



Handling Variable-Output Kernels

17 February 2004



Problem

- Kernel outputs a variable number of elements for each input element it processes
 - Number isn't known at compile time
 - Isn't even known at initialisation time when datasets are loaded
 - Data-dependent
- Example: Polygon rendering

Problem mainly affects the compiler

- SRF allocation
 - Needs to do strip mining and SRF scheduling despite not knowing the lengths of some streams
- Kernel compilation
 - Stream's length is unknown, creating different boundary cases for kernels on the last partial strip
 - May need to compile several versions of each kernel to handle the different amounts of padding that might be present

Also impacts architecture

- What mechanisms can be added to the hardware to either enable compiler actions or to help performance?

Types of variable output kernels (1)

- Maximum number of outputs per input is bounded and not “too large”
 - Can simply allocate for the maximum

Types of variable output kernels (2)

- Maximum number of outputs per input is bounded but large compared to expected number
 - May be able to allocate for the maximum
 - Assuming it fits in the SRF
 - Even if it does fit, better performance may be obtained by allocating for less than the max. and having some mechanism to handle the cases when this overflows

Types of variable output kernels (3)

- Maximum number of outputs per input is unbounded
 - After reading a single input element, kernel may produce arbitrarily many outputs
 - Impossible to allocate for maximum output
 - SRF space may run out in the middle of processing an input element

“Solution” currently used

- Profiling is used in Imagine tools to handle variable output kernels
 - Run program once with full dataset in debug mode and record the actual lengths of all streams
 - Now compile the program, using the exact lengths
 - Re-run the program in cycle-accurate simulator
- Obviously, not a general solution
 - Especially since when we report perf. numbers we don't count the time to profile (with the full dataset)

Profiling as a length estimator

- Perhaps it can still be used in a more general way as a length estimation technique
 - Profile on a small sample dataset rather than the full data set
 - Estimate stream lengths
 - Have a mechanism to handle cases when estimate is wrong

Mechanisms

- The basic strategy for handling variable output kernels is for the compiler to somehow “guess” a length, and then have some mechanism to handle the run-time situation of the allocated space being too small

Mechanism: Double buffering (1)

- At run-time, when it becomes known that a SRF stream has reached some length before the producer kernel is finished, start DB
 - Split SRF stream into two half-buffers
 - When one is filled, write it to mem. while the kernel is filling the other half
 - Stall kernel if next half-buffer isn't ready yet
 - When producer kernel is done, DB the stream back from memory to the consumer kernel

Mechanism: Double buffering (2)

- This is a completely general solution to the SRF allocation problem
- Problem is that it can stall the kernel while waiting on memory
- Loses the producer-consumer locality exploited by the SRF

Mechanism: Double buffering (3)

- Two ways to do DB
 - Symmetric: Split SRF stream into two half-buffers all the time, and DB as soon as the first half-buffer is full
 - Tries to maximize overlap of kernels and mem. transfers
 - Conditional: Only initiate DB once SRF stream has filled completely
 - Only does DB when necessary

Mechanism: Spilling other streams

- Problem with DB is that the output stream may not be best choice to spill
- May be able to spill a different stream altogether and use that SRF space to hold excess outputs, and then re-load the spilled stream later
- Can use restartable streams to effectively chain SRF strips together

Mechanism: Kernel swapping (1)

- Idea is that if the producer kernel fills the output stream before finishing, then suspend it and run the consumer kernel to clear the buffer before resuming the producer kernel

Mechanism: Kernel swapping (2)

- When suspending, just stalling kernels isn't sufficient since the kernel state needs to be copied between clusters (LRFs etc.) and SRF
 - When suspending, kernel would be in charge of writing its state out
 - Similarly for resuming
 - Compiler would insert handler code into kernels

Mechanism: Kernel swapping (3)

- Overhead of DB and spilling in general is stalling kernel while waiting on memory; overhead of kernel swapping is the cost to suspend and resume a kernel
- Choice of which mechanism is better is dependant on characteristic of application

Mechanism: Kernel swapping (4)

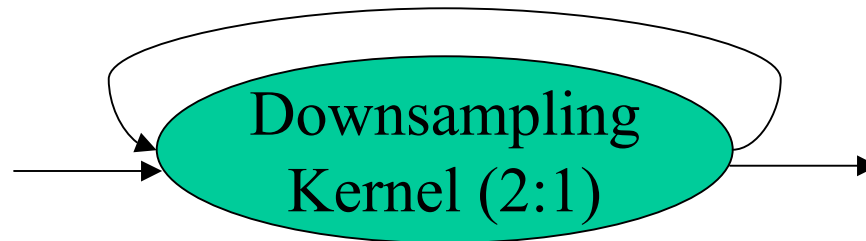
- A possible optimization would be to allow multiple cluster resident kernels (i.e.) partition the cluster state (LRFs etc.) at compile time so that different kernels used different cluster resources
- Would much reduce kernel suspend/resume time, making it close to 0
 - Can now basically just stall kernel rather than having to copy state around

