Outsourcing database indexes to the GPU

Jeff Hagen & Tim Kaldewey
Jack Baskin School of Engineering, UC Santa Cruz
{jhagen,kalt}@cs.ucsc.edu
Video Cards & Databases?

- Video cards offer very good parallel performance.
- Databases are used concurrently by many users in parallel.
  - Make use of the video card, to speed up common operations on the database.
- What are common operations?
  - Searching!
Searching Databases

- Index structures to speed up searching
- Common data structures: trees, bitmaps, hashes
- Usage depends on application,
  e.g. B+trees exploit locality in consecutive queries
CPUs use pipelining and branch prediction

- 50% prediction accuracy using binary search

⇒ Stalls

How about BIG databases?

- Index structures don't fit cache

⇒ Longer stalls
Searching B+tress – on the Video card

- Bus speed for video and memory is ~same

  BUT video card is a powerful (multi)processor

- Less concerned with locality ... but parallelism

- Binary search in B+tress is inherently serial, while multiple searches map well [1]

  ➔ Require different algorithms & data structures

Simple GPU Index Search (1)

• Load null-terminated strings into VRAM as a texture (2D images)

• Load search string in a separate texture

• All pixels are processed independently

• Parallelism is transparent (# of fragment processors)
Searches are performed in parallel

All pixels are processed independently

Parallelism is transparent

In most cases, a pixel does not contain the beginning of a search string.
  • Many processing elements are wasted!
Simple GPU Index Search (3)

- Fragment program “reduces” 4 pixels into 1
- Texture size = x^4 keeps 4 fragment processors busy

Choose texture size dependent on #processors

```c
float4 reduce (float2 coords: WPOS,
    uniform samplerRECT texCgFrag2) : COLOR
{
    float2 topleft = coords * 2.0;

    float4 val1 = texRECT(texCgFrag2, topleft);
    float4 val2 = texRECT(texCgFrag2, topleft+float2(1,0));
    float4 val3 = texRECT(texCgFrag2, topleft+float2(1,1));
    float4 val4 = texRECT(texCgFrag2, topleft+float2(0,1));

    if (val4.r == 254.0) result = val4;
    if (val3.r == 254.0) result = val3;

    ...
```
Experimental Setup

- BerkeleyDB - Test001.tcl
  - 10,000 insert/delete operations
  - Every operation searches

- Hardware Setup
  - Dual-core 2.2ghz AMD64 w/ 3GB Main RAM
  - nVidia 7900GS 256MB Video Ram
    - 7 vertex proc.
    - 20 fragment proc.

- Measure Performance of searching 1 B+tree node GPU VS CPU
Results

Runtime for a small test increases significantly...

Speed Comparison

<table>
<thead>
<tr>
<th>10k Insert/Deletes</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDB</td>
<td>BDB+Video card</td>
</tr>
</tbody>
</table>

- BDB
- BDB+Video card
Performance analysis

Where do we spend most of the GPU time?

Timing Breakdown

- OpenGL Overhead
- OpenGL Startup
- Data re-order
- Data transfer
- Shader programs

Speed Comparison

<table>
<thead>
<tr>
<th>Seconds</th>
<th>10k Insert/Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>2.50</td>
<td></td>
</tr>
<tr>
<td>5.00</td>
<td></td>
</tr>
<tr>
<td>7.50</td>
<td></td>
</tr>
<tr>
<td>10.00</td>
<td></td>
</tr>
<tr>
<td>12.50</td>
<td></td>
</tr>
<tr>
<td>15.00</td>
<td></td>
</tr>
<tr>
<td>17.50</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion and future work

• Demonstrated index search using the GPU
• Performance requires improvement
• High potential for improvement
  ⇒ hardware, algorithms and data structures

• Future Work:
  • Evaluate new algorithms & data structures
  • Load full index into VRAM at DB startup
  • Add Functionality: Sorting, Updates
Questions ?