R-Stream: Enabling efficient development of portable, high-performance, parallel applications

Peter Mattson, Allen Leung, Ken Mackenzie,
Eric Schweitz, Peter Szilagi, Richard Lethin
{mattson, leunga, kenmac, schweitz, szilagyi,
lethin @ reservoir.com}

R-Stream Compiler

Reservoir Labs Inc.

Goals and Benefits

Portable, consistently high-performance compiler for streaming applications on morphable architectures

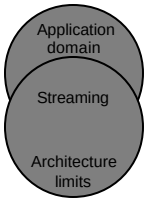
Relative to fixed-function hardware, R-Stream and a morphable architecture yields:

- Cost savings
- Shorter time to deployment
- Multipurpose hardware
- More advanced algorithms
- Dynamic mission changes
- In-mission adaptation
- Multi-sensor
- Bug fixes and upgrades less costly

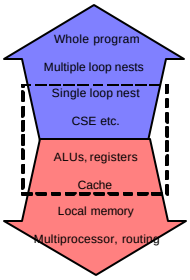
Streaming Compilation

- **Streaming is a *pattern* in efficient implementations of computation- and data-intensive applications**
  - Three key characteristics:
    - Processing is largely parallel
    - Data access patterns are apparent
    - Control is high-level, steady, simple
  - Caused by intersection of application domain and architecture limits:

	Applications process, simulate, or render physical systems	Architectural limits
Processing is largely parallel	*Independence of spatially and/or temporally distributed data*	*VLIW, SIMD, multiprocessor demand for parallelism*
Data access patterns are apparent	*Continuous, N-dimensional spatial/temporal data*	*Small local memories*
Control is high-level, steady, simple	*Few tasks per application, few unpredictable events*	*Handling unpredictability in hardware is expensive*



- **Streaming languages and architectures designed for this pattern are emerging:**
  - Streaming languages enforce the pattern, making parallelism and data-flow apparent
  - Streaming architectures provide maximum parallel resources and minimize control overhead by exposing resources to the compiler
- **R-Stream uses these to expand the scope of optimization and resource choreography**
  - Streaming languages avoid limits on compiler analysis (e.g. aliasing), enabling top-down optimizations
  - Streaming architectures allow the compiler to lay out all computation, data, and communication



Implementation Plan

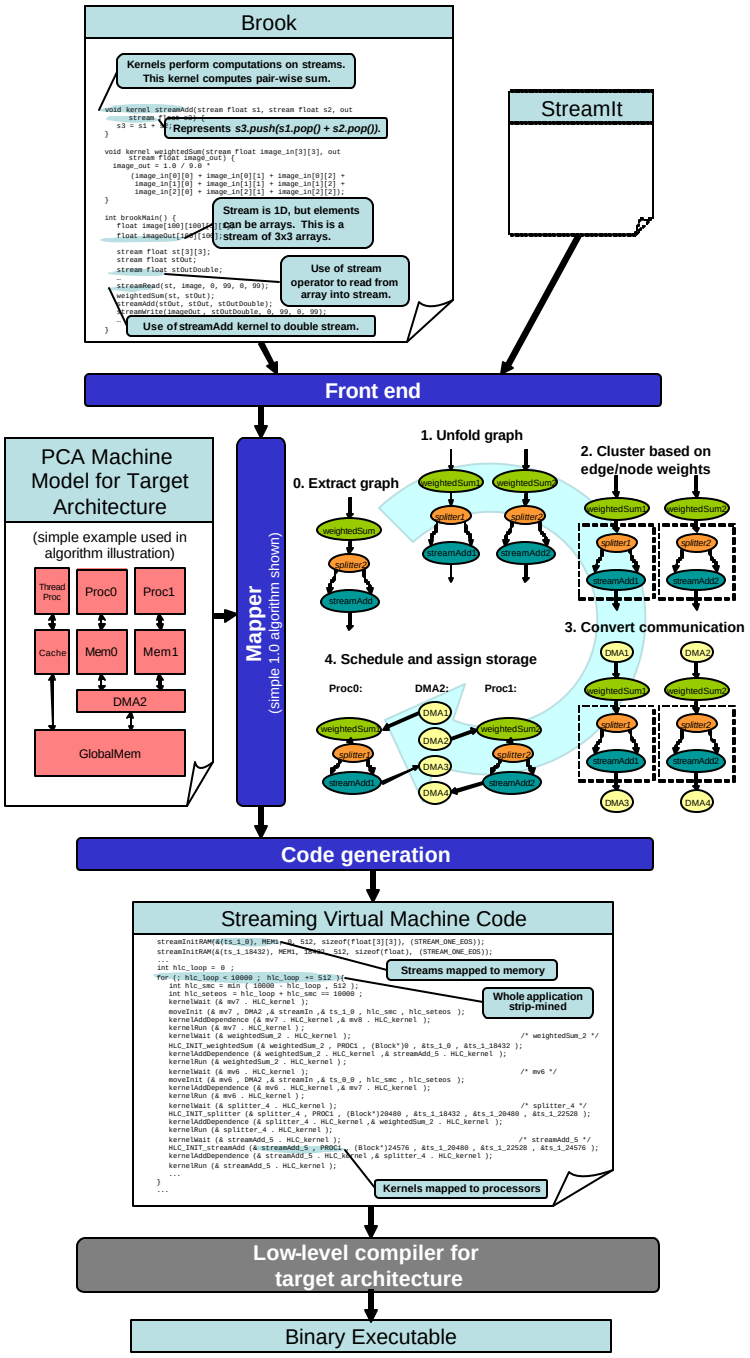
- Spiral development
  - Helps to ensure eventual results
  - Enables early usage by partners
- Three major releases:
  - Release 1.0: Functionality, but no performance optimization
  - Release 2.0: Common-case, phase-ordered performance optimization, static morphing
  - Release 3.0: General, unified performance optimization, dynamic morphing

