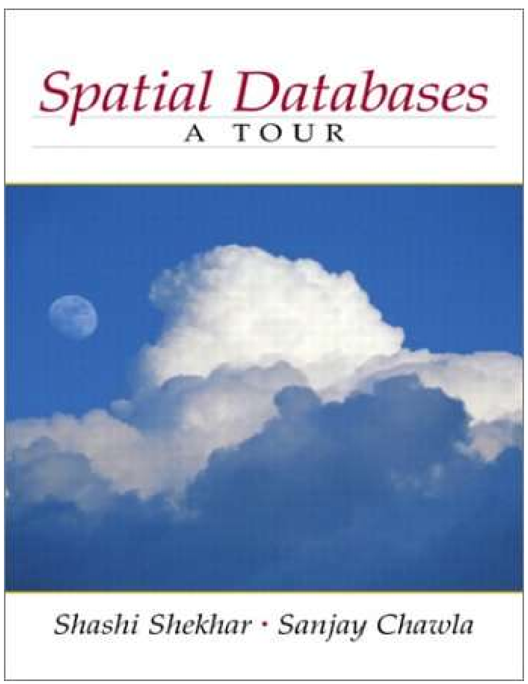
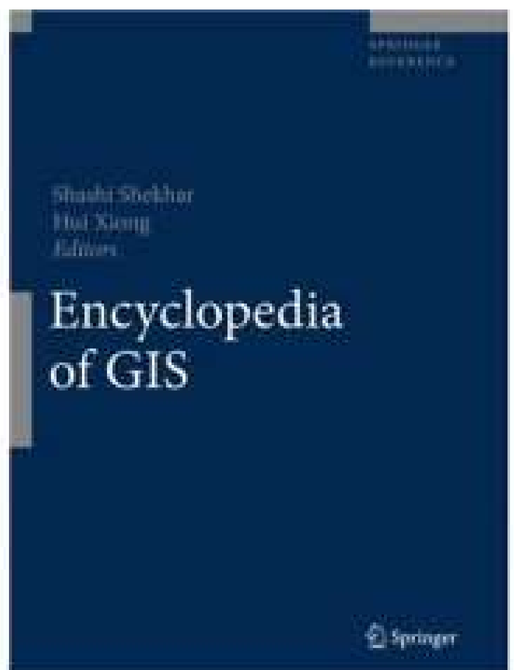
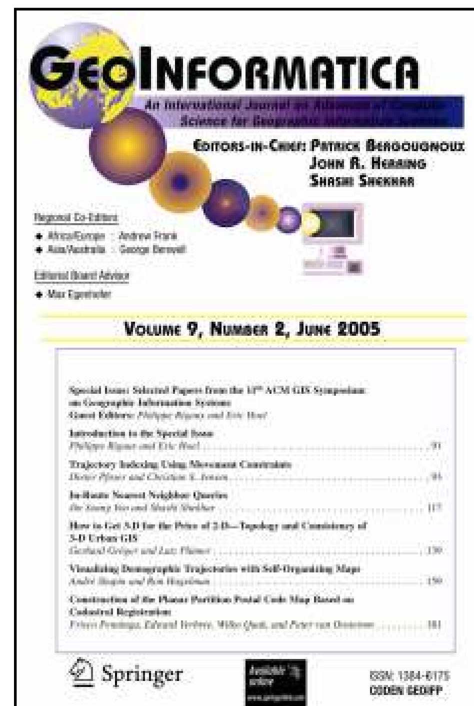
High Performance Computing With Spatial Datasets

Shashi Shekhar
McKnight Distinguished University Professor
Department of Computer Science and Engineering
University of Minnesota
www.cs.umn.edu/~shekhar





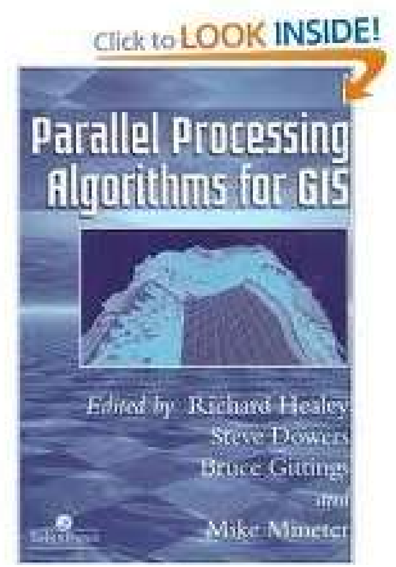


Acknowledgments

- HPC Resources, Research Grants
  - Army High Performance Computing Research Center-AHPCRC
  - Minnesota Supercomputing Institute - MSI
- Spatial Database Group Members
  - Mete Celik, Sanjay Chawla, Vijay Gandhi, Betsy George, James Kang, Baris M. Kazar, QingSong Lu, Sangho Kim, Sivakumar Ravada
- USDOD
  - Douglas Chubb, Greg Turner, Dale Shires, Jim Shine, Jim Rodgers
  - Richard Welsh (NCS, AHPCRC), Greg Smith
- Academic Colleagues
  - Vipin Kumar
  - Kelley Pace, James LeSage
  - Junchang Ju, Eric D. Kolaczyk, Sucharita Gopal

Overview

- Motivation for HPC with Spatial Data
  - **Ex. Application: Disaster Relief, Evacuation Planning**
  - Ex. Application: Geo-spatial Intelligence
- Case Study 1: Simple to Parallelize
- Case Study 2 – Harder
- Case Study 3 – Hardest
- Wrap-up



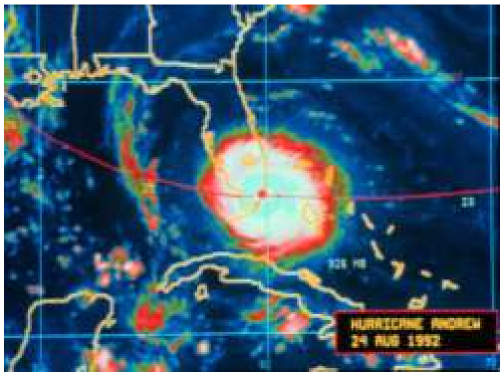
Example Application : Situation Assessment

- Where are the affected people?
- Which roads are navigable?
- Where may relief supplies to air-dropped?
- Where should distribution centers be located?
- What is the environment? What is the impact?

Example Application: Evacuation Planning

Hurricane Andrew

Florida and Louisiana, 1992



(National Weather Services)

(www.washingtonpost.com)

- Lack of effective evacuation plans
- Traffic congestions on all highways
- Great confusions and chaos

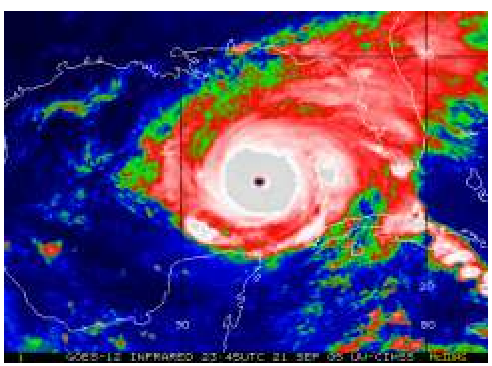
"We packed up Morgan City residents to evacuate in the a.m. on the day that Andrew hit coastal Louisiana, but in early afternoon the majority came back home. **The traffic was so bad that they couldn't get through Lafayette**."

Mayor Tim Mott, Morgan City, Louisiana

(http://i49south.com/hurricane.htm)

Hurricane Rita

Gulf Coast, 2005

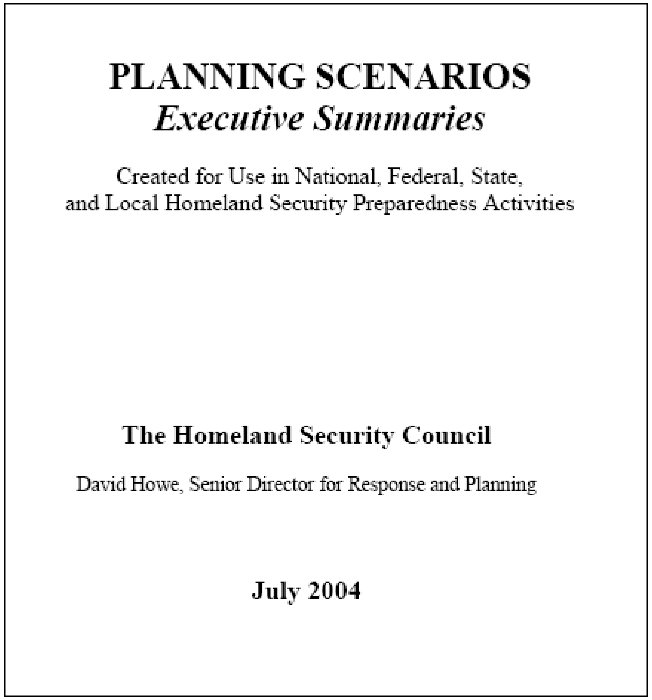


(National Weather Services)

(FEMA.gov)

Hurricane Rita evacuees from Houston clog I-45.

Homeland Defense & Evacuation Planning



PLANNING SCENARIOS

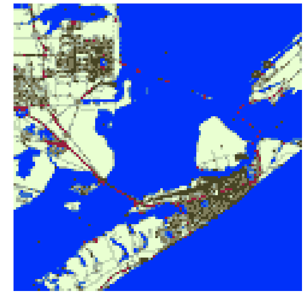
Executive Summaries

Created for Use in National, Federal, State, and Local Homeland Security Preparedness Activities

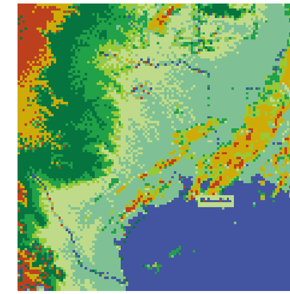
The Homeland Security Council

David Howe, Senior Director for Response and Planning

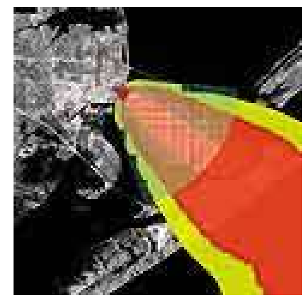
July 2004



Base Map

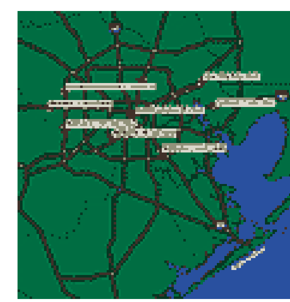


Weather Data



Plume Dispersion

Demographics Information



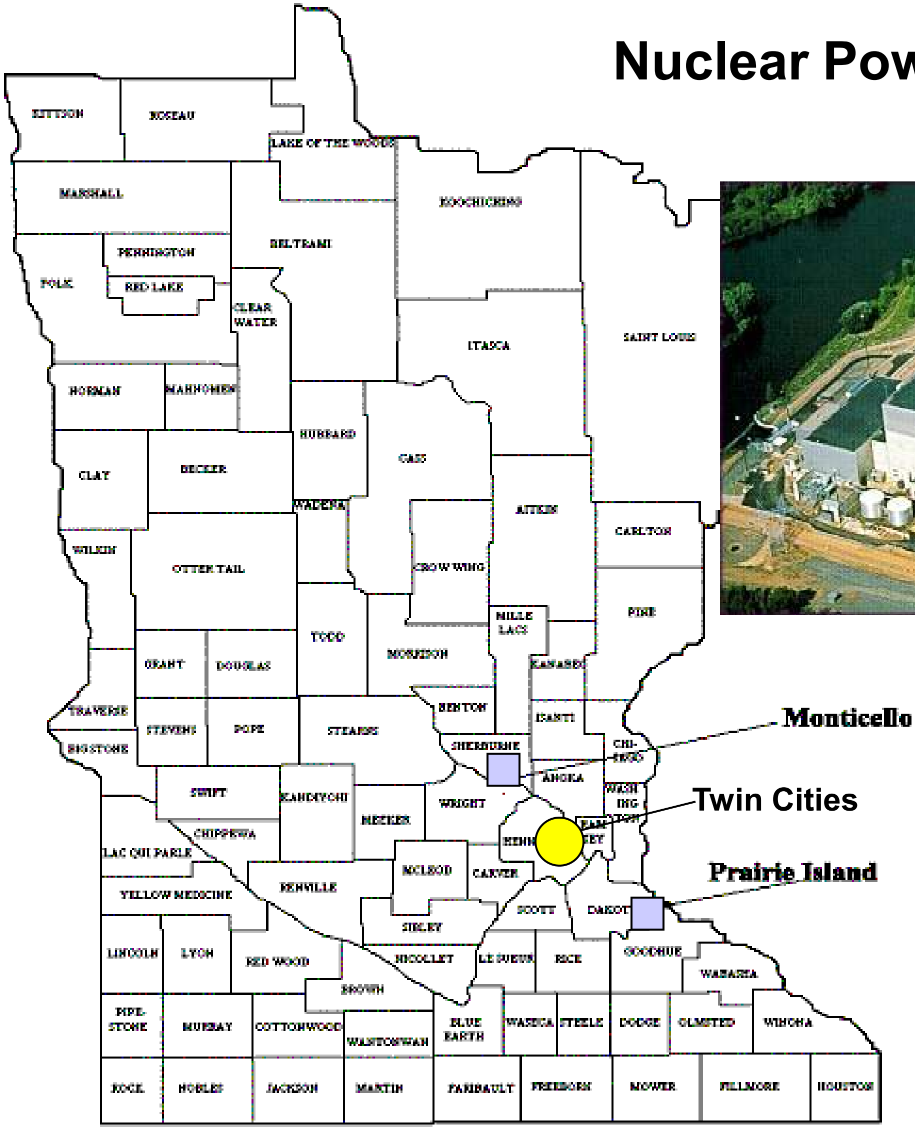
Transportation Networks

- Preparation of response to a chem-bio attack
- Plan evacuation routes and schedules
- Help public officials to make important decisions
- Guide affected population to safety

(*Images from www.fortune.com*)

