Stream-based concurrent computational models and programming tools

Leonel Sousa

with

Shinichi Yamagiwa
Outline

1. Many-core platforms based on GPU’s

1. GPGPU: Computation Models and Programming tools
   1. Stream based computing
   2. Massively parallelism based on Multithreading
   3. APIs and Programming tools

2. Caravela Project
   1. Flow-Model and Caravela Platform
   2. Caravela Tools for programming GPUs (locally and remotely)
   3. Optimizations for current GPUs/Systems
   4. Future Work
Graphics Processing Units

- Graphics Processing Units (GPUs)
  - Available in all computers
  - Unused high computational capacity
  - Manycore processing systems

GPGPU - General-Purpose computation on GPUs

- Usage of GPUs for GPGPU
  - Graphics APIs are not tuned for general-purpose applications
  - Programmer has to learn irrelevant graphics concepts
  - Data copy from main memory to video memory is slow
    - PCI-E system bus
GPU Structure

- Stream-based processing with four elements
  - Vertex processor: x, y, z, w
  - Pixel processor: operates on pixel data in a vector approach, issuing instructions to operate concurrently on the multiple color components of a pixel - R(ed), G(reen), B(lue) and A(lpha)
- Vertex and Pixel processors are programmable
  - DirectX assembly language and HLSL
  - OpenGL Shader Language (GLSL)
Texture mapping example

ps_2_0 ← DirectX assembly language
Pixel Shader Model 2.0

def c0, 0.5,0.5,0.5,0 ← α
def c1, 1,1,1,1

dcl_2d s0
dcl_2d s1

dcl t0.xy ← Coordinates of textures
dcl t1.xy

texld r2, t0, s0
texld r3, t1, s1

mov r5, c1
sub r5, r5, c0
mul r2, r2, r5
mad r4, r3, c0, r2

mov oC0, r4 ← Output of results

P' = Pa(1-α)+Pbα
Alpha blending
GPU Performance

- GPU drastically improves performance in the last 5 years
• **GPU supports general purpose processing (data-parallelism)**
  – with high number of arithmetic calculations per memory access

• **Examples** ([www.gpgpu.org](http://www.gpgpu.org))
  – Physics simulation
  – Signal processing
  – Computational geometry
  – Database management
  – Computational biology
  – Computational finance
  – Computer vision
  – …..
GPU architecture

GeForce 8800 [source: NVIDIA]
GPU architecture
Case study: GeForce 8800

330 Gflop/s (issue rate for MAC), 86.4 GB/s peak mem. bandwidth

- 128 stream processors: 8 clusters of 16 SPs
- SPs aren't vertex or pixel shaders: generalized floating-point processors capable of operating on vertices, pixels, or any data
  - most GPUs operate on pixel data in a way (R,G,B,A) but the G80's SP is scalar
- SPs are clocked at a relatively speedy 1.35GHz, while most of the rest of the chip is clocked independently at 575MHz
  - GeForce 8800: a tremendous amount of raw floating-point processing power

- The cores in a cluster share:
  - local memory (L1)
  - banks of specialized hardware (TF) for implementing texture fetch operations

- High performance access to the frame buffer memory (FB)
  - to store both texture data and rendered images
Computation Models: Stream processing

- Input data is streamed in from one or more input arrays, processed by a **stream kernel**, and then streamed out to one or more output arrays.
- A **stream kernel** can be thought of as:
  - function that is applied in parallel to every element of one or more input arrays and produces one or more output arrays.
Applications can easily be limited by memory bandwidth
  - Restrictions: memory accesses oriented to pixel processing
  - Only gather: can read data from other pixels

- No scatter: (Can only write to one pixel)
**SPMD + SIMD Model**
- Data-parallel portions of an application are executed as **kernels** which run in parallel on many threads

**A kernel is executed as a grid of thread blocks**
- A thread block is a batch of threads that can cooperate with each other through shared memory

**Two threads from two different blocks cannot cooperate**
### Computation Models: Multithreading

- **Massive parallelism for GPUs to hide memory access and pipeline latencies**
  - For instance, a single processing element in a GPU might run several threads at once and switch between them whenever a high-latency operation is encountered.

