

# Portable and Ordinal Memory Complexity Analyses through Data Movement Distance

---

Wesley Smith

*University of Rochester*

Chen Ding

*University of Rochester*

# Introduction

- Data movement now most important factor in performance/energy consumption
- Dominant assumption: accessing a bit anywhere in memory has no quantitative difference in cost
  - programming via virtual memory
  - asymptotic time/space complexity
- Why is this issue well-suited for attack?
  - fully automatic solutions brittle/suboptimal as scale/complexity of memory increases
  - urgent demand to reduce energy consumption in computing to address climate crisis
  - death of Moore's Law -> more specialization, including co-design of application/memory implementation

# Memory access complexity





















- goals for metric:
  - quantitative and ordinal (optimization)
  - symbolic and parameterized by input (problem size)
  - include effect of caching (hierarchical memory)
- past metrics inadequate?
  - miss ratio, I/O complexity
- our approach:
  - introduce abstract memory hierarchy model
  - use model to build metric

# Past metrics/analyses: why are they inadequate?

- Miss ratio
  - specific to (program, input, cache implementation)
  - rarely symbolic
  - even when symbolic, MRCs non-ordinal
- result: difficult to use miss ratio to analyze/compare algorithms across problem sizes
- Effect of hierarchical memory beyond single cache?
- Ordinality?
  - total order on functions?

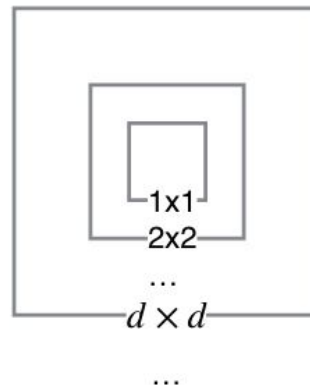
## Past metrics cont.

- I/O complexity: asymptotic expressions of lower-bound miss count
  - symbolic in cache and program size
  - not precise: big-O notation fails to differentiate between constant factor performance differences
  - portability issue: I/O complexity for algorithms derived from difficult ad hoc proofs
- *PLDI '20: Automated Derivation of Parametric Data Movement Lower Bounds for Affine Programs*
  - compiler technique to automatically produce lower bound
  - one approach to solving portability, precision issues
- Ordinality?
- Effect of hierarchical memory?

	Symbolic in input size	Portable	Distinguishes constant-factor performance differences	Ordinal	Considers effect of hierarchical caching
Miss ratio curves					
I/O complexity					
Olivry et al. PLDI '20					
???					

# Geometric stack

- abstract memory hierarchy model
- infinite level cache hierarchy
  - each level stores a single datum
  - processor resides at level 0
  - behavior generalizes effect of hierarchical memory
- two-dimensional planar shape
  - exact shape not important: memory size quadratic in distance to processor
  - contrast to classic (Mattson et al. '70) stack model
  - distance from datum with stack distance  $n$  to processor: proportionate to  $\sqrt{n}$



(a) A geometric stack



(b) A linear stack

# Data movement distance

**DEFINITION 1 (GEOMETRIC STACK DISTANCE (GSD)).** For an access sequence  $a$  and any stack algorithm, let  $l_i$  be the stack distance for  $a_i$  in the linear stack. Its GSD  $d_i$  is then

$$d_i = \sqrt{l_i}$$

**DEFINITION 2 (DATA MOVEMENT DISTANCE (DMD)).** For a program  $p$  with all data accesses  $a_i$  and any stack algorithm  $A$ , let its GSD of  $a_i$  be  $d_i$ . The DMD for  $p$  under caching algorithm  $A$  is

