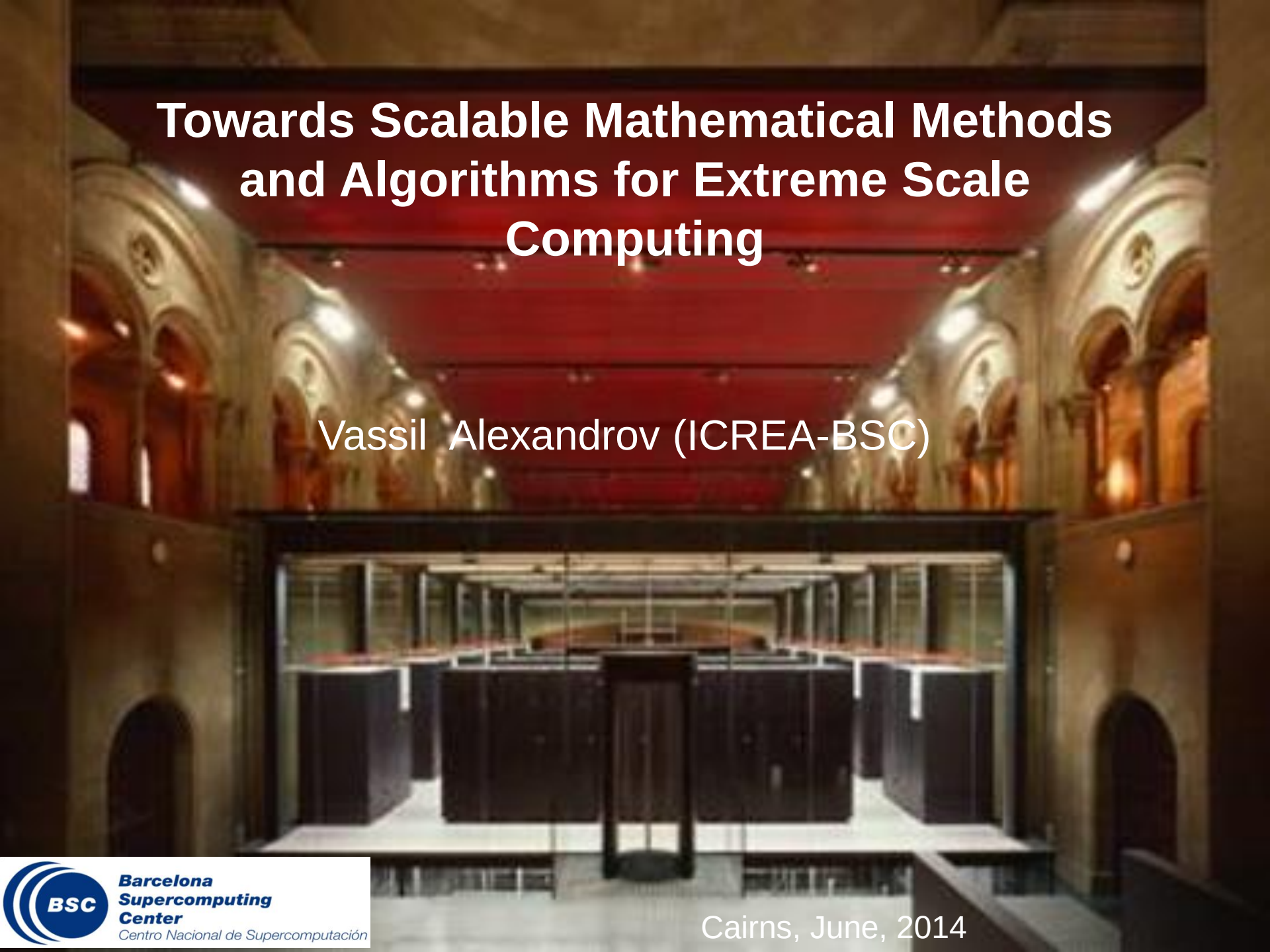
# Towards Scalable Mathematical Methods and Algorithms for Extreme Scale Computing

