Homework 4

Abhijit Chowdhary

April 15, 2019

Matrix Vector Operations on a GPU

2D Jacobi method on a GPU

Here I implemented Jacobi iteration naively on a GPU, just running each Jacobi step on the GPU. The kernel written looks like:

```
__global__ void jacobi_step_gpu(
                double *u, const double *u0, const double *f,
                const long N)
{
  int idx = (blockIdx.x) * blockDim.x + threadIdx.x + 1;
  int jdx = (blockIdx.y) * blockDim.y + threadIdx.y + 1;
  double h = 1.0 / (double)N;
  u[idx*N + jdx] = 0.25 * ( h*h*f[idx*N + jdx] + u0[(idx-1)*N + jdx] +
                                                 u0[idx*N + (jdx-1)] +
                                                 u0[(idx+1)*N + jdx] +
                                                 u0[idx*N + (jdx+1)] );
}
```

Pitch your final project

For my final project, I plan on working solo to try and implement Parareal, a numerical ODE system solver which is parallel in time. The basic idea behind the solver is to solve the system with a course and inexpensive solver, and use it's answers to correct in parallel with a finer solver. Repeat this process until desired stopping point. With this final project, I intend to:

- Implement as efficiently as possible with OpenMP (and CUDA/MPI time and/or method permitting) and verify the correctness of the method.
- Test it on a few different model ODEs, and solve a parabolic or elliptic PDE (discretized with the method of lines approach), noting efficiency.
- Experiment with hardware, verifying theoretical estimates for speedup relative to resources given.
- Experiment with choices of fine and course solvers, with the intent of getting the fastest solution for a given accuracy.

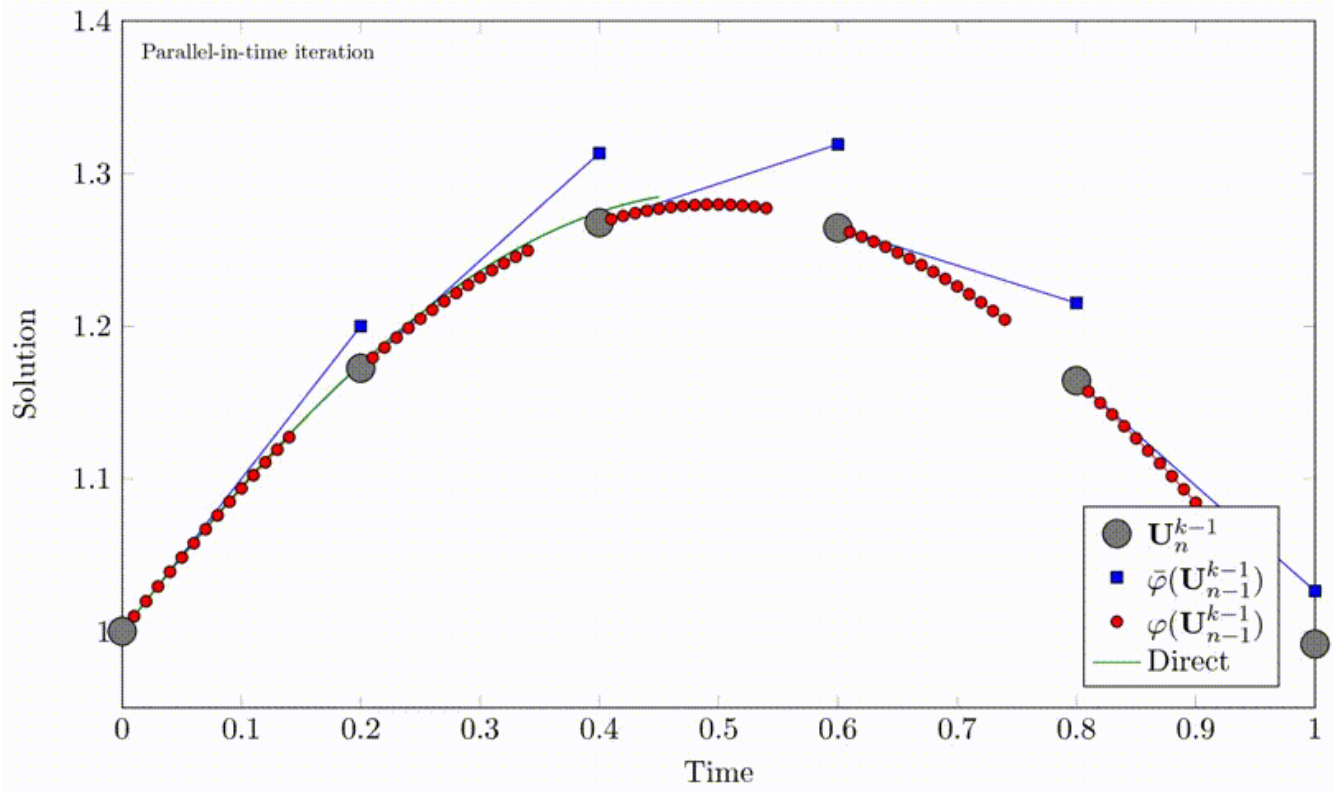

Figure 1: A sample Parareal solve mid iteration. Check out parareal.gif for gif version for the above (very slow). Credits wikipedia.