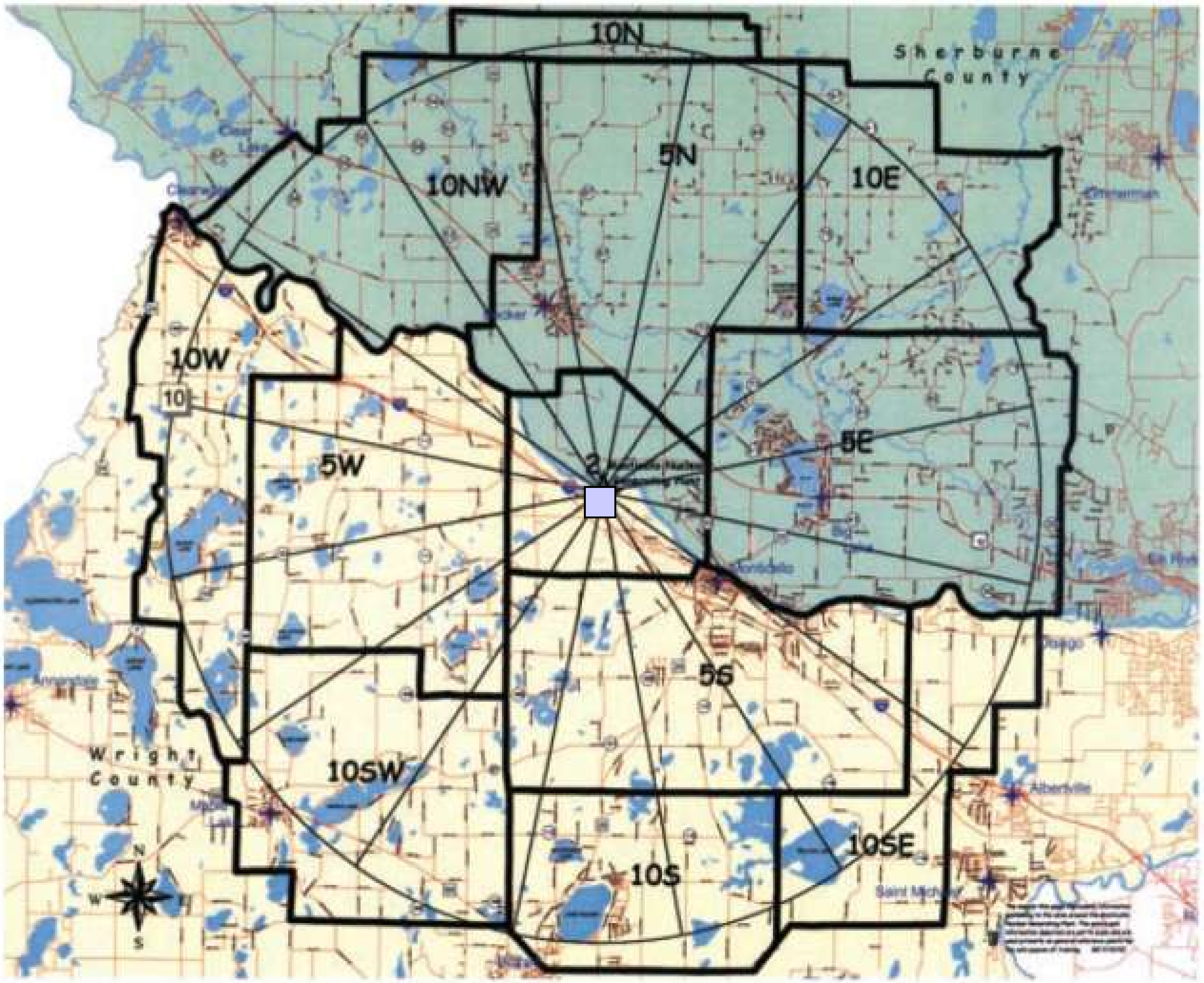
A Real Scenario

Nuclear Power Plants in Minnesota



Monticello Emergency Planning Zone

Emergency Planning Zone (EPZ) is a 10-mile radius around the plant divided into sub areas.



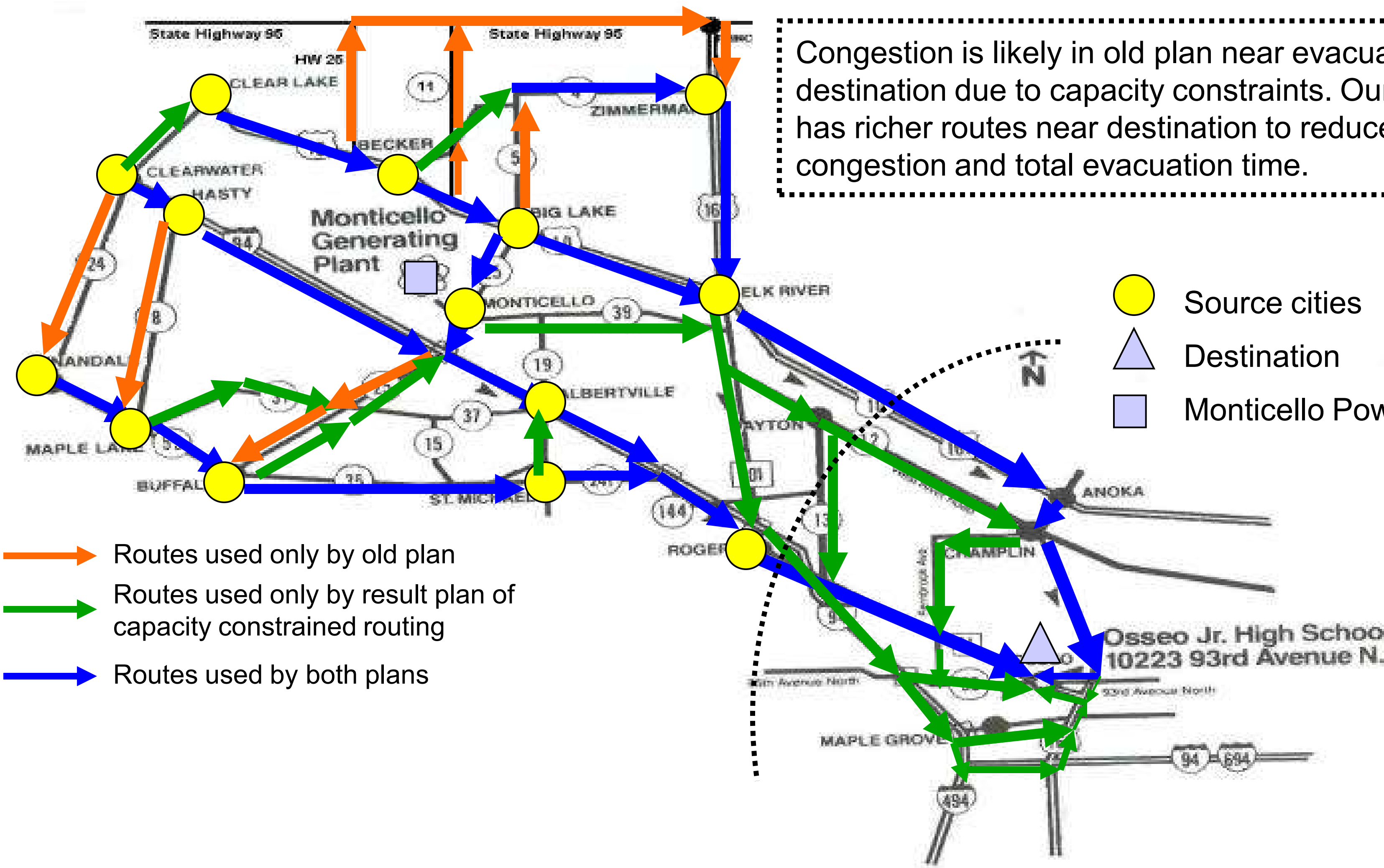
Monticello EPZ

Subarea	Population
2	4,675
5N	3,994
5E	9,645
5S	6,749
5W	2,236
10N	391
10E	1,785
10SE	1,390
10S	4,616
10SW	3,408
10W	2,354
10NW	707
Total	**41,950**

Estimate EPZ evacuation time:
Summer/Winter (good weather):
3 hours, 30 minutes
Winter (adverse weather):
5 hours, 40 minutes

Data source: Minnesota DPS & DHS
Web site: http://www.dps.state.mn.us
http://www.dhs.state.mn.us

A Real Scenario (Monticello): Result Routes



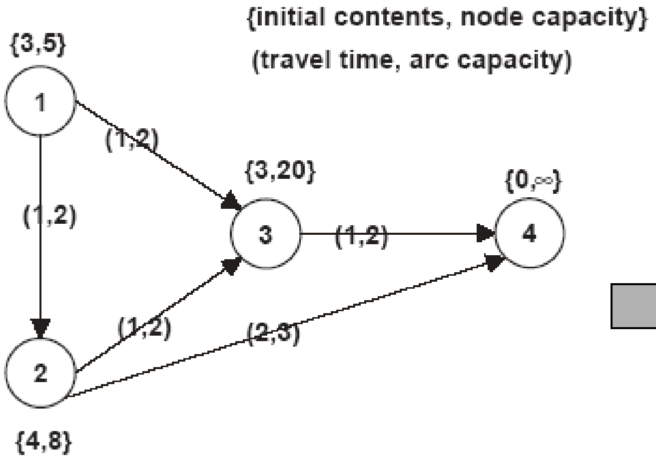
Congestion is likely in old plan near evacuation destination due to capacity constraints. Our plan has richer routes near destination to reduce congestion and total evacuation time.

- Source cities
- Destination
- Monticello Power Plant

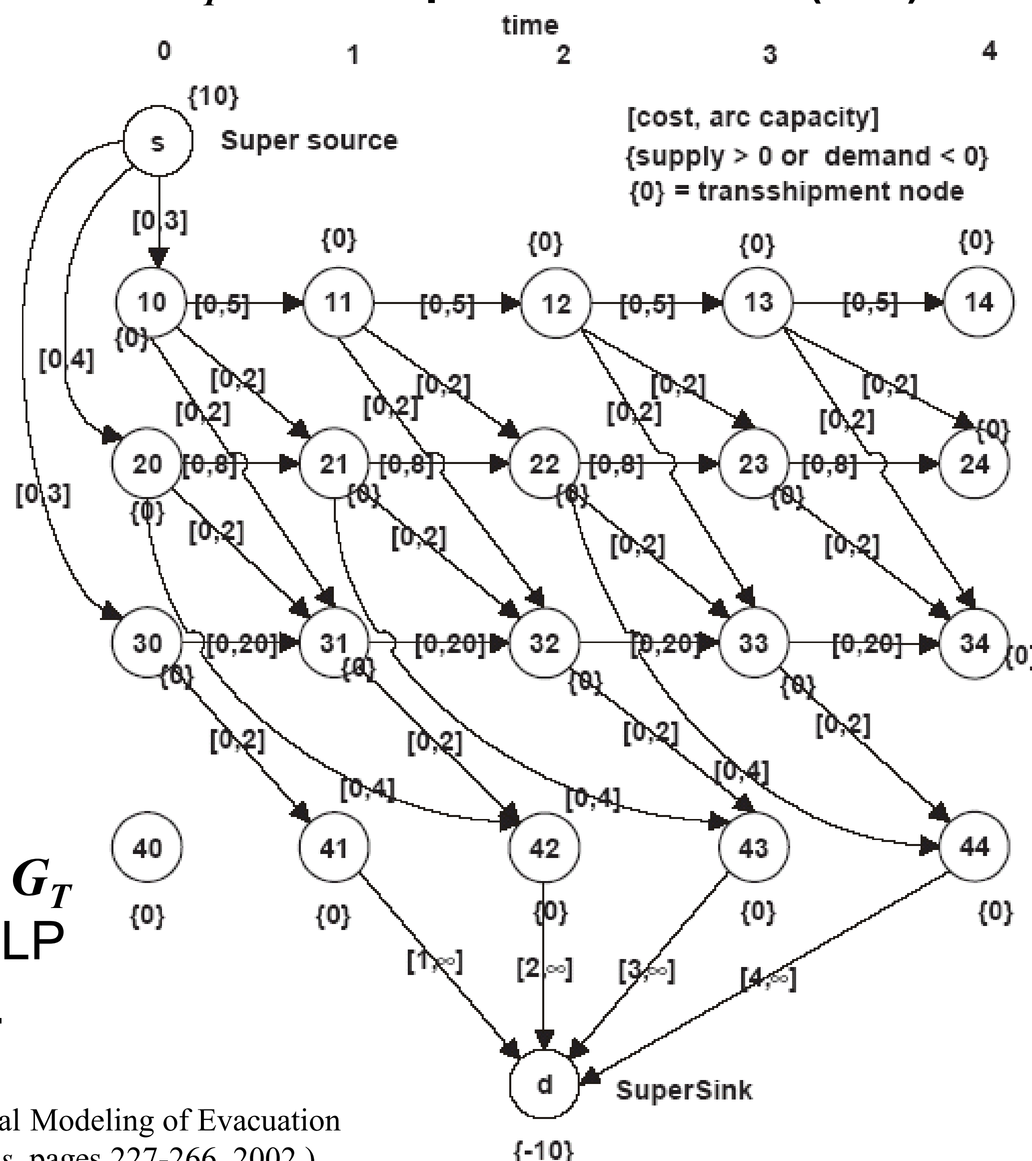
- Routes used only by old plan
- Routes used only by result plan of capacity constrained routing
- Routes used by both plans

Related Works: Linear Programming Approach

G : evacuation network



G_T : time-expanded network (T=4)



Step 1: Convert evacuation network into time-expanded network with user provided time upper bound **T**.

Step 2: Use time-expanded network G_T as a flow network and solve it using LP min. cost flow solver (e.g. NETFLO).

(Source of network figures: H. Hamacher, S. Tjandra, Mathmatical Modeling of Evacuation Problems: A state of the art. *Pedestrian and Evacuation Dynamics*, pages 227-266, 2002.)

Related Works

Linear Programming Approach

- Optimal solution for evacuation plan
- e.g. EVACNET (U. of Florida), Hoppe and Tardos (Cornell University).