Mechanism: Kernel merging (1)

- Kernel merging is a compiler transformation which converts SRF locality into LRF locality
 - Although with LRF spilling implemented this distinction is less clear
- Similar to multiple cluster-resident kernels, but uses LRFs to stage intermediate stream rather than SRF

Mechanism: Kernel merging (2)

- Double buff. and kernel swapping are both general solutions
 - Kernel merging is not
- For example, the following stream graph can't be merged into a single kernel:

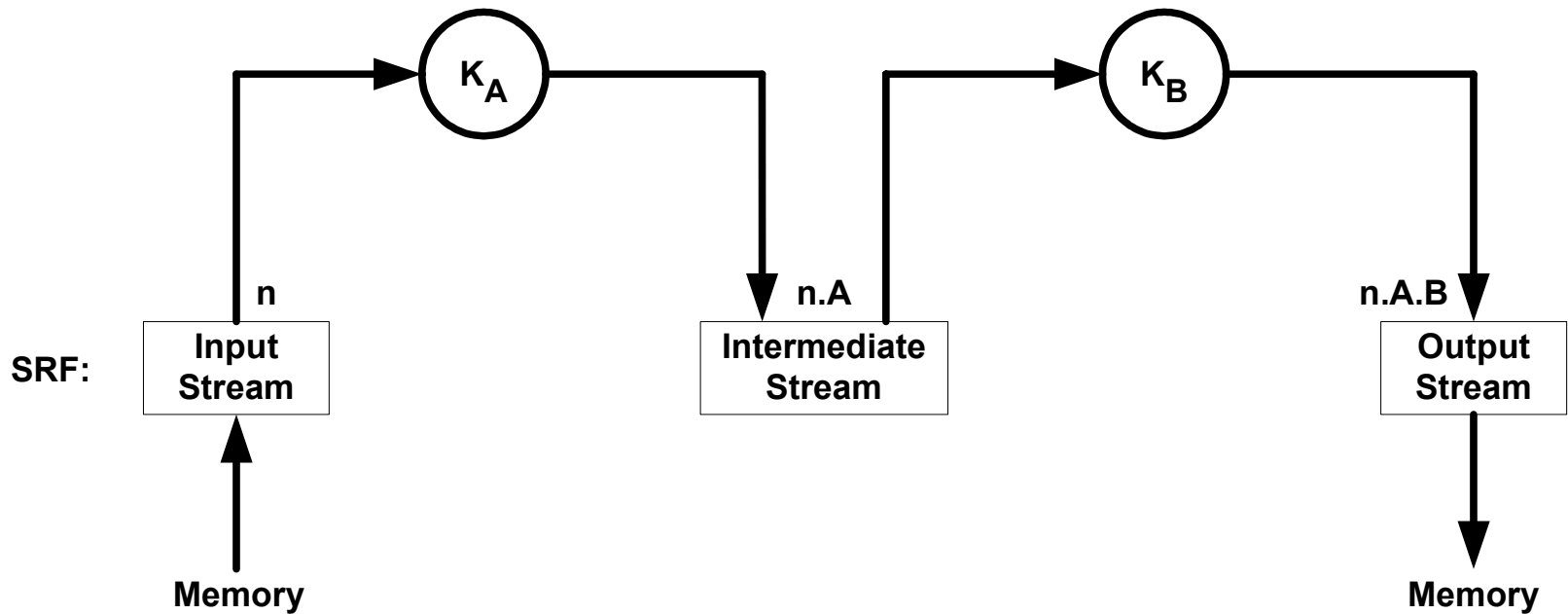


Evaluation of mechanisms

- Created a simple parameterized model of stream application and stream architecture
- Can explore which mechanism gives better performance for which kinds of applications
- Ideally, can fit an application to the model to determine which mechanism would be best suited

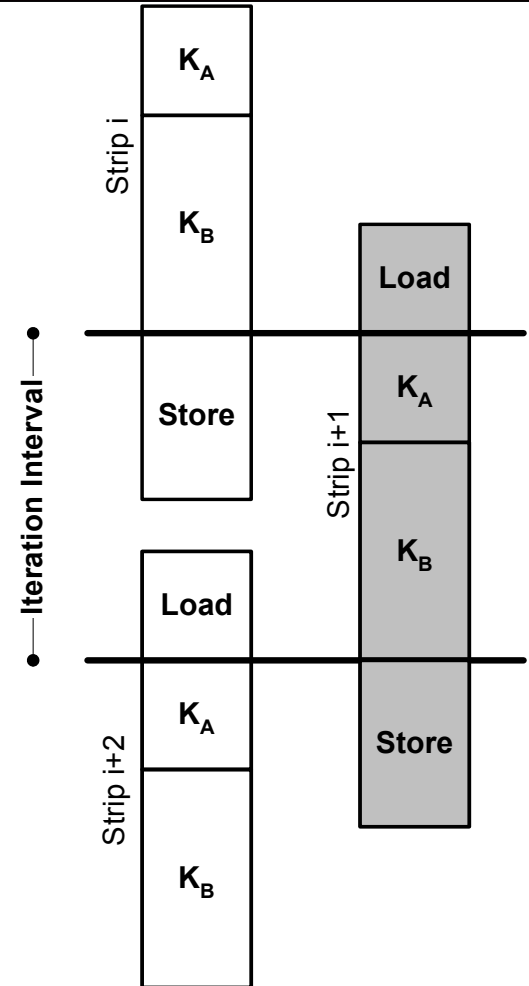
Model (1)

