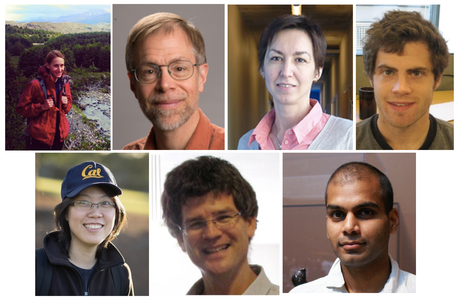
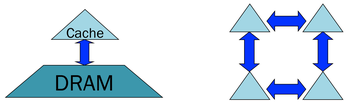
Write-Avoiding Algorithms

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

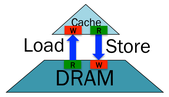


Motivation / Model

- 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 #words 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



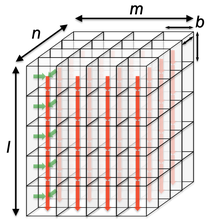
- Writes may be much more expensive than reads:
  - Nonvolatile Memory (NVM)
    Flash, PCM, STT-RAM, ReRAM, …
    STT-RAM read = .14ns / 10^{-15} Joules vs. write = 10ns / 10^{-12}J
    Faster wear-out with writes than reads
  - When intermediate results written to disk for fault tolerance (e.g. Spark)
  - Writes may cause more coherency traffic in a shared memory environment

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

- **Theorem 1**:
  #writes 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 (W-A).**

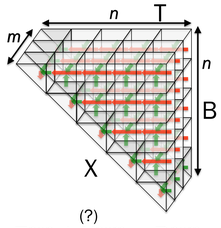
Write-Avoiding Linear Algebra and N-Body Algorithms



$C^{mxn} = A^{mxl} * B^{lxn}$

Classical Matrix Multiplication

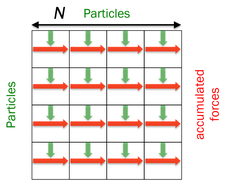
Writes to fast memory:
ml + 2mnl/b
Stores to slow memory:
mn



(?)
$T^{nxn} * X^{nxm} = B^{nxm}$

Triangular Solve

Loads to fast memory:
mn + n²/2+ mn²/b
Stores to slow memory:
mn



Direct N-body

Loads to fast memory:
N²/b
Stores to slow memory:
N

- 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
  - extend to multiple levels of memory
  - assume explicit control over data movement
  - special cases of CA algorithms
- Not all communication avoiding algorithms are W-A
- Conjecture: should work for many other familiar algorithms

Krylov Subspace Methods

Communication-Avoiding CG:

```
while (not converged)
   Compute and store

   Compute and store
    for i=1:s
        (update coefficient vectors p', r', x')
    end for
    Recover [p, r]=V[p', r']
end while
```

Reads per s steps: *O(nnz + n)*
Writes per s steps: *O(ns)*

Write-Avoiding CG:

```
while (not converged)
   Interleave computation of  and construction of
   for i=1:s
        (update coefficient vectors p', r', x')
   end for
   Recompute V (w/out writing), use to compute [p, r]=V[p', r']
end while
```

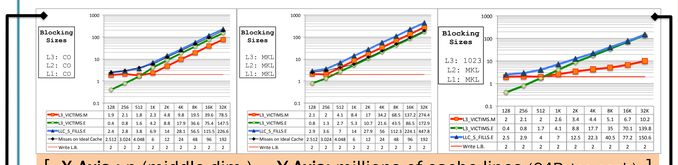
Reads per s steps: *O(nnz + n)*
Writes per s steps: *O(n)*

(1 extra matrix powers kernel)

[See Erin Carson's poster for details]

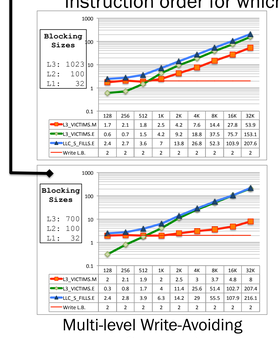
Hardware Perf Counters & Cache Replacement Policy

- 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 n x 4000 matrix mult
  - Intel Nehalem-EX Xeon 7560, MESIF coherence, pseudo-LRU L3
  - 24MB L3, 256KB L2, 32KB L1 (Output: 122MB = 2.0mln cache lines)
  - `hugectl` used to allocate huge pages, avoid TLB misses
- C-box events accessed via customized Intel PCM 2.4
  - L3_VICTIMS.M = #stores from L3 to DRAM (Modified L3 evictions)
  - L3_VICTIMS.E = #L3 evictions without DRAM writes (Exclusive lines)
  - LLC_S_FILLS.E = #loads from DRAM to L3 in Exclusive state

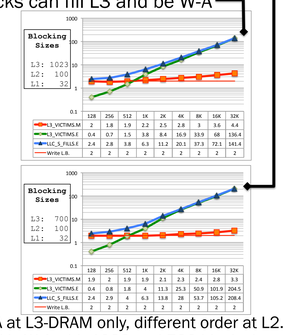


[**X-Axis** : n (middle dim.) **Y-Axis**: millions of cache lines (64Bytes each)]

