I/O-Efficient Algorithms for GIS Problems on Grid-based Terrains

Lars Arge, Laura Toma, Jeffrey S. Vitter

Duke University

January 2000

Computation on Terrains

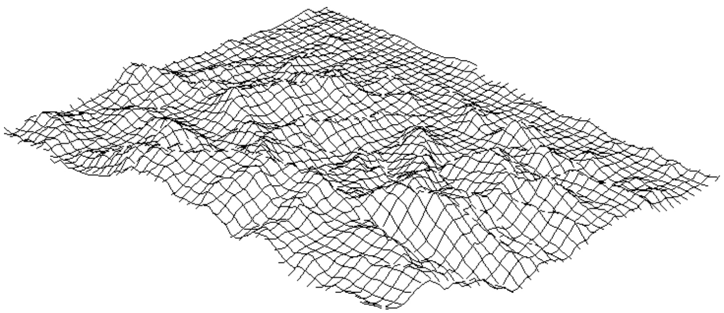
- ☆ Important in environmental analysis:
  - used in modeling processes like: *erosion, infiltration, drainage, solar radiation distribution, vegetation structure, species diversity*
- ☆ Terrain often modeled by grid of height values
  - readily available from remote sensing devices
  - yields simple algorithms
- ☆ High resolution data available for much of earth's surface
  - 30m resolution available for (almost) all US; 10m resolution becoming available; 1m resolution coming soon
  - NASA's Shuttle Radar Topography Mission: collect data for 80% of earth's land mass (10 terabytes)

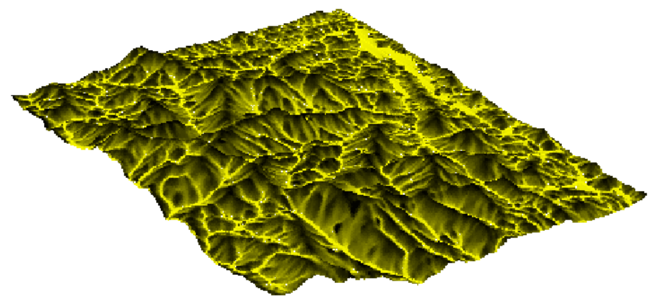
Flow Accumulation Problem

• initially one unit of water on each grid point

• every point distributes water to downslope neighbors proportional to height difference

Problem: Compute the total amount of flow through each grid point





☆ basic hydrologic attribute

- used to determine other hydrologic attributes: *drainage network, watersheds, topographic convergence index etc.*

Massive data–scalability

☆ Scalability problems

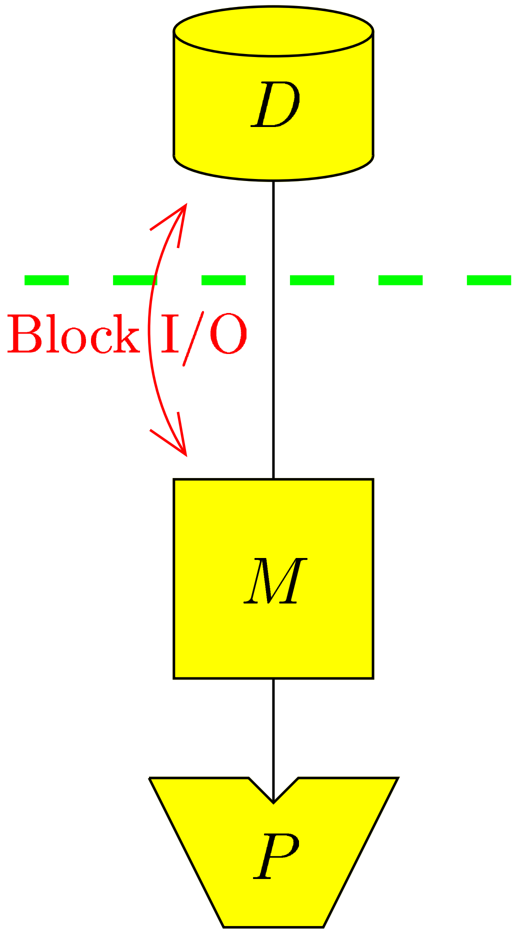
- current GIS algorithms minimize CPU time
- I/O is the bottleneck in computation on massive data

☆ Example: Duke environmental researchers—computing flow accumulation on Appalachian Mountains (800km $\times$ 800km)

- at 100m resolution: 500MB $\longrightarrow$ 14 days with 512MB memory
- at 30m resolution: 5.5GB
- at 10m resolution: 50GB
- at 1m resolution: 5TB !!

