SVM API

Francois Labonte

Modifications of the meeting in Blue
Single Node Big Picture

Scalar Unit
Runs Threads

Stream Instructions

Shared Registers
- Parameter Reg
- Stream SRF Reg
- Stream Mem Reg

Stream Unit
- Stream Engine (clusters)
  Executes Kernels
- SRF
  Working Set Memory
- Stream Memory Unit
  Memory Transfers to SRF
SVM Programs

Control Thread
C thread with
• Special Registers
• Special Functions
• Call Stream Instructions on Stream Unit

Kernel Foobar
Kernel Foo
Similar to KernelC with cluster abstraction
Deals with 1D streams
Control Thread – Special Functions

1. Creating Parallel Control Threads
   When multiple nodes are working on a subset of the problem doing reductions across these nodes

   \[
   \text{mythreadID} = \text{SpawnParallelCThreads}(8);
   \]

   Here if the control thread was running on node \(x\), copies of the control thread have been created on nodes \(x+1\) to \(x+7\) each returning their thread ID
Control Thread – Stream Instructions

• The Control Thread launches Stream Instructions executed on Stream Unit (StreamLoad, StreamStore, KernelStart, …).

• It needs to specify the dependencies of these instructions on each other.

• It also needs to synchronize on the completion of these operations (ex: expecting a reduction value).
Control Thread – Stream Instructions

- Each Stream Instruction launched returns a fence ID.
- The dependencies between Stream Instructions are specified with a list of fence ID (which need to complete before it can be executed)
- The Control Thread can query (non-blocking) or sync (blocking) on these fences too.

```c
fence[87] = StreamLoad (StreamMemRegA, StreamSRFRegA, Fence[86], Fence[85]);
fence[88] = KernelStart (KernelPtrFoo, StreamSRFRegA, StreamSRFRegB, fence[87]);

if (query(fence[87]))
    // Stream Load has completed already

synch(fence[88]); // Will only return once KernelStart has completed
```
Control Thread – Shared Registers

• The Shared Registers are used to specify arguments being reused by stream instructions (closer to machine working)

• Ex: Stream Memory Register describes how a stream is laid out in main memory, used by Streamloads and Streamstores
Control Thread – Stream Memory Register

```c
#define STREAM_MEM_RF 32 // could be a SVM parameter, not really critical
#define STRIDED_STREAM 0
#define INDEXED_STREAM 1

struct StreamMemReg {
    int stream_access_mode; // STRIDED_STREAM or INDEXED_STREAM
    int stream_stride_or_index; // the stride or which stream is indexed
    int stream_ptr; // address at which the stream access is to start.
    int stream_record_size; // Size in Words of records
    int stream_length; // Number of records
};

StreamMemReg StreamMemRF[STREAM_MEM_RF];

• StreamLoads and StreamStore refer to a stream described in the StreamMem Register file
```
Control Thread – Stream SRF Register

#define STREAM_SRF_RF 32 // could be a SVM parameter, not really critical

struct StreamSRFReg {
    int stream_ptr;    // address at which the stream starts in SRF.
    int stream_record_size; // Size in Words of records
    int stream_length;   // Number of records
};

StreamReg StreamSRF_RF[STREAM_SRF_RF];

- Used By StreamLoads, StreamStore and KernelStart to tell location of stream IOs of kernel.
This is where the scalar (non-stream) kernel parameters are passed.

Also where the reductions are written by the stream unit and read by the control thread.
Control Thread - Reductions

#define REDUCTION_ADD_INT 0;
#define REDUCTION_ADD_FP 1;
...

LocalReduction = KernelParamRf[11];

GlobalReduction = Reduce(REDUCTION_ADD_INT, localReduction);

- Doing reduction across multiple nodes (control threads).
- Leave some room for faster machine specific implementation of reduction
- Leave choice for communication pattern (radix of three)
- MODIFICATION: non-blocking reduction and reductions that return the value to a specific control thread only
Stream Instructions – StreamLoad

Fence[87] = StreamLoad (StreamMemRegA, StreamSRFRegA, Fence[86], Fence[85]);

• Arguments:
  – Stream MEM Register Index
  – Stream SRF Register Index
  – List of fence IDs which need to complete before it can execute

