PARALLEL PROGRAM FOR IMAGE CLONING

CUDA Programming Approach

Presenter: Yan Shen
Instructor: Dr. Russ Miller
University at Buffalo

University at Buffalo The State University of New York

OUTLINE

- Problem Statement
- Algorithm Design
- Source Availability
- Implementation Work Flow
- Performance Analysis
- Conclusion

Image Cloning

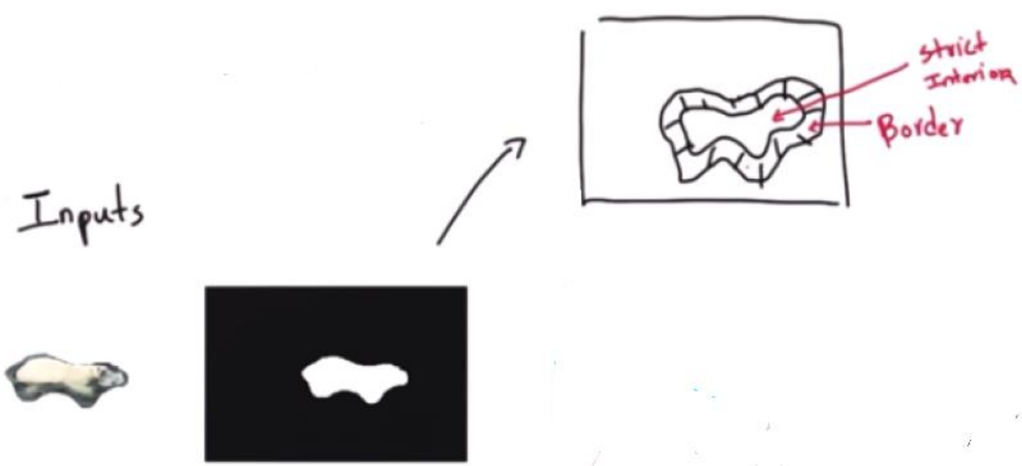
Definition:

Seamless placing a **source image** patch into a **target image**, which **smoothly** interpolates the **discrepancies between the boundary** of source patch and the target across the entire cloned area.

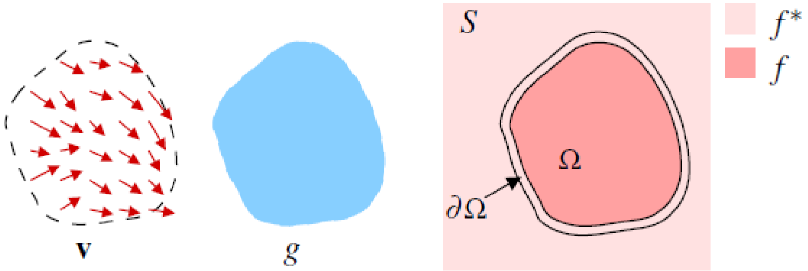
Image Preprocessing

Segment Source Image Patches into three type:

- Border
- Stricter interior
- Background



Discrete Possion Solver for Guided Interpolation



- Membrane interpolation problem under a guidance field is defined as a solution to a minimization problem

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

- Written in discrete from:

$$\min_{f|_\Omega} \sum_{\langle p,q\rangle \cap \Omega \neq \emptyset} (f_p - f_q - v_{pq})^2, \text{ with } f_p = f_p^*, \text{for all } p \in \partial\Omega,$$

$$\text{for all } \langle p,q\rangle, v_{pq} = g_p - g_q,$$

- Solution of a minimization problem satisfies the following equation:

$$\text{for all } p \in \Omega, \quad |N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq}.$$

Discrete Possion Solver for Guided Interpolation

- Solving the following Linear Equations:

$$\text{for all } p \in \Omega, \quad |N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p} v_{pq}.$$

$$\text{for all } \langle p, q \rangle, v_{pq} = g_p - g_q,$$

- Jacoby Iterative Solution for Linear Equations:

$$\text{for all } p \in \Omega, \; f_p^1 = \frac{\sum_{q \in N_p \cap \partial\Omega} f_q^* + \sum_{q \in N_p \cap \Omega} f_q + \sum_{q \in N_p} v_{pq}.}{|N_p|}$$