Disk Model

[Aggarwal & Vitter 88]



N = # grid points.

B = # grid points per disk block.

M = # grid points that fit into memory.

- Linear I/O is $\frac{N}{B} << N$
- sorting takes
  $\text{sort}(N) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os

Our Results

☆ Flow accumulation

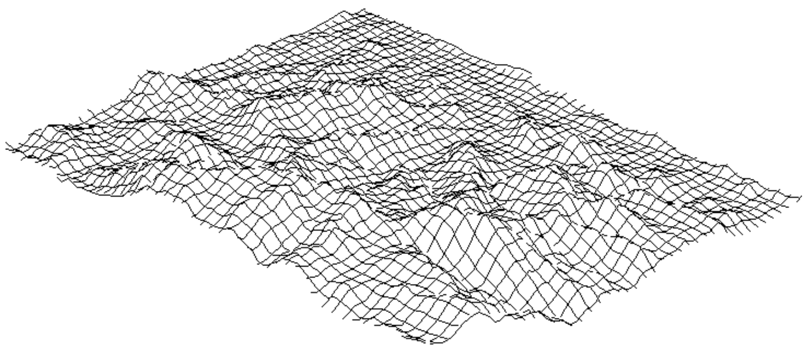
- we show that previous algorithms use $O(N)$ I/Os
- we develop new algorithm that uses $O\big(\mathrm{sort}(N)\big)$ I/Os
- we perform experiments showing practical efficiency on real-life data
  e.g. Appalachian Mountains in 2 hours!

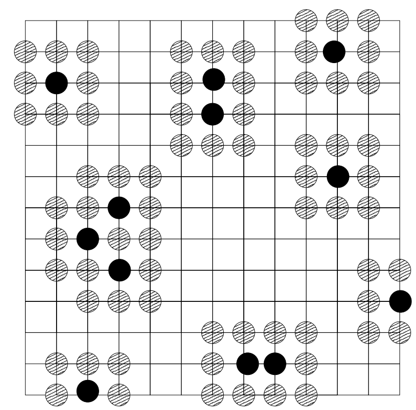
☆ Other problems on grid-graphs with terrain applications

Problem	Previous upper bound	Our result
SSSP	$O\left(N + \frac{N}{B}\log_2 \frac{N}{B}\right)$ [KS96]	$O(\mathrm{sort}(N))$
BFS	$O(N + \mathrm{sort}(N))$ [MR99]	$O(\mathrm{sort}(N))$
CC	$O(\mathrm{sort}(N))$ [CGGTVV95]	$O(\mathrm{scan}(N))$

Flow Accumulation–Internal memory algorithm

☆ Process (sweep) points in decreasing order of heights, distributing flow to neighbors
one sweep enough $\Longrightarrow O(N \log N)$ time





☆ Problem: algorithm uses $O(N)$ I/Os if height and flow stored as grids (not fitting in memory)

- points with same height are distributed over the terrain
$\Longrightarrow$ scattered accesses to (1) elevation grid (2) flow grid

Flow Accumulation–I/O-efficient algorithm

☆ Eliminate scattered accesses to elevation grid:

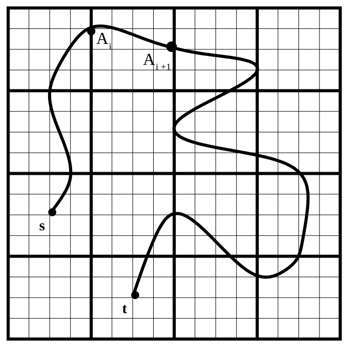
- augment each height with heights of 8 neighbors (trade space for I/Os)

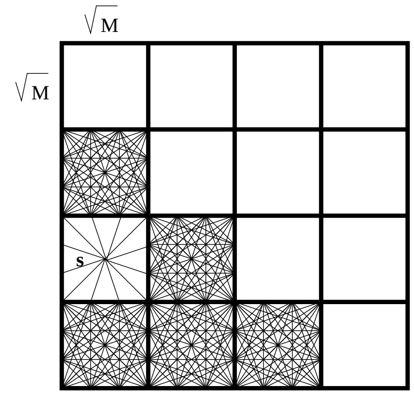
☆ Eliminate scattered accesses to flow grid

