Streaming Virtual Machine API

MIT
Stanford
Reservoir Labs
What is the SVM API?

• A **Virtual Machine** is a description of a specific architecture using a common set of architectural primitives

• **Stream Virtual Machine API** is a set of constructs used to express the streaming portion of an application mapped to a specific VM
Goals for SVM API

1. Express high performance mappings for multiple applications and architectures
2. Separate computation- and data-intensive code from control code
   - Configure/build hardware well-suited to each role
3. Explicitly manage memory for data-intensive code
   - Use DMA instead of caches
4. Represent streaming nature of application
   - Explore hardware support for streaming
5. Easy to translate by low-level compiler
6. Human comprehensible and writable
SVM API Constructs

• **Kernel**
  – Computation- and data- intensive function executed on a single stream processor (one at a time)
  – Generated by high-level compiler, may not correspond to a “conceptual” kernel

• **Stream and Block**
  – Data stream/block operated on by kernel, bound to a specific resource

• **Graph**
  – Set of kernels which concurrently read/write streams in a synchronized fashion
  – Defined and controlled by a thread
Kernels

• Each kind of kernel is a class that extends Kernel:

```cpp
class Amplifier : public Kernel {
    ...
    IStream input1;
    OStream output1;
    work() {
        output1.push(input1.pop() * gain);
    }
}
```

• Each instance of a kernel is like a remote function call on specific processor:

```cpp
Amplifier amp1(PROC0, scratch, stream1, stream2, amp1Gain);
```
Streams and Blocks

- Kernels operate on Streams and Blocks

- A Stream is a queue of records buffered in memory or registers:
  - `Stream<type> s1(MEMORY1, 0x0, 8, 4)`
  - Push, pop, and peek
  - Capacity, `length`, `totalLength`

- A Block is an array of records in a specific memory or registers:
  - `Block<type> b1(MEMORY1, 0x0, 8)`
  - Read, write
  - Capacity
Graphs

- Set of kernels which concurrently read/write streams in a synchronized fashion
- Threads control graphs using run() and wait()
Pre-defined kernels for data movement

- Copy(PROC, srcStream, destStream, length)
- StridedGather(PROC, srcBlock, destStream, stride, itemsPerChunk, length)
- IndexedGather(PROC, srcBlock, indexStream, destStream, itemsPerChunk, length)
- Send(PROC, srcStream, connection, length)
- (Also StridedScatter, IndexedGather, Receive)
SVM Mapping Example

• Application:

• Architectures:
Mappings with Space-Multiplexing

• Efficient for kernels with well-matched data rates
Mapping with Time-Multiplexing

- Efficient for kernels with different and/or dynamic data rates
- Required for kernels > number of processors
Combination Mapping (reality)
Kernel Details

• Status:
  – UNSTARTED → WAITING → RUNNING → FINISHED
     ↑     ↓
     SUSPENDED

• Fields:
  – Scalar values, small arrays
  – Public fields read/write by thread when not WAITING/RUNNING

• Control methods:
  – addDependence(Kernel& dependence)
    • Dependence must be FINISHED before becomes RUNNING
  – terminate()
    • Best-effort attempt to stop execution, set status to FINISHED

• Computation methods:
  – work(), prework(), postwork()
Thread/Graph/Kernel Interaction

Thread:

... 
graph.run() {
    with all UNSTARTED kernels
    {
        send public fields
    }
}
...

Kernel: (bold = by compiler/user)

receive public fields
status = WAITING
wait for dependences to be cleared
status = RUNNING
newStatus = prework();

While (newStatus != FINISHED) {
    newStatus = work();
}

postwork();
status = FINISHED
clear dependences on this kernel
Thread:

```java
... 
graph.run() {
    with all UNSTARTED kernels
    {
        send public fields
    }
}
... 
graph.wait() {
    with all FINISHED kernels
    {
        receive public fields
    }
}
... 
```

Kernel:

```java
receive public fields
status = WAITING
wait for dependences to be cleared
status = RUNNING
newStatus = prework();

While (newStatus != FINISHED) {
    newStatus = work();
}
postwork();
status = FINISHED
clear dependences on this kernel
send public fields
```
Thread/Graph/Kernel Interaction

Thread:

graph.run()
...

Kernel:

receive public fields
status = WAITING
wait for all dependences to be cleared
status = RUNNING
newStatus = prework();

While (newStatus != FINISHED) {

newStatus = work();
}

postwork();
status = FINISHED
clear dependences on this kernel
send public fields

graph.wait()
### Thread/Graph/Kernel Interaction

**Thread:**

- `graph.run()`
- ...
- `graph.wait()` {
  - with all FINISHED or SUSPENDED kernels
    - receive public fields
  }
- ...
- `graph.run()` {
  - with all UNSTARTED or SUSPENDED kernels
    - send public fields
  }
- ...
- `graph.wait()`

**Kernel:**

receive public fields

status = WAITING

wait for all dependences to be cleared

status = RUNNING

newStatus = prework();

While (newStatus != FINISHED) {
  If (newStatus == SUSPENDED) {
    status = SUSPENDED
    send public fields
    receive public fields
    status = RUNNING
  }
  newStatus = work();
}

postwork();

status = FINISHED

clear dependences on this kernel

send public fields
Graph/Kernel/Stream Interaction

- Each kernel instance can only be in one graph.
- Each stream can have at most one writer and one reader at a time, must be in same graph.
- Push() to a “full” stream stalls if it has unFINISHED reader in graph.
- Pop() from an “empty” stream stalls if it has unFINISHED writer in graph.
- canPush(N)/canPop(N) return true if push/pop N items will not stall forever.
High-level / low-level compiler for SVM

• High-level compiler
  – Extracts/merges/splits kernels, streams, blocks, graphs
  – Maps kernels to processors
  – Maps streams and blocks to memory
  – Performs global dependency analysis and inserts addDependency() and wait() as needed

• Low-level compiler
  – Compiles kernels
  – Translates SVM constructs to hardware
Summary

• Stream Virtual Machine API is an API used to express the streaming portion of an application mapped to a VM

• API constructs (Kernels, Streams, Blocks, and Graphs) are well-defined

• Data movement uses pre-defined kernels

• Public fields/SUSPENDED for thread-kernel communication

• SVM constructs express high-level compiler mapping, translated by low-level compiler
SVM Revision Process

• Participants:
  – Saman Amarasinghe, Bill Thies, Mark Horowitz, Lance Hammond, Francois Labonte, Rich Lethin, Eric Schweitz, Charlie Garrett, Mike Vahey, Mike Dahlin, interim meeting participants

• Meeting at MIT
• Interim Morphware meeting
• Numerous telephone calls
• 100+ emails
• 8 drafts