- **Read/write per-thread**
  - registers, local memory

- **Read/write per-block**
  - shared memory

- **Read/write per-grid**
  - global memory

---

**Grid**

<table>
<thead>
<tr>
<th>(0, 0)</th>
<th>(1, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shared Memory</strong></td>
<td><strong>Shared Memory</strong></td>
</tr>
<tr>
<td>Registers</td>
<td>Register</td>
</tr>
<tr>
<td>Thread (0, 0)</td>
<td>Thread (1, 0)</td>
</tr>
<tr>
<td>Local Memory</td>
<td>Local Memory</td>
</tr>
<tr>
<td>Thread (0, 0)</td>
<td>Thread (1, 0)</td>
</tr>
<tr>
<td>Local Memory</td>
<td>Local Memory</td>
</tr>
</tbody>
</table>

**Host**

- Global Memory
- Constant Memory
- Texture Memory

---

*University of Murcia*
• CUDA API is an extension to the C language
  – extensions to target portions of the code for execution on the device
  – a runtime library split into
    • a common component providing built-in vector types and a subset of the C runtime library supported in both host and device codes
    • A host component to control and access one or more devices from the host
    • A device component providing device-specific functions
APIs and Programming tools: Heterogeneous Multi-core Parallel Programming (HMPP)

• The GPU is always viewed as a computing device that:
  – is a coprocessor to the CPU or host
  – has its own DRAM (device memory)

• Approach similar to OpenMP, but designed to handle hardware accelerators
  – application source code portable
    • sequential binary -> traditional compiler

• CAPS HMPP is:
  – a set of compiler directives and runtime software for multicore programming in C
Caravela: Motivation

- A new execution model for local and remote computation is required
- Stream computing is the expected for the next high performance computing method
- GPU never touches resources on host machine using stream-based computation, so security can be guaranteed

Stream-based computation on GPU can be applied to distributed computing

Caravela: A new platform for distributed computing
Caravela Project: Project Roadmap

Basic Concept

Flow-model

Distributed Computing

Caravela Platform

GPU

Implementation dependent

Emulator

Dedicated Hardware

Algorithm development for different applications

Task Scheduling in Distributed environment

Pipeline-model

Meta-pipeline

Performance Optimization

Execution Optimization on GPU

Stream-based concurrent computational models and programming tools

University of Murcia
Caravela Platform: Flow-model

- Memory effect by introducing feedback
- Program does not touch other resources beyond I/O streams
- Flow-model encapsulates a task object
- Flow-model can be fetched from remote site.

Caravela provides a set of tools for executing a flow-model unit.

FlowModelCreator and Caravela Library
Caravela Platform: Runtime Environment

• Resource definition in Caravela library
  – **Machine**: has Adapter(s)
  – **Adapter**: has Shader(s)
  – **Shader**: Pixel Processor(s)

• Programming steps in application
  1. Acquire shaders
  2. Define flow-models
  3. Map flow-models to shaders
  4. Setup input streams
  5. Fire flow-models
  6. Get output data streams
Remote execution runtime supports:
  - **Worker server**: executes flow-models.
  - **Broker server**: maintains routing information to worker servers.
Caravela Platform: Caravela library

- Initialization and Finalization
  CARAVELA_Initialize(RUNTIME), CARAVELA_Finalize(RUNTIME)
- Flow-model creation
  flow-model  CARAVELA_CreateFlowModelFromFile(filename)
- Machine creation
  machine  CARAVELA_CreateMachine(machine_type)
- Getting Shader
  shader  CARAVELA_QueryShader(machine)
- Mapping Flow-model into Shader
  fuse  CARAVELA_MapFlowModelIntoShader(shader, flow-model)
- Initialization for input data stream
  input data stream buffer  CARAVELA_GetInputData(flow-model)
- Execution of Flow-model
  CARAVELA_FireFlowModel(fuse)
- Getting output data stream
  output data stream buffer  CARAVELA_GetOutputData()

machine_type is “REMOTE” for remote execution.
Caravela Platform: 1D FIR Filter

\[ y_n = \sum_{i=0}^{15} b_i \cdot x_{n-i} \]

```c
void main()
{
    int i, j;
    float inv = 1.0/Const4.x;
    vec4 res = vec4(0.0, 0.0, 0.0, 0.0);

    vec2 coord = gl_TexCoord[0].xy;
    vec4 data0 = texture2D(CaravelaTex0, coord);
    coord.x += inv;
    vec4 data1 = texture2D(CaravelaTex0, coord);
    coord.x += inv;
    vec4 data2 = texture2D(CaravelaTex0, coord);

    // for x value
    for( j=0; j<4; j++ )
    {
        res.x += data0[j] * Const0[j];
        res.x += data1[j] * Const1[j];
    }

    // for y value
    for( j=1; j<4; j++ )
    {
        res.y += data0[j] * Const0[j-1];
        res.y += data1[0] * Const0[3];
        for( j=1; j<4; j++ )
        {
            res.y += data1[j] * Const1[j-1];
            res.y += data2[0] * Const1[3];
        }
    }
    gl_FragData[0] = res;
}
```