- idea: neighbor only needs the distributed flow when the sweep plane reaches its elevation
- use a $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ priority queue [A95, BK98]
  - ∗ distribute flow by inserting it in priority queue with priority equal to neighbor's height (and grid position as secondary key)
- $O(N)$ priority queue operations $\Rightarrow O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ algorithm

SSSP on Grid-graphs

☆ previous bound: $O(V + \frac{E}{B} \log_2 \frac{V}{B}) = O(N)$ I/Os [KS96]





☆ sparcification: replace each $\sqrt{M} \times \sqrt{M}$ subgrid with a full graph on the "boundary vertices": $\Theta(\frac{N}{\sqrt{M}})$ vertices, $\Theta(N)$ edges

- the weight of an edge is the shortest path between the two boundary vertices **in the subgrid**
- $O(\frac{N}{B} \log_2 \frac{N}{\sqrt{M}B})$ I/Os

☆ improved to $O(\text{sort}(N))$ I/Os

- Dijkstra algorithm and graph blocking

Empirical Results

- ⋆ We implemented internal and external algorithms
- ⋆ Implementation facilitated by use of TPIE
  - toolkit for efficient implementation of external memory algorithms
  - primitives: scanning, sorting
  - available at `http://www.cs.duke.edu/~tpie`
  - we implemented external priority queue in TPIE using [Brodal & Katajainen 98]
- ⋆ Experimental platform
  - 450MHz Intel PII running FreeBSD 4.0
  - striped disk array

Datasets

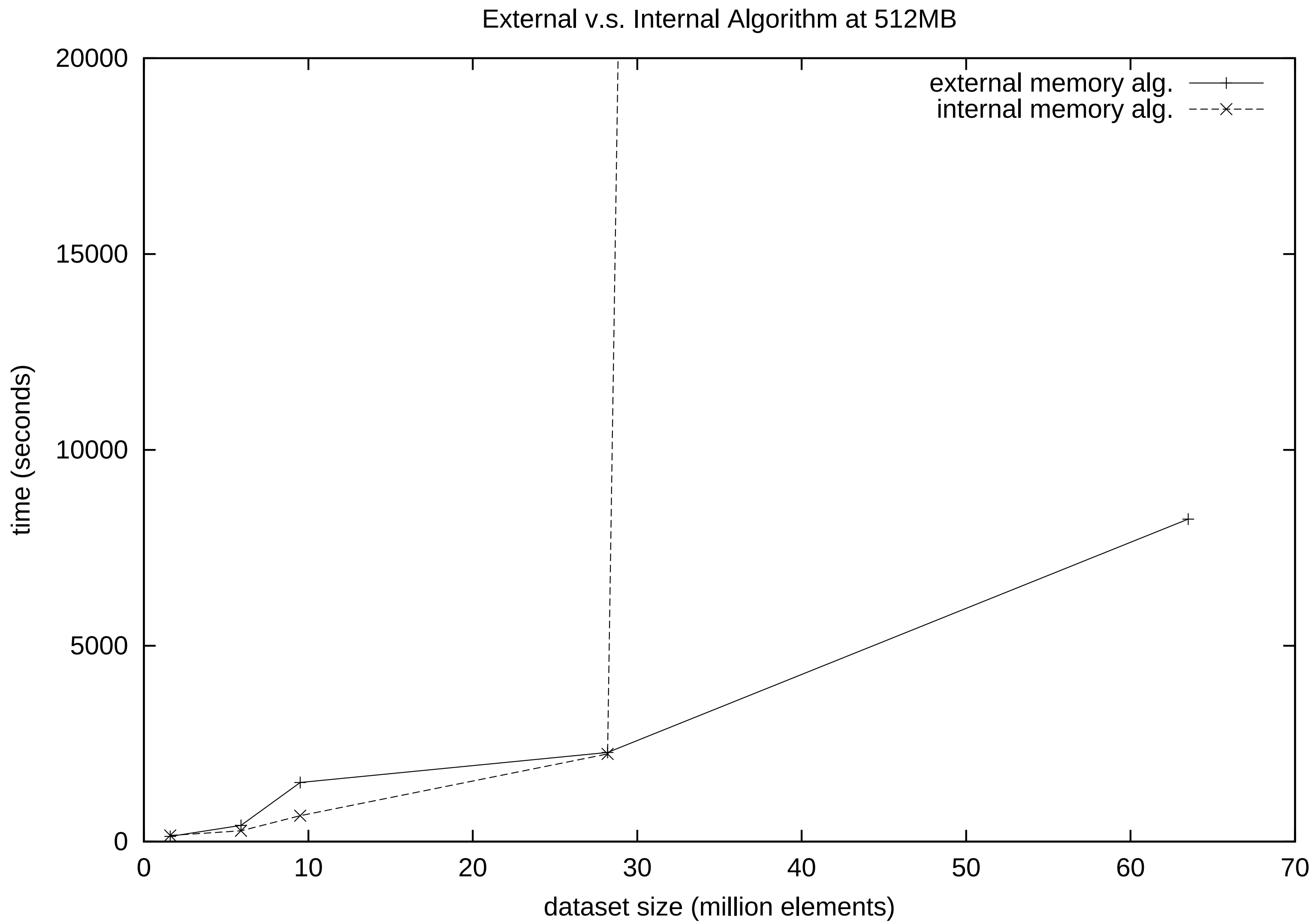
☆ 3 national parks (provided by NSoE)