$$\text{for all } \langle p, q \rangle, v_{pq} = g_p - g_q,$$

GPU Resource

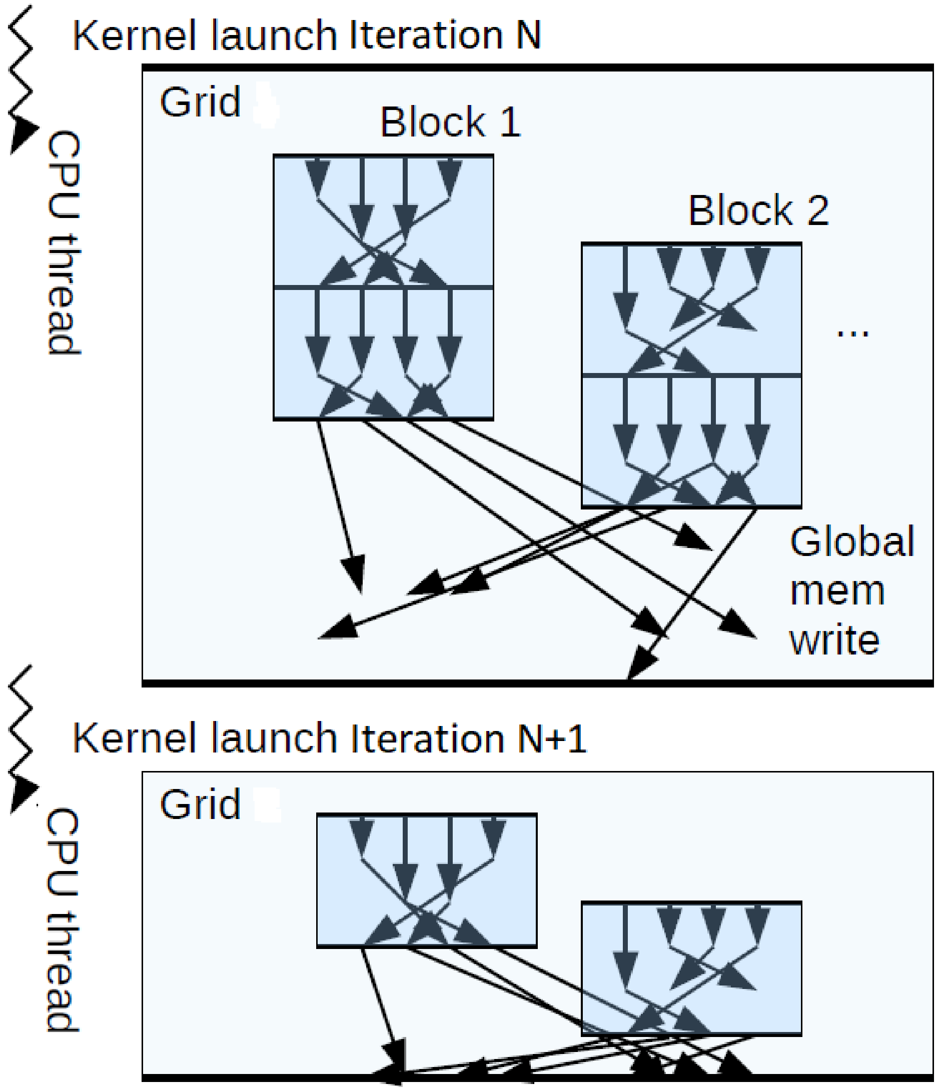
Hardware Spec in CCR:

- Name: Tesla V100
- CUDA Version: 6.5
- Shared memory per block: 49152
- Total constant memory: 165536
- Regs per block:32768
- Max threads per block: 1024
- Max threads per dim: 1024,1024,64
- Max grid size: 65535, 65535, 65535
- Multi processor count: 14

Work Flow

CPU	GPU
Load source patch and target image	
Preprocess source patch to interior and boundary	
Allocate space in GPU and copy source patch, target image and interior/boundary labels into GPU	
Create two buffers for iteration and copy the target image into one buffer as initial guess	
Launch a kernel for one iteration	Complete one iteration in GPU
Launch a kernel to swap memory between two buffers	Swap memory in GPU
Launch a kernel for one iteration	……
Copy back from iteration buffer	

Synchronization



Sbatch Script

- #!/bin/bash.
- #SBATCH --partition=gpu
- #SBATCH --time=01:00:00
- #SBATCH --qos=gpu
- #SBATCH --nodes=1
- #SBATCH --ntasks-per-node=1
- #SBATCH --output=slurmNAMD.out
- #SBATCH --job-name=cuda
- module load cuda/6.5
- module load python
- module load opencv
- srun image_cloning source.png destination.png

Parallel Reduction

Average Running Time on CPU:

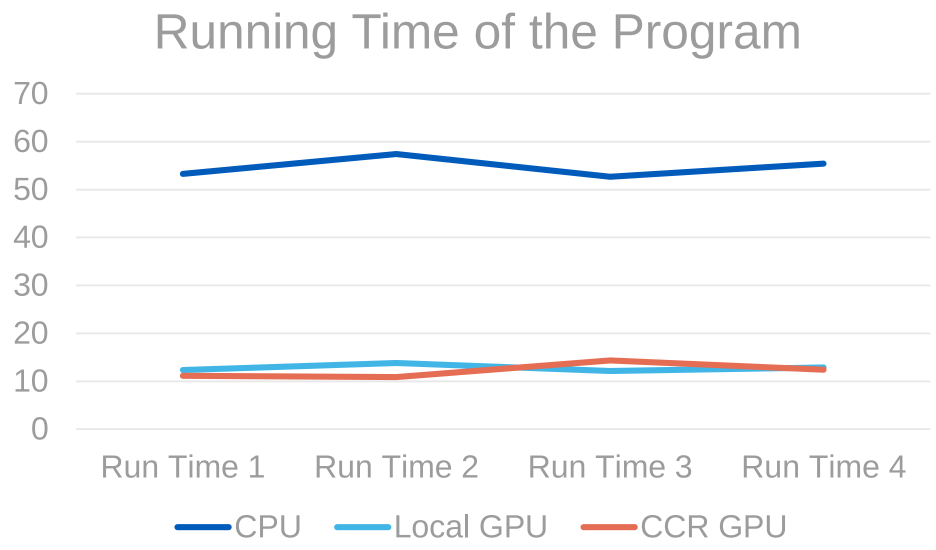
- 54.71ms

Average Running Time on Local GPU

- 12.78ms

Average Running Time on CCR GPU

- 12.19ms

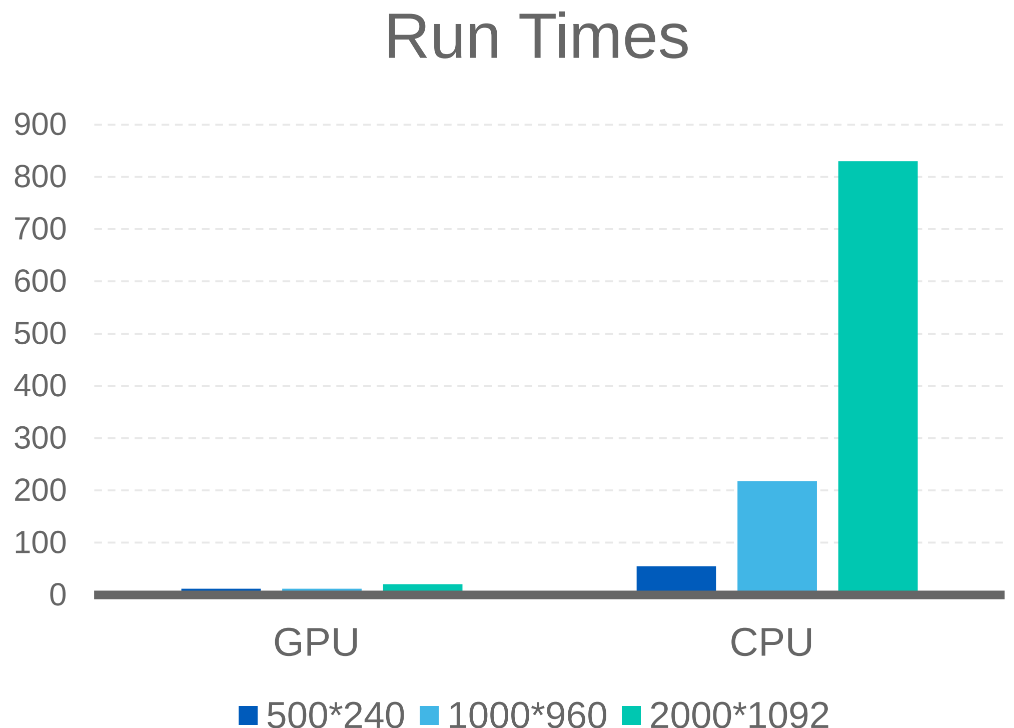


Varying Block Size

	Grid Size	
Block Size	1	2
128	40.12ms	35.23ms
256	22.45ms	18.17ms
512	12.19ms	10.08ms
768	8.78ms	6.98ms

Varying Image Size

	500*240	1000 *960	2000*1920
GPU	12.19ms	12.31ms	20.54ms
CPU	54.71ms	218.13ms	830.12ms



Conclusions

- For pure mathematical computations, GPU offers huge speed up by offering huge parallelism
- Synchronization cost is a big overhead
- GPU memory is an important constrains
- Data I/0 cost from GPU to CPU and GPU to CPU is higher than higher than intra memory I/O
- Set reasonable grid and block size


Thank You!