Parallelized Jacobi Method for Fast Poisson Image Blending

Amatur Rahman

Department of Computer Science and Engineering, Pennsylvania State University

E-mail: aur1111@psu.edu

Introduction

Goal:
Seamlessly cut and paste a source image on a target image, where the source and target are visually different.

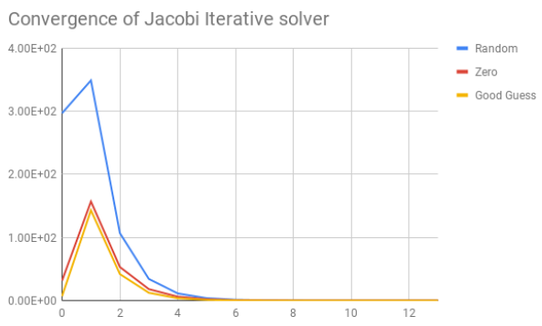
- Gradient domain processing of image
- Computationally expensive, even more for gigabyte sized images.

Why parallel?

- Sequential algorithms do not scale well to solve these problems.
- Too large image size to fit into one node

Direct vs. Indirect Solvers

- Our 'A' matrix is sparse, so indirect solver Jacobi does better.
- Sparse matrices do not generally have sparse LU decomposition, so it gets harder to fit the matrices in memory as the problem size goes larger.
- For Jacobi method, we store the A matrix in sparse representation.
- We tweaked the initial conditions to achieve fast convergence of direct solver Jacobi.



Effect of Parallelizing the Code

For Image size = 40 x 40 and 10 processors

Optimized Serial Runtime: 10.6 second
Parallel Jacobi Runtime: 1.21 second

8X speedup

Parallel version is performing better

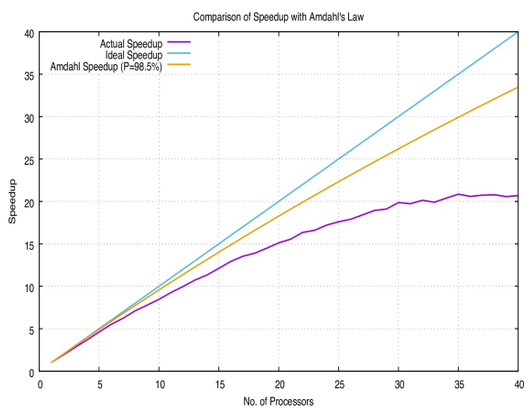
For smaller problem size, the problem size is much smaller to make effective use of parallelism.

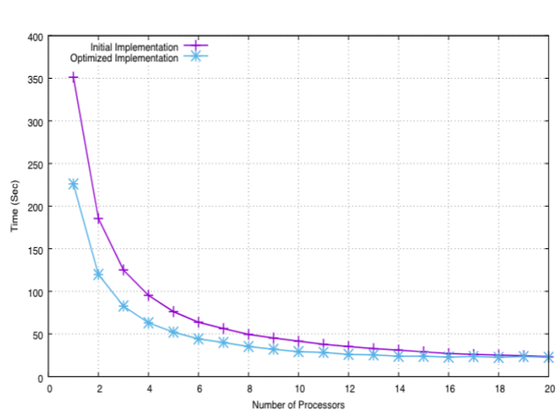
For dimension 20*20=400, non-MPI serial version actually does better.

For trivial problems, setting up the overhead of costly parallel communication is not helpful, rather it increases runtime.

Strong Scaling

For an image with 6400x6400 pixel, the parallel jacobi iterative method implemented using MPI can show upto **20x speedup** (over optimized serial code).

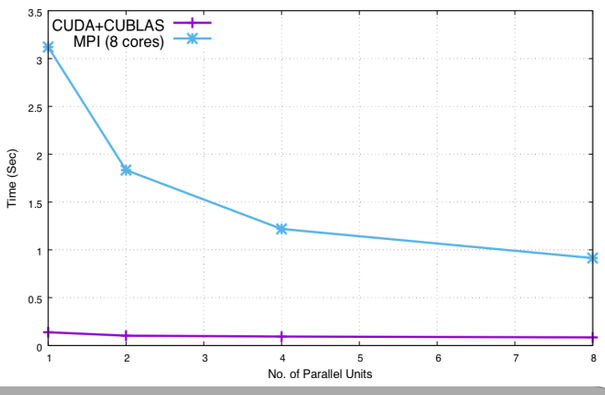




GPU and Coupling to External Library

Use GPU Programming: CUDA

Highly parallel structure of Jacobi iterative method: utilize that structure to the fullest



Coupling to library is useful:

- Makes coding simpler
- Cleaner memory management
- Easy learning curve
- Faster math operations

However, watch out for the **overheads** of moving data in and out of GPU device!

