VSIPL++ Acceleration Using Commodity Graphics Processors

Dan Campbell
Georgia Tech Research Institute
7220 Richardson Road
Smyrna, GA 30080
dan.campbell@gtri.gatech.edu

High Performance Embedded Computing (HPEC) Workshop
21 September 2006
Signal Processing on Graphics Processors

• GPUs are fixed function: turn 3-D polygons into 2-D pixels
• Also a cheap & plentiful source of FLOPs
  • Leverages volume & competition in entertainment industry
    • Worldwide GPUs: $5B, 10M units per year
    • U.S. Video Games: $7.5B, 250M units 2004
    • Holds down unit-price, drives advancement
  • Primary application highly parallel, very regular
  • Opaque, stable abstraction (graphics APIs)
  • Recent changes make GPUs usable for signal processing
• Outstripping CPU capacity, and growing more quickly
GPU Performance Trends: Fragment Processor

CPU & GPU Capacity Growth

GFLOPS (Peak Theoretical)

- ATI
- NVIDIA
- Pentium 4
- "Moore's Law"


- R580: 244
- NV40: 122
- Dual Core: 30
GPU Graphics Stream

- GPU Dispatch
- Vertex Processor
- Transformation & Lighting
- Primitive Assembly
- Rasterization, Interpolation
- Fragment Processor
- Pixel Shading, Texturing, Illumination
- Commands, vertex stream
- Camera-space vertices w/ light & texture info
- Polygons, lines, points
- Fragments
- Pixel Color Updates

- 48 dual-issue Pipelines 650 MHz 4-Way SIMD
GPU Graphics Stream

- **Commands, vertex stream**
  - GPU Dispatch
  - Vertex stream
    - **Vertex Processor**
      - Camera-space vertices w/ light & texture info
    - **Primitive Assembly**
      - Polygons, lines, points
    - **Rasterization, Interpolation**
    - **Fragment Processor**
      - Fragments
      - Signal processing & general purpose calculations
      - Fast Signal Processing & general purpose calculations

- **Output Buffer Updates**
  - 240+ GFLOPS
GPGPU (Simple) Concept of Operations

- Arrays ➔ Textures
- Render polygon with the same pixel dimensions as output texture
- Execute with fragment program to perform desired calculation
- Move data from output buffer to desired texture

```c
void main(float2 tc0 : TEXCOORD0,
          out float4 col : COLOR,
          uniform samplerRECT B,
          uniform samplerRECT C)
{
    col = texRECT (B, tc0) + texRECT (C, tc0);
}
```
GPUs are difficult to Program

- Forced to put computations into graphics context
- Requires journeyman-level expertise with *Graphics* APIs
- Optimizations obscure, hidden, and a moving target
- At the mercy of video game market
- Lots of restrictions in execution model!
  - Output driven model – no random write, no accumulate
    (Scatter/Gather/Reduction costly & complicated)
  - Dynamic branching heavily restricted and costly
  - Program length limits
  - Precision restrictions
  - In-place operations not supported
  - ...
GPU Programming Approaches

- Low level, single-use programming
- Reusable kernels
- New Languages
  - BrookGPU, Sh, Gumdrop
- Domain specific libraries
  - Quick insertion, appropriate abstractions, stable APIs
  - GPU-FFTW, others…
  - GPU-VSIPL++
VSIP - Vector Signal Image Processing Library

- Portable API for linear algebra, image & signal processing
- Originally sponsored by DARPA in mid ’90s
- Targeted embedded processors – portability primary aim
- Open standard, Forum-based
- Initial API approved April 2000

- Functional coverage
  - Vector, Matrix, Tensor
  - Basic math operations, linear algebra, solvers, FFT, FIR/IIR, bookkeeping, etc
High Performance Embedded Computing Software Initiative

• Extend VSIPL and other industry standard APIs
• Develop a unified computation & communication framework
• Measurably improve embedded application development:
  • Portability: 3x
  • Productivity: 3x
  • Performance: 1.5x

• C++: VSIPL++ (approved August 2004)
• Data-Parallel: ||VSIPL++ (approved April 2006)
**VSIPL++ Highlights**

- Same functional coverage as C-VSIPL
- Templates, function overloading, operator overloading
  - Simplified API
  - Expression templates for loop fusion, improved performance
  - Smooth path to encapsulated data-parallel

```cpp
vsip_vadd_f
vsip_vadd_i
vsip_cvadd_f
vsip_rcvadd_f
vsip_madd_f
vsip_madd_i
vsip_rcmadd_f
vsip_cmadd_f

_f: one of 4+ floating point precisions
_i: one of 40+ integer precisions

add (v1, v2, result)

result = v1 + v2
```
VSIPL & GPUs

