Extending a Stream Programming Paradigm to Hardware Accelerator Platforms

Sek Chai  
Motorola  
Schaumburg, IL  
sek.chai@motorola.com

Abelardo López-Lagunas  
Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Toluca, México  
abelardo.lopez@itesm.mx

Nikos Bellas  
University of Thessaly, Greece  
bellas@inf.uth.gr
Contributors

Nikos Bellas
Sek Chai
Silviu Chiricescu
Malcolm Dwyer
Ray Essick
Dan Linzmeier
Abelardo Lopez-Lagunas

Brian Lucas
Phil May
Kent Moat
Jim Norris
Michael Schuette
Ali Saidi
Agenda

• Separation of concerns between computation and memory access
• RSVP and Proteus streaming accelerators
• Results and summary
Hardware Acceleration

In embedded applications:
- Loops take most of the execution time
- Access patterns are usually uniform and static

- Use hardware acceleration for compute intensive loops
- Keep single processor programming flow

Identify Loops By Profiling C Code

Host Calls

Host Cycles

Loop 1

Loop 2

Loop 3

Rcode loops as for acceleration
Stream Processing

Example processing chain as stream kernels

**Characteristics** | **Architecture implications**
---|---
Computation is repetitive on localized data regions | Explicit parallelism. Overlap data movements to accelerator.
Kernels are independent and self contained | Low temporal locality for data. Traditional caches are not effective.
Large (possibly infinite) amount of data | Memory access patterns are deterministic.
Limited lifetime of datum |
Kernel Parallelism

Kernels read and write streams (no global variables)

Optimal exploitation of Instruction & Data Level parallelism via loop unrolling modulo scheduling, etc

Lack of dynamic memory schedule allows the compiler to produce optimal code for the given hardware resources

Chain *functional units* based on stream consumption and production rate
Communication Locality

Inter-kernel communication through a producer-consumer model.

System-level task scheduling is easier because the communications are explicit in the program.

Memory wall problem can be significantly reduced.

Chain *hardware accelerators* based on stream consumption and production rate.

Needed if Kernel C reads data in a different rate or pattern.
Decoupled memory accesses and computations:
• enable better optimization of hardware
• simplify compiler tasks
**Hardware Acceleration of Stream Kernels**

*Stream Descriptors* help define memory subsystem structure. *Stream Kernels* define hardware accelerators.
Streaming Accelerator Template

System Bus

Arbiter & Bridge

Input Stream

Bus Line Buffer

Stream Buffer

Data alignment

To other Input Stream

To Output Stream

Stream Unit

Addr Queue

AGU

Streaming Data

Multiplexer Tree

FU

FU

Acc

Line buffers

Control Registers

Reg

Reg

Constants

Data Path

Programmer Visible Architectural Elements

- Host Cycles
- Loop 1
- Loop 2
- Accelerator Cycles + accelerator calls
- ARM Cycles
- Accelerator calls
- Memory Subsystem
- Input SU
- Output SU
- Accumulators
- Control
- Scheduler
- Functional Units
- Constants
- Interconnect

Identify Loops By Profiling C Code

Speedup

Identify Loops By Profiling C Code
Describing Computation

Components of sDFG (stream data flow graph)

- Nodes in graph represent computation (performed by functional units)
- Edges in graph represent data movement between functional units
- Multiple computation elements arranged across the data streams

Algorithm Dataflow
Datapath Construction

Resource constraints:
- 192 ALU bits,
- 128 MUL bits,
- 64 SHIFTER bits
- 128 NAMED REG bits
- 4, 32-bit INSTREAM PORTS
- 1, 32-bit OUSTREAM PORT

Map sDFG to functional units
Hardware mechanisms ensure that the prologue and the epilogue are executed correctly.
Describing Memory Access

A method to move data efficiently using known shape of data

For data prefetch, staging, and reuse

- Efficient data movement
- Utilize unused bandwidth
- Less sensitive to peak bandwidth

2-D Subarrays (row)
(SA, Stride, span, skip)
(4, 1, 4, 97)

Mapped to “stream unit” or smart DMA in hardware accelerators
Stream Unit

Keep bus busy with requests.

Prefetch data using Stream Descriptors.

Align and order stream elements for data path.

Larger queues and buffers allow more aggressive prefetching of stream elements.

Store bus transfers in transit.
Stream Unit

Stream Unit

Input Stream

System Bus

Arbiter & Bridge

Bus Line Buffer

Stream Buffer

Addr Queue

AGU

Data alignment

Stream shape & access pattern

Bus latency to memory

Required stream bandwidth

Stream shape & access pattern

Required stream bandwidth
Reconfigurable Streaming Vector Processor

A software programmable vector accelerator based on a “streaming dataflow” programming model

ARM946+RSVP™-I SoC

SoC (ARM946+RSVP) in 0.18 µm which contains 9.5M transistors in a 5.04 x 9.03 mm² die. Power consumption is 587mW (1.8V, 120MHz core, 60MHz bus).