- Kaweah River: foothills of Sierra Nevada Range
- Sierra: Sequoia/Kings Canyon National Park
- Appalachian Mountains

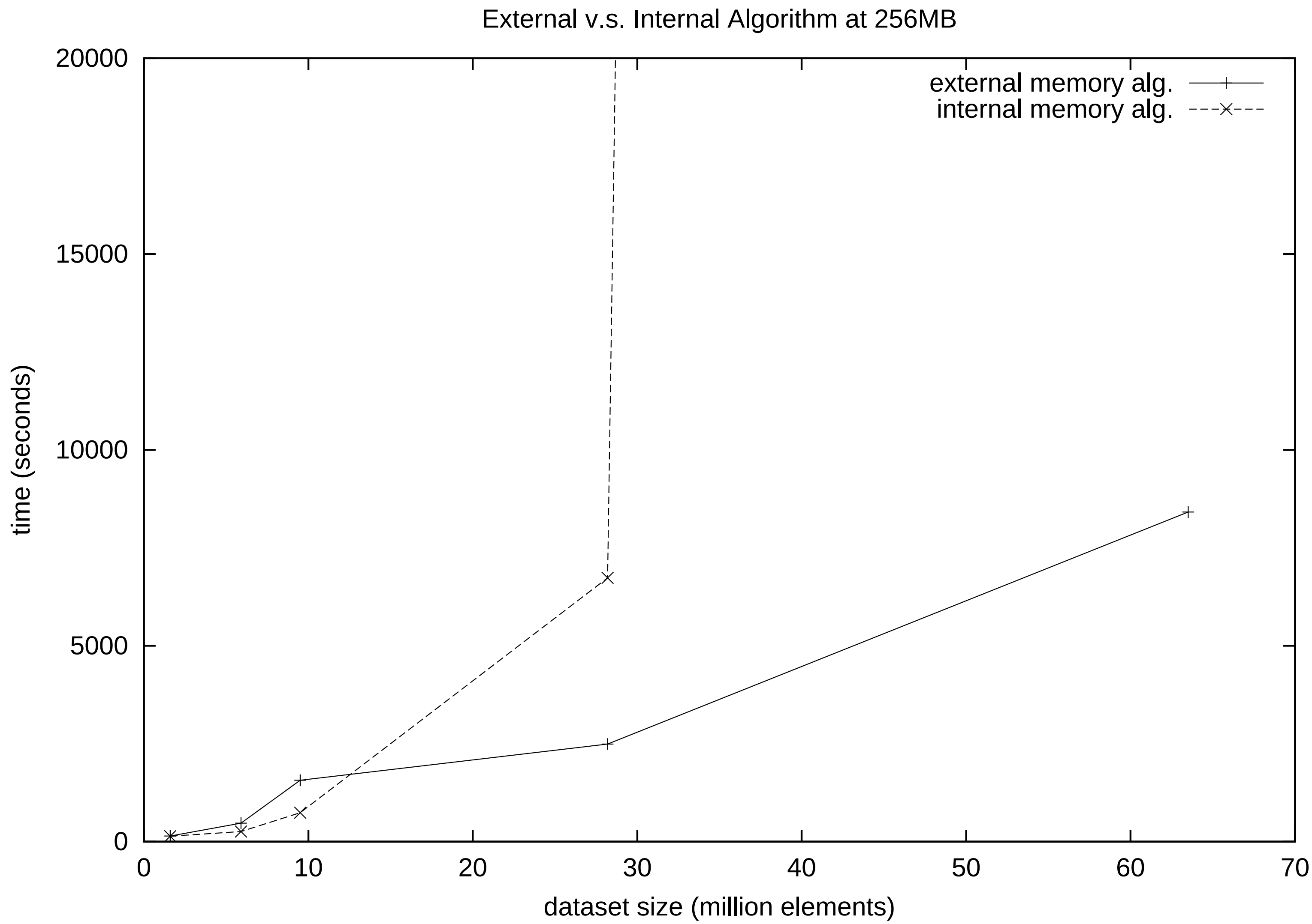
☆ 2 low-relief island-states: Hawaii and Puerto Rico