Vassil Alexandrov (ICREA-BSC)



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Cairns, June, 2014

# Overview

- Needs and Motivation
- Overview - Monte Carlo Hybrid Methods
- Parallel Approach
- Monte Carlo vs SPAI/MSPAI
- Experimental results
- Other Examples
- Conclusions

# Exascale Computing Challenge

Exascale System Architecture with a cap of $200M and 20MW, (Jack Dongarra – SC13, Denver. Nov 2013 )

| Systems | 2013<br>Tianhe-2 | 2020-2022 | Difference<br>Today & Exa |
|---|---|---|---|
| **System peak** | **55 Pflop/s** | **1 Eflop/s** | **~20x** |
| **Power** | **18 MW**<br>**(3 Gflops/W)** | **~20 MW**<br>**(50 Gflops/W)** | **O(1)**<br>**~15x** |
| System memory | 1.4 PB<br>(1.024 PB CPU + .384 PB CoP) | 32 - 64 PB | ~50x |
| Node performance | 3.43 TF/s<br>(.4 CPU +3 CoP) | 1.2 or 15TF/s | O(1) |
| Node concurrency | 24 cores CPU +<br>171 cores CoP | O(1k) or 10k | ~5x - ~50x |
| Node Interconnect BW | 6.36 GB/s | 200-400GB/s | ~40x |
| System size (nodes) | 16,000 | O(100,000) or O(1M) | ~6x - ~60x |
| Total concurrency | 3.12 M<br>12.48M threads (4/core) | O(billion) | ~100x |
| MTTF | Few / day | Many / day | O(?) |

# Current Key Reports

- Dongarra et al, *Applied Mathematics Research for Exascale*, March 2014.

- ESSI  II , annual report, Sept 2013.

# Needs

- Novel scientific algorithms that improve performance, scalability, resilience and power efficiency
- Novel mathematical modeling methods leading to increased algorithmic scalability
- Developing Scientific algorithms that can exploit extreme concurrency (e.g. 1 billion for exascale by 2020)
- Naturally fault tolerant, self-healing or fault oblivious scientific algorithms
- Programming model and system software that support algorithm scalability and resilience

# Scalable Algorithms: Motivation/Drivers

- Bridging the Performance Gap while dealing with Hybrid Architectures
- Increased Scalability
- Highly fault-tolerant  and fault-resilient algorithms
- Need to calculate with higher precision without restart

# Important Properties

- Efficient Distribution of the compute data.
- Minimum communication/ communication reducing algorithms
- Increased precision is achieved adding extra computations (without restart) .
- Fault-Tolerance achieved through adding extra computations


# MONTE CARLO METHODS

# Idea of the Monte Carlo method

- Wish to estimate the quantity α
- Define a random variable ξ
- Where ξ has the mathematical expectation α
- Take N independent realisations $\xi_i$ of ξ
  - Then $$\bar{\xi} = \frac{1}{N} \sum_{i=1}^{N} \xi_i$$
  - And according to the Law of Large Numbers (LLN) $\bar{\xi} \approx \alpha$

# Motivation: MC for Linear Algebra

- Many scientific and engineering problems revolve around:
  - inverting a real n by n matrix (MI)
    - Given A
    - Find $A^{-1}$
  - solving a system of linear algebraic equations (SLAE)
    - Given A and b
    - Solve for x, $Ax = b$
    - Or find $A^{-1}$ and calculate $x = A^{-1}b$

# Motivation cont.

- Traditional direct Methods with dense matrices
  - Gaussian elimination
  - Gauss-Jordan
  - Both take $O(n^3)$ steps