Limitation:

- High computational complexity
- Cannot apply to large transportation networks

Number of Nodes	50	500	5,000	50,000
EVACNET Running Time	0.1 min	2.5 min	108 min	> 5 days

Capacity-ignorant Approach

- Simple shortest path computation
- e.g. EXIT89(National Fire Protection Association)

Limitation:

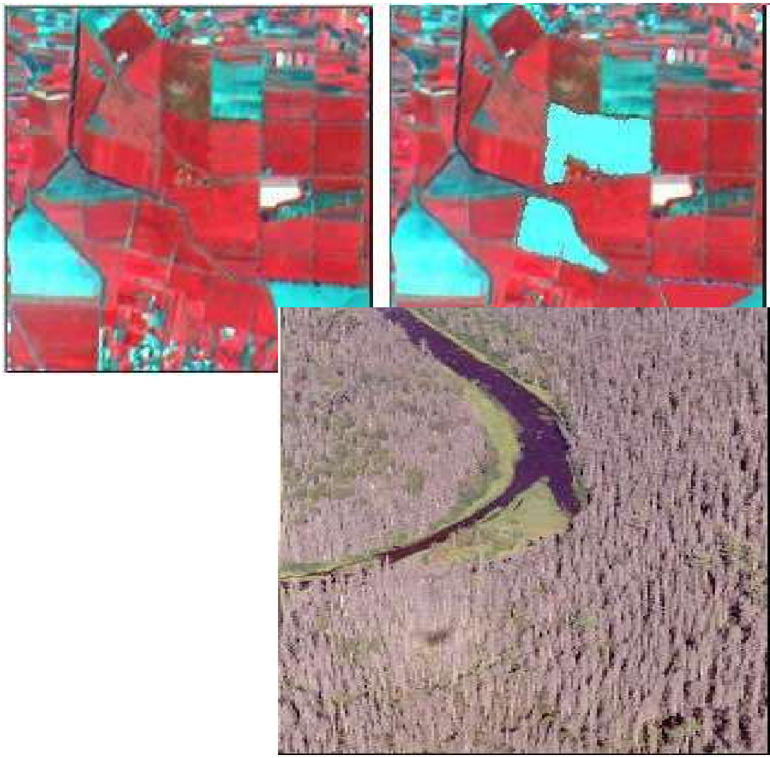
- Do not consider capacity constraints
- Very poor solution quality

Overview

- Motivation for HPC with Spatial Data
- Case Study 1: Simple to Parallelize
  - Multi-Scale Multi-Granular Classification
  - Motivation, Problem Definition
  - Serial Version
  - Alternative parallelization
  - Evaluation
- Case Study 2 – Harder
- Case Study 3 – Hardest
- Wrap-up

Multiscale Multigranular Image Classification

- Applications
  - Land-cover change Analysis
  - Environmental Assessment
  - Agricultural Monitoring

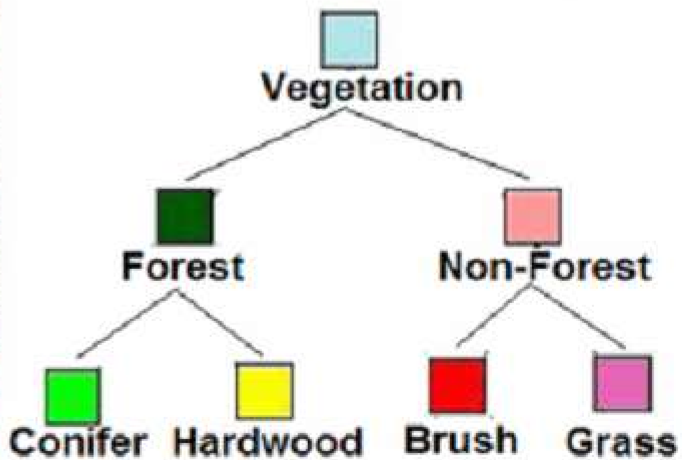


- Challenges
  - Expensive computation of Quality Measure (i.e. likelihood)
  - Large amount of data
  - Many dimensions

Spatial Applications: An Example

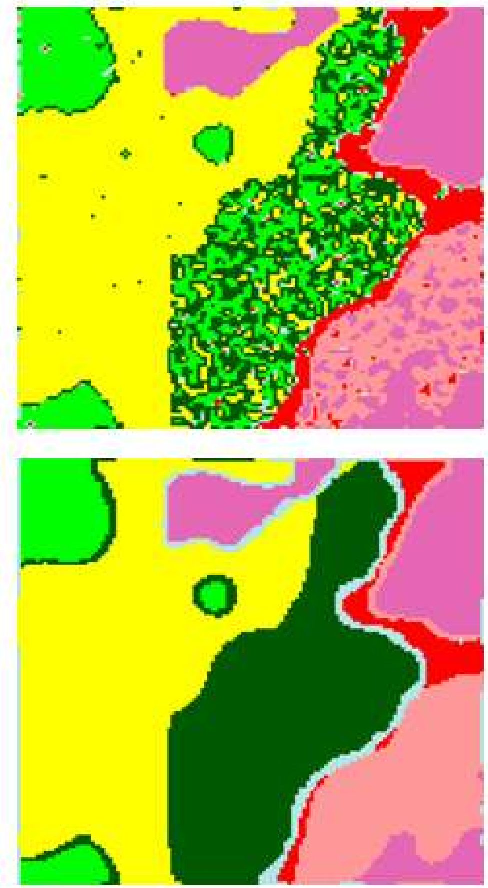
- Multiscale Multigranular Image Classification

Remotely-sensed Satellite Image



Multigranular Land-use Classification

Inputs



Output Images at Multiple Scales

MSMG Classification – Formulation

- Model
  - $$\hat{M} \equiv \arg\max_{M} \{ l(x \mid M) - \lambda\, pen(M) \}$$
    - *x* : observations
    - *M* : a classification model
    - *l(x / M)* : log-likelihood (Quality Measure) of M
    - *pen* : Penalty function

  - Calculation of log-likelihood of M
    - Uses Expectation Maximization
    - Computationally Expensive
      - 7 hours of Computation time for an input image of size 512 x 512 pixels with 4 Classes

Pseudo-code : Serial Version

1. Initialize parameters.
2. **for** each **Class**
3.   **for** each **Spatial Scale**
4.   **for** each **Quad**
5.   Calculate Quality Measure (i.e., log-likelihood)
8.   end **for** Quad
9.   end **for** Spatial Scale
10. end **for** Class
11. Post-processing

Q? What are the options for parallelization?

Parallelization – Problem Definition

- Given
  - Serial version of a Spatial Data Mining Algorithm
  - Likelihood of each specific class at each pixel
  - Class-hierarchy
  - Maximum Spatial Scale
- Find
  - Parallel formulation of the algorithm
- Objective
  - How effective is UPC in parallelizing spatial applications? (Speedup)
  - How effective is UPC in improving productivity of researchers in spatial domain?
- Constraints
  - Platform: Parallel Global Address Space (PGAS), Unified Parallel C (UPC)

Challenges in Parallelization

Description of work

– Compute Quality Measure for combinations of Class-label, Scale, Quad (Spatial Unit)

Challenges

a) **Variable workload** across computations of quality measure

b) **Many dimensions** to parallelize

    i.e. Class-label, Scale, Quad

c) **Dependency** across scales

Quad-level Parallelization

1. Initialize parameters and memory
2. **for** each Spatial Scale
3.   **upc_forall** Quad
4.   **for** each Class
5.   Calculate Quality Measure
6   end **for** Class
7.   end **upc_forall** Quad
8.   **upc_barrier**
9. end **for** Spatial Scale
10. Post-processing

Quad-level Parallelization

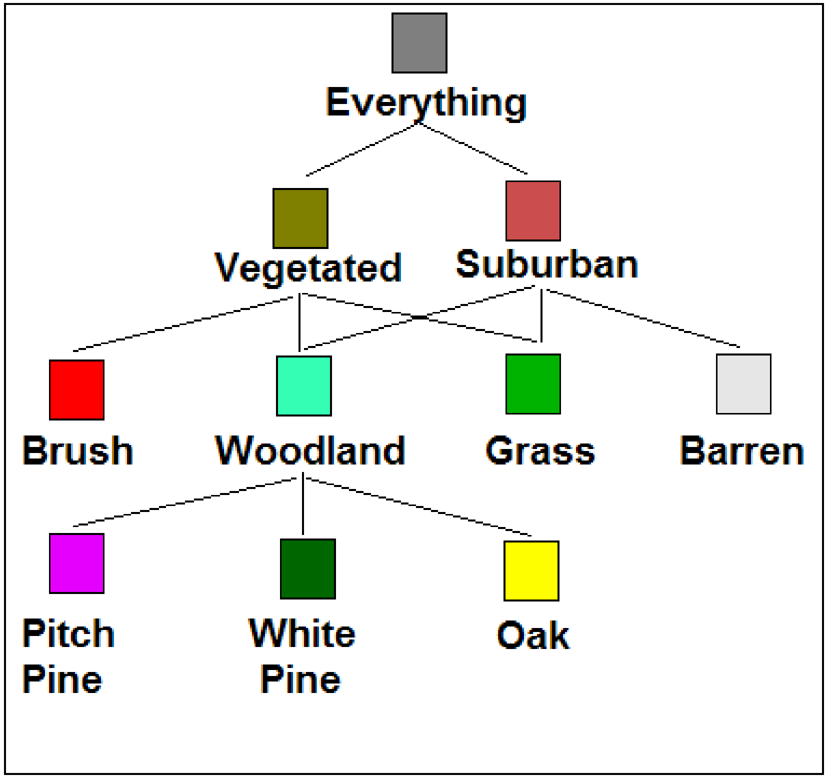
- Advantages:
  - Workload distribution is more even
  - Greater number of processors can be used
  - Number of Quads = f (Number of pixels)
    - Example
      - Input: 4 Classes, Scale of 6

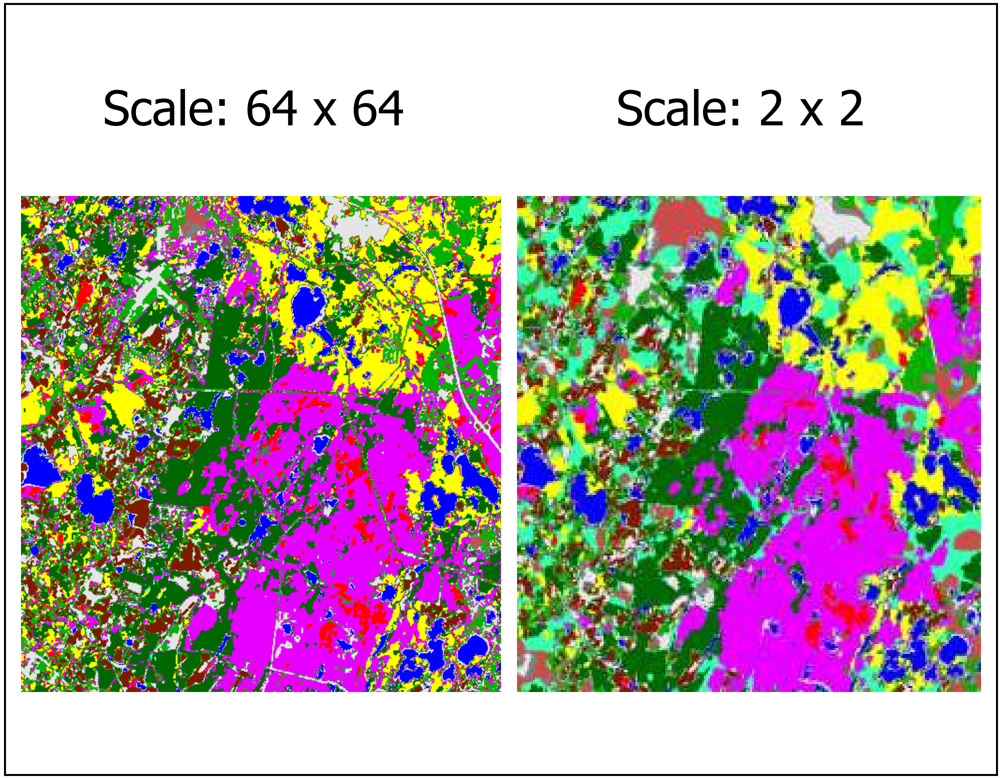
Input Image Size	Number of Quads
64 x 64	98,304
128 x 128	393,216
512 x 512	6,291,456
1024 x 1024	25,165,824

Experimental Design

Input	• 64 x 64 pixels image (Plymouth County, Massachusetts) • 4 class labels (Everything, Woodland, Vegetated, Suburban)
Language	UPC
Hardware Platform	Cray X1
Number of Processors	1-8

Workload

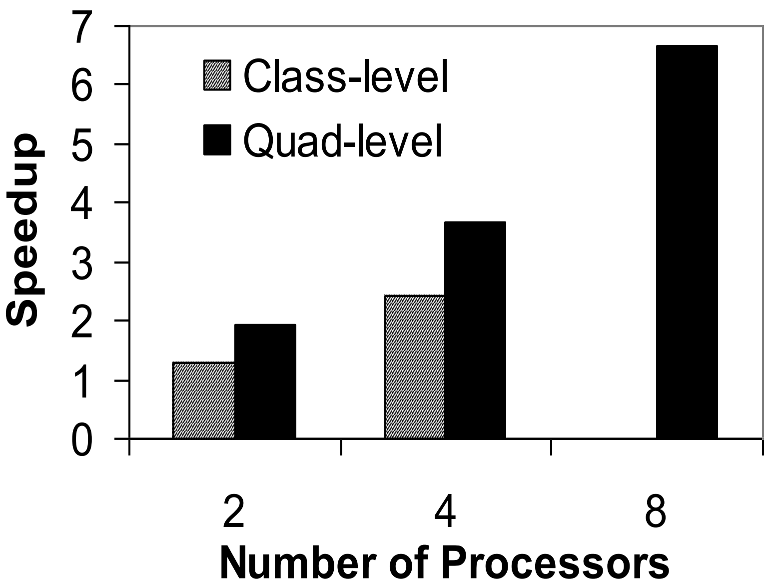


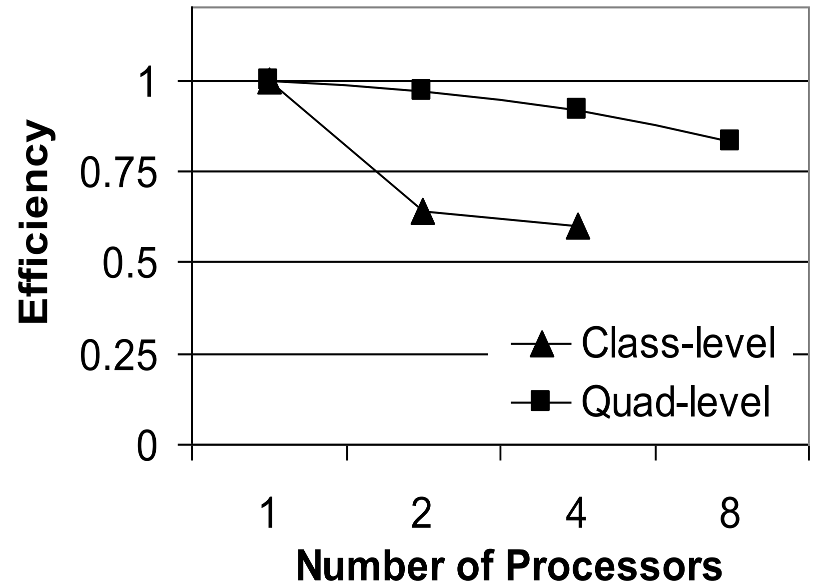


Input class hierarchy

Output Images at Multiple scales

Effect of Number of Processors



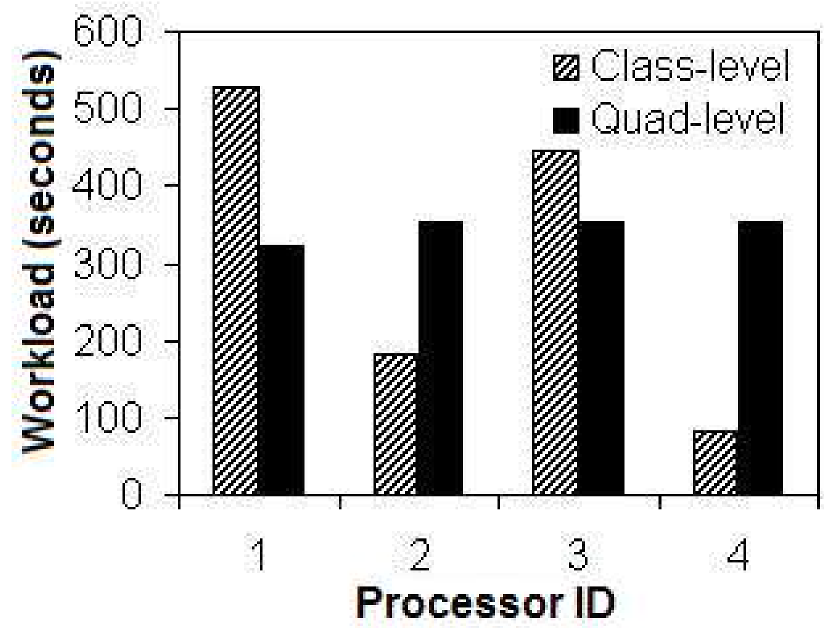


- Quad-level parallelization gives better speed-up
- Room for Speed-up for both approaches

- Q? Class-level << Quad-level. Why?