Dataset	**Coverage [km×km]**	**Grid size**	**Approx. size**	**Res**
Kaweah	34 × 42	1163 × 1424	1.6 $\times 10^6$, 13MB	30m
Puerto Rico	445 × 137	4452 × 1378	5.9 $\times 10^6$, 47MB	100m
Sierra	112 × 80	3750 × 2672	9.5 $\times 10^6$, 76MB	30m
Hawaii	678 × 4369	6784 × 4369	28.2 $\times 10^6$, 225MB	100m
Appalachian	847 × 785	8479 × 7850	63.5 $\times 10^6$, 508MB	100m

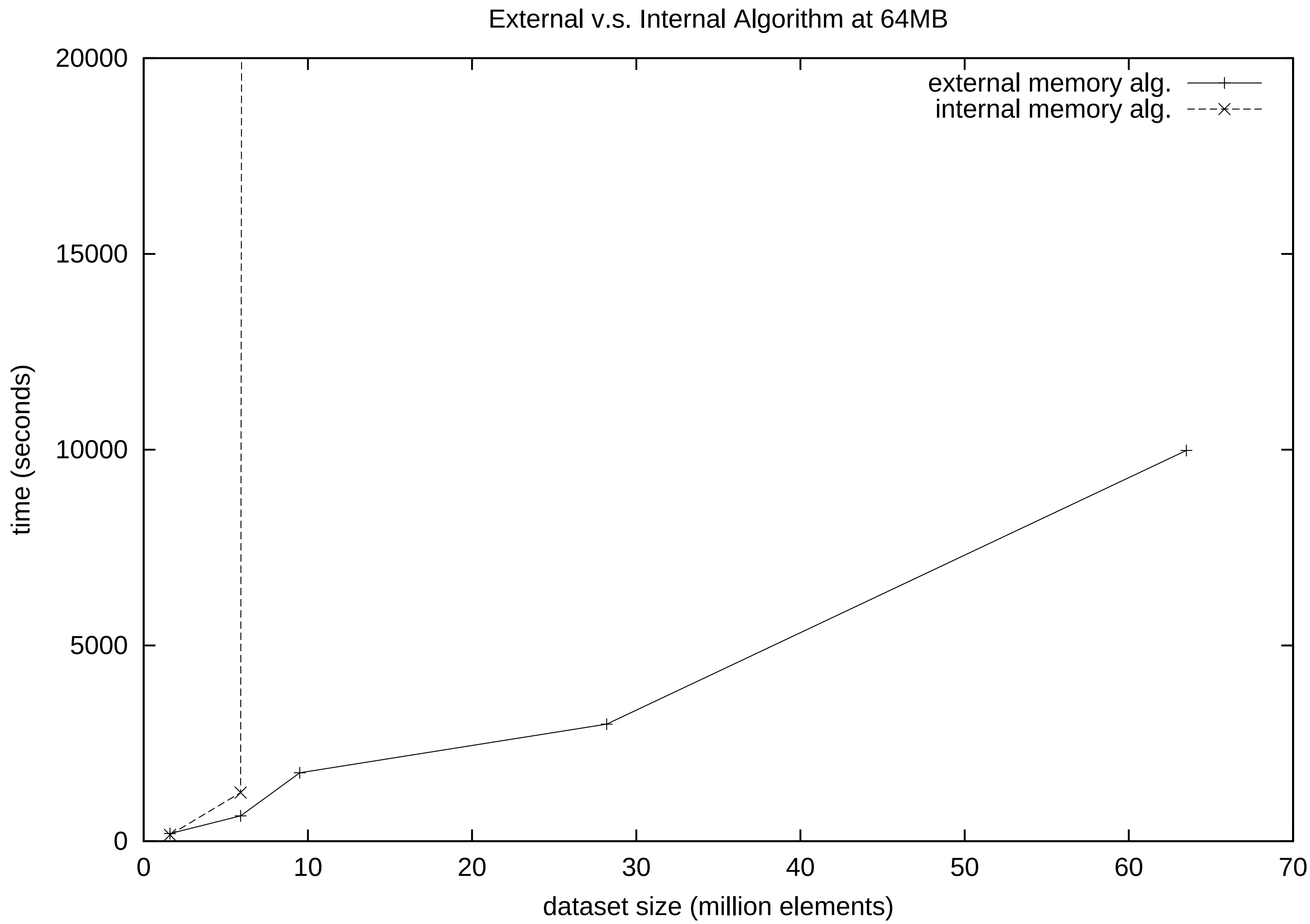
Experimental Results–512MB



Experimental Results–256MB

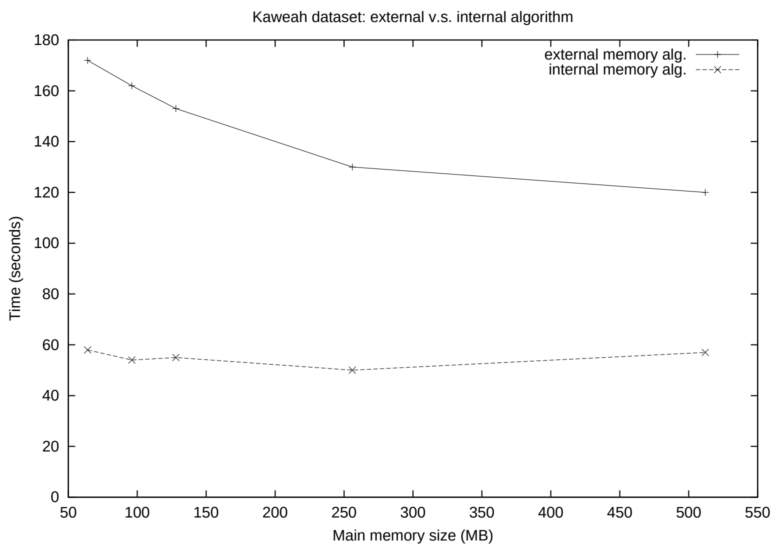


Experimental Results–64MB

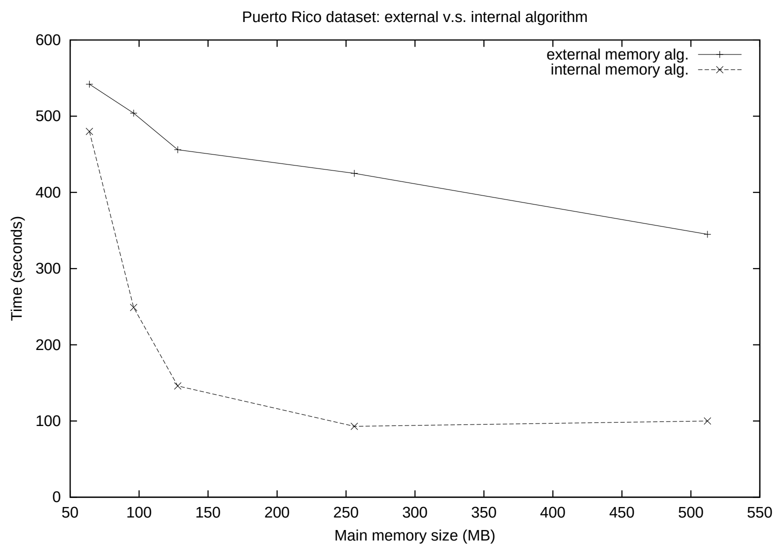


More graphs–Results for each dataset

Kaweah [13MB]

