PARALLEL PROGRAM FOR IMAGE CLONING

CUDA Programming Approach

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

\[
\min_{f} \iint_{\Omega} |\nabla f - v|^2 \text{ with } f|_{\partial \Omega} = f^*|_{\partial \Omega},
\]

- Written in discrete form:

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

- 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:

\[
\forall p \in \Omega, \quad |N_p| f_p - \sum_{q \in N_p \cap \Omega} f_q = \sum_{q \in N_p \cap \partial \Omega} f_q^* + \sum_{q \in N_p} v_{pq}. 
\]

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

- Jacoby Iterative Solution for Linear Equations:

\[
\forall p \in \Omega, \quad f_p^{(1)} = \frac{\sum_{q \in N_p \cap \partial \Omega} f_q^* + \sum_{q \in N_p \cap \Omega} f_q + \sum_{q \in N_p} v_{pq}}{|N_p|}. 
\]

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

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
Launch a kernel to swap memory between two buffers
Launch a kernel for one iteration
Copy back from iteration buffer

GPU

Complete one iteration in GPU
Swap memory in GPU

……
Synchronization

Kernel launch Iteration N

CPU thread

Grid Block 1 Block 2

Global mem write

Kernel launch Iteration N+1

CPU thread
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

Running Time of the Program
Varying Block Size

<table>
<thead>
<tr>
<th>Block Size</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>40.12ms</td>
<td>35.23ms</td>
</tr>
<tr>
<td>256</td>
<td>22.45ms</td>
<td>18.17ms</td>
</tr>
<tr>
<td>512</td>
<td>12.19ms</td>
<td>10.08ms</td>
</tr>
<tr>
<td>768</td>
<td>8.78ms</td>
<td>6.98ms</td>
</tr>
</tbody>
</table>
Varying Image Size

<table>
<thead>
<tr>
<th></th>
<th>500*240</th>
<th>1000*960</th>
<th>2000*1920</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>12.19ms</td>
<td>12.31ms</td>
<td>20.54ms</td>
</tr>
<tr>
<td>CPU</td>
<td>54.71ms</td>
<td>218.13ms</td>
<td>830.12ms</td>
</tr>
</tbody>
</table>

Run Times
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/O 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!