Write-Avoiding Algorithms

Erin Carson, James Demmel, Laura Grigori, Nicholas Knight, Penporn Koanantakool, Oded Schwartz, Harsha Gardhan Simhadri

Motivation / Model

- Moving data (communication) most expensive operation (in time or energy), so avoid it
- Work so far on Communication-Avoiding Algorithms
- Provably minimize #loads and #stores between levels of memory hierarchy, and I/Os sent over network
- Big speedups in theory and practice

- Loads and Stores are not equal
  - Load = read from slow mem, write to fast mem
  - Store = read from fast mem, write to slow mem

- Writes may be much more expensive than reads:
  - Nonvolatile Memory (NVM)
  - Flash, PCM, STT-RAM, ReRAM, ST-RAM read = 14ns / 10^12 Joules vs. write = 50ns / 10^12
  - Faster wear-out with writes than reads
  - When intermediate results written to disk for fault tolerance
  - Writing may cause more coherency traffic in a shared memory environment

- Can we minimize #writes, not just #loads/#stores?

Theorem 1: Permits to fast memory ≥ ½ (#loads + #stores)
#writes to slow mem ≥ size of output
Take-away: may be able to do fewer writes to slow memory!
When is this (much smaller) lower bound attainable?

Definition: An algorithm that is CA and also attains an asymptotically smaller lower bound on #writes to slow memory is called Write-Avoiding (WA).

Write Lower Bounds

- W-A algorithms don’t always exist
- CDAG of an algorithm and its input is a directed graph
  - vertices = arguments (inputs, outputs, intermediate data)
  - edges = direct dependencies
- Theorem 2: If the out-degree of a CDAG of an algorithm (or large portions of it) is bounded by d, then

  #writes to slow memory ≥ (1/d) #reads from slow memory

  [see paper for a precise statement ]
- Corollary: Algorithms with bounded out-degree CDAGs like Cooley-Tukey FFT (≤ 2) and Strassen’s Algorithms (≤ 6) cannot be made write-avoiding by instruction ordering.

Write-Avoiding Linear Algebra and N-Body Algorithms

- There are sequential W-A algorithms for classical matrix multiplication, Triangular Solve, Cholesky Factorization, and direct N-body algos.
  - Based on appropriate loop reordering
  - All use
  - Explicit blocking based on cache size
  - Extends to multiple levels of memory
  - Assumes explicit control over data movement special cases of CA algorithms
  - Not all communication avoiding algorithms are WA
  - Conjecture: should work for many other familiar algorithms

Krylov Subspace Methods

- Communication-Avoiding CG:
  - Not: write (not converged)
    - Compute matrix-vector product
    - Update vector coefficients $\beta$ and $\alpha$
    - Recompute $\beta$ after writing,
    - Uses compact $[\beta(\alpha)^T, \beta]$ format
  - Write-Avoiding CG:
    - Not (converged)
      - Interleave computation and explicit construction of
        - Update vector coefficients $\beta$ and $\alpha$
        - Reuse $\beta$ after writing
    - Uses compact $[\beta(\alpha)^T, \beta]$ format (1 extra matrix powers kernel)

Hardware Perf Counters & Cache Replacement Policy

- W-A algorithms above assume explicit control over data movement (Ex: User accesses Flash over PCIe)
- Do cache policies like LRU enable WA algorithms?
- Experimental Data for 4000 x x 4000 matrix mult
  - Intel Nehalem-EX Xeon 7560, MESIF coherence, pseudo-LRU L3
  - 24MB L3, 256KB L2, 32KB L1 (Output: 122MB = 2.09in cache lines)
  - hbuf=1 used to allocate huge pages, avoid TLB misses
  - C-boxes and events accessed using customized Intel PCM 2.4
  - L3 vic = from L3 to DRAM (Modified L3 evictions)
  - L3 vic = # evictions without DRAM writes (Exclusive lines)
  - LLC vic = from L3 vic and LLC vic

Parallel W-A Algorithms

Model 1:
- L1 = cache
- L2 = DRAM
- Network (NW) connects L2s
- Can we minimize NW communication and L2 writes from L1?

Natural idea:
- Use CA algorithm to minimize NW communication
- Use WA algorithm locally on each processor
- Applies to Matmul, TRSM, Cholesky, N-Body, ...

Does it work for Matmul?
- Goals: n/p^2/p words moved on NW, n/p^2/p words to L2=L1
- Yes for NW, but n/p^2/p writes to L2 from L1
- Probably OK, since dominated by NW costs
- Can attain both lower bounds, but with p^2/p^2 times as much L2, probably not worth it

Model 2.1:
- L3 = NVM
- Data fits in L2
- Is it worth using L3?

Idea: use 2.5D algorithms that replicate data to reduce NW traffic
- Ex: 2.5DM moves n^2/p*(p/1)^2 words over NW if enough memory for a copies of data
- Using L3 may let us increase c, at cost of L3 writes

Performance model
- c : # copies using L2, c_1 copies using L3, so c_1 > c
- b_L = network BW, b_L2 = L3 write BW, b_L3 = L3 read BW
- Potential speedup = (c_1/c)^1.5*b_L/(b_L + b_L2 + b_L3)

Model 2.2:
- Same architecture as Model 2.1
- Data fits in L3, not L2
- Can we minimize NW communication and L3 writes from L2?
- Can we attain all lower bounds?
  - W_m = 0 (n^2/p^2) words communicated over network
  - W_m = 0 (n^2/p^2) words written to L3 from L2

Theorem 4 (bad news): It is impossible to attain both lower bounds (See paper for details)

Good news: There are algorithms that can attain either bound (but not the other)

- Alg 1: W_m = O(n^2/p^2) but W_m = W_L2
- Alg 2: W_L2 = O(n^2/p^2) but W_m = O(n^2/p^2)
- W_m is best depends on algorithmic & NW parameters
- Extends to L1

Theorem 8.1: If 5 blocks fit in L3 cache (not just 3 blocks required in explicit blocking), LRU matches lower bound for any instruction order of block multiplication within L3 cache.

If we do not require WA at each cache level, there exists an instruction order for which 3 blocks fit in L3 and be W-A.

Conclusions / Future Work

Possible to asymptotically reduce #writes to lowest level memory hierarchy for many algorithms:
- Enables saving time and energy for nonvolatile memory
- Works for many of the 7 dwarfs:
  - Dense/Sparse L1, Unstructured grids, N-body, (not FFT)

Future Work:
- Extend WA approach to other algorithms
- Conjecture: extends to nested loops accessing arrays (HBL)
- Conjecture: For direct N-body, need to double flops
- Conjecture: W-A impossible for O(n log n) sorting or FFT
- Extend theory, scheduling algorithms to shared memory
- Implement on Firebox++
  - When does ‘single node + NVM beat ‘cluster + DRAM’?