- Time prohibitive if
  - large problem size
  - timely solution required

# Monte Carlo Methods

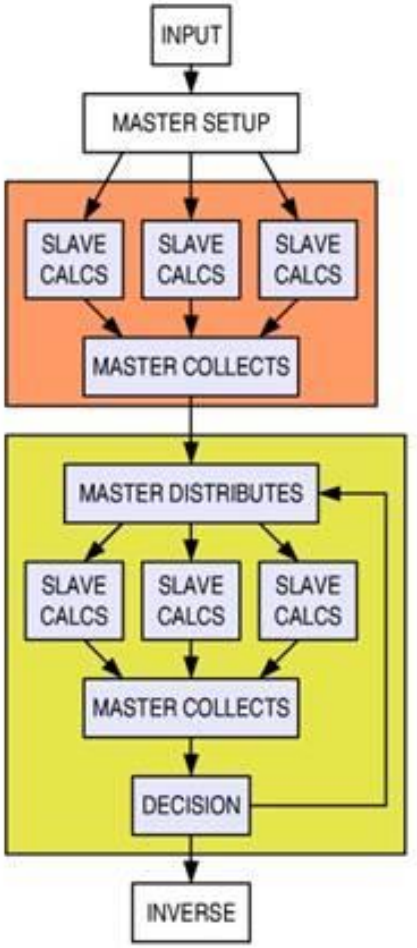
- Fast stochastic approximation
- Very efficient in finding a quick rough estimation
  - element or row of inverse matrix
  - component of solution vector

# Reason for using Monte Carlo

- O(NT) steps to find an element of the
  - inverse matrix A
  - solution vector x
- Where
  - N Number of Markov Chains
  - T length of Markov Chains
- Independent of n - size of matrix or problem
- Algorithms can be efficiently parallelised

# Parallel Algorithms

- Multi-tiered process
- Using parallel Monte Carlo to find a rough inverse of A
- Original algorithm for diagonally dominant matrices
- Extension to the general case non-diagonally dominant matrices with ||A|| < 1
- Parallel iterative refinement to improve accuracy and retrieve final inverse



# Parallel Algorithm

- Start with a diagonally dominant matrix ^B
- Make the split ^B = D - $B_1$
  - D has only the diagonal elements of ^B
  - $B_1$ includes only off-diagonal elements
- Compute A = $D^{-1}B_1$
- If we had started with a matrix B that was not diagonally dominant then an additional splitting would have been made at the beginning, B = ^B - (^B - B), and a recovery section would be needed at the end of the algorithm to get $B^{-1}$ from $\hat{B}^{-1}$

# Parallel Algorithm cont.

## Matrix Inversion using Markov Chain Monte Carlo

Each element in the inverse matrix is

$$c_{rs} = \frac{1}{N} \sum_{s=1}^{N} \left( \sum_{(j|s_j=r)} W_j \right)$$

where:

- $N = \left( \frac{.6745}{\varepsilon(1-||A||)} \right)^2$ is the number of Markov Chains
- $(j|s_j = r)$ means that only the $W_j = \frac{a_{ss_1} a_{s_1 s_2} \cdots a_{s_{j-1} s_j}}{p_{ss_1} p_{s_1 s_2} \cdots p_{s_{j-1} s_j}}$ are included for which $s_j = r$ (i.e. the Markov Chain terminates at $r$)

# Parallel Algorithm cont.

- Having used MC for inverting diagonally dominant matrices the obvious next extension is to see how this algorithm can be extended to invert general matrices. For this, assume the general case where ||B|| > 1 and consider the splitting

$$B = \hat{B} - C$$

- From this it is then necessary to work back and recover $B^{-1}$ from $\hat{B}^{-1}$.
- To do this an iterative process $(k = n-1, n-2, \ldots, 0)$ is used on $\hat{B}^{-1}$

$$B_k^{-1} = B_{k+1}^{-1} + \frac{B_{k+1}^{-1} S_{k+1} B_{k+1}^{-1}}{1 - trace\left(B_{k+1}^{-1} S_{k+1}\right)};$$

# Refinement Process

- Given a non-singular matrix $A.$, and its inverse $A_0$, if we define $R_0 = I - AA_0$ then we perform the following steps for more accurate inverse computation:

$$A_i = A_{i-1}\left(I + R_{i-1}\right), \quad R_i = I - AA_i \quad i = 1, 2, \ldots, n$$

