CS7960 L23 : GPU | Bridging Model

GPU

Parallel processor
- Many cores
- Small memory

memory transfer overhead

What is missing? GPU has a large hierarchy.
G80 series:
  16 multiprocessors
  8 processor units
  6 steps in pipeline
+ GPU Memory
+ CPU Memory
+ Disk

Locality important in hierarchy.

Bridging Model (Les Valiant 2010)
  "Multi-BSP"

d = depth of hierarchy (=5)
4d parameters
  {(p_i, g_i, L_i, m_i)}_{i=1..d}
At level i:
  p_i = number of components at next level
        (p_5 = 1, p_4 = 1, p_3 = 16, p_2 = 8, p_1 = 6)
  g_i = bandwidth = ratio of (operation) / (words moved between level i and i-1)
        (assumed g_1 = 1)
  L_i = latency of barrier synchronization at level i.
        Algorithm proceeds in rounds, at each level, followed by synch.

```
  m_i = memory size at level i.
        Assumed m_i >= m_{i-1}*p_i

  Let P_i = prod_{j=1}^i p_j  = number of processors below level i
  Let M_i = m_i + p_i m_{i-1} + p_i p_{i-1} m_{i-2} + ...
            total memory available at level i
            (although, usually just replicate parts of m_i at
m_{i-1})
  Let G_i = sum_{j=1}^i g_j
            communication cost of two processors w/ LCA at level i

Assumes size of input is n, and that m_d = O(n), so data just fits
in memory.
   Otherwise, we need more external memory.
   We also assume g_{d+1} = infinite (FedEx, perhaps), not
connected...


----------
Van Neumann :  d=1  (p_1=1, g_1=infty, L_1=0, m_1=M)
PRAM  :        d=1  (p_1 >= 1, g_1=infty, L_1=0, m_1=M)
BSP(P,G,L) :   d=2  (p_1=1, g_1=G, L_1=0, m_1=M)
                    (p_2=P, g_2=infty, L_2=L, m_2=M')
----------

Goal of Valiant paper:
 - make work/Ptime basically optimal:  (1+o(1)) * W_sequential

 - make communication cost (based on g_i) and synchronization cost
(based on L_i)
     constant-factor within optimal.

 - adapts known lower bound results to this setting.

*** Wants algorithms that are optimal at all levels
simultaneously.
    Needs recursive structure in algorithms, so each level looks
identical
          unto value of parameters
----------
```

Results:
  (all results for synch similar, often just replace g with L)

  Adding(n) : requires n-1 adds
    comm = sum_{i=1}^d n (g_i / P_i)

  Matrix-Multiply(n x n) : basic algorithm, uses n^3 ops
    comm = n^3 / sum_{i=1}^d (g_i m_i^{-1/2} / P_i)

  FFT(n) : requires n log_2 n ops
    comm = sum_{i=1}^d (n log n)(g_i / (P_i log m_i))

  Sorting : requires n log n comparisons
    comm = sum_{i=1}^d (n log n)(g_i / (P_i log m_i))

Comparison Sorting (only count comparisons)

  At level i, get a set A of n_i = n * P_i / P unsorted points
  1. Use O(n)-Median-like procedure to compute set S of p_i splits
     so almost (about) n_i/p_i points between each s_j,s_{j+1} in
S
  2. Send elements {a in A | s_j <= a < s_{j+1}} to subnode j
     (if does not fit in memory, compute n_i/m_{i-1} sets, and
take turns...
  3. Call recursively on each subnode.

Recall, computing splits, requires some sorting
   - break into n_i/5 sets of size 5 (or some constant)
   - compute median of each set of 5 -> r
      let set of medians be R
   - sort R
   - compute p_i evenly spaced points in R -> S

Sorting of R is done by above approach recursively on sizes n_i/5

In practice: replace median with random distribution?
             (off by factor in size, but simpler and less
overhead)
             maybe only at lower levels, switch to this version...
             ...or switch to bitonic sort at bottom