OpenGL (GLSL)

```c
void main( in float2 t0 : TEXCOORD0, out float4 oC0 : COLOR0 )
{
    int j;
    float inv = 1.0/Const4.x;
    float4 res = 0;
    float2 coord = t0;

    float4 data0 = tex2D(CaravelaTex0, coord);
    coord.x += inv;
    float4 data1 = tex2D(CaravelaTex0, coord);
    coord.x += inv;
    float4 data2 = tex2D(CaravelaTex0, coord);

    // for x value
    for( j=0; j<4; j++ )
    {
        res.x += data0[j] * taps[j][0];
        res.x += data1[j] * taps[j][1];
    }

    // for y value
    for( j=1; j<4; j++ )
    {
        res.y += data0[j] * taps[j-1][0];
        res.y += data1[0] * taps[3][0];
        for( j=1; j<4; j++ )
        {
            res.y += data1[j] * taps[j-1][1];
            res.y += data2[0] * taps[3][1];
        }
    }
    oC0 = res;
}
```

DirectX (HLSL)
## Caravela Platform: Experimental Results

<table>
<thead>
<tr>
<th></th>
<th>Machine1</th>
<th>Machine2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>AMD Opteron 2GHz 2GB DDR400</td>
<td>Intel CoreDuo 1.66GHz 1GB DDR2</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>NVIDIA GeForce 7300GS 256MB DDR</td>
<td>NVIDIA GeForce Go 7400 128MB DDR2</td>
</tr>
</tbody>
</table>

University of Murcia
Caravela Platform: Experimental Results

- 1D FIR Filter
  - Input: 1M samples × 30 iterations
- S = 4-10 times regarding to CPU

Caravela platform speeds up local processing
Local Optimizations: Recursive processing

- Recursive processing with flow-model
  - Output streams must be copied to input streams
    → performance degrades due to the copy overhead
- Example: IIR Filter
  - Output “y” is feed-forwarded to input recursively.
    \[ y_n = \sum_{i=0}^{7} b_i \cdot x_{n-i} + \sum_{k=1}^{8} b_k \cdot y_{n-k} \]

Execution time (sec)
Local optimizations: Swap mechanism

- Swap mechanism: Optimization for recursive I/O
  - \( \text{Pair} \leftarrow \text{CARAVELA\_CreateSwapIoPair}(\text{input\_index}, \text{output\_index}) \)
  - \( \text{CARAVELA\_SwapFlowmodelIo}(\text{Pair}) \)

```
Pair = CARAVELA\_CreateSwapIoPair(0,1)
While(…){
    Fire flow-model…
    CARAVELA\_SwapFlowmodelIo(pair)
}
```
Local optimizations: Implementation of Swap mechanism

Conventional method

Swap mechanism

Copy method (DirectX)
Output stream copied
VRAM → CPU memory and
CPU memory → VRAM

Swap method (OpenGL)
Exchanges pointers of I/O
buffers in the GPU side.
Local optimizations: Swap mechanism

- OpenGL is used as the graphics runtime:
  - CARAVELA_SwapFlowmodelIO() for swap mechanism
- Swap:

  Improves performance 55-60%

Swap mechanism is an effective optimization technique.
Local optimizations: Remap method

- I/O overhead of GPGPU application
  - Copy operation among CPU memory-VRAM
  - Overhead in GPU at writing output stream to VRAM
  - Overhead in Pixel processor at reading textures

Smaller texture size may result in better performance.
Local optimizations: Remap method

- Iterating with 3000x3000 texture input and applying Swap mechanism
  - Spot depending on the number of iterations of Swap mechanism
  - GeForce7300: 1500 iterations
  - GeForce7900: 2000 iterations

Swap iteration should be reset at the spot!
Local optimizations: Remap method

- For the applications which calculation size decreases,
  - Flow-model should be mapped again after the input texture sizes are reduced
  - Applying a threshold number of iterations for Swap, flow-model is mapped again at the spot
Local optimizations: LU decomposition

(A) Normalization of diagonal elements
(B) Orthogonalization
(C) Normalization

Elements previously calculated are forwarded to the output data stream without any calculation