• Possible Dependencies
  – StreamStore, wait for SRF Space to be freed
  – KernelStart, Stream is being consume to free SRF space

• MODIFICATION: Specify the number of dependencies as an argument, for all stream instructions
Stream Instructions – StreamStore

Fence[87] = StreamStore (StreamMemRegA, StreamSRFRegA, Fence[86]);

• Arguments:
  – Stream MEM Register Index
  – Stream SRF Register Index
  – List of fence IDs which need to complete before it can execute

• Possible Dependencies
  – KernelStart, Stream is being produced
Fence[87] = KernelLoad (KernelStartMEMAdd, KernelStartKMEMAdd, length, Fence[85]);

• Arguments:
  – Kernel Start address in memory
  – Kernel Start address in the kernel memory
  – Kernel length
  – List of fence IDs which need to complete before it can execute

• Possible Dependencies
  – KernelStart, some kernel code won’t be needed, liberating some space in kernel memory

• MODIFICATION: no KernelLoad needed, just call the kernel
Fence[87] = KernelStart (KernelStartKMEMAdd, KernelParamStartIndex, StreamSRFRegA, StreamSRFRegB, StreamSRFRegC, Fence[85], Fence[86]);

• Arguments:
  – Kernel Start address in the kernel memory
  – The index of the first argument in the parameter register (they are contiguous, wraparound)
  – List of Indices in the Stream SRF register for the stream Inputs and Outputs
  – List of fence IDs which need to complete before it can execute

• Possible Dependencies
  – StreamLoad loading input stream
  – StreamStore, KernelStart liberating SRF space that will be required by output stream.
Stream Instructions – StreamBarrier

Fence[87] = StreamBarrier();

• All the previous instructions have to be completed before the next one can start.
SVM Kernels

- General C,
  - For, while, if –else
- Section before and after the stream element loop unlike Brook which only has the main loop
  - To handle corner cases like starting and stop special cases.
SVM Kernels – Stream IO

• Like KernelC, use the C++ syntax to get the next element from the stream and output the next element.

• Also use the conditional input and output stream construct.
  – Because the number of clusters is not defined, the order is not guaranteed when using conditional streams.
SVM Kernels - Stencils

- Stencils are regular accesses within a stream which can be propagated through the inter-cluster communication switch.
- SVM kernel supports only 1D streams.
- Within a 1D stream, can access relative elements anywhere:

```c
Int prev;
Prev << StreamA[INSTANCE-1]
```

- `INSTANCE` gives which element of the stream you are at.
- Makes C-syntax happier than negative indices.
SVM Kernels – Test Conditions

• Test Conditions:
  – Input Stream: End of Stream (EOS)
  – Output Stream: Full

• The end of stream is used to loop on an input stream until it is empty.

• The full output stream could for kernels with undefined number of outputs to suspend and restore.
SVM Kernels – Suspend and Restore

- Basically supported explicitly in a kernel
  - The kernel has an extra input stream to restore state
  - An extra output stream to save state
  - Within the kernel it can test to see if the kernel is full (maybe more flexibility would be useful, like number of words left)
  - The kernel would have an argument to know if it needs to restore
- The details need to be worked out, does the stream instructions need to be resend, or does the barrier has a special state for suspend
- Also what happens for Software pipelined kernels which need the stream data
Multinode Issues - Naming

• Global address space
  – Physical: Node + Offset
  – Virtual: specify where within the address the node number is inserted (not as flexible as segments) elements can be dispersed every power of 2 across nodes

• Implementation detail: For each stream in memory, specify where the node id is inserted in the virtual address
Multinode Issues - Coherence

• Basically all the coherence has to be explicit:
  – You could get coherence across everything with an expensive operation: MemFlush

• Can specify higher levels of tolerance through typedefs in the Control thread:
Multinode Issues – Stream Communications

• Desirable to send stream from one SRF to another without going to memory: SSS supports this?
• More Stream Instructions like StreamSend, StreamReceive
• Use MPI-like protocol
Multinode Issues - Synchronization

• Pthread would be great to use, as a framework, but the primitives are limited:
  – Mutex
  – Conditional Variables
• UPC … Parallel C looks more suited to us.