Best performance on data within a region of memory, not requiring too much movement

- Coupled our code to "thrust" and "cuBLAS" library
- Shows 5x speedup over a parallel MPI implementation with 8 processors.

Code Availability

Seial and Parallel Code:
https://github.com/amatur/cse597_parallel_solver

Coupling to External Library and GPU Code:
https://github.com/amatur/cuda597

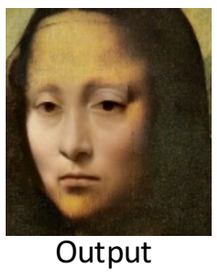
Poisson Image Blending

Seamlessly blend two images into a single one.

Used in: Mainly Digital Image Processing, Data augmentation in deep neural networks, Medical Imaging.


Source, S


Target, T


Output

Optimize: this equation to find out value f(x,y)

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|\partial\Omega$$

Formulated as a classical Ax=b problem, solved using Jacobi iterative method.

By discretizing the equation we get
A is a matrix of size NxN, a sparse matrix having -4 in diagonals, 1 in off-diagonals

$$b[i] = div(G(Source(x,y))) + Neighbor(target[i])$$

For simplification:
We use greyscale images

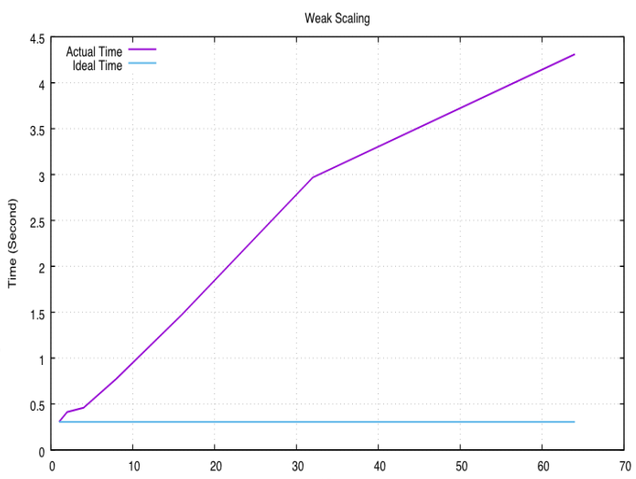
v = gradient of a region in an image,
g = selected region of source
f* = set of known functions that exist in domain S
f = unknown function f(x,y) that exist in domain Ω
Ω = region g after placing it on source image
∂Ω = boundaries between the source and target regions

Weak Scaling

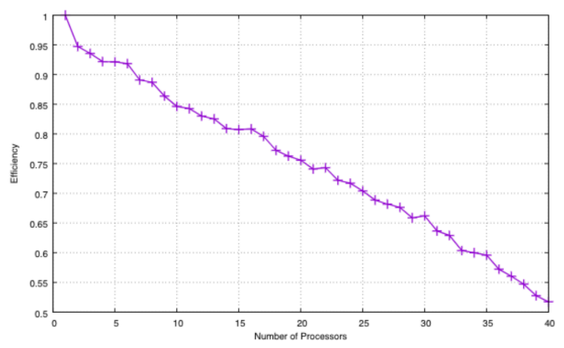
Keeping the workload per process constant, we aim to show the rate of increase in runtime.

In real world, linear speedup is impossible. Because of the instrumentation code and MPI communication overheads, as number of processors increase, runtime increases as well.

So actual time is far from the ideal expectation.



Amount of Resources Required



Efficiency drops as number of processors increase. So we are just bringing more resources, while a lot of those resources are just sitting idle and wasting their compute time. This results in loss of efficiency. We would want to be as efficient as possible, because with more processors come more cost. So around **70-80%** efficiency is good enough. For that we pick the number of processing units from **15-20**.

Results Overview

- Iterative solver runs **faster** than direct solver, but gives inexact solution.
- Parallelization works well, but up to a point.
- Among serial and parallel method, parallel method proves it usefulness when the compute time and memory is exceeded in a single computer node.
- Coupling to GPU (CUDA) and external library (cuBLAS) gives us performance boost on runtime over MPI code even in processors with 8 cores: **5x** speedup is obtained.

Acknowledgements

ICS-ACI and XSEDE computing resources were used to perform the computations.

All resources required for completing this project were provided by Dr. Adam Lavely and Dr. Chris Blanton.

References

[1] http://eric-yuan.me/poisson-blending/
[2] http://www.ctralie.com/Teaching/PoissonImageEditing/
[3] Barker, B. Message passing interface (mpi). In Workshop: High Performance Computing on Stampede (2015), vol. 262.
[4] Perez, P., Gangnet, M., and Blake, A. Poisson image editing. ACM Transactions on graphics (TOG) 22, 3 (2003), 313–318