Parallel PySAL

Sergio J. Rey Jason Laura

GeoDa Center for Geospatial Analysis and Computation
School of Geographical Sciences and Urban Planning
Arizona State University

AHM 2013

1. Motivation
   - PySAL
   - CyberGIS Goals
   - Parallel PySAL (pPySAL)
2. Problem
   - Fisher-Jenks Optimal Choropleth Map Classification
   - Parallel Optimizations
3. Summary and Next Steps

Outline

PySAL Team

Serge Rey	Luc Anselin
Charlie Schmidt	Xinyue Ye
Phil Stephens	David Folch
Xun Li	Mark McCann
Julia Koschinsky	Andrew Wilson
Myunghwa Hwang	Dani Arribas
Pedro Admaral	Ran Wei
Jason Laura	Nicholas Malizai
Rob Pahle	Wenwen Li

PySAL Objectives

Leverage Existing Tools Development

- GeoDa/PySpace
- STARS

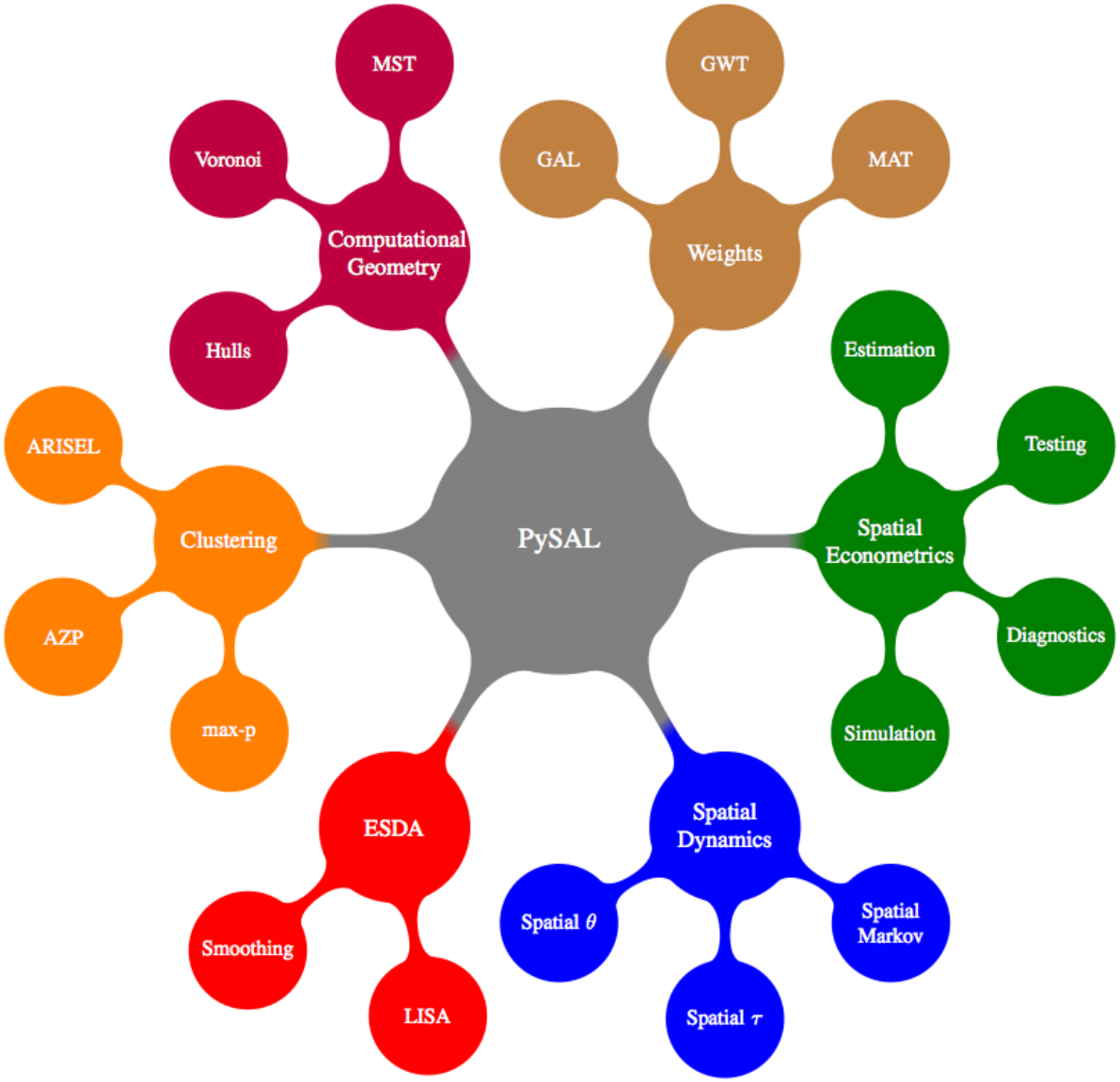
Develop Core Library

- spatial data *analytical* functions
- enhanced specialization, modularity
- fill void in geospatial Python libraries

Flexible Delivery Mechanisms

- interactive shell
- GUI
- **Toolkits**
- webservices

Components


PySAL
Computational Geometry
MST
Voronoi
Hulls
Weights
GWT
GAL
MAT
Clustering
ARISEL
AZP
max-p
Spatial Econometrics
Estimation
Testing
Diagnostics
Simulation
ESDA
Smoothing
LISA
Spatial Dynamics
Spatial θ
Spatial τ
Spatial Markov

Outline

CyberGIS Research Questions

1. What is fundamentally different about how we approach GIS problems based on CI?
2. What new science questions does CyberGIS allow us to address?
3. Will we gain new understanding of geospatial problems with more computing power and richer data?
4. How do we adapt CI to better work with geospatial computation and scientific questions?
5. What needs to be done to adapt existing GIS approaches (e.g., algorithms, data models, etc.) to take advantage of CI?

Outline

PySAL Parallelization Project Goals (pPySAL)

- To create scalable spatial algorithm implementations
- To improve solution quality for currently implemented algorithms
- To explore parallelization techniques for SMP and cluster environments
- To develop a taxonomy of parallelization methods for spatial analysis

pPySAL

Approach