- Therefore,

$$\begin{aligned} R_n = I - AA_n &= I - AA_{n-1}\left(I + R_{n-1}\right), \\ &= I - \left(I - R_{n-1}\right)\left(I + R_{n-1}\right) = R_{n-1}^2 = R_{n-2}^4 = \cdots = R_0^{2n}, \end{aligned}$$

- Obviously we have,

$$A_n = A^{-1}(I - R_0^{2n})$$

# Refinement Process

- The formula shows that $A_n$ approaches $A^{-1}$ when the convergence of the process is very rapid. We can estimate the error at step $n$ of this procedure:

$$A^{-1} = IA^{-1} = (A_0 A_0^{-1})A^{-1} = A_0(AA_0)^{-1} = A_0(I - R_0)^{-1},$$

$$||A_N - A^{-1}|| = || - A^{-1}R_0^{2m}|| = || - A_0(I - R_0)^{-1}R_0^{2m}||$$

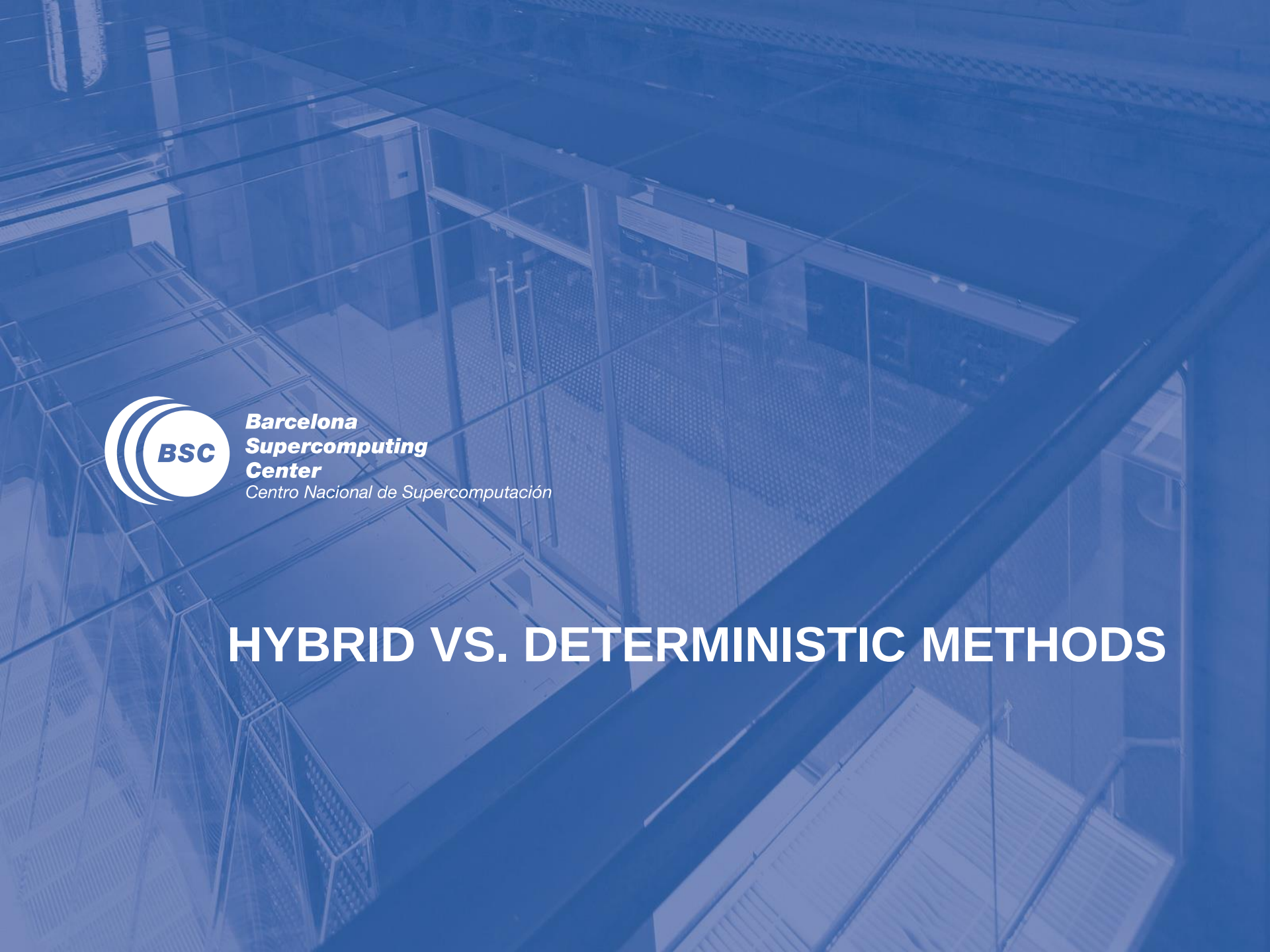
$$\leq ||A_0|| \quad ||(I - R_0)^{-1}|| \quad ||R_0^{2m}||$$