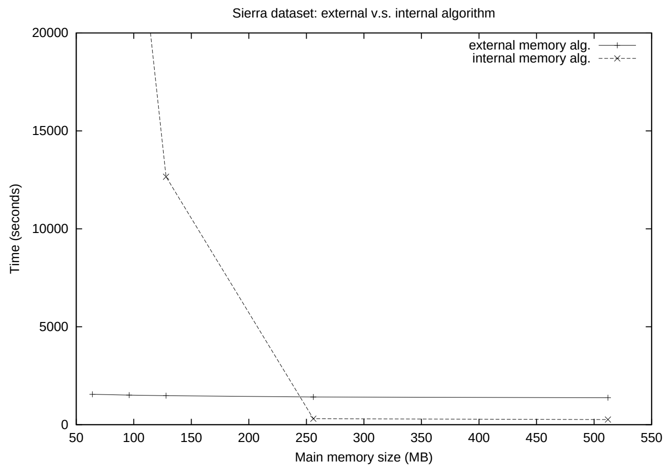


Puerto Rico [47MB]

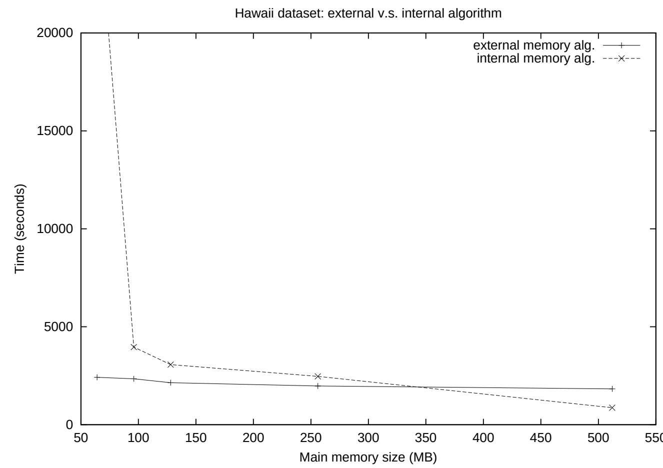


More graphs–Results for each dataset

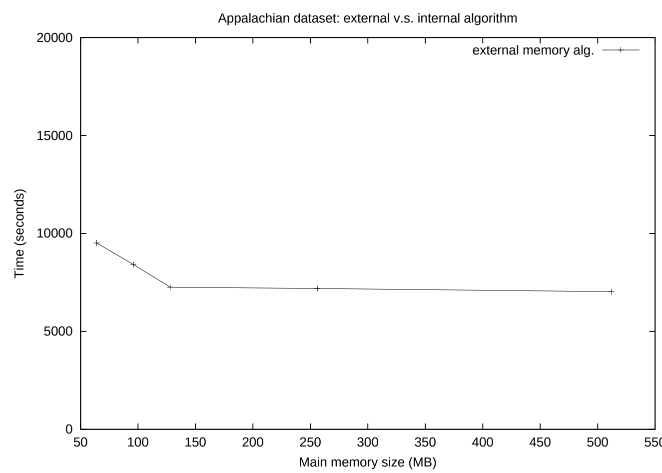
Sierra [76MB]



Hawaii [225MB]



Appalachians [508MB]



Experimental Results–Comments

☆ Time and CPU utilization reflect the I/O bottleneck

- e.g.: time on Sierra-Nevada

memory=	512MB	128MB	64MB
internal algorithm	4 mins	3.5 hours	∞
external algorithm	22 mins	24 mins	25 mins

- e.g. CPU utilization on Sierra-Nevada

memory=	512MB	128MB	64MB
internal algorithm	79%	2%	NA
external algorithm	84%	79%	77%

☆ Internal algorithm: bottleneck is sweeping (not sorting)

☆ Sweeping depends on terrain properties

Conclusions

- ☆ I/O-efficient algorithms for grid-graph problems on terrain applications
  - severely I/O-bound when dataset does not fit in memory
  - external algorithm scales very well
- ☆ Future directions:
  - extend algorithm to handle depressions (currently depressions are assumed filled in preprocessing step)
  - develop more realistic flow models (e.g. subsurface flow)
  - flow accumulation on TINs