Workload Distribution

Fixed Parameter
- Number of processors: 4



- Quad-level parallelization provides better load-balance
- Probably because of large number of Quads (~100,000)

Findings - I

- How effective is UPC in parallelizing Spatial applications?
  - Quad-level parallelization
    - Speed-up of 6.65 on 8 processors
    - Large number of Quads (98,304)
  - Class-level parallelization
    - Speed-ups are lower
    - Smaller number of Classes (4)

Findings - II

- How effective is UPC in improving productivity of researches in spatial domain?
  - Coding effort was reduced
    - 20 lines of new code in program with base size of 2000 lines
    - 1 person-month
  - Analysis effort refocused
    - Identify units of parallel work i.e. Quality Measure
    - Identify dimensions to parallelize i.e. Quad, Class, Scale
    - Selecting dimension(s) to parallelize
      - Dependency Analysis (Ruled out Scale)
      - Number of Units (Larger the better)
      - Load Balancing
    - 6 person-month

Future Work

- Improve Efficiency
  - Explore Dynamic Load Balancing
  - Other parallel formulations

Overview

- Motivation for HPC with Spatial Data
- Case Study 1: Simple to Parallelize
- Case Study 2 – Harder
  - Spatial Databases: Parallelizing Range Query
  - Basic Concepts & Problem Formulation
  - Declustering Spatial Data
  - Dynamic Load Balancing (DLB) for Spatial Data
- Case Study 3 – Hardest
- Wrap-up

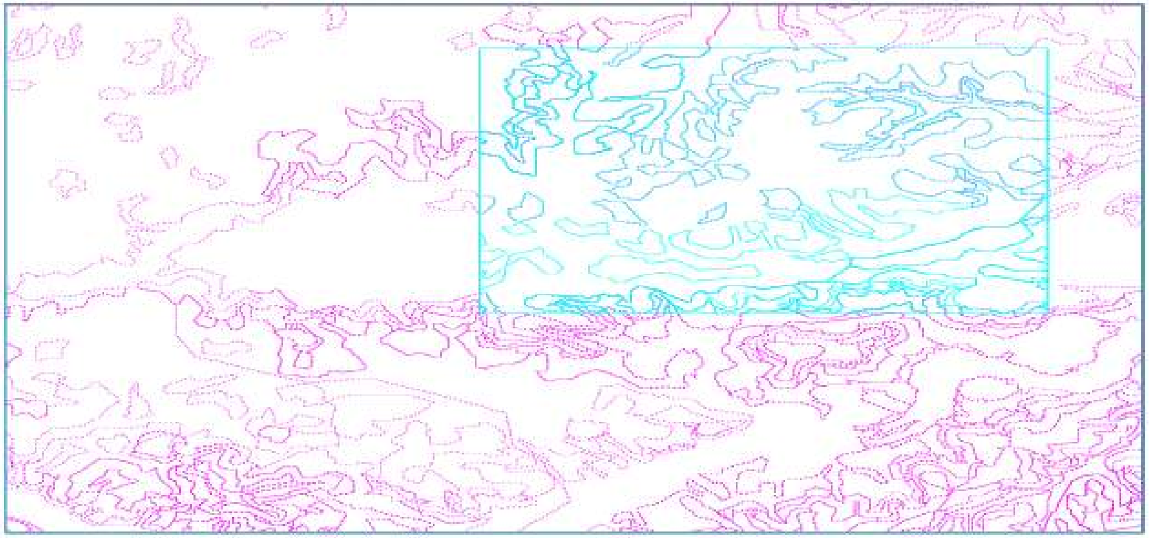
Range Query - Motivation

- GIS Based Augmented/Virtual Reality
  - GIS fetches polygons (feature-data plus elevation-data)
  - Transform polygons to 3-D objects (e.g. trees, buildings, etc.)
  - Render 3-D objects

- Other Applications
  - RealTime Terrain Visualization
  - Interactive Situation Assessment
  - Interactive Spatial Decision Making, etc.

Range Query Motivation - 2

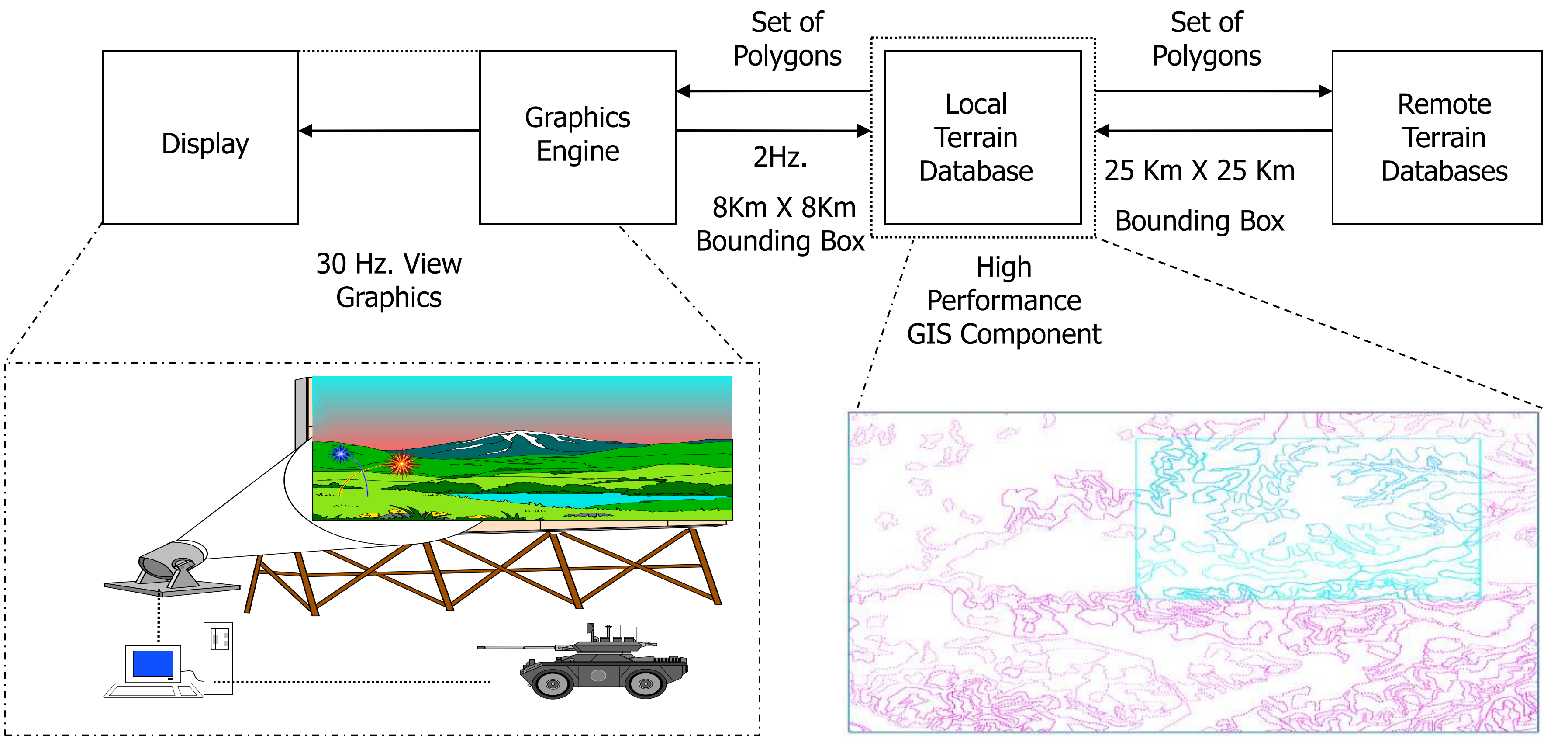
- High Performance Requirements (mid-1990s)
  - limit on response time: < 1/2 sec
  - latest processors can process < 1500 polygons in 1/2 sec
  - typical requirement: process more than 10,000 polygons
- An Example Map



- Q?. Are Sequential Approaches Adequate?
- Q?. Can Parallel Solutions Meet the Requirements?

A Case Study: High Performance GIS

- (1/30) second Response time constraint on Range Query
- Parallel processing necessary since best sequential computer cannot meet requirement
- Blue rectangle = a range query, Polygon colors shows processor assignment

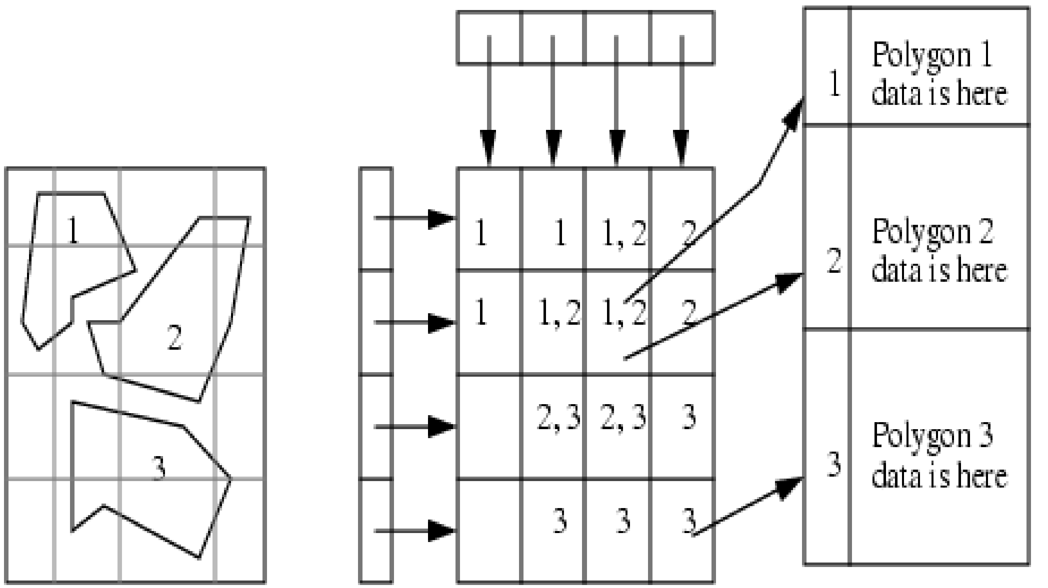


A Sequential Algorithm

- Sequential Solution: 3-parts
  - Approximate but Fast Polygon Filtering
    - Point-Data
      - Search Trees: K-d trees, Range-trees [Preparata,Shamos], etc.
      - Access Methods: GridFile [Nievergelt].
    - Polygon-Data
      - Extend plus Customize point-based access methods: R-Trees [Guttman].
- Intersection Computation
  - Edge-Filter and Brute-Force
  - Plane Sweep [Bentley,Ottmann]
  - Iterative-Based [Sugihara]
- Polygonization
  - Scan clipped line-segments
  - Scan intersection points

Approximate but Fast Polygon Filtering

- Grid-Directory



(a) Polygons in 2-d space

(b) Grid Directory and Buckets

(c) Data Structure for storing polygon data

- Each grid cell contains the list of polygons intersecting the cell
- Indices on both x and y axis

Problem Formulation

- Parallelize range-query
  - Goal: Minimize response time for a set of range-queries
- Alternative Approaches
  - Function-Partitioning
    - each task executes a certain function
    - uses specialized data structures
    - e.g., parallel plane-sweep
  - Data-Partitioning
    - each task executes the same function, but on different data
    - divides data among different processors
    - execute the sequential algorithm at each processor
- Focus: Data-Partitioning Approaches

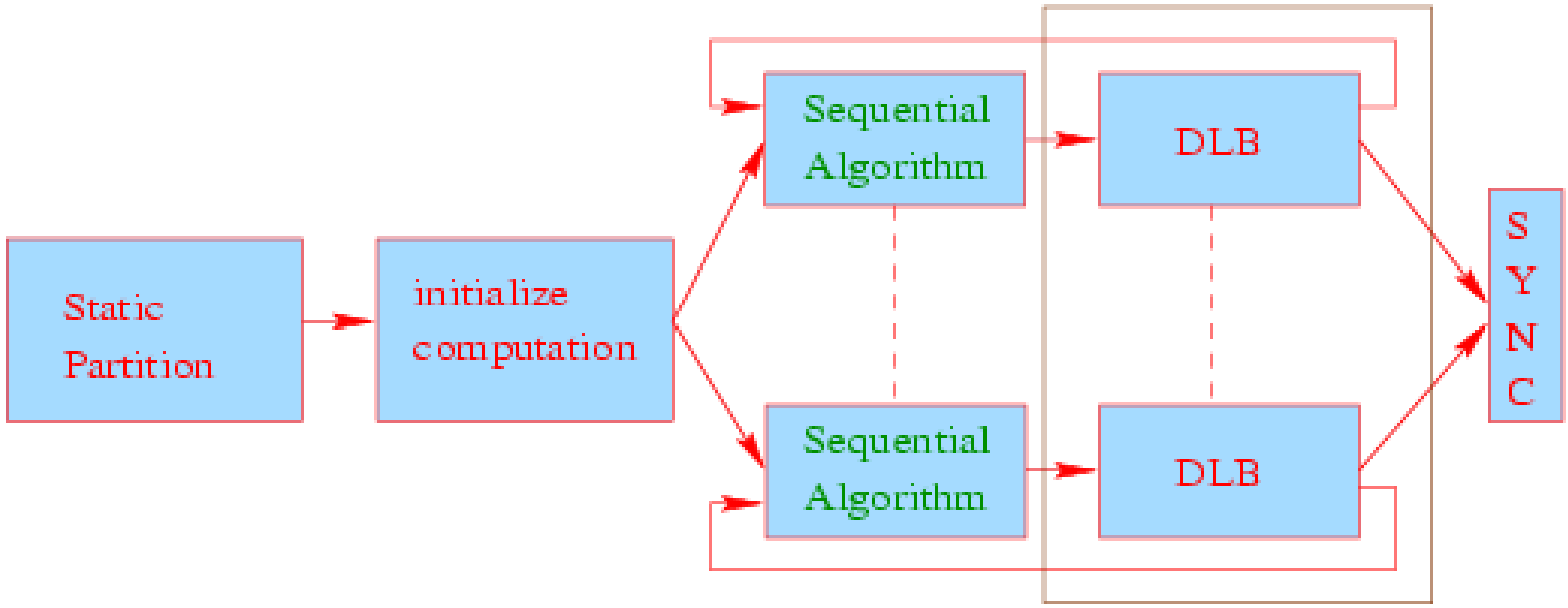
Issues in Data-Partitioning

- Partitioning the data
  - partition polygons at different levels
    - edges, sub-polygons, polygons
  - partition range-query
    - smaller range-queries, edges
- Focus
  - partition data at polygon level
  - Range-query is not partitioned

		Options for dividing polygon data			
		No Division	Subsets of polygons	Subsets of small polygons	Subsets of edges
Options for dividing range-query	No Division	I	II	III	IV
	Divide into small boxes	III	III	III	IV
	Divide into Edges	IV	IV	IV	IV

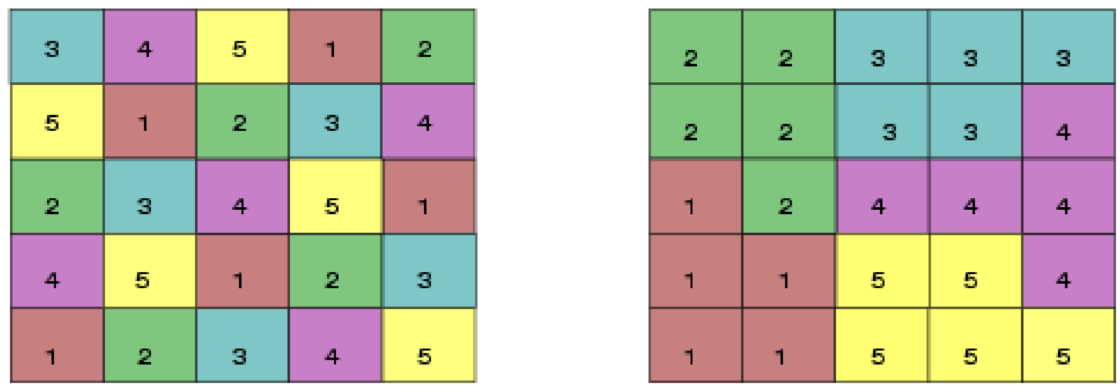
Data-Partitioning Approach

- Initial Static Partitioning
- Run-Time dynamic load-balancing (DLB)



Static-Partitioning & DLB

- Partitioning
  - different from classical parallel computing
  - declustering, not clustering
  - related objects scattered, not grouped
- What is declustering ?



