)(N-1)+2} = -\Delta y^2 \end{aligned}$$

- (i,j) with $3 \leq i \leq N-1$ and $3 \leq j \leq N-1$

$$\begin{aligned} u_{(i-3)(N-1)+j-1} + u_{(i-2)(N-1)+j-2} - 4u_{(i-2)(N-1)+j-1} \\ +u_{(i-1)(N-1)+j-1} + u_{(i-2)(N-1)+j} = -\Delta y^2 \end{aligned}$$

- (i,N) with $3 \leq i \leq N-1$

$$\begin{aligned} u_{(i-3)(N-1)+N-1} + u_{(i-2)(N-1)+N-2} - 3u_{(i-2)(N-1)+N-1} \\ +u_{(i-1)(N-1)+N-1} = -\Delta y^2 \end{aligned}$$

- $(N,2)$

$$u_{(N-3)(N-1)+1} - 3u_{(N-2)(N-1)+1} + u_{(N-2)(N-1)+2} = -\Delta y^2$$

- (N,j) with $3 \leq j \leq N-1$

$$\begin{aligned} u_{(N-3)(N-1)+j-1} + u_{(N-2)(N-1)+j-2} - 3u_{(N-2)(N-1)+j-1} \\ +u_{(N-2)(N-1)+j} = -\Delta y^2 \end{aligned}$$

- (N,N)

$$u_{(N-2)(N-1)} + u_{(N-2)N} - 2u_{(N-1)^2} = -\Delta y^2$$

So what we end up with is an $(N-1)^2 \times (N-1)^2$ matrix A and a solution vector b with $(N-1)^2$ entries. Moving across a row we start at $(2,2)$, to increase j by one we move to the right 1 entry, to increase i by 1 we move the right $(N-1)$ entries, such that we hit every value of j first, then move to the next i.

Along the diagonals of the matrix A we have -4 except in the following locations:

- rows $\alpha(N-1)$ the diagonal entry is -3 for $1 \leq \alpha \leq N-2$
- rows $(N-2)(N-1)+\beta$ the diagonal is -3 for $1 \leq \beta \leq N-2$
- row $(N-1)^2$ the diagonal is -2

We also have 4 other sub-diagonals that will all contain ones except where noted. These represent the following location:

- $j-1$ which is directly below the diagonal
  In Julia these are the locations: `[2:(N-1)²,  1:(N-1)²-1]`
  Exception: the locations $(\alpha(N-1)+1, \alpha(N-1))$ should be zero for $1 \leq \alpha \leq N-2$
- $j+1$ which is directly above the diagonal
  In Julia these are the locations: `[1:(N-1)²-1,  2:(N-1)²]`
  Exception: the locations $(\alpha(N-1), \alpha(N-1)+1)$ should be zero for $1 \leq \alpha \leq N-2$
- $i-1$ which are exactly $N-1$ below the diagonal
  In Julia these are the locations: `[N:(N-1)²,  1:(N-1)²-(N-1)]`
- $i+1$ which are exactly $N+1$ above the diagonal
  In Julia these are the locations: `[1:(N-1)²-(N-1),  N:(N-1)²]`

Once A is created it is probably easier to create a column vector of length $(N-1)^2$ in which every location contains $-\Delta y^2$. Once a solution is found via $u = A \backslash b$ or CG, then u can be reshaped to the correct dimensions either manually—`i = floor((k-1)/(N-1)) + 2`; `j = mod(k-1,N-1) + 2`—or via the reshape function transposed—`U = reshape(u,N-1,N-1)'`. Using reshape without the transpose puts the solution in meshgrid format (with y as the columns and z as the rows) similar to looking at a cross-section.

4 Example program run:

Direct solve on CPU
0.042633 seconds (8.16 k allocations: 11.199 MiB, 51.33norm between our solution and the exact solution = 1.443430e-05

Native Julia CG solve on CPU
15.070843 seconds (34 allocations: 1.985 MiB)
norm between our solution and the exact solution = 1.443430e-05

Direct solve on GPU
1.129700 seconds (209 allocations: 9.750 KiB)
norm between our solution and the exact solution = 1.443427e-05

CG solve on GPU
3.191565 seconds (4.40 M allocations: 172.127 MiB, 0.92norm between our solution and the exact solution = 1.443432e-05

---

5 Code Listing

Here is the code listing:

```
using Pkg
Pkg.add("LinearAlgebra")
Pkg.add("SparseArrays")
Pkg.add("IterativeSolvers")
Pkg.add("Printf")
Pkg.add("CuArrays")
Pkg.add("CUDAnative")
Pkg.add("CUDAdrv")

using LinearAlgebra
using SparseArrays
using IterativeSolvers
using Printf

using CuArrays
CuArrays.allowscalar(false)
using CUDAnative
using CUDAdrv

z = 5e-5
z = 0:z:1
N = length(z)


function exact(z)
    return -1/2 * z.^2 + z
end
```

```
# create sparse matrix A
Il = 2:N-1
Jl = 1:N-2
Iu = 1:N-2
Ju = 2:N-1

dl = ones(N-2)
du = ones(N-2)
d = -2*ones(N-1)
A = sparse(Il,Jl,dl,N-1,N-1) + sparse(Iu,Ju,du,N-1,N-1) + sparse(1:N-1,1
A[N-1, N-1] = -1
A .= A / (z^2)

b = -ones(N-1, 1)


#   Perform LU solve
println("Direct solve on CPU")
u_dummy_DSCPU = A \ b
@time u_int_DSCPU = A \ b
u_DSCPU = [0; u_int_DSCPU]
# @show umod
@printf "norm between our solution and the exact solution = \x1b[31m %e
println("-----------")
println()

#    Perform CG solve
println("Native Julia CG solve on CPU")
u_dummy_CGCPU = cg(-A, -b)
@time u_int_CGCPU = cg(-A, -b)
u_CGCPU = [0; u_int_CGCPU]
@printf "norm between our solution and the exact solution = \x1b[31m %e

println("-----------")
println()


d_A = CuArray(A)
d_b = CuArray(b)

println("Direct solve on GPU")
u_dummy_DSGPU = d_A \ d_b
@time u_int_DSGPU = d_A \ d_b
u_int_DSGPU_reg = Array(u_int_DSGPU)
u_DSGPU = [0; u_int_DSGPU_reg]
@printf "norm between our solution and the exact solution = \x1b[31m %e
println("-----------")
println()
```

```
# CG solve on GPU
d_A = CuArrays.CUSPARSE.CuSparseMatrixCSC(A)
d_b = CuArray(b)
u_cg = CuArray(zeros(size(d_b)))
u_dummy = CuArray(zeros(size(d_b)))

println("CG solve on GPU")
# u_dummy_CGCPU = cg(-d_A, -d_b)
cg!(u_dummy, d_A, d_b)
# @time u_int_CGGPU = cg(-d_A, -d_b)
@time cg!(u_cg, d_A, d_b)
u_int_CGGPU_reg = Array(u_cg)
u_CGGPU = [0; u_int_CGGPU_reg]
@printf "norm between our solution and the exact solution = \x1b[31m %e
println("-----------")
println()
```