$$\leq ||A_0|| \frac{k^{2^m}}{1 - k}$$

- We see from the inequality that as long as the initial approximate inversion satisfies the condition $||R_0|| \leq \rho(R_o) \leq 1$ the number of correct decimal figures increases with a power $2^m$

# Parallel Algorithm cont.

- Almost linear speedup for large enough problem sizes
- Minimal inter-process communication necessary
- Interleaving and balancing computation and communication balanced
- For smaller problem sizes need data replication in order to obtain good performance




# HYBRID VS. DETERMINISTIC METHODS

# Combination of Monte Carlo and SPAI

- SPAI – SParse Approximate Inverse Preconditioner
  - Computes a sparse approximate inverse M of given matrix A by minimizing $\| AM - I \|$ in the Frobenius norm
  - Explicitly computed and can be used as a preconditioner to an iterative method
  - Uses BICGSTAB algorithm to solve systems of linear algebraic equations Ax = b

- Sparse Monte Carlo Matrix Inverse Algorithm
  - Computes an approximate inverse by using Monte Carlo methods
  - Uses an iterative filter refinement to retrieve the inverse

# Monte Carlo and SPAI cont.

- Idea: using a Monte Carlo computed approximate matrix inverse as a pre-conditioner

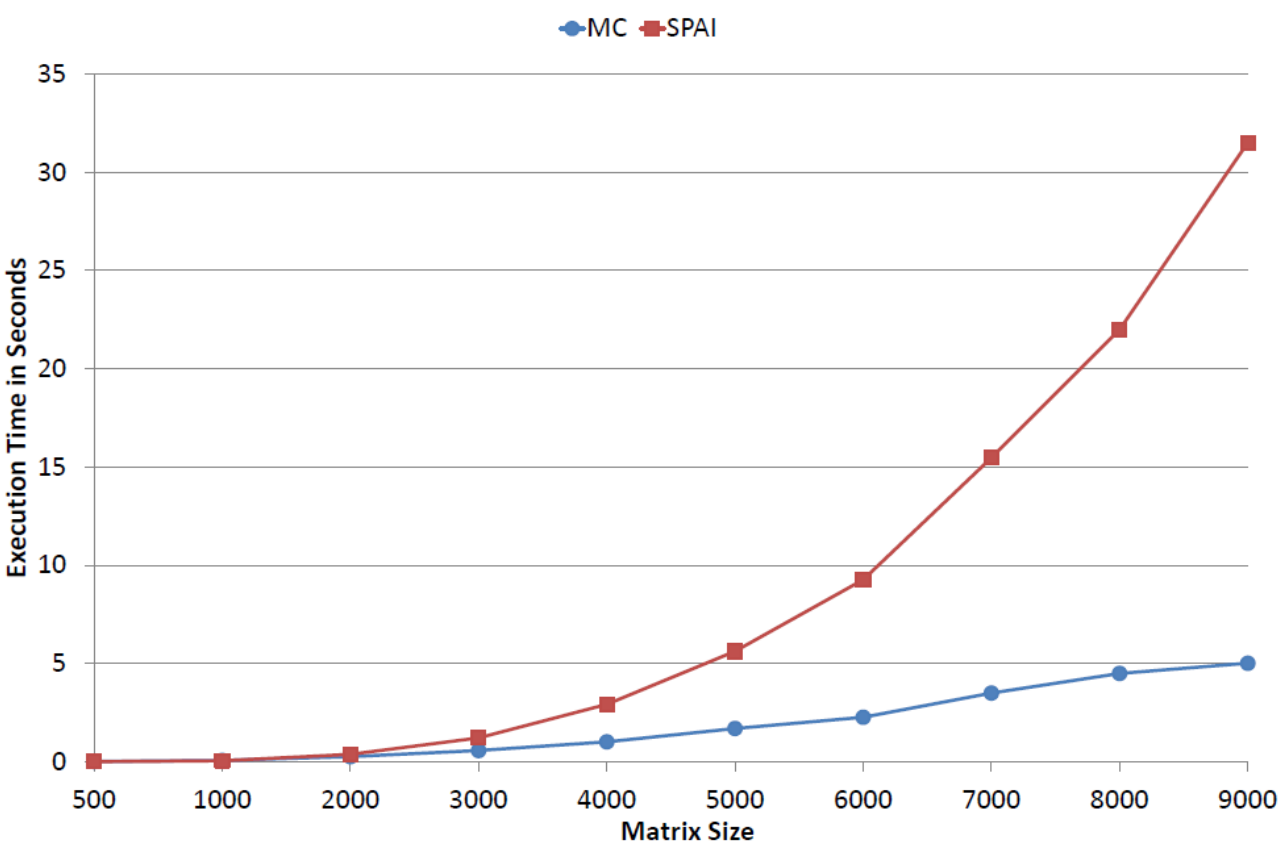
- Considering
  - computation time for MC solution
  - Suitability of rough inverse
  - Matrix types that are not invertible via Frobenius norm
  - Time savings, especially for larger problem sizes

