Post Processing C66x Benchmarking

Sorting and Ordered Statistics

Test Setup

- Use Appleton EVM board for C66x benchmarking
- Use 2M of MSMC for input and output
  - Configured as cachable with prefetch on
- L1D and L1P configured as 32K cache
- No L2 cache
- Code and small memory are in L2
- Only did for the ascending order. Descending order will have similar code and the same benchmarks.

General Requirements

- Single-precision floating-point input.
- Also requires output of indices in sorted order.
- Vector size up to 512.

Sorting Module(1) :Simple Sort

- Each complete inner loop find the minimum and index, then swap with the current one.
- Implementation details:
  - For each search of next minimum, per loop 4 parallel searches are performed for the local minimum, then final comparison.
    - This minimize the loop carrier dependency bound for the inner loop
  - Inner loop is .LSD bounded @ ii = 3 for 4 searches.
- Size 512 vector takes 122k cycles
  - Comparing to flat memory estimation of 511 * 256 * 3 / 4 = 98k cycles
  - Comparing to Conti requirement of 800ns which is 800 cycles for 1GHz device.

Sorting Module(1) :Bubble Sort

- Per inner loop swap the bigger value towards the end of the buffer. Perform the same for the indices buffer.
- Implementation details:
  - There is not much we can parallelize or unroll the loop to mitigate the loop carrier dependency bound.
  - To make things worse, the conditional execution (either NOP added, or truly executed) of the inner loop from makes the cost the same as always swapping (worst case) even statistically only half of the time we may need swap.
  - The only saving is if there is early exit if no swap happened in the inner loop.
  - Inner loop is loop carrier dependency bounded @ ii = 15 per swap.
- Size 512 vector takes 1960k cycles
  - Comparing to flat memory estimation of 511 * 256 * 3 / 4 = 1960k cycles
  - Comparing to Conti requirement of 800ns which is 800 cycles for 1GHz device.

Sorting Module(1) :Merge Sort

- https://en.wikipedia.org/wiki/Merge_sort
  - Details and animation
  - Generic Code used from the example code.
    - Uses recursive calls
- C66x Implementation details:
  - Reused the recursive porting for compare and merge for size bigger than 4
    - This part of the code has loop of ii = 11 for 2 compare and merge op. This is .LSD bounded
  - The basic pair-wise compare and swap is recoded, with ii = 11 for 4 compare and swap op. This is also .LSD bounded
  - The final last compare and merge cannot be parallelized and it’s loop carrier dependency bounded @ ii = 10 for single compare and merge op.
- Size 512 vector takes 38.5k cycles
  - Comparing to flat memory estimation of 511 * 256 * 3 / 4 = 26.8k cycles
  - Comparing to Conti requirement of 800ns which is 800 cycles for 1GHz device.

Ordered Statistics Module

- Reused the simple sort, stop outer loop searching at desired rank.
- Implementation details:
  - For each search of next minimum, per loop 4 parallel searches are performed for the local minimum, then final comparison.
    - This minimize the loop carrier dependency bound for the inner loop
  - Inner loop is .LSD bounded @ ii = 3 for 4 searches.
- Size 512 vector rank = 16 takes 7.3k cycles
  - Comparing to flat memory estimation of 511 * 256 * 3 / 4 = 6.2k cycles
  - Comparing to Conti requirement of 800ns which is 80 cycles for 1GHz device.