- Parameterized producer and consumer kernels
- Input-output multiplicative parameters A and B

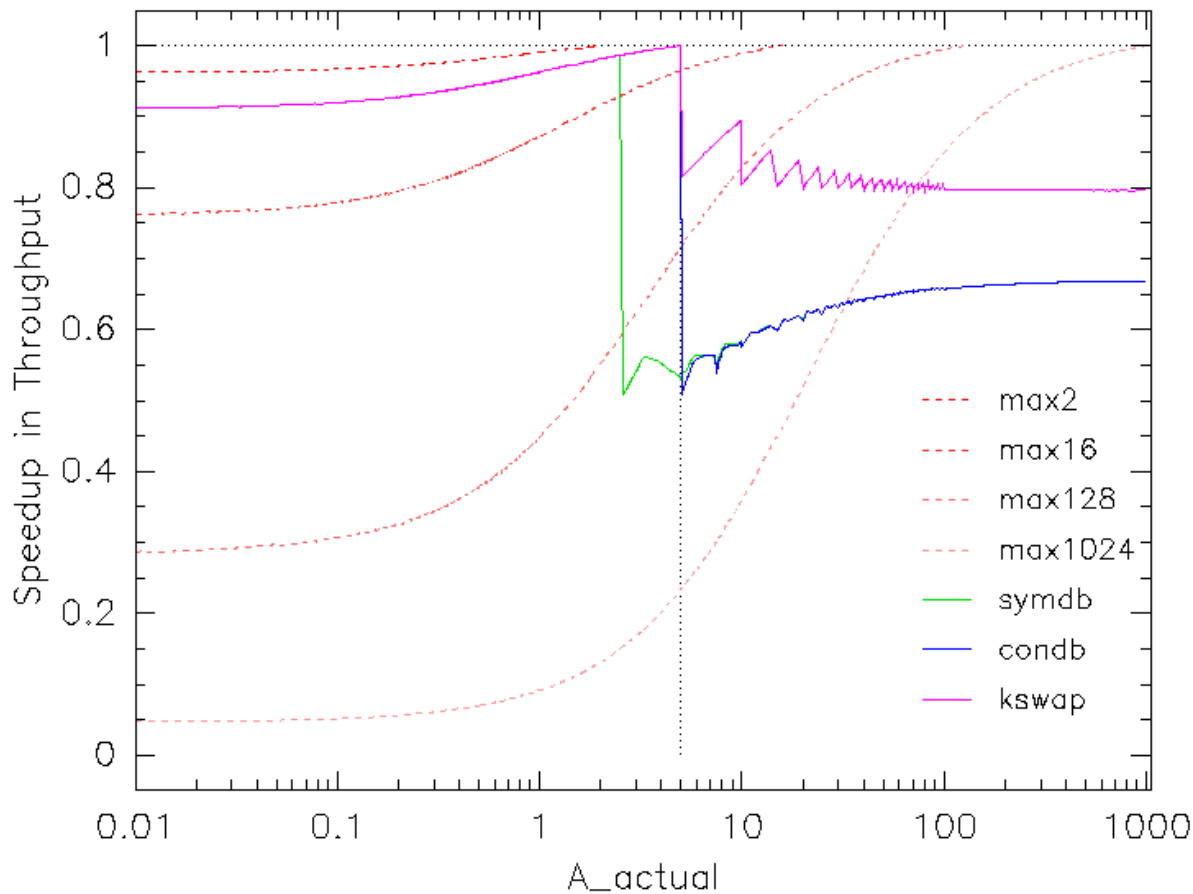


Model (2)

- Software-pipelined at stream level
- Metric is the increase in execution time of the iteration interval of SWP loop
- Baseline is case where the actual output lengths of K_A and K_B are exactly allocated for



Comparison, for example params



- Suspend and resume each take 10 cycles
- 100 cyc latency, 2w/cyc memory
- 50 cyc kernel startup
- 16 cyc kernel main loop
- Allocated for $A=5$
- $B = 1/2$

Last slide ...

- Different HW mechanisms can be used by compiler to support variable-length streams
 - Wasn't covered in talk, but the HW cost for each is basically just more complex control, wouldn't be much of a difference in area
- Challenges for compiler:
 - Guess the output length well
 - Choose the best mechanism to handle overflows based on application characteristics