Data Parallel Computing on Graphics Hardware

Ian Buck
Stanford University
Brook
General purpose *Streaming* language

- DARPA Polymorphous Computing Architectures
  - Stanford - Smart Memories
  - UT Austin - TRIPS Processor
  - MIT - RAW Processor
- Stanford Streaming Supercomputer
- Brook: general purpose *streaming* language
  - Language developed at Stanford
  - Compiler in development by Reservoir Labs
- Study of GPUs as Streaming processor
Why graphics hardware

Raw Performance:

Pentium 4 SSE Theoretical:

\[ 3\text{GHz} \times 4 \text{ wide} \times 0.5 \text{ inst / cycle} = 6 \text{ GFLOPS} \]

GeForce FX 5900 (NV35) Fragment Shader Obtained:

\[ \text{MULR R0, R0, R0: 20 GFLOPS} \]

Equivalent to a 10 GHz P4

And getting faster: 3x improvement over NV30 (6 months)

2002 R&D Costs:

Intel: $4 Billion
NVIDIA: $150 Million

*from Intel P4 Optimization Manual
GPU: Data Parallel

– Each fragment shaded independently
  • No dependencies between fragments
    – Temporary registers are zeroed
    – No static variables
    – No Read-Modify-Write textures
  • Multiple “pixel pipes”

– Data Parallelism
  • Support ALU heavy architectures
  • Hide Memory Latency

[Torborg and Kajiya 96, Anderson et al. 97, Igehy et al. 98]
Arithmetic Intensity

Lots of ops per word transferred

Graphics pipeline

– Vertex
  • BW: 1 triangle = 32 bytes;
  • OP: 100-500 f32-ops / triangle

– Rasterization
  • Create 16-32 fragments per triangle

– Fragment
  • BW: 1 fragment = 10 bytes
  • OP: 300-1000 i8-ops/fragment

Courtesy of Pat Hanrahan
Arithmetic Intensity

- Compute-to-Bandwidth ratio
- High Arithmetic Intensity desirable
  - App limited by ALU performance, not off-chip bandwidth
  - More chip real estate for ALUs, not caches

Courtesy of Bill Dally
Brook
General purpose Streaming language

Stream Programming Model
- Enforce Data Parallel computing
- Encourage Arithmetic Intensity
- Provide fundamental ops for stream computing
Brook
General purpose Streaming language

- Demonstrate GPU streaming coprocessor
  - Make programming GPUs easier
    - Hide texture/pbuffer data management
    - Hide graphics based constructs in CG/HLSL
    - Hide rendering passes
  - Highlight GPU areas for improvement
    - Features required general purpose stream computing
Streams & Kernels

• Streams
  – Collection of records requiring similar computation
    • Vertex positions, voxels, FEM cell, …
  – Provide data parallelism

• Kernels
  – Functions applied to each element in stream
    • transforms, PDE, …
  – No dependencies between stream elements
    • Encourage high Arithmetic Intensity
Brook

- **C with Streams**
  - API for managing streams
  - Language additions for kernels

- **Stream Create/Store**

  ```
  stream s = CreateStream (float, n, ptr);
  StoreStream (s, ptr);
  ```
Kernel Functions

- Pos update in velocity field
- Map a function to a set

```c
kernel void updatepos (stream float3 pos,
        float3 vel[100][100][100],
        float timestep,
        out stream float newpos) {
    newpos = pos + vel[pos.x][pos.y][pos.z]*timestep;
}
```

`s_pos = CreateStream(float3, n, pos);`
`s_vel = CreateStream(float3, n, vel);`
`updatepos (s_pos, s_vel, timestep, s_pos);`
Fundamental Ops

• Associative Reductions
  
  KernelReduce(func, s, &val)
  
  – Produce a single value from a stream
  – Examples: Compute Max or Sum

<table>
<thead>
<tr>
<th>8</th>
<th>6</th>
<th>3</th>
<th>7</th>
<th>2</th>
<th>9</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
</table>

40
Fundamental Ops

- **Associative Reductions**
  \[\text{KernelReduce} (\text{func}, \text{s}, \&\text{val})\]
  - Produce a single value from a stream
  - Examples: Compute Max or Sum

- **Gather:** \[p = a[i]\]
  - Indirect Read
  - Permitted inside kernels

- **Scatter:** \[a[i] = p\]
  - Indirect Write
  - Last write wins rule
    \[\text{ScatterOp} (\text{s\_index}, \text{s\_data}, \text{s\_dst}, \text{SCATTEROP\_ASSIGN})\]
GatherOp & ScatterOp

Indirect read/write with atomic operation

• **GatherOp:** \( p = a[i]++ \)
  
  \[
  \text{GatherOp}(s\_index, s\_data, s\_src, \text{GATHEROP\_INC})
  \]