- What is DLB
  - redistributes work at runtime
  - data may be transferred between processors

Scope of this Work

- No pre-computation except index
- Main-Memory database
- Each range-query is independent
- Data-partitioning, not function-partitioning
- Granularity of partition is polygon

Declustering Polygonal Maps

Example:

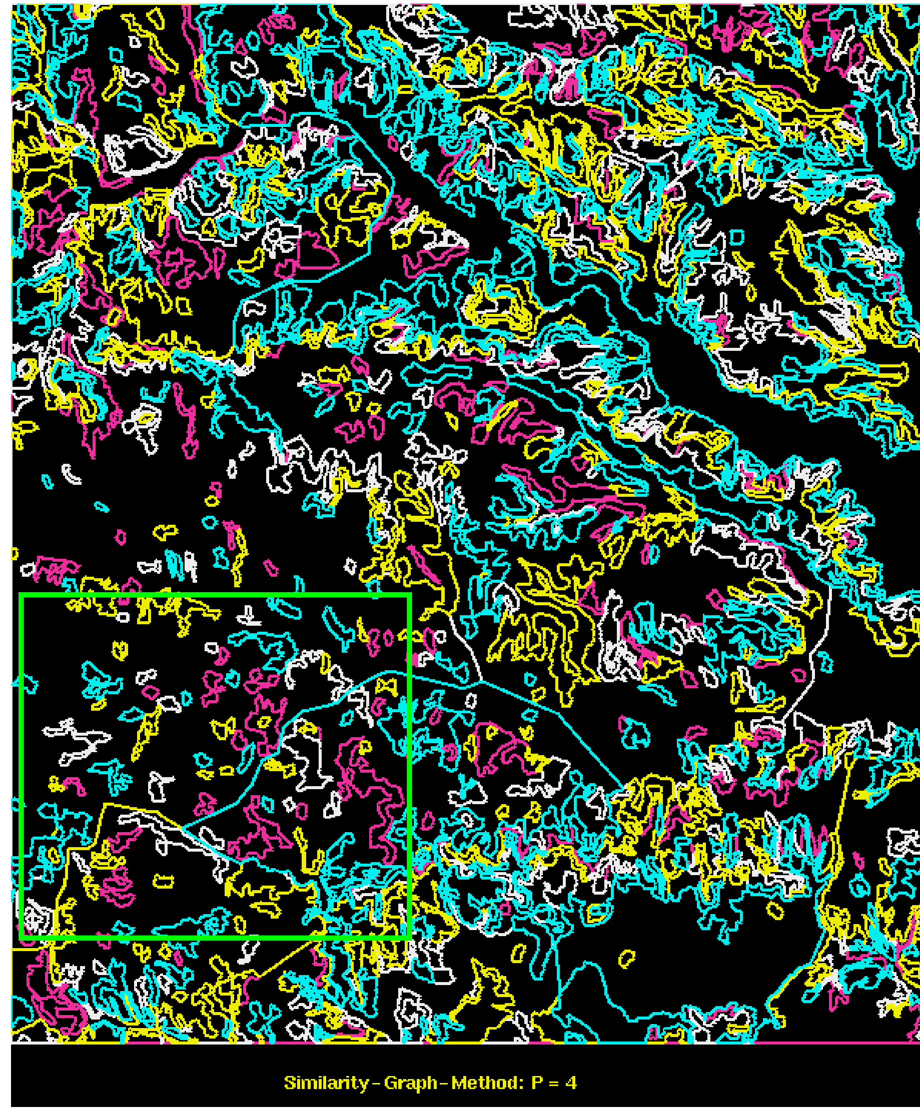
- Dividing a Map among 4 processors.
- Polygons within a processor have common color
- Green rectangle = a range query

Goals of declustering

- balance load acrosss proessors
- for arbitrary range query

Challenges

- Large polygons – upper right
- Variation in polygon sizes



Declustering Spatial Data

- Goal of Declustering
  - partition the data so that each partition imposes exactly the same load for any range-query
- **<u>Theorem: Declustering is NP-hard</u>**

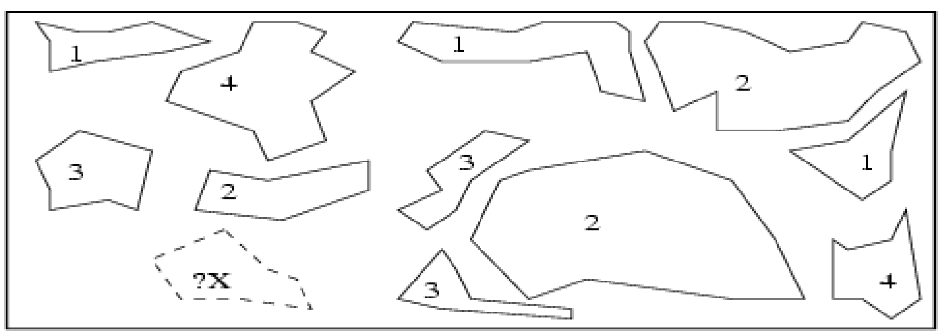
- Definition: Declustering Problem
  - Given:
    - Set S of extended-objects
    - P processors
    - Set $Q = (Q_1 \ldots. \; Q_n)$ of n range-queries
  - Partition the set S among P processors
  - Such that:
    - load at each processor is balanced for all $Q_i \in Q$
  - Where load of an object $x \in S$ for a given range-query Q_i is given by
    - $f_Q^i : \rightarrow Z$
    - Z is the set of nonnegative integers

Issues in Declustering Polygonal Maps

- Declustering method
- Work-Load metric
- Spatial-Extent of workload
- Distribution of the workload over spatial-extent

Declustering Methods

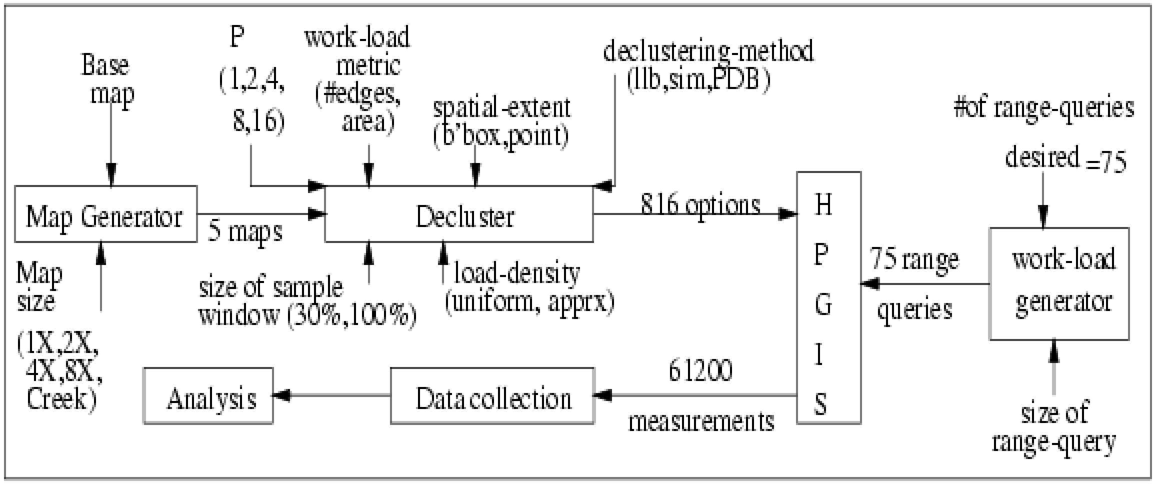
- Space-Partitioning
  - Hilbert [Bially (69), Jagadish (90)]
- Local Load-Balance [kamel,faloutsos(92)]
  - Geometric-Based, e.g. area, count
  - f(data-distribution, window-size, insertion-order)
- (Dis-)Similarity-Graph [liu(95)]
  - Topological and geometric based
  - f(query, neighbor, window-size, insertion-order)



- Count load-balance: ?X = 4
- Area load-balance: ?X = 3
- (Dis-)Similarity: ?X = 1
- Q. Which is the better choice ?

Experiment Design

- Key Questions
  - Which workload metric is better?
  - How to approximate the spatial-extent?
  - Which declustering method is better?
- Experimental Methodology

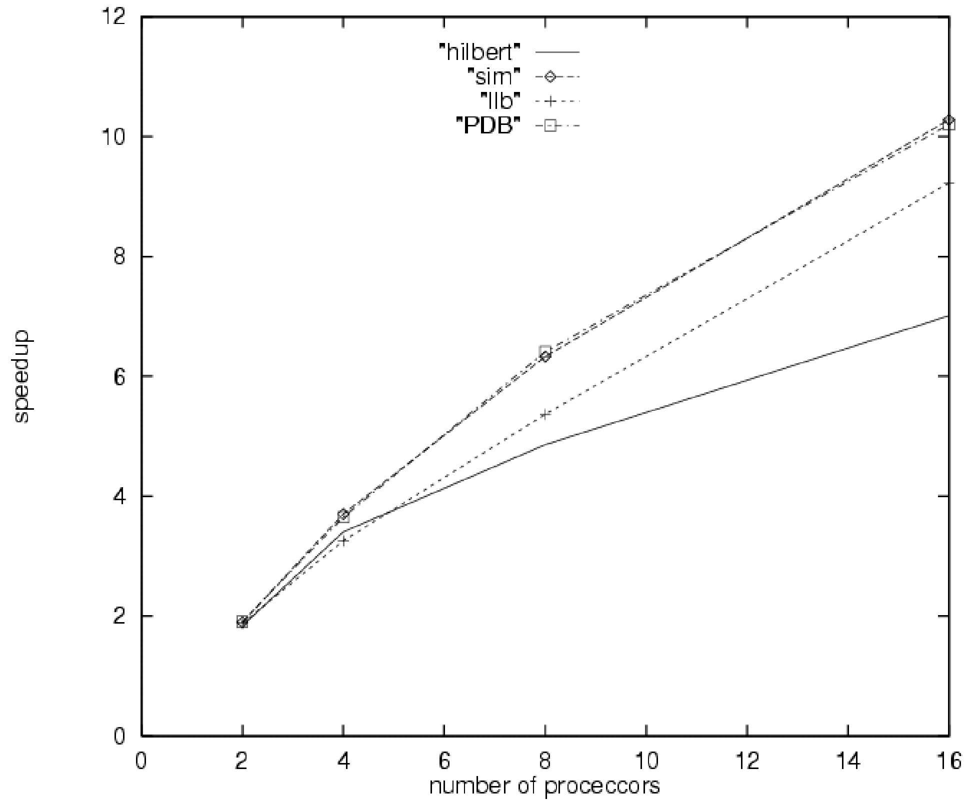


- Experiments conducted using vector data

Map	#objects	#edges	range-query	
			size	number
1X	729 polygons	41162	25%	75
2X	1458 polygons	82324	25%	75
4X	2916 polygons	164648	25%	75
8X	5832 polygons	329296	25%	75
16X	11664 polygons	658592	25%	75
Creek	9667 chains	188678	20%	75

Comparison of Declustering Methods

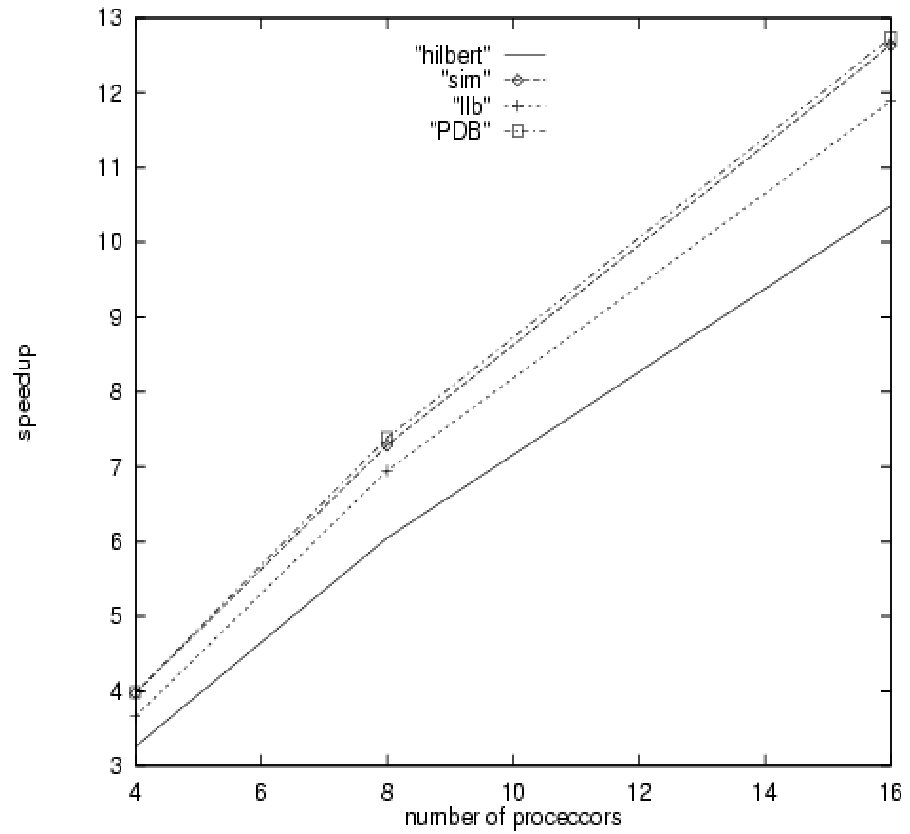
- Fixed Parameters
  - Map = 4X
  - 75 range-queries; each fetching 12KM x 12KM
  - Load-balancing = static only (no DLB)
  - work-load metric = no. of edges, work-load dist. = uniform
  - Spatial-extent = MBR