University of Murcia

Stream-based concurrent computational models and programming tools

27-05-2008
Caravela Platform: Applying remap method to LU decomposition

- **GeForce7300**
  - Remap flow-model every 1500 Swap iterations
- **GeForce7900**
  - Remap flow-model every 2000 Swap iterations

- Reduction of 80% in execution time
  - Remap method further improves performance in the top of the swap mechanism
Remote execution: Meta-pipeline

- Executing the flow-model in a remote machine:
  - Sending input data to the remote machine,
  - receiving output data from the remote machine,
  - scheduling the execution
Remote execution: Pipeline model

- I/O ports of the Pipeline-model
  - ENTRANCE port
  - EXIT port
  - INTERMEDIATE port
- When all input streams are ready, flow-model is executed
- Deadlock might occur if feedback edges exist
  - INITONCE port
Remote execution: Extension of Caravela library

- Extended functions for Caravela library
  - CARAVELA_CreatePipeline()
  - CARAVELA_AddShaderToPipeline()
  - CARAVELA_AttachFlowModelToShader()
  - CARAVELA_ConnectIO()
  - CARAVELA_Specify[InitOnce | Exit | Intermediate]Port()
  - CARAVELA_ImplementPipelineModel()
    - CARAVELA_SendInputDataToPipeline()
    - CARAVELA_ReceiveOutputDataFromPipeline()

During local execution: it promotes pipeline execution.
During remote execution: communication with worker servers.
Remote execution: 2D DWT

- 2D Discrete Wavelet Transform
  - Image compression (JPEG2000), denoising, edge detection, enlarge...

\[
\begin{align*}
LL_n &= \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} LL_{n-1}(2i + k, 2j + m)l(m)l(k) \\
HL_n &= \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} LL_{n-1}(2i + k, 2j + m)h(m)l(k) \\
LH_n &= \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} LL_{n-1}(2i + k, 2j + m)h(m)h(k) \\
HH_n &= \sum_{k=0}^{K-1} \sum_{m=0}^{M-1} LL_{n-1}(2i + k, 2j + m)h(m)h(k)
\end{align*}
\]

2 decomposition level
Remote execution: 2D DWT

S = LL₀

Level 1 Flow-model

LL₁ → HL₁, LH₁, HH₁

Level 2 Flow-model

LL₂ → HL₂, LH₂, HH₂

Level 3 Flow-model

LL₃ → HL₃, LH₃, HH₃

Level N Flow-model

LLₙ → HLₙ, LHₙ, HHₙ

void main()
{
    float delta = 1/NUMDATA;
    vec4 tmp, tmp0, tmp1, result;
    vec2 coord = gl_TexCoord[0].xy;
    vec2 caux; int i;
    coord += coord;
    caux = coord;
    // horizontal direction
    for (i=0; i<4; i++, coord.y += delta){
        tmp.x = texture2D(CaravelaTex0, coord).x;
        coord.x += delta;
        tmp.y = texture2D(CaravelaTex0, coord).x;
        coord.x += delta;
        tmp.z = texture2D(CaravelaTex0, coord).x;
        coord.x += delta;
        tmp.w = texture2D(CaravelaTex0, coord).x;
        coord.x = caux.x;
        tmp0[i] += dot(tmp, const0);
        tmp1[i] += dot(tmp, const1);
    }
    // vertical direction
    result.x = dot(tmp0, const0);
    result.y = dot(tmp0, const1);
    result.z = dot(tmp1, const0);
    result.w = dot(tmp1, const1);
    // LL sub-band stream
    gl_FragData[0] = result;
    // LH, HL and HH sub-bands stream
    gl_FragData[1] = result;
}
Remote execution: PipelineModelCreator tool
Future Work

- MPI + flow-model = CaravelaMPI
- Caravela platform operated in command line mode (operating system)
- Attach other hardware platforms to the Caravela platform (co-processors on FPGAs, …)
- Test Meta-Pipeline with large real problems
  - Japan-Cyprus-Portugal
Publications

• Papers
  5. Shinichi Yamagiwa and Diogo Ricardo Cardoso Antao and Leonel Sousa, Design and Implementation of a Graphical User Interface for Stream-based Distributed Computing, the IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2008), Feb. 2008

• Book chapter

• Patent
  1. “Program execution method applied to data streaming in distributed heterogeneous computing environment”, Portuguese national patent
For more detailed information, please visit: http://www.caravela-gpu.org