• **ScatterOp:** \( a[i] \ += \ p \)
  
  \[
  \text{ScatterOp}(s\_index, s\_data, s\_dst, \text{SCATTEROP\_ADD})
  \]

• Important for building and updating data structures for data parallel computing
Brook

• C with streams
  – kernel functions
  – CreateStream, StoreStream
  – KernelReduce
  – GatherOp, ScatterOp
Implementation

• Streams
  – Stored in 2D fp textures / pbuffers
  – Managed by runtime

• Kernels
  – Compiled to fragment programs
  – Executed by rendering quad
Implementation

• Compiler: brcc

• Source to Source compiler
  – Generate CG code
    • Convert array lookups to texture fetches
    • Perform stream/texture lookups
    • Texture address calculation
  – Generate C Stub file
    • Fragment Program Loader
    • Render code
Force Function (~90% compute time): 

\[ F_i (r_{ij}) = \left( \frac{1}{4\pi \varepsilon_0} \frac{q_i q_j}{r_{ij}^2} + 12 \frac{C_{12}}{r_{ij}^{12}} - 6 \frac{C_6}{r_{ij}^6} \right) \frac{r_{ij}}{r_{ij}} \]

Energy Function:

\[ V_{nb} = \sum_{i,j} \left( \frac{1}{4\pi \varepsilon_0} \frac{q_i q_j}{r_{ij}} + \left( \frac{C_{12}}{r_{ij}^{12}} - \frac{C_6}{r_{ij}^6} \right) \right) \]

Gromacs
Molecular Dynamics Simulator
Eric Lindhal, Erik Darve, Yanan Zhao

July 27th, 2003
Ray Tracing
Tim Purcell, Bill Mark, Pat Hanrahan
Finite Volume Methods

Joseph Teran, Victor Ng-Thow-Hing, Ronald Fedkiw

\[ \sigma = pI + 2 \left\{ \left( W_1 + I_1 W_2 \right) B - W_2 B^2 \right\} + W_4 a \otimes a \]

\[ W_i = \partial W / \partial t_i \]
Applications

Sparse Matrix Multiply
Batcher Bitonic Sort

\[
\begin{pmatrix}
3 & 0 & 0 & 2 \\
0 & 0 & 0 & 1 \\
0 & 4 & 0 & 0 \\
6 & 0 & 0 & 8
\end{pmatrix}
\begin{pmatrix}
8 \\
3 \\
6 \\
2
\end{pmatrix}
\]

elem: (3 2 1 4 6 8)
ipos: (0 3 3 1 0 4)
start: (0 2 3 4)
len: (2 1 1 2)
v: (8 3 6 2)
Summary

• GPUs are faster than CPUs
  – and getting faster
• Why?
  – Data Parallelism
  – Arithmetic Intensity
• What is the right programming model?
  – Stream Computing
  – Brook for GPUs
GPU Gotchas

NVIDIA NV3x: Register usage vs. GFLOPS

July 27th, 2003
GPU Gotchas

- ATI Radeon 9800 Pro
- Limited dependent texture lookup
- 96 instructions
- 24-bit floating point
Summary

“All processors aspire to be general-purpose”
– Tim van Hook, Keynote, Graphics Hardware 2001
GPU Issues

• Missing Integer & Bit Ops
• Texture Memory Addressing
  – Address conversion burns 3 instr. per array lookup
  – Need large flat texture addressing
• Readback still slow
• CGC Performance
  – Hand code performance critical code
• No native reduction support
GPU Issues

• No native Scatter Support
  – Cannot do p[i] = a (indirect write)
  – Requires CPU readback.
  – Needs:
    • Dependent Texture Write
    • Set x,y inside fragment program

• No programmable blend
  – GatherOp / ScatterOp
GPU Issues

• Limited Output
  – Fragment program can only output single 4-component float or 4x4 component float (ATI)
  – Prevents multiple kernel outputs and large data types.
Implementation

- Reduction
  - $O(\log(n))$ Passes
- Gather
  - Dependent texture read
- Scatter
  - Vertex shader (slow)
- GatherOp / ScatterOp
  - Vertex shader with CPU sort (slow)
Acknowledgments

- NVIDIA Fellowship program
- DARPA PCA
- Pat Hanrahan, Bill Dally, Mattan Erez, Tim Purcell, Bill Mark, Eric Lindahl, Erik Darve, Yanan Zhao
Status

• Compiler/Runtime work complete
• Applications in progress
• Release open source in fall
• Other streaming architectures
  – Stanford Streaming Supercomputer
  – PCA Architectures (DARPA)