- Trends
  - [Similarity, PDB] > LLB > Hilbert
  - Efficiency < 75% for P >= 16

Comparison of Declustering Methods

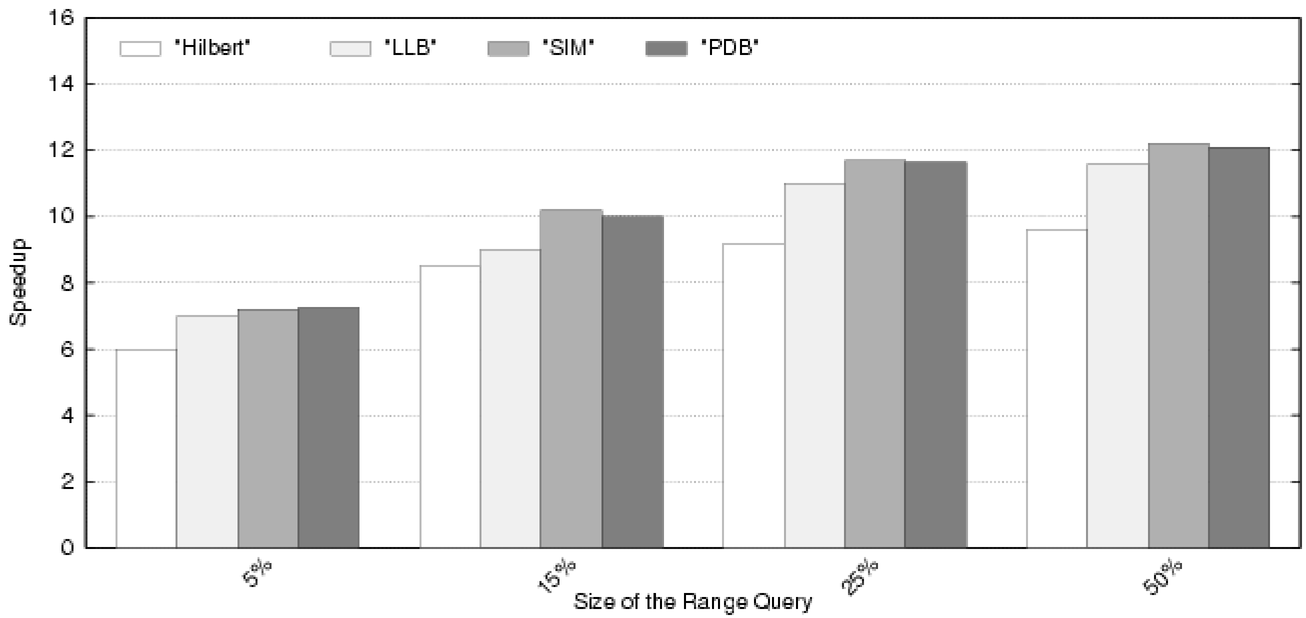
- Fixed Parameters
  - Map = Creek
  - 75 range-queries; each fetching 12KM x 12KM
  - Load-balancing = static only (no DLB)
  - work-load metric = no. of edges, work-load dist. = uniform
  - Spatial-extent = MBR



- Trends
  - [Similarity, PDB] > LLB > Hilbert
  - Efficiency < 75% for P >= 16

Effect of Range-Query Size

- Fixed Parameters
- Map = 4X
- Load-balancing = static only (no DLB)
- workload metric = no. of edges, workload dist. = uniform
- Spatial-extent = MBR



Dynamic Load-Balancing (DLB)

- DLB is used if static declustering methods fail to balance the load
- Issues in DLB
  - Methods for transferring work
  - Partitioning method & Granularity of work transfers
  - Which processor should an idle processor ask for more work?

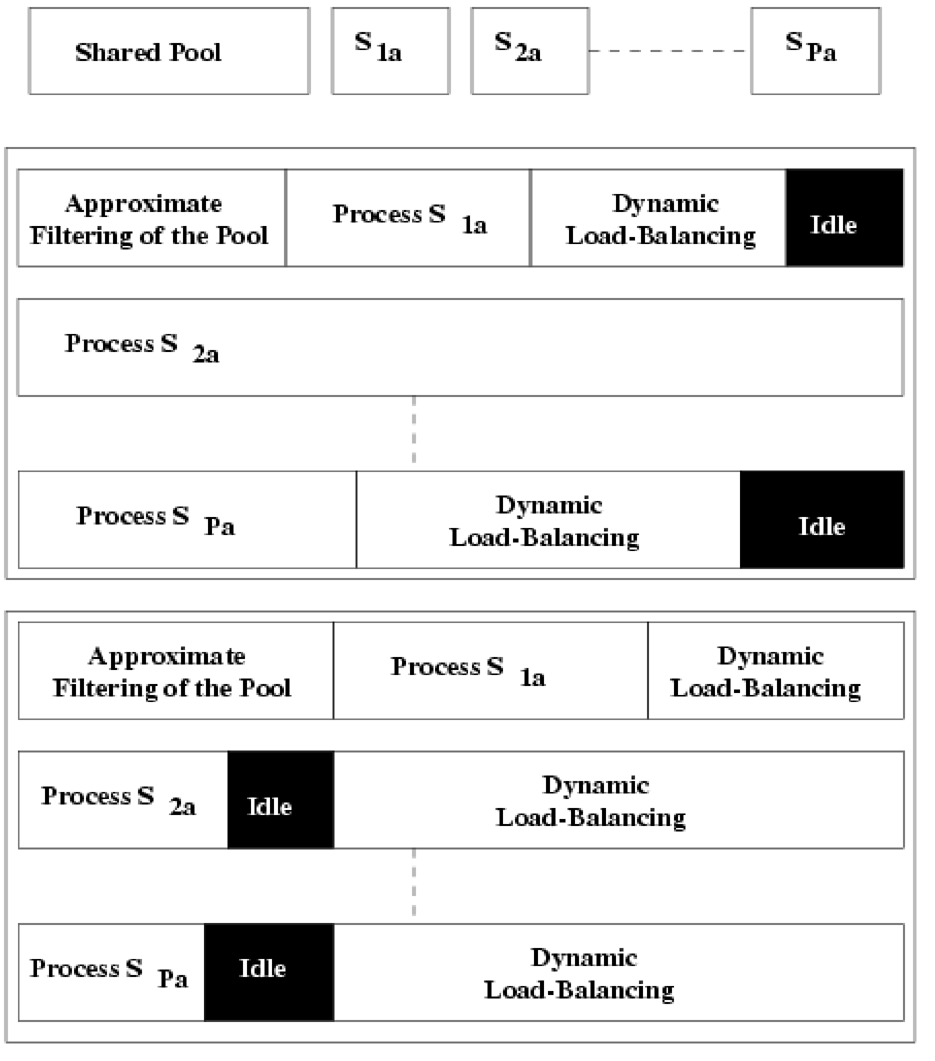
DLB: Methods for Transferring Work

- Extended spatial-objects are large (ß 1K)
- Cost of transfer ? cost to solve the problem locally
  - cost of transfer is linear
  - cost of solving the problem is sub-linear due to filtering

Time (sec)	Q_1	Q_2	Q_3	Q_4	Q_5	Avg. over 75 queries
T_t	0.78	0.63	1.06	0.84	0.46	0.764 ± 0.022
T_s	0.39	0.22	0.53	0.51	0.33	0.362 ± 0.086

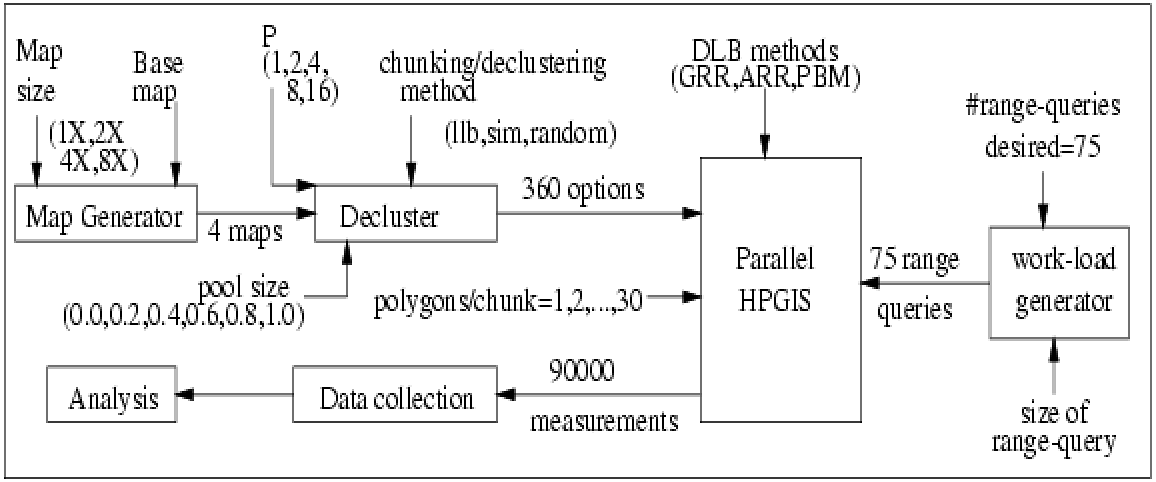
- A possible solution?
  - transfer only the polygon IDs
  - selectively duplicate the polygon data at different processors

Pool-Size Choice is Challenging!



DLB Methods: Experiment Design

- Key Questions:
  - What is the chunk size of work transfer ?
  - How to partition work into chunks ?
  - Who to ask for more work ?
- Experimental Methodology



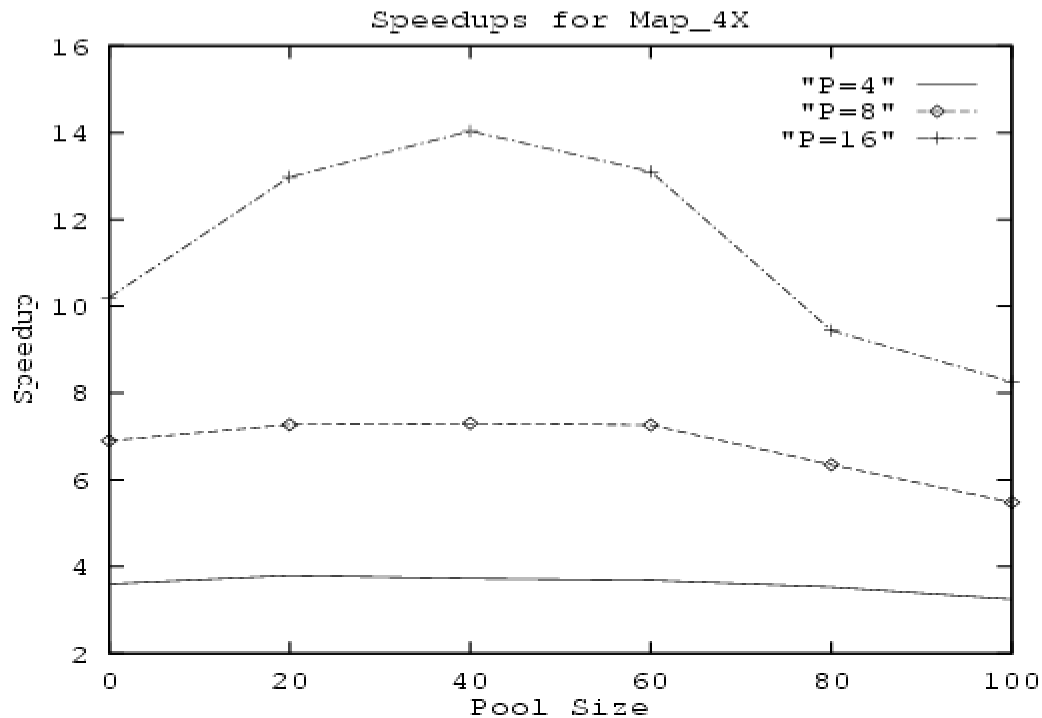
- Experiments conducted using vector data

DLB: Experiment Design

- Hardware
  - CrayT3D (mid-1990s)
    - Distributed memory with fast interconnection network
    - Each node is a DEC-Alpha (150MHz)
  - SGI Challenge
    - Shared-Memory with fast bus
    - Each node is a MIPS 4400 (200MHz)
- Software
  - Sequential code (8K lines) plus communication routines
  - C processes with partial shared address space
  - Shared-Memory library (both T3D and SGI)

Effect of Pool Size on PBM methods

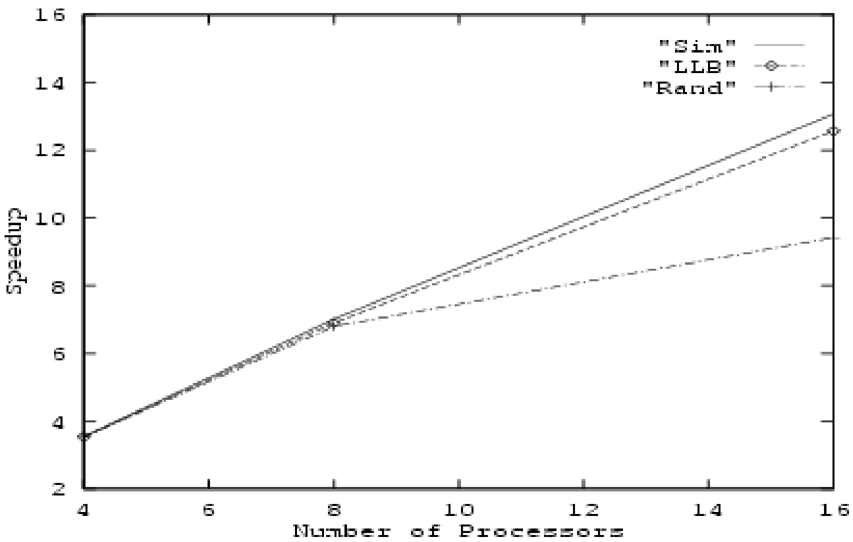
- Fixed Parameters
  - Map size = 4X; Map = Killeen, TX (25KM x 25KM)
  - 75 range-queries; each fetching 12KM x 12KM
  - Load-balancing = static, then dynamic



- Note
  - 0% pool is SLB; 100% pool is pure DLB
- Trends
  - Pool-Size around 40%60% gives peak speedup
  - SLB then DLB >= [SLB, DLB] for speedup

How to Partition the work into chunks?