- Map different types of **parallelization** to different **spatial analytics**
- Map different types of **parallelization** to different **modules** in Python
- Explore alternative possibilities using PySAL
  - many parts of the spatial analytical stack
  - more than one-off proof of concept

Existing Work

- Parallel LISA (Local Moran) - Trestles implementation
- Max-p regionalization - SMP implementation
- spreg - spatial regime specifications - SMP implementation
- Fisher-Jenks - SMP (Rey et al. 2013 IJGIS)
  - multiprocessing
  - parallel python
  - pyOpenCL

Outline

Choropleth Mapping

Define a set of k classes, $C_1, C_2, \ldots, C_k$, for n objects such that:

$$\sum_{j=1}^{k} \phi_{i,j} = 1 \; \forall \; i \tag{1}$$

$$\phi_{i,j} = \begin{cases} 1 & \text{if } i \in C_j, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

$$\sum_{j=1}^{k} n_j = n \tag{3}$$

$$\phi_{i,j} = \begin{cases} 1 & \text{if } C_j^l < X_i \leq C_j^u \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where C_j^l and C_j^u are the upper and lower bounds of interval j, and X_i is the observation for enumeration unit i on variable X.

Fisher-Jenks Formulation: Optimal Classification $O(n^k)$

The sequential algorithm has four steps which depart from the sorted attribute values $X_1, X_2, \ldots, X_n$:

1. Compute the diameter $D(I, J)$ for the contiguous group $(I, I+1, \ldots, J)$, for all I, J such that $1 \leq I \leq J \leq n$. The diameter can be based on the variation about the means or the absolute deviations about the class medians. Here we use the total variation.
2. Compute the errors of the optimal partitions, $2 \leq I \leq n$, by $e[P(I,2)] = min[D(1, J-1) + D(J, I)]$ over the range $2 \leq J \leq I$.
3. For each $L : [3 \leq L \leq K]$ compute the errors of the optimal partitions $e[P(I,L)] : (L \leq I \leq n)$ by $e[P(I,L)] = min\{e[P(J-1, L-1)] + D(J,I)\}$ over the range $L \leq J \leq I$.
4. The optimal partition $P(n,k)$ is recovered from the table of errors $e[P(I,L)] : (1 \leq L \leq K, 1 \leq I \leq n)$ by first finding J such that $e[P(n,k)] = e[P(J-1, K-1)] + D(J,n)$.