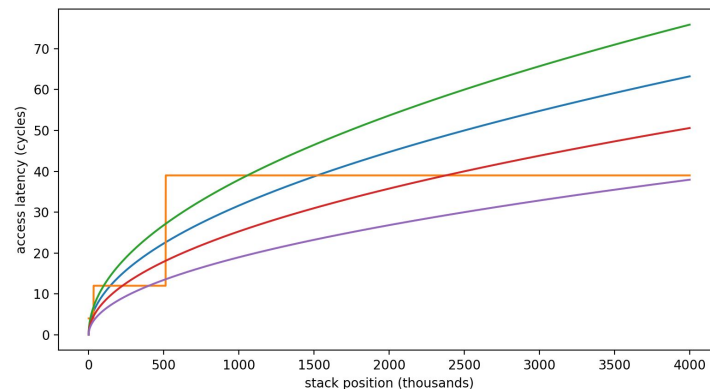
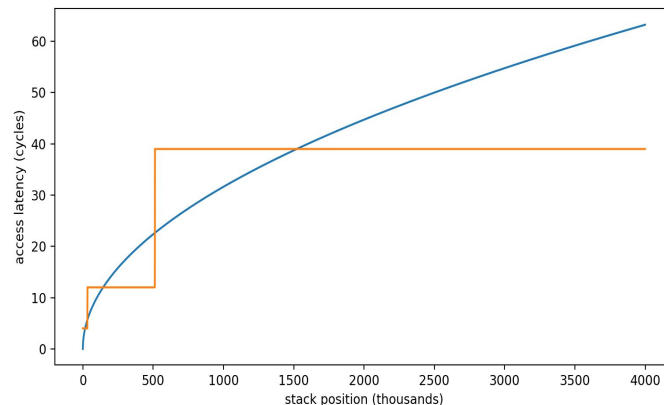
$$DMD(p, A) = \sum_i d_i$$

- DMD complexity: sum of distances in geometric (2D) stack model
  - symbolic in input size
  - ordinal for single input size!
    - geometric stack gave “exchange rate” for misses in caches of different sizes
- quantified by asymptotic equivalence
  - similar to big-O notation, but retains primary factor coefficient
  - additional precision necessary to distinguish access patterns known to be practically different



# Why $\sqrt{n}$ ?

- reflects physical (2D) memory layout
- reflects microarchitectural tradeoffs
  - Interpreting distance as cost function:
  
- AMD Zen2 architecture,  $\sqrt{n}$ : access latency vs. stack position
- goodness of fit?
  - unimportant: capturing trend in step function, not predicting exact values
  - care about family of functions  $a*\sqrt{n}$ 
    - total order preserved, DMD ratios preserved for  $DMD = a*\sqrt{n}$



# Data Movement Distance: Application to simple algorithms

- proposed metric: DMD
  - use stack distances in 2+ dimensional stack to create “exchange rate” between cache misses in caches of different sizes
- two data traversal patterns: *cyclic* and *sawtooth*
  - *cyclic*: *abc...mabc...m...*
  - *sawtooth*: *abc..m...cbabc...*
- two stack algorithms: LRU and OPT

## DMD: Application to simple algorithms cont.

- symbolic measure of data movement as function of input size
- time and space complexity unable to distinguish
  - difference obvious w.r.t memory systems
- LRU, OPT DMD asymptotically equivalent for *sawtooth*
  - property of great practical interest: **better caching cannot reduce data movement complexity**

	DMD per elem per traversal	
	LRU	OPT
cyclic	$\sim (\sqrt{m})$	$\sim (\frac{2}{3}\sqrt{m})$
sawtooth	$\sim (\frac{2}{3}\sqrt{m})$	$\sim (\frac{2}{3}\sqrt{m})$

## Rest of MEMSYS '21 position paper

- define memory access optimality through DMD
- prove upper bound on LRU, OPT stacks
- discuss the relationship between memory dimensionality and benefit of caching

# Ongoing/Future Work

- application to more algorithms
  - focus: algorithms where traditional time/space analyses are inadequate to understand performance
    - matrix multiplication: naive, recursive, Strassen, tiled...
  - FFT
- applications to performance optimization
  - compiler technique to compute DMD
  - report % improvement of optimization suite w.r.t. data movement
    - “cache oblivious” optimization
- feedback/suggestions welcome!
  - algorithm targets
  - applications
  - related work

# Questions