- Fixed Parameters
  - Map = 4X; Hardware = Cray T3D
  - 75 range-queries; each fetching 12KM by 12KM
  - Load-balancing = static, then dynamic
  - DLB Method: GRR, SLB method: sim, llb, random



- Trends
  - Similarity ? LLB ? Random (see color maps)
  - Ranking of methods same in DLB and SLB
  - Random declustering is worse than informed declustering
- Other Results
  - DLB has better speedup than SLB: 13 for P=16

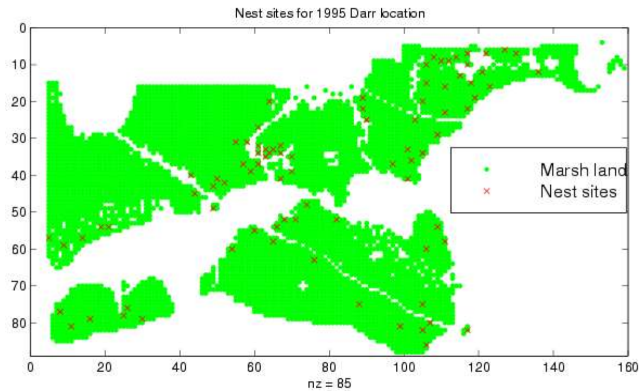
Research Contributions

- Static Load-Balancing - Declustering Spatial Data
  - proposed new declustering methods outperform traditional methods
  - neither static declustering nor DLB alone are sufficient to provide good speedups
  - declustering can be used to improve the performance of DLB
- Dynamic Load-Balancing (DLB)
  - How to manage work transfers?
    - selectively duplicate the polygons at different processors
    - transfer the polygon Ids
  - How to partition the work?
    - Chunks-cheduling is interesting for spatial data
    - declustering can be used to partition the work into chunks
  - DistributedMemory vs SharedMemory
    - Able to solve bigger maps on sharedmemory machine
    - Declustering is interesting for both the architectures

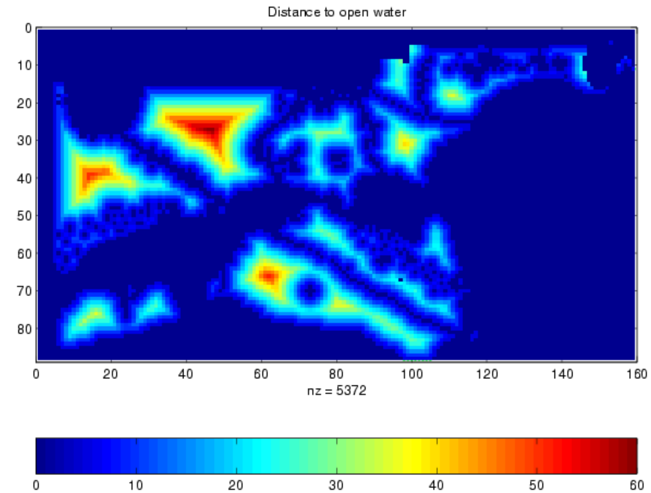
Overview

- Motivation for HPC with Spatial Data
- Case Study 1: Simple to Parallelize
- Case Study 2 – Harder
- <u>Case Study 3 – Hardest</u>
  - Spatial Data Mining: Parallelizing Spatial Auto-regression
    - Key Concepts & Problem Definition
    - Parallel Spatial Auto-Regression
    - Experimental Results
- Wrap-up

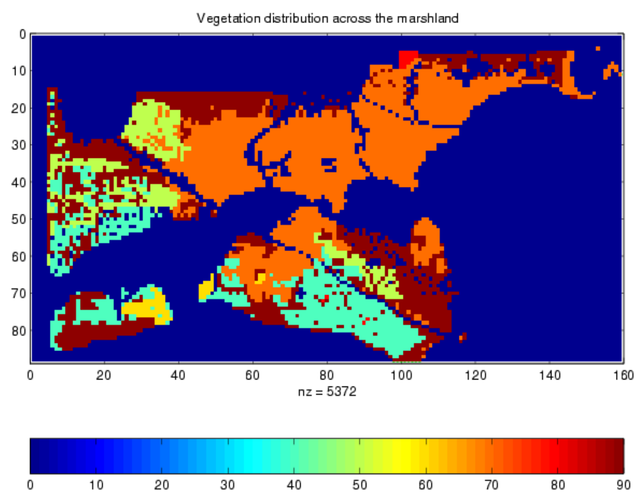
Motivation – Location Prediction



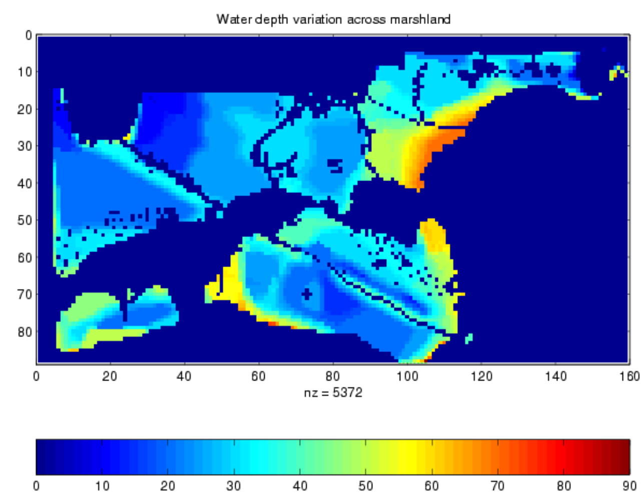
Nest locations



Distance to open water



Vegetation durability



Water depth

Classical and New Data-Mining Techniques

Name	***Model***	***Classification Accuracy***
Classical Linear Regression	$\boxed{\mathbf{y} = \mathbf{x}\boldsymbol{\beta} + \varepsilon}$	Low
Spatial Auto-Regression	$\boxed{\mathbf{y} = \rho\mathbf{W}\mathbf{y} + \mathbf{x}\boldsymbol{\beta} + \boldsymbol{\varepsilon}}$	High

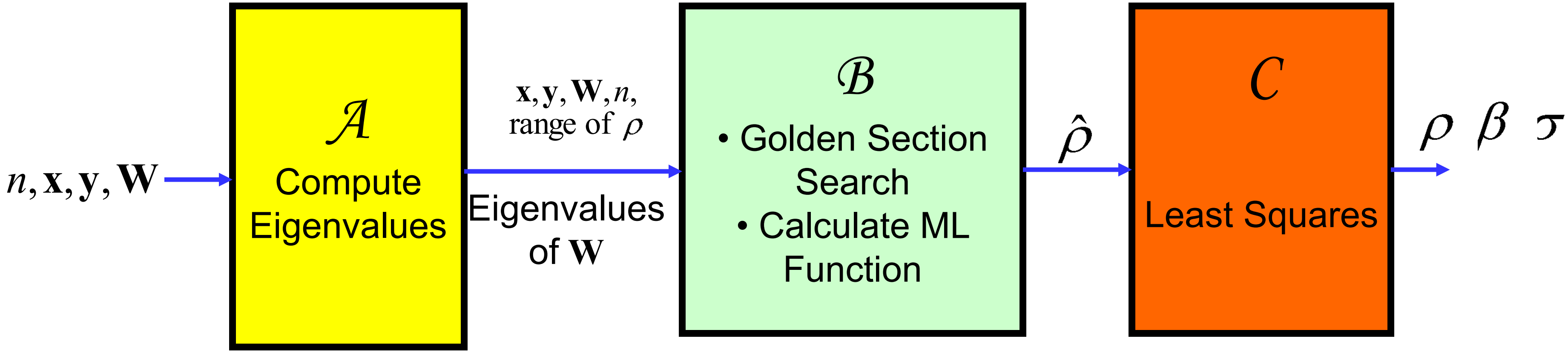
ρ : the spatial auto-regression (auto-correlation) parameter

$\mathbf{W}$: n-by-n neighborhood matrix over spatial framework

- Solving Spatial Auto-regression Model
  - $\rho = 0, \boldsymbol{\varepsilon} = 0$: Least Squares Problem
  - $\boldsymbol{\beta} = 0, \boldsymbol{\varepsilon} = 0$: Eigenvalue Problem
  - General case: Computationally expensive

- **<u>Maximum Likelihood Estimation</u>** $\ln(L) = \boxed{\ln\left|\mathbf{I} - \rho\mathbf{W}\right|} - \frac{n\ln(2\pi)}{2} - \frac{n\ln(\sigma^2)}{2} - SSE$
- Need parallel implementation to scale up

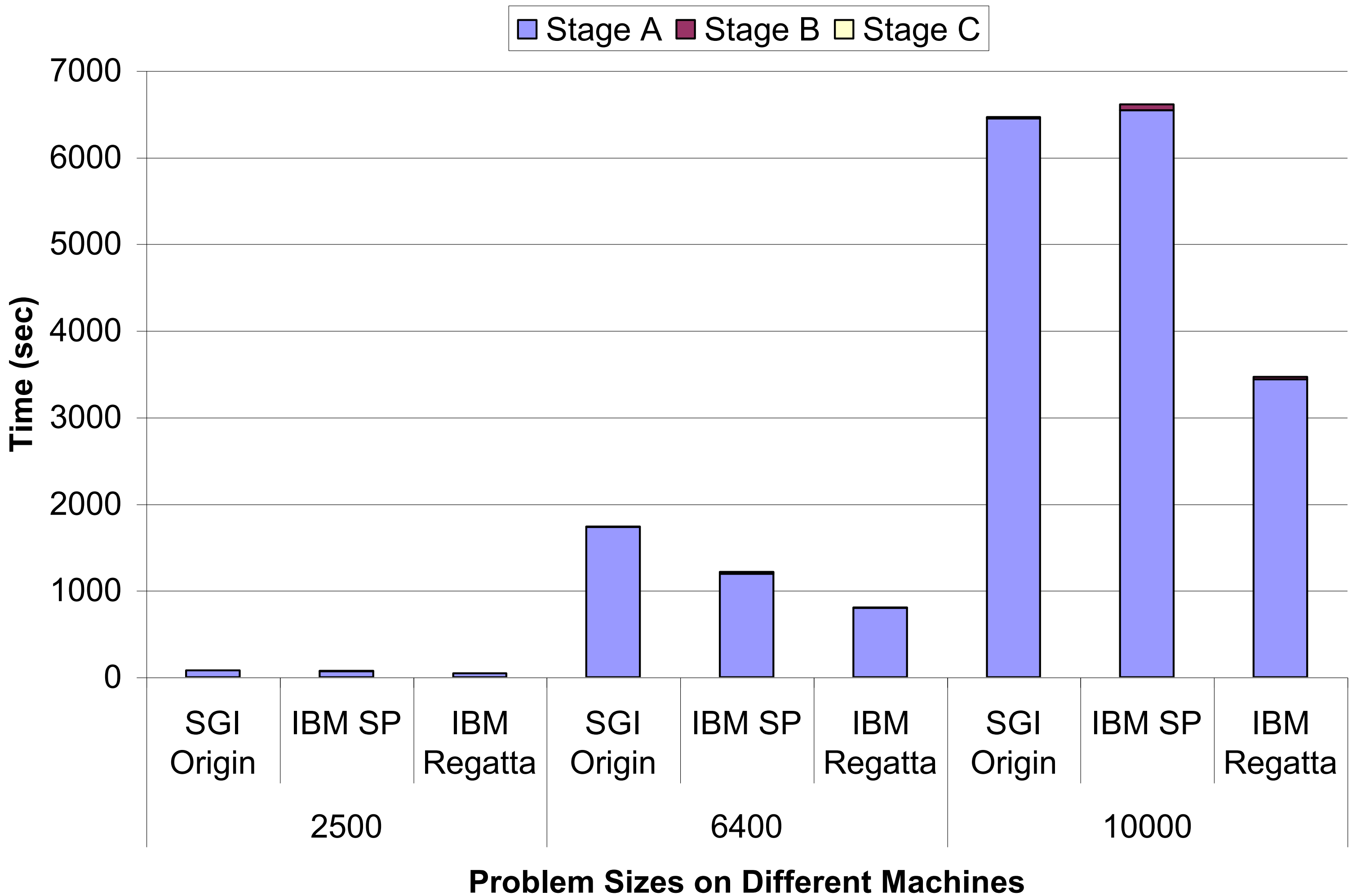
A Serial Solution



- Compute Eigenvalues (Stage $\mathcal{A}$)
  - Produces dense $\mathbf{W}$ neighborhood matrix
  - Forms synthetic data $\mathbf{y}$
  - Makes $\mathbf{W}$ symmetric
  - Householder transformation
    - Convert dense symmetric matrix to tri-diagonal matrix
  - QL Transformation
  - Compute ***all*** eigenvalues of tri-diagonal matrix

Serial Response Times (sec)

- Stage $\mathcal{A}$ is the bottleneck & Stage $\mathcal{B}$ and $\mathcal{C}$ contribute very small to response time



Problem Definition

Given:

- A Sequential solution procedure: "Serial Dense Matrix Approach" for one-dimensional geo-spaces

Find:

- Parallel Formulation of Serial Dense Matrix Approach for multi-dimensional geo-spaces

Constraints:

- $\boldsymbol{\varepsilon} \sim N(0,\sigma^2\mathbf{I})$ IID
- Reasonably efficient parallel implementation
- Parallel Platform
- Size of $\mathbf{W}$ (large vs. small and dense vs. sparse)

Objective:

- Portable & scalable software

Related Work & Our Contributions

- Related work: Li, 1996
  - Limitations: Solved 1-D problem
- Our Contributions
  - Parallel solution for 2-D problems
  - Portable software
    - Fortran 77
    - An Application of Hybrid Parallelism
    - MPI messaging system
    - Compiler directives of OpenMP

Our Approach – Parallel Spatial Auto-Regression

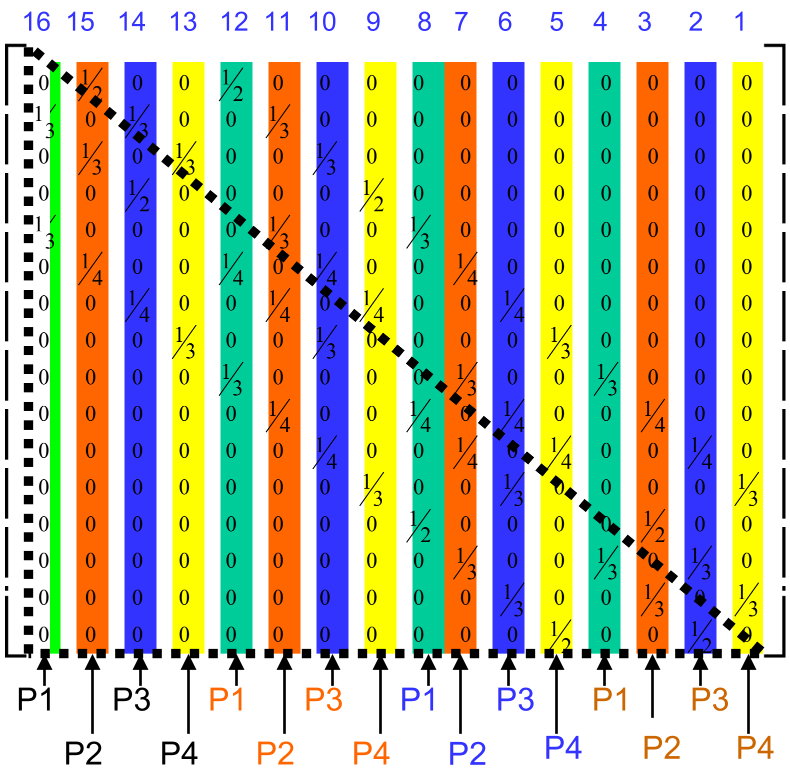
- Function vs. Data Partitioning
  - Function partitioning: Each processor works on the same data with different instructions
  - Data partitioning (**applied**): Each processor works on different data with the same instructions
- Implementation Platform:
  - Produces dense **W** neighborhood matrix
  - Fortran with MPI & OpenMP API's
- No machine-specific compiler directives
  - Portability
  - Help software development and technology transfer
- Other Performance Tuning
  - Static terms computed once