Outline

Improved SMP Implementation

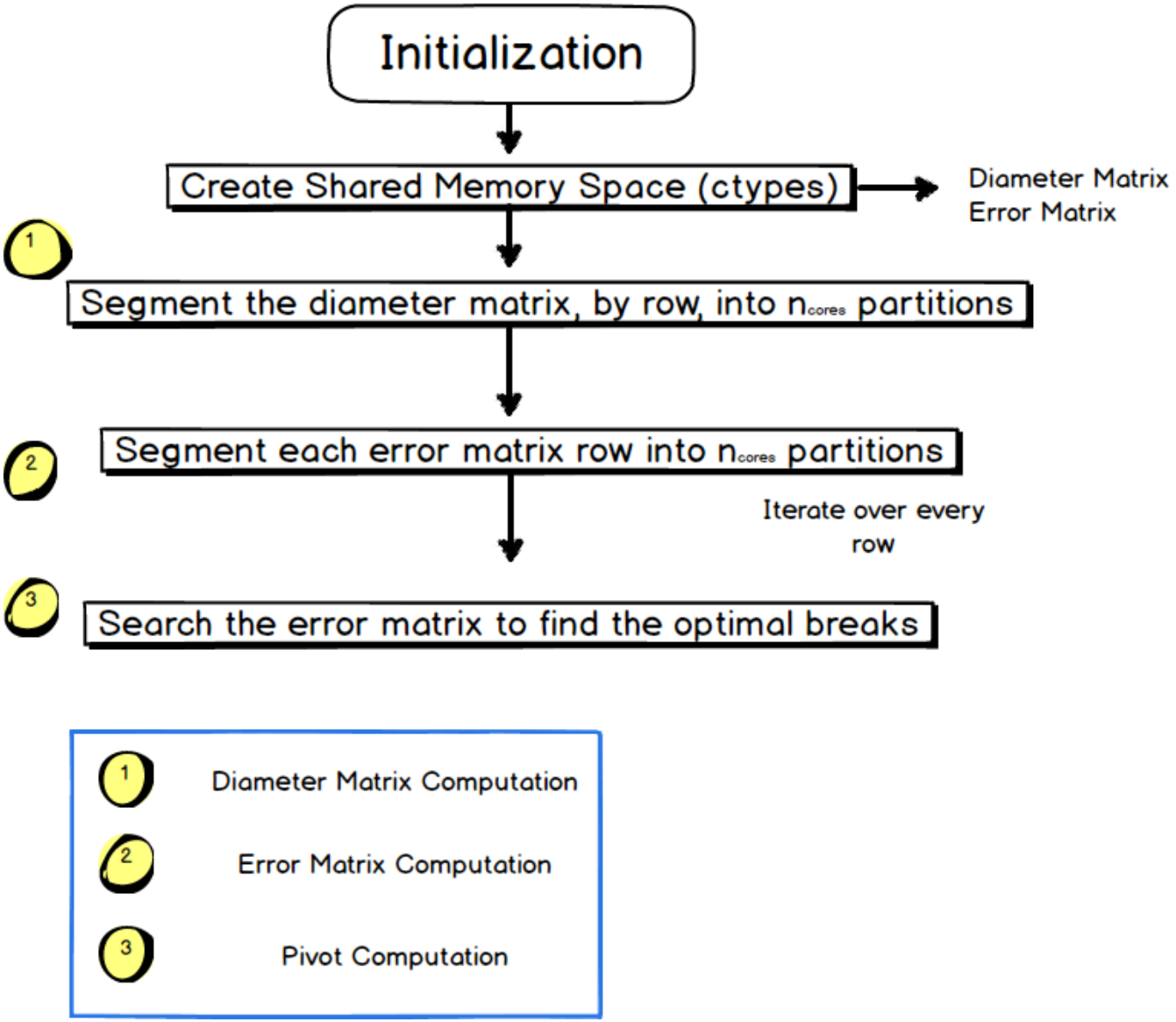
Point of Departure: Rey et al. (2013)