Proteus Streaming Accelerator Design Flow
(FPGA)

1. FU Alloc
2. Modulo Scheduling
3. Build Data Path
4. Generate Verilog
5. 3rd party Synthesis & P&R

Streaming DFG + Stream Descriptors + Resource Constraints + System Constraints

FU instantiation + Iteration Interval Determination

vld, vadd
vmul
vsub, vshl

.v

3rd party Synthesis & P&R

Generate Verilog
Proteus Scheduler Features

- Do not require separate prolog and epilog code
- Can handle nested loop constructs without having to create different schedules for different parts of the nested loop construct
- Supports tightly coupled memories (e.g. doubled buffered LUT)
- Handle resources (FUs, fabric, queues) for different application optimization
Hardware File Template

Proteus HW File features

- Structured intermediate format (IF) for streaming data path
- Facilitates debugging with consistent naming/labels
- Describes resources such as FUs, fabric, queues, LUTs.
- Scalable with new ISA updates
Smart Cameras read license plates and compare against database.


**Automotive**

**Smart Cameras** detect and track lanes. Warn drivers on hazards and unexpected lane departures.

- **Scene Calibration**
  - Capture Images
  - Convert to gray scale
  - Noise Filter
  - Perspective Mapping

- **Scene Analysis** (to find lines on the road)
  - Edge Detection
  - Line Detection (Hough Transform)

- **Lane Extraction** (to find lane and road markings)
  - Lane Extraction
  - Road modeling
  - Driver Warning

---

Lens correction

Addresses issues related to filtering and lens-distortion correction for visual communications

Stereoscopic geometry of Fisheye Lenses

• Fisheye lenses refract the incident light rays towards the central perspective point

\[
\alpha_1 = \beta_1 \quad \alpha_2 = \beta_2
\]

Rectilinear Projection

\[
\frac{\alpha_i}{d_i} = \frac{\alpha_i}{d_i}
\]

Fisheye Projection
Algorithmic steps (A)

From rectilinear to fisheye space coordinates (inverse mapping)

\[
\begin{bmatrix}
X_c \\
Y_c \\
Z_c
\end{bmatrix} = \begin{bmatrix}
r_{11} & r_{12} & r_{13} \\
r_{21} & r_{22} & r_{23} \\
r_{31} & r_{32} & r_{33}
\end{bmatrix}
\begin{bmatrix}
i \\
j \\
1
\end{bmatrix}
\]

\[
x = \frac{2R}{\pi} a \tan\left( \frac{\sqrt{(Xc)^2 + (Yc)^2}}{Zc} \right) + d_x + x_h
\]

\[
y = \frac{2R}{\pi} a \tan\left( \frac{\sqrt{(Xc)^2 + (Yc)^2}}{Zc} \right) + d_y + y_h
\]
Algorithmic steps (B)

- Approximate pixel values in fractional positions in *Fisheye space*
- Complex memory access pattern due to non-linear projection trajectory
Algorithmic steps (B)

Bicubic interpolation

- **Bicubic interpolation** uses a 4x4 window of pixels to approximate intermediate points
- Interpolation weights depend on the relative position of the intermediate point
Architectural Optimization Strategies

Use **block tiling** to correct a block of pixels at a time.
Parallelism extracted by Proteus

- Instruction Level Parallelism (ILP) naturally expressed in streaming
  - About 400 executed instructions/cycle
  - Modulo scheduler with II=2
- Task level parallelism
  - Concurrent
  - Pipeline
### Hardware details
Virtex-4, LX-80 FPGA

<table>
<thead>
<tr>
<th><strong>SPEED</strong></th>
<th>Clock freq.</th>
<th>62.5 MHz (single clock)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>22 frames/sec</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>AREA</strong></th>
<th>Logic Slices</th>
<th>11082 (30%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP48 units</td>
<td>71 (88%)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BRAMs</th>
<th>109 (54.5%)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>BRAM types (number per type)</th>
<th>4096x8 (15)</th>
<th>13728x50 (1)</th>
<th>256x16, 512x7, 256x17, 256x7, 256x17, 256x7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>6864 x 8 (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6864 x 16 (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3432 x 8 (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3432 x 16 (2)</td>
</tr>
</tbody>
</table>
Performance

Bounded by available FPGA SRAM

(unpublished results)
Performance

Module Runtime Breakdown

- Inverse Mapping
- Bicubic Interpolation
- Low Pass Filter

Floating point intensive

(unpublished results)
Research summary

Separation of concerns
- Memory access patterns are defined explicitly by programmer for RSVP and Proteus

Extend stream descriptors

Plan to open source Proteus tool