# Experiments

- Selected test sets
  - The University of Florida Sparse Matrix Collection
  - Matrix Market
- Parameter and setting selection
  - Computation of pre-conditioner to same accuracy
  - Utilized in BiCGSTAB solver
  - RHS generated from input matrix
- Monte Carlo approach without refinement filter
  - Rough inverse sufficient for quick convergence in most cases

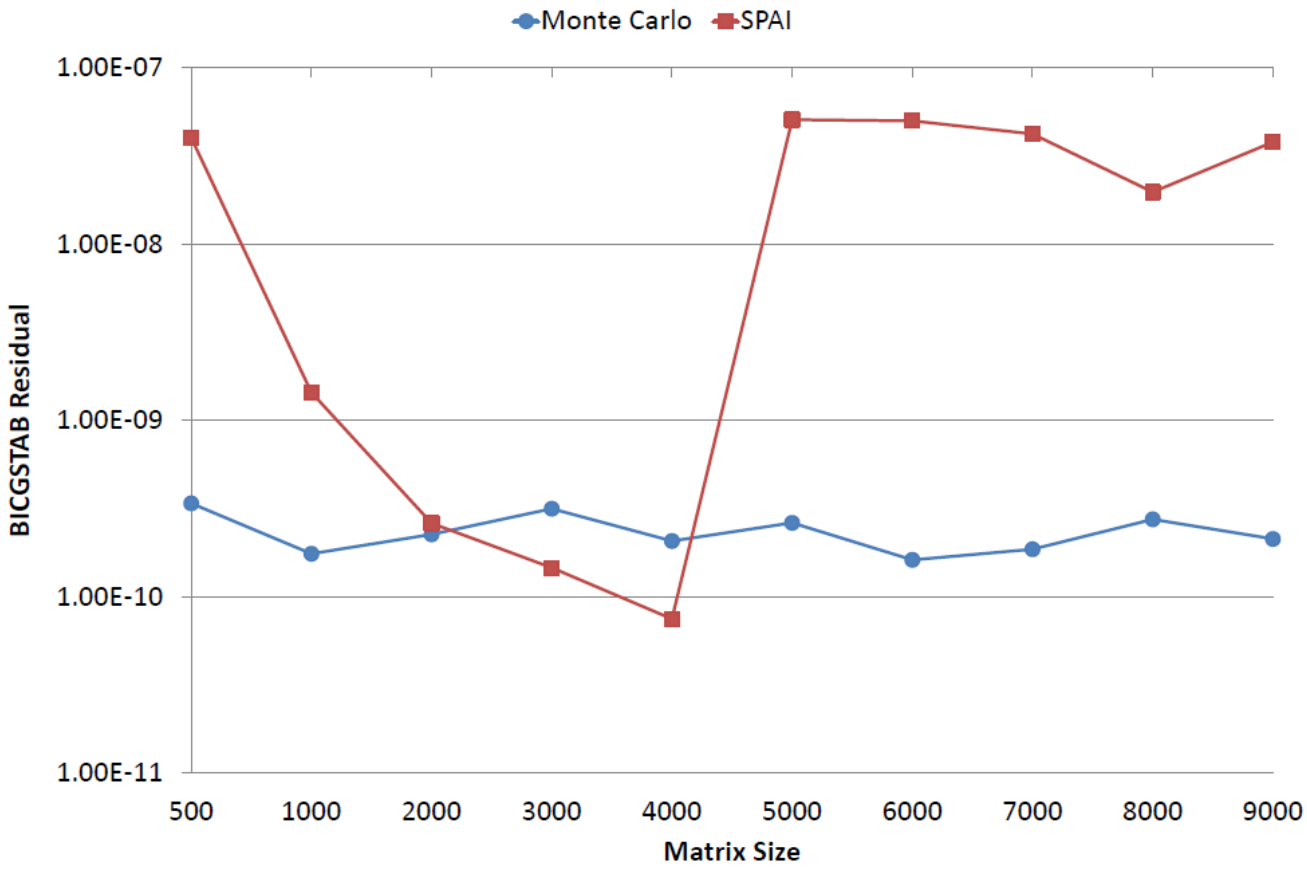
# Monte Carlo and SPAI

- Experiments show significant performance improvement, especially using Monte Carlo for larger problem sizes
  - Considering original serial implementation
  - Parallel MSPAI and Monte Carlo showing comparable performance

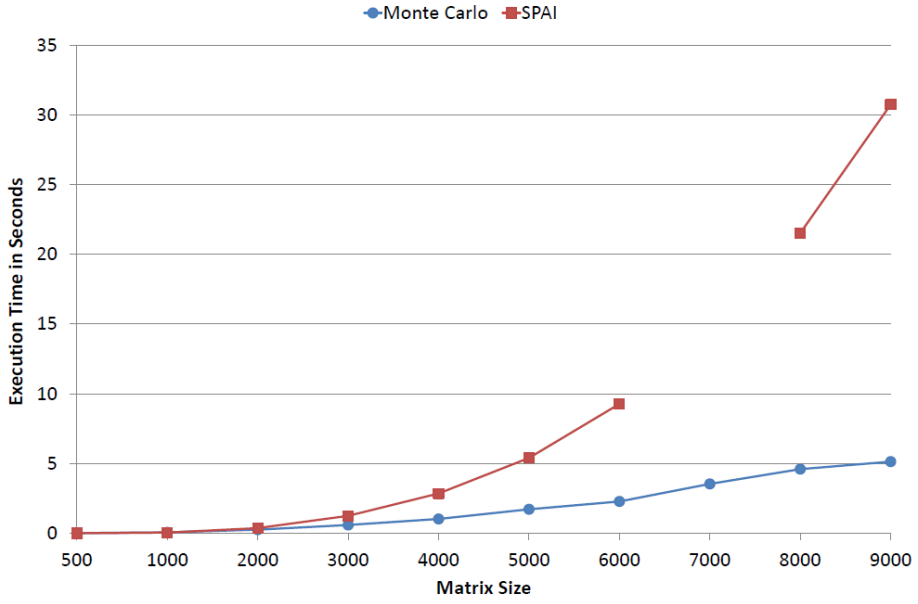