- Focused on parallel implementation of diameter matrix
- $D(i,j) = \sum_{k=i}^{j} \left(x_k - \sum_{l=i}^{j} x_l/(j-i+1) \right)^2$

Optimizations

- Shared memory for Diameter and Error Matrix
- Vectorization for Diameter and Error Matrix
- Parallelization of Diameter and Error Matrix

Algorithm Flow



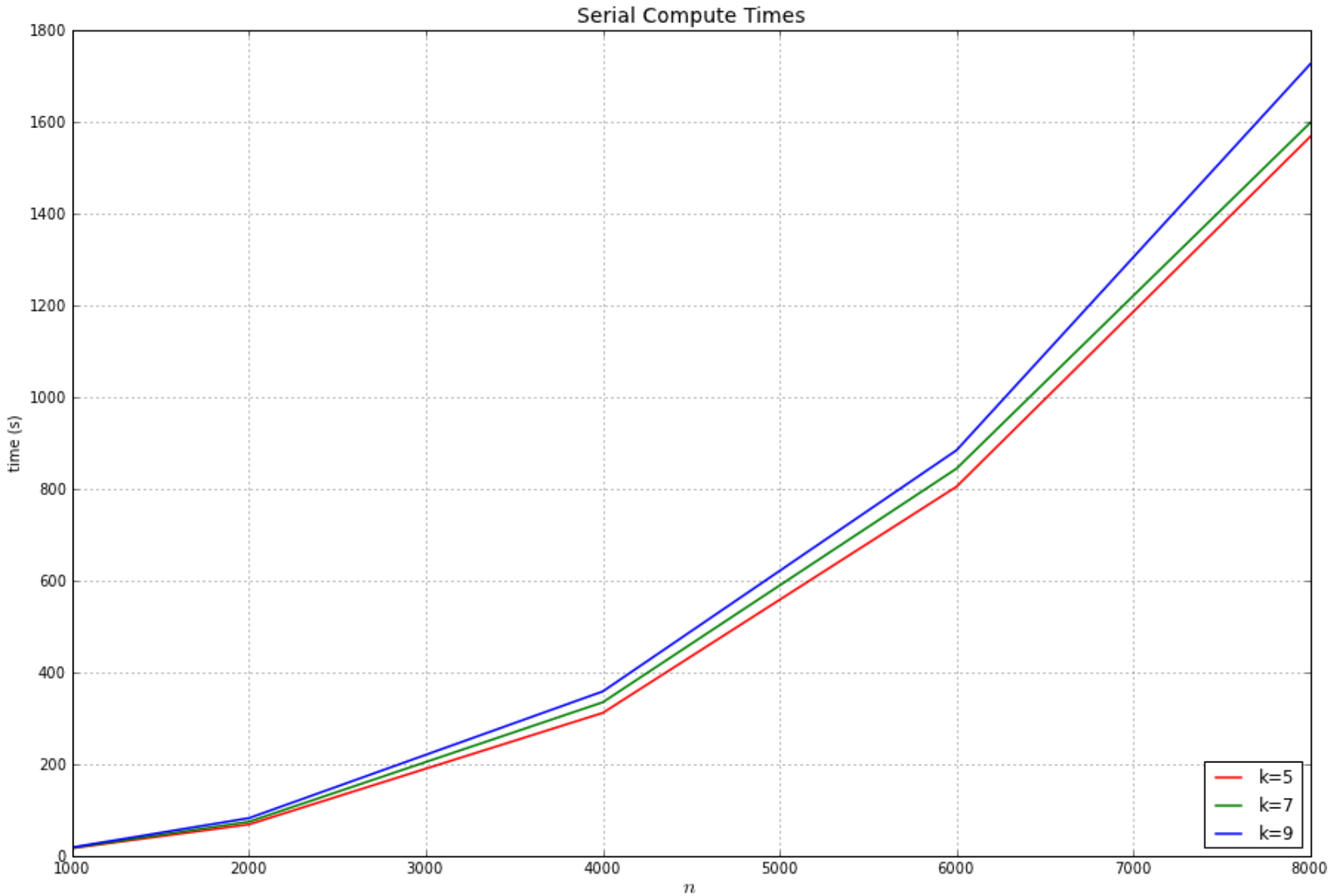
Test Data

- Range of n tested from 2000 - 42000. Cutoffs set for different implementations due to runtimes and memory constraints.
- k tested for $k = 5, 7, 9$
  - The selection of k is data set dependent, but generally values greater than 9 will introduce artificial breaks

