Parallelized Graphics Rendering Using Software Implementation Of OpenGLES

Rakesh Shaji Lal
Srivatsan Bhaskar
Bingnan Huang
OpenGLES

- **Open Graphics Library for Embedded Systems**
  - API for rendering graphics in mobile devices

- Often interacts with GPU for accelerated graphics rendering
Objective

• Parallelize the graphics rendering of a test application using software implementation of OpenGLES (implemented in C++)

• Useful for accelerated graphics rendering in the absence of hardware accelerators like GPU
for (i = 0; i < N; i++)
{
    Cube Geometry
    Graphics Pipeline
    Pixel Array
    X Window System
    Rotates cube by an angle determined by i
}
Parallelization Methodologies

1. Rotation Parallelization using MPI

2. Cube Face Rendering Parallelization Using OpenMP

3. Rasterization Parallelization using MPI
for( i = myrank; i < N; i = i + num_processors )
{
    
    Processor #0
    Cube Geometry
    Graphics Pipeline
    Pixel Array
    X Window System

    Processor #1
    Cube Geometry
    Graphics Pipeline
    Pixel Array

    ... ...

    Processor #(num_processors - 1)
    Cube Geometry
    Graphics Pipeline
    Pixel Array

    Overwrite in order - Synchronization

}
### Rotation Parallelization Using MPI

<table>
<thead>
<tr>
<th>Description</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>0.8%</td>
</tr>
<tr>
<td>Parallel</td>
<td>99.2%</td>
</tr>
<tr>
<td>Synchronization</td>
<td>33.1% of the parallel section</td>
</tr>
</tbody>
</table>

#### Speedup Values
- **1 core**: 0.99
- **2 cores**: 1.48
- **4 cores**: 2.02
Parallelization Methodologies

1. Rotation Parallelization using MPI

2. Cube Face Rendering Parallelization Using OpenMP

3. Rasterization Parallelization using MPI
Cube Face Rendering Parallelization using OpenMP

```c
#pragma omp section
// Rewrote the matrix operations
// to handle privatized matrices per face
{
  Front face
  Top face
  Back face
```

<table>
<thead>
<tr>
<th>Graphics Pipeline</th>
<th>Graphics Pipeline</th>
<th>Graphics Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front face</td>
<td>Top face</td>
<td>Back face</td>
</tr>
<tr>
<td>Pixel Array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X Window System</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

60% of code under critical section

99.1% of execution time under critical section
<table>
<thead>
<tr>
<th>Description</th>
<th>Time in microseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical Section in Parallelized Code with 1 core</td>
<td>21110</td>
</tr>
<tr>
<td>Same Section in original Sequential Code</td>
<td>9965</td>
</tr>
</tbody>
</table>
Parallelization Methodologies

1. Rotation Parallelization using MPI

2. Cube Face Rendering Parallelization Using OpenMP

3. Rasterization Parallelization using MPI
Rasterization Parallelization

Cube Geometry

Graphics Pipeline

Pixel Array

X Window System

Screen (a collection of pixels)

Rasterization

99% of pipeline execution time
Rasterization Parallelization using MPI

Screen (a collection of pixels)

Rasterization

N Splits

Split #i

Processor #i owns this screen space
for( i =0; i < N; i++)  //10 Induction Variables Substituted
{
    Cube Geometry
    Graphics Pipeline (rasterize owned screen space)
    Pixel Array for owned screen space
    Pixel Array for entire screen space
    X Window System
    
    . . . . . .
    
    Cube Geometry
    Graphics Pipeline (rasterize owned screen space)
    Pixel Array for owned screen space
    Send Pixel Array for owned screen space
    
    . . . . . .
    
    Cube Geometry
    Graphics Pipeline (rasterize owned screen space)
    Pixel Array for owned screen space
}
Rasterization Parallelization using MPI

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>0.97</td>
</tr>
<tr>
<td>2 cores</td>
<td>1.43</td>
</tr>
<tr>
<td>4 cores</td>
<td>2.29</td>
</tr>
</tbody>
</table>
For 4 cores:
- Rotation Parallelization Speedup (2.02) < Rasterization Parallelization Speedup (2.29)
- Counterintuitive
  - Rotation Parallelization has more parallel execution than Rasterization Parallelization

Rotation Parallelization vs Rasterization Parallelization

Less Rasterization for Rank 0
Work Offloading

Rotation Parallelization

Rasterization Parallelization
Extra Slides
Induction Variable Substitution Schemes

- **Sequential**
  
  \[
  k = 0; \\
  \text{for}(i = 1; \ i < \ N; \ i++) \\
  \quad k = k + 2
  \]

- **Parallel Scheme 1**
  
  \[
  \text{for}(.....) \\
  \quad k = i*2;
  \]

- **Parallel Scheme 2**
  
  \[
  k = \text{myrank} \times \text{chunk\_size} \times 2 \\
  \text{for}(.....) \\
  \quad k = k + 2;
  \]

---

**Parallel Scheme 1 vs Parallel Scheme 2**

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Speedup (Parallel Scheme 1)</th>
<th>Speedup (Parallel Scheme 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 core</td>
<td>0.96</td>
<td>0.97</td>
</tr>
<tr>
<td>2 cores</td>
<td>1.42</td>
<td>1.43</td>
</tr>
<tr>
<td>4 cores</td>
<td>2.23</td>
<td>2.29</td>
</tr>
</tbody>
</table>

- Blue: Parallel Scheme 1
- Red: Parallel Scheme 2
Linux Build Creation

OpenGLES source code
(Build for Windows)
http://sourceforge.net/projects/ogl-es/files/ogl-es/1.00/

Test Application and Linux OpenGLES Emulator
(no source code for OpenGLES)
http://malideveloper.arm.com/develop-for-mali/tools/software-development/

Relevant libraries

OpenGLES source code with Test Application
(Build for Linux)
Future Work

• Parallelize Rasterization using OpenMP
  - will help eliminate critical section in Cube Face Rendering Parallelization

• Split only the Rasterized screen space evenly among processors using MPI
  - requires OpenGL programming knowledge

• Analyze the parallel engine using more complex test applications
  - Speedup is likely to increase
References