Data Partitioning in a Smaller Scale

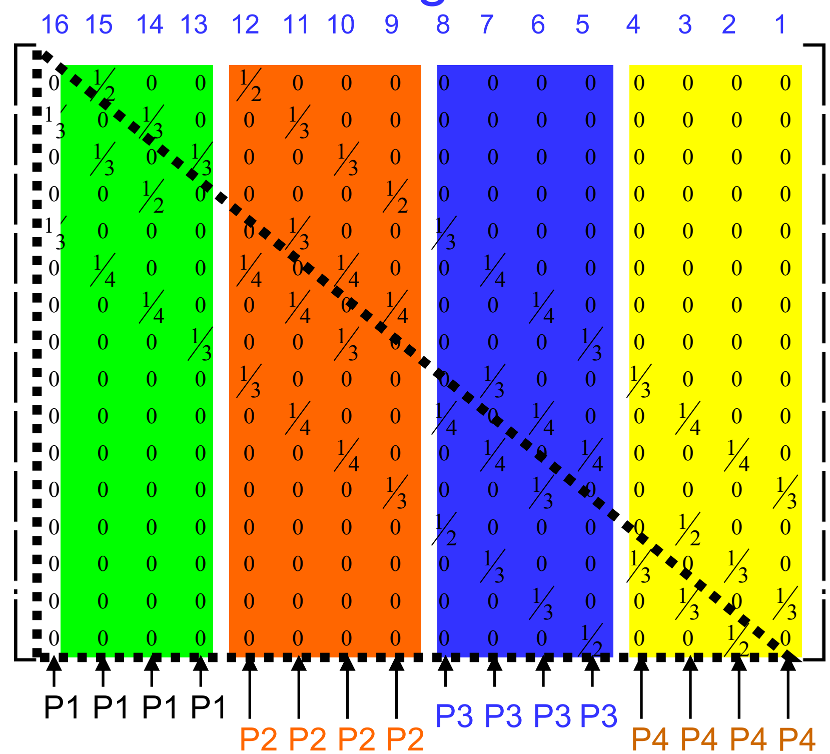
- 4 processors are used and
  chunk size can be determined by the user
- **W** is 16-by-16 and partitioned across processors

P1- (**40** vs. **58**)
P2- (36 vs. 42)
P3- (32 vs. 26)
P4- (**28** vs. **10**)

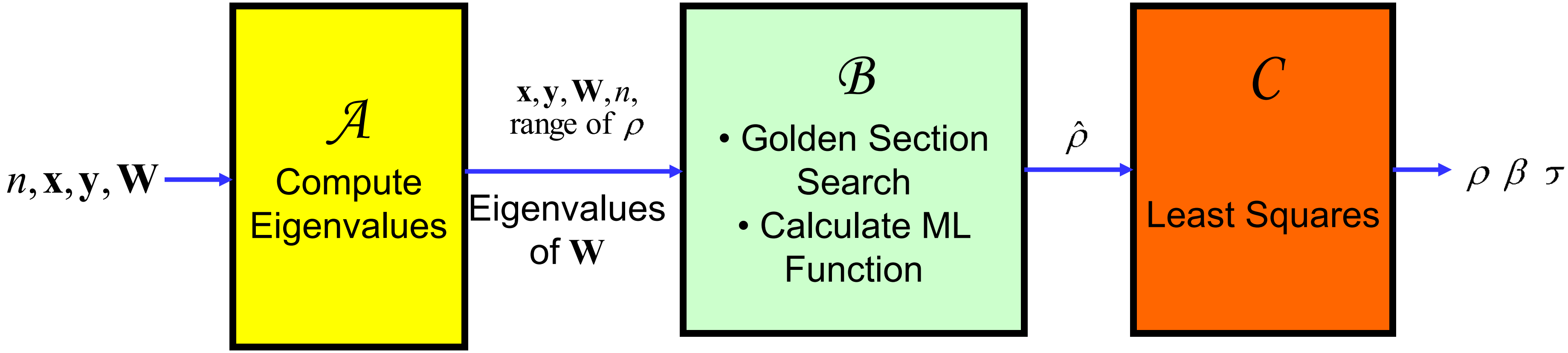
Round-robin with chunk size 1



Contiguous



Data Partitioning & Synchronization



- $\mathcal{A}$: Contiguous for rectangular loops & round-robin with chunk-size 4
- $\mathcal{B}$: Contiguous
- $\mathcal{C}$: Contiguous
- The arrows are also synchronization points for parallel solution

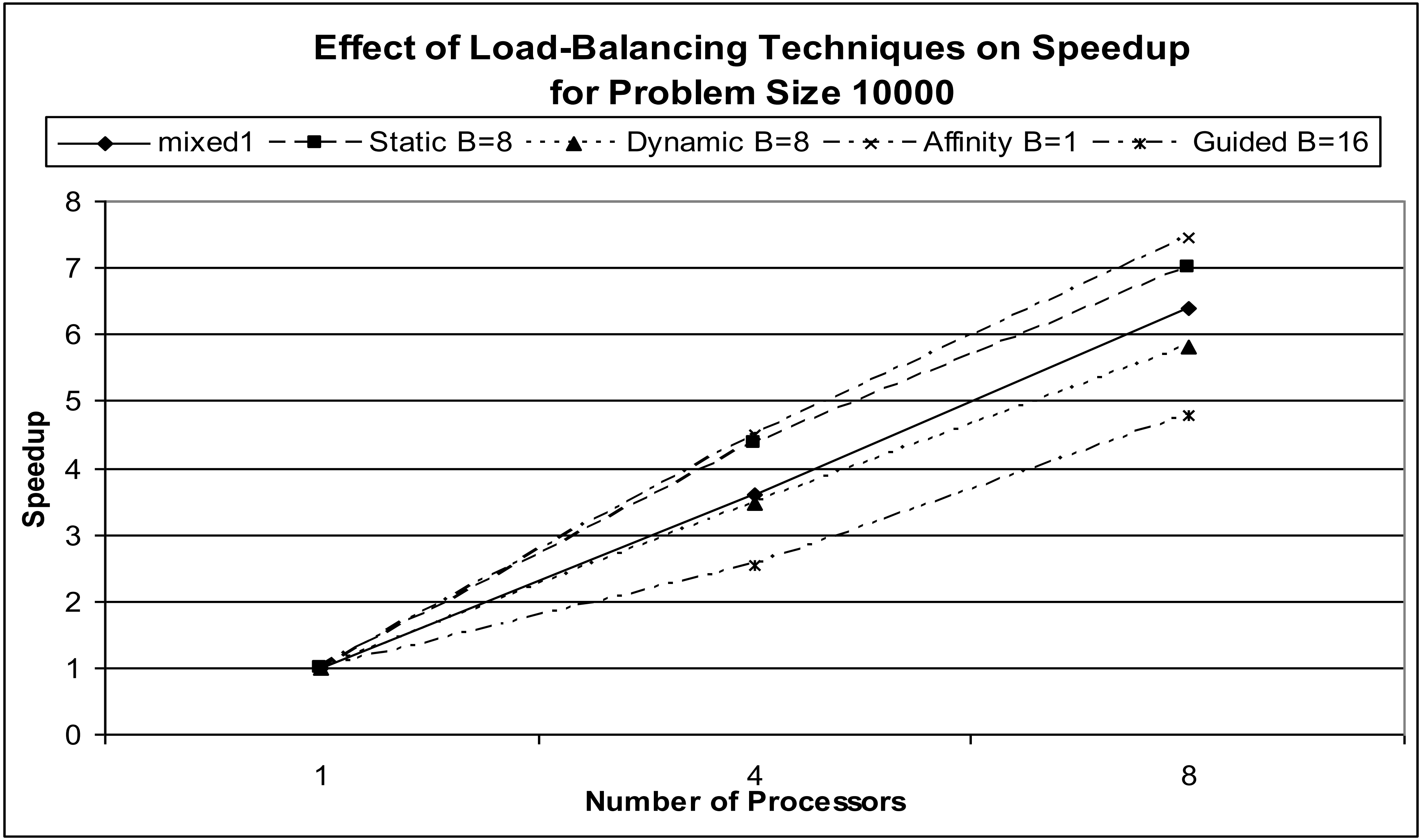
$$\mathcal{A} \rightarrow \mathcal{B} \rightarrow \mathcal{C}$$

- There are synchronization points within the boxes as well

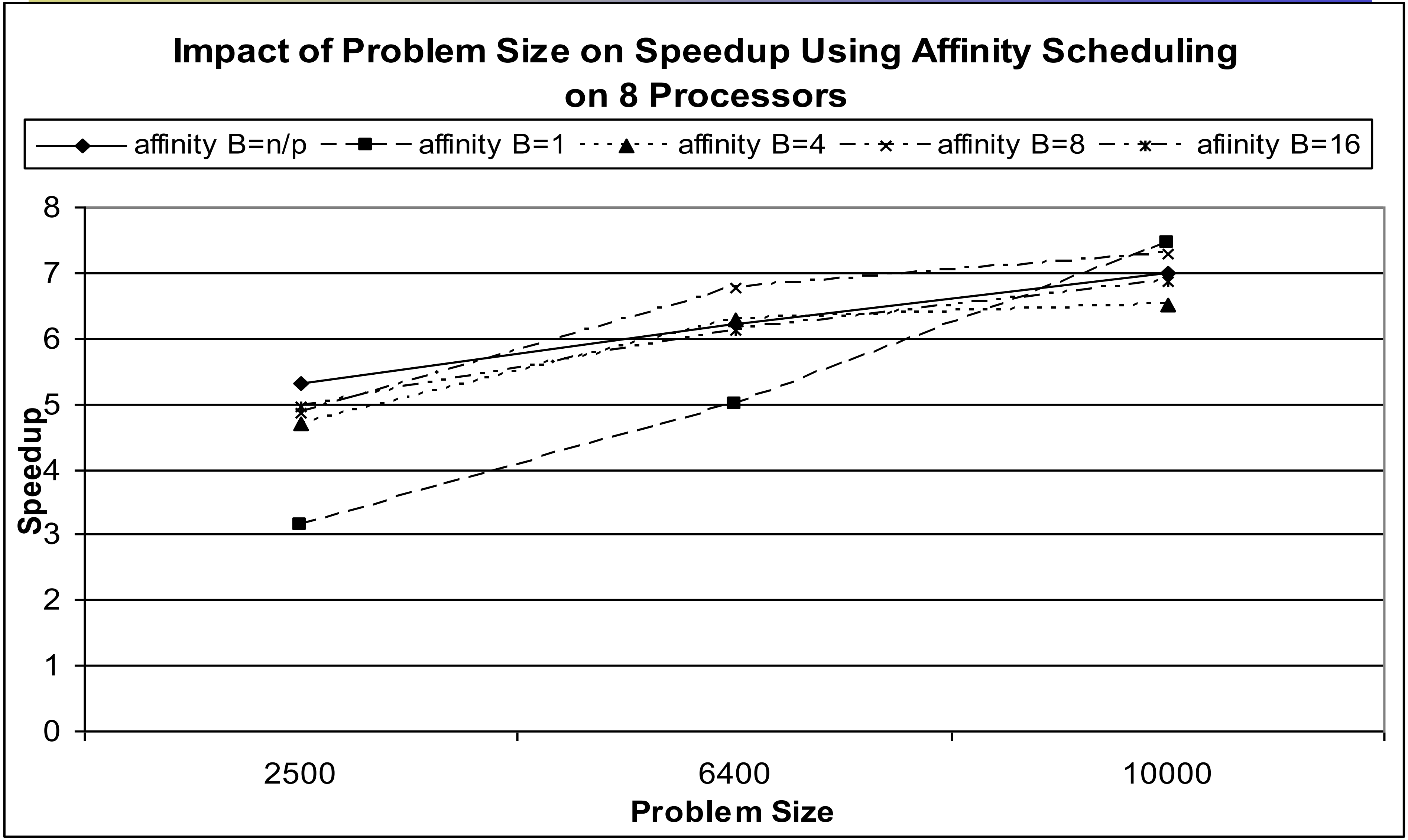
Experimental Design

Factor Name	Parameter Domain	
Language	f77 w/ OpenMP & MPI	
Problem Size (n)	2500,6400 and 10000 observation points	
Neighborhood Structure	2-D w/ 4-neighbors	
Method	Maximum Likelihood for exact SAM	
Auto-regression Parameter	[0,1)	
Load-Balancing	*SLB*	Contiguous ($B=n/p$)
		Round-robin w/ $B=\{1,4,8,16\}$
		Combined (Contiguous+Round-robin)
	DLB	Dynamic w/ $B=\{n/p,1,4,8,16\}$
		Guided w/ $B=\{1,4,8,16\}$
	MLB	Affinity w/ $B=\{n/p,1,4,8,16\}$
Hardware Platform	IBM Regatta w/ 47.5 GB Main Memory; 32 1.3 GHz Power4 architecture processors	
Number of Processors	1,4, and 8	

Experimental Results – Effect of Load Balancing



Experimental Results- Effect of Problem Size



Parallel SAR - Summary

- Developed a parallel formulation of spatial auto-regression model
- Estimates maximum likelihood of regular square tessellation 1-D and 2-D planar surface partitionings for location prediction problems
- Used dense eigenvalue computation and hybrid parallel programming

Parallel SAR - Future Work

1. Understand reasons of inefficiencies
   - Algebraic cost model for speedup measurements on different architectures
2. Fine tune implemented parallel formulation
   - Consider alternate parallel formulations
3. Parallelize other serial solutions using sparse-matrix techniques
   - Chebyshev Polynomial approximation
   - Markov Chain Monte Carlo Estimator

Overview

- Motivation for HPC with Spatial Data
- Case Study 1: Simple to Parallelize
- Case Study 2 – Harder
- Case Study 3 – Hardest
- Wrap-up

Wrap-Up

- Motivation for HPC with Spatial Data
  - Large volume of data, e.g. geo-spatial intelligence
  - Time-critical applications, e.g. moving targets, disaster response
  - Intellectual Challenges!

- Presented a few case Studies
  - Not all spatial problems are hard to parallelize!
    - Local operations on raster data – MSMG Classification
  - Spatial Databases - Parallelizing Range Query
    - Partitioning spatial data is different!
  - Spatial Data Mining - Parallelizing Spatial Auto-regression
    - Computing determinants (or all Eigenvalues) of large matrices is challenging

- Work has barely begun, many open problems remain!
  - Persistence Surveillance (USDOD)
  - Situation Assessment after a disaster US-DHS (FEMA), …
  - Predictive analysis, knowledge discovery, …