MCMD L21 : GPU Algorithms

[Sitchinava + Weichert 13]

p multiprocessors MP
w scalar processors in each multiprocessor SP
w*p total scalar processors SP

 (ignores steam processor level of hierarchy)

MP i has cache C_i of size M (local to MP i)
  “unlimited” global memory G (6 to 12 GB)

Move data from G to C_i in blocks of size B. B in [w,4w]
  can move p blocks to all p caches in 1 step.
  CRCW (CW broken arbitrarily)

  Like BSP (I/O rounds can be synced, with barrier)

At each step, each SP in MP i, runs the same instruction, but on different in C_i

Each cache has b=w memory banks, and only one memory bank can be accessed at once.

Complexity:
  + parallel I/O complexity q_k
      = I/O steps   ~~ max I/O step per MP
  + PTime complexity t_k
      = scalar time steps   ~~ max time steps per SP
  + # barrier synchronizations
      = rounds of MP computation

Total time: (really)
   weighted sum of all of these.

lambda = latency of one block
sigma = latency of one barrier sync

sigma ~= (10^{1} or 10^{2}) lambda
lambda ~= (10^{2} or 10^{3})

 T(n) = sum_{k=1}^{R(n)} (t_k(n) + lambda * q_k(n) + sigma)

roughly Parallel External Memory [PEM] model of Arge etal. 2008.
  - remaining challenge: no memory bank conflicts
      small PTime so like PRAM

Cache C_i is organized as b x (M/b) cells, as a matrix. (b=w rows)
When block is called to C_i, stored in B/b adjacent columns.
  Assume M = c b^2 for c>=1 small integer.
  Example b x b matrix can be transposed in b=w PTime

Prefix Sum:
  O(n/(wp) + lambda * n/wp + sigma)

extends to colored case:
  + each element has a color [1...16] d=16
  + sum each color independently
 (if not careful, time increases by d -> bank conflicts)

Have w x d matrix in each C_i.
 + Each thread sums up one subset for each color.
 + transpose matrix
 + add each color
 Send d color counts to create global subset sums, which percolate back down.

Sorting:
2 phases:
  1. decompose data into chunks of size M
  2. for each chunk, load + sort on SP

1. They suggest merge sort, since get exact size M
    Could use sampling (e.g. terasort), but not exact size, need padding

2. More interesting for GPUs
  Store M numbers in w * w matrix.
   - use Shear sort (like boitonic sort)
   alternate (log w times):
     (a) sort each column in alternating order
          (first ascending, second descending, third ascending, …)
     (b) sort rows in ascending order.
    Leaves in column-major order.
   O(w^2 log^3 w) work, O(w log^3 w) parallel time.

    Each row/column sort using sorting network in t = O(w log^2 w) time
      Batcher's sorting network
      ** no bank conflicts ** (transpose between each phase in round)