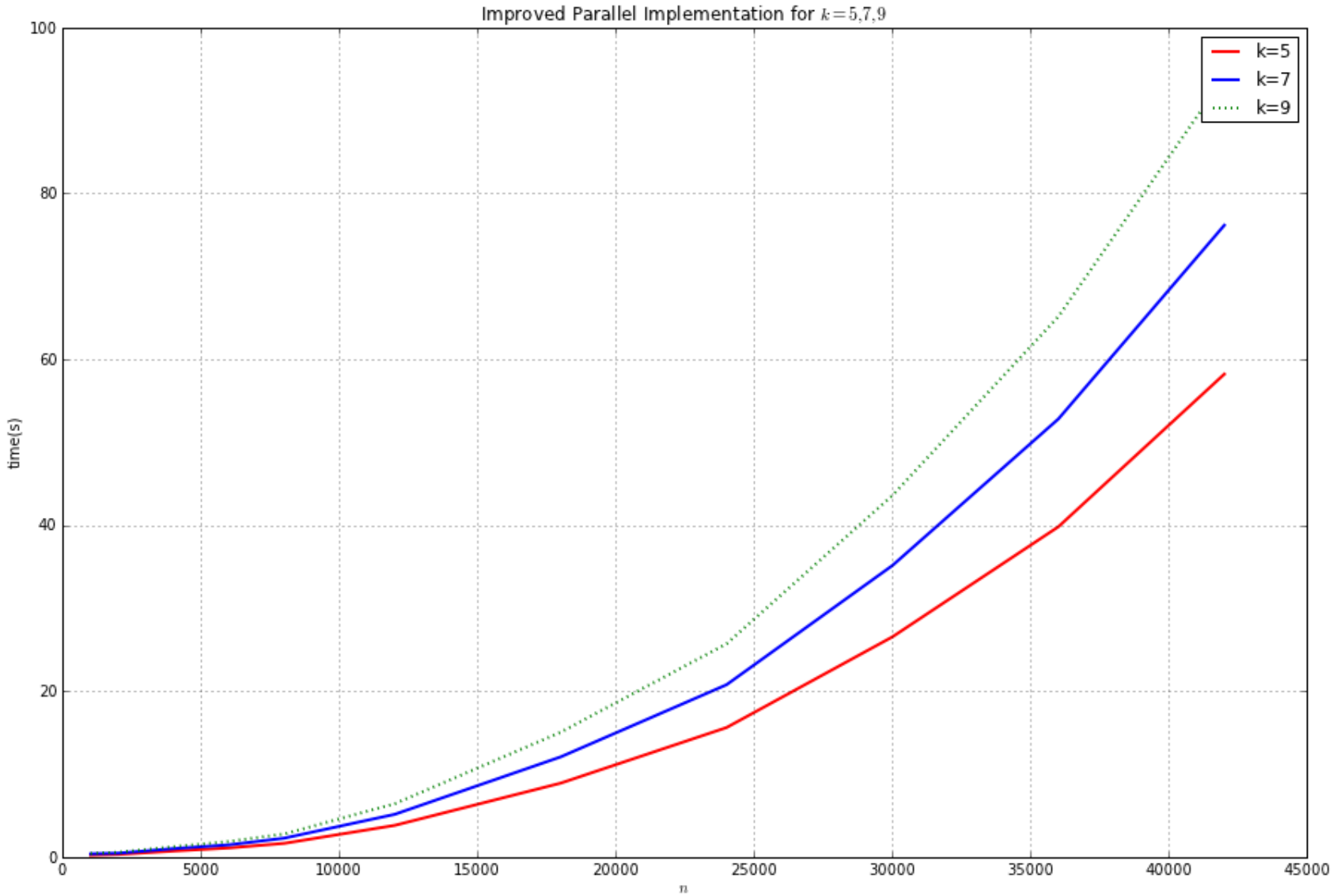
Test Machine

2 x 2.66GH 6-Core Intel Xeon Processors with 64GB RAM available and no other non-system processes running.