- Measurements for Cache-Oblivious, MKL and W-A
  - Cache-Oblivious optimizes for reads, but not writes to DRAM
  - MKL optimizes for neither (better time though)
  - W-A at L3-DRAM (with b=1023, 3 blocks fit in L3) + MKL for L1 and L2 does fewer writes, but still not very close to optimal. Gap between lower bound and writes due to replacement policy.
- **Proposition 6.1**: 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 W-A at each cache level, there exists an instruction order for which 3 blocks can fill L3 and be W-A



Multi-level Write-Avoiding (left column)



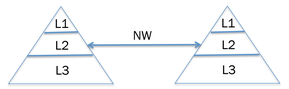
W-A at L3-DRAM only, different order at L2. LRU is W-A with 1023 and 700 block sizes.

Parallel W-A Algorithms

- Model 1:
  - L1 = cache
  - L2 = DRAM
  - Network (NW) connects L2s

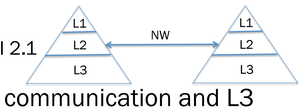
  - Can we minimize NW communication and L2 writes from L1?
- Natural idea:
  - Use CA algorithm to minimize NW communication
  - Use WA algorithm locally on each processor
  - Applies to Matmul, TRSM, Cholesky, N-Body, …
- Does it work for Matmul?
  - Goals: $n^2/P^{1/2}$ words moved on NW, n^2/P writes to L2←L1
  - Yes for NW, but $n^2/P^{1/2}$ writes to L2 from L1
  - Probably OK, since dominated by NW costs
  - Can attain both lower bounds, but with $P^{1/2}$ 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: 2.5DMM moves $n^2/(cP)^{1/2}$ words over NW if enough memory for c copies of data $(1 \le c \le P^{1/3})$
  - Using L3 may let us increase c, at cost of L3 writes
- Performance model
  - c_2 = #copies using L2, c_3 = #copies using L3, so $c_3 > c_2$
  - β_{NW} = network BW, β_{23} = L3 write BW, β_{32} = L3 read BW
  - Potential speedup = $(c_3/c_2)^{1/2}$ [β_{NW} /(β_{NW} + 1.5β_{23} + β_{32})]

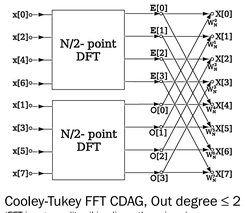
- Model 2.2
  - Same architecture as Model 2.1
  - Data fits in L3, not L2



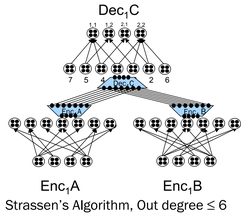
  - Can we minimize NW communication and L3 writes from L2?
- Can we attain all lower bounds?
  - $W_{NW} = \Omega\ (n^2/(cP)^{1/2})$ words communicated over network
  - $W_{23} = \Omega\ (n^2/P)$ words written to L3 from L2
- **Theorem 4** (bad news): It is impossible to attain both lower bounds (See paper for details)
- Good news: There are algorithms that can attain either bound (but not the other)
  - Alg 1: $W_{NW} = O\ (n^2/(cP)^{1/2})$ but $W_{23} = W_{NW}$
  - Alg 2: $W_{23} = O\ (n^2/P)$ but $W_{NW} = O\ (n^3/(P(\text{size\_of\_L2})^{1/2})$
  - Which one is best depends on algorithmic & HW parameters
  - Extends to LU

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 more precise statement]
  - Corollary : Algorithms with bounded out-degree CDAGs like Cooley-Tukey FFT (d ≤ 2) and Strassen's Algorithms (d ≤ 6) cannot be made write-avoiding by instruction ordering.



Cooley-Tukey FFT CDAG, Out degree ≤ 2
(FFT image credit: wikipedia; author: virens)



Strassen's Algorithm, Out degree ≤ 6

Cache-Oblivious → Not W-A

- **Definition**: A Cache-Oblivious (C-O) algorithm does not depend on cache-size parameters
  - There are well-known sequential Cache-Oblivious algorithms for classical matrix multiplication, TRSM, Cholesky, etc. that are communication avoiding for general nested memory hierarchies
- Can CO algorithms be Write-Avoiding?
  - No!
- **Theorem 3**: For a large class of problems, Cache-Oblivious algorithms do at least a constant fraction as many writes to slow memory as reads from slow memory
  - "Large class" means "smells like 3-nested loops"

- Conjecture: extends to nested loops accessing arrays
- Empirical measurements of C-O Matrix Multiplication are in line with Theorem 3

Conclusions / Future Work

- Possible to asymptotically reduce #writes to lowest level of memory hierarchy for many algorithms
  - Enables saving time and energy for nonvolatile memory
  - Works for many of the 7 dwarfs:
    Dense/Sparse LA, Un/structured 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 DFT
  - Can we minimize #messages too?
  - Extend theory, scheduling algorithms to shared memory
  - Implement on Firebox++
    When does 'single node + NVM' beat 'cluster + DRAM'?