• VSIPL and GPUs are a natural match
  • Execution model similar
  • VSIPL functions map well to GPU capabilities
  • Data management models similar – explicit data movement prevents unnecessary transfers over slow bus

• VSIPL++ Even better – Loop fusion helps
  • Communication and per-loop FLOP costs higher on GPUs
  • Reduction of large temporaries
  • Reduction of number of loops
  • Reduction of texture accesses
GPU-VSIPL++ Implementation: Components

• VSIPL++ Reference Implementation (wrapper version)
  • Open source, thin wrapper over a C-VSIPL implementation
  • Developed by CodeSourcery, LLC - VSIPL++ functional prototype

• GPU-VSIPL (partial)
  • Provides backend to VSIPL++ Reference Implementation
  • Allows acceleration of C-VSIPL applications

• OpenGL, Cg
  • Graphics APIs and fragment programs
GPU-VSIPL++ Implementation: Methodology

- VSIPL Blocks ➔ OpenGL Textures
  - Behaviorally Analogous
  - Leverage drivers for memory management – heavily optimized

- Simple math operations ➔ single render operations
  - Example vsip_vsma_f fragment program:

```c
void main (float2 tc0 : TEXCOORD0,
           out float4 col : COLOR,
           uniform float4 beta,
           uniform samplerRECT A,
           uniform samplerRECT C)
{
    col = texRECT (A, tc0) * beta + texRECT(C, tc0);
}
```
GPU-VSIPL++ Implementation: Methodology

- More generalized operation is restricted
  - Scatters, gathers, reductions tricky
  - Random-write not possible directly
  - No ordering control/knowledge
  - In-place operations out-of-spec for OpenGL
- Various approaches used
  - Multipass for reductions
  - Dynamic fragment program generation
  - Vertex processors, fixed function elements (Not yet)
- Adjustments to VSIPL++ Reference Implementation
  - Lazy operators, additional GPU-VSIPL hooks
GPU-VSIPL++ Benchmarking

- Tested several VSIPL++ functions on 4 platform configurations

- Platforms:
  - Baseline: Athlon64 2.4GHz 1GB RAM Desktop PC
  - GPUs: ATI Radeon 1900XT, nVidia 7800GTX single/SLI

- Software:
  - VSIPL++ Functions: vsip_vadd_f, vsip_vsma_f, vsip_firflt_f
  - Implementations: GPU-VSIPL++, best-case pure software
  - Vector sizes 4->16M (4M ATI)
VSIP++ Sample: vsip_vsma_f benchmark

Vector<vsip_scalar_f> C (vector_size) = 0.2f;
Vector<vsip_scalar_f> B (vector_size) = 0.3f;
Vector<vsip_scalar_f> A (vector_size);

tic();
for (j=0; j<iterations; ++j)
{
    A = B * 3.4f + C;
}
toc();
Results: vsip_vadd_f

![Graph showing speedup for different vector sizes and hardware configurations. The graph compares ATI X1900XT, Nvidia 7800GTX, and Nvidia 7800GTX SLI. The y-axis represents speedup, and the x-axis represents vector size (4, 16, 64, 256, 1K, 4K, 256K, 1M, 4M, 16M). The data shows a significant speedup for Nvidia 7800GTX SLI, with a 20x increase at certain vector sizes.]}
Results: vsip_vsma_f

Vector Size

Speedup

100

10

1

0.1

0.1

0.01

0.01

0.001

4

16

64

256

1K

4K

16K

64K

256K

1M

4M

16M

20x
Results: vsip_firflt_f

<table>
<thead>
<tr>
<th>Vector Size</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>79x</td>
</tr>
<tr>
<td>64</td>
<td>48x</td>
</tr>
<tr>
<td>256</td>
<td>25x</td>
</tr>
<tr>
<td>1K</td>
<td></td>
</tr>
<tr>
<td>4K</td>
<td></td>
</tr>
<tr>
<td>16K</td>
<td></td>
</tr>
<tr>
<td>64K</td>
<td></td>
</tr>
<tr>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>1M</td>
<td></td>
</tr>
<tr>
<td>4M</td>
<td></td>
</tr>
<tr>
<td>16M</td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows the speedup for different vector sizes.
Summary

• So far: prototype - basic functionality & promising speed
  • GPUs excel on larger vectors
• Future Work:
  • More functions needed: support, math – core lite+
  • More optimization cases
  • Full compliance with all VSIPL conditions, edge cases, etc
  • Direct integration with Sourcery VSIPL++ - faster, parallel
• Helpful links
  • VSIPL: http://www.vsipl.org
  • HPEC-SI: http://www.hpec-si.org
  • GPGPU: http://www.gpgpu.org
  • Cg: http://developer.nvidia.com/cg
  • OpenGL: http://www.opengl.org