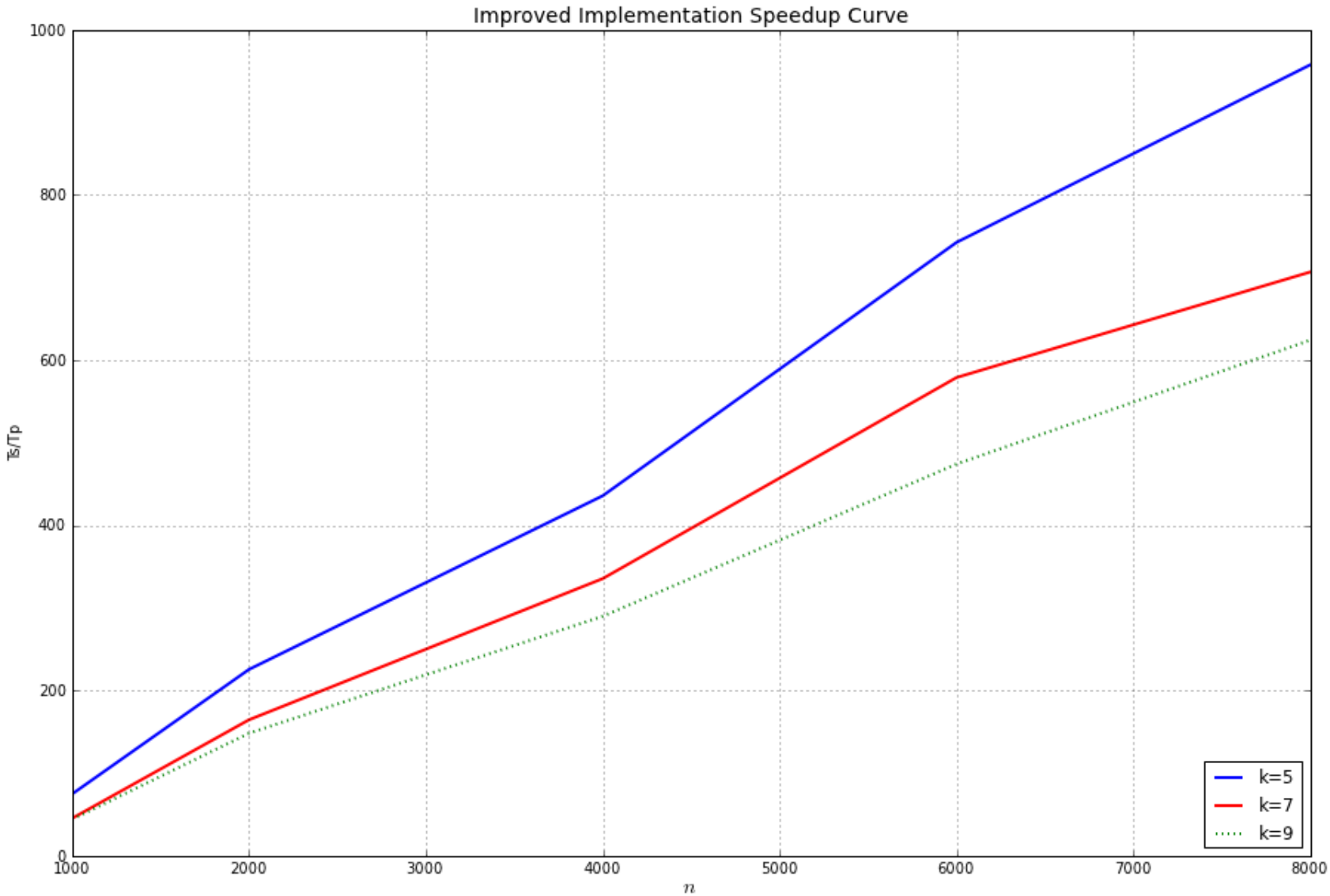
Serial Computation Time



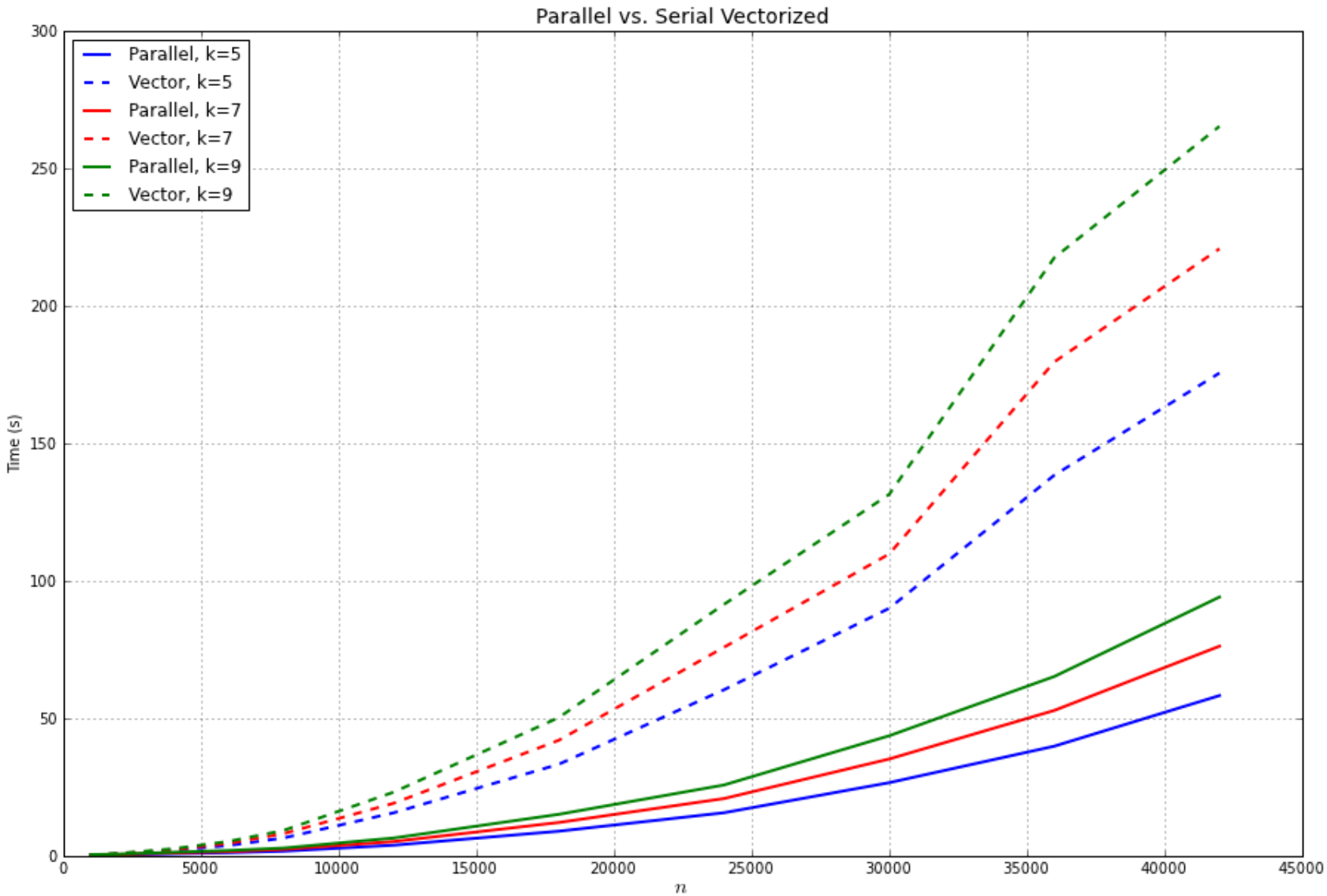
Parallel Computation Time



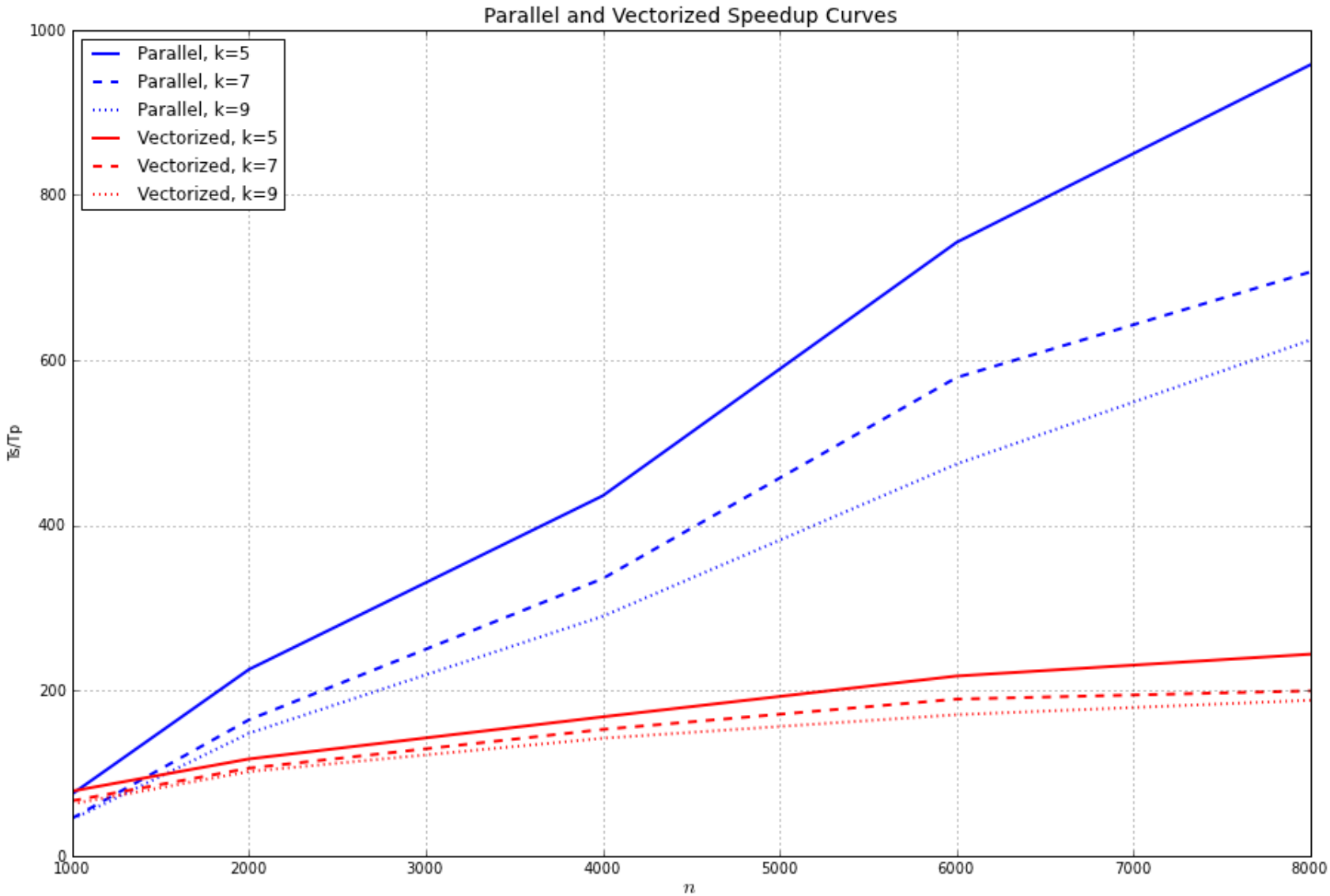
Parallel Computation Time



Parallel vs. Vectorized Computation Time



Parallel vs. Vectorized Computation Time



Lessons Learned: Fisher-Jenks

- Vectorization
  - might be “fast enough" for small n without the overhead of using `multiprocessing`
  - can be human development time intensive to refactor
- Parallelization
  - `ctypes` is key to avoiding in memory duplications for parallel `NumPy` processing
  - Embarrassingly parallel problems on regularly shaped arrays are rapid to develop and deploy

Next Steps

SMP Environment

- Address Memory Bound
  - Caching
  - On-demand recalculation
  - Jagged arrays

Non-SMP Environments

- Clusters
- NUMA

General Portability

- Move pPySAL implementation into PySAL proper
- **Move into Toolkit**