Year	03			04				05			
Quarter	2	3	4	1	2	3	4	1	2	3	4
Release			1.0			2.0			3.0		
Front-end											
Streaming IR											
Mapping											
Code generation											

Non-existent
Discardable draft
Draft
Final

DARPA

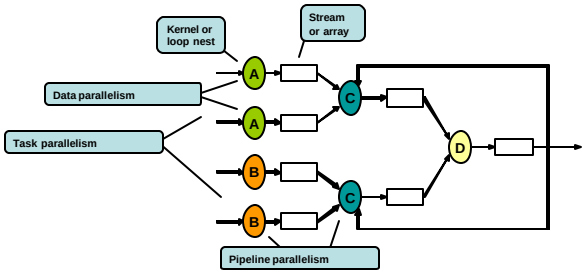
Compiler Flow



Technologies

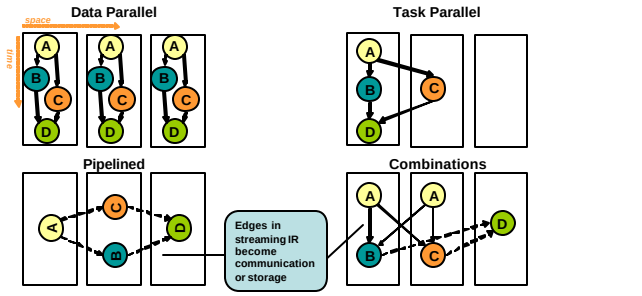
Streaming Intermediate Representation (IR)

- Represents an application within the compiler as a series of kernels or loop nests connected by streams or arrays
- Enables the compiler to directly analyze and optimize streaming applications
- Makes task, pipeline, and data parallelism readily apparent:



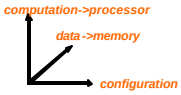
Top-down, unified, streaming IR mapping

- Mapping will include top-down optimizations, such as software pipelining an entire application to cover communication latency
- Mapping of computation, data, and communication layout will be performed by a single, unified pass for greater efficiency
- Mapping will exploit all degrees of parallelism exposed by the streaming IR:



Compiler driven architecture morphing

- Morphable architectures reconfigure high-level resources such as processors, memories, and networks to meet application demands
- Compiler must choose a configuration and map to that configuration – this expands the compiler mapping space:



- Compiler can employ one or multiple configurations:

1. *Static Morphing*: pick one configuration and mapping for whole application
2. *Dynamic Morphing*: pick multiple configurations and mappings for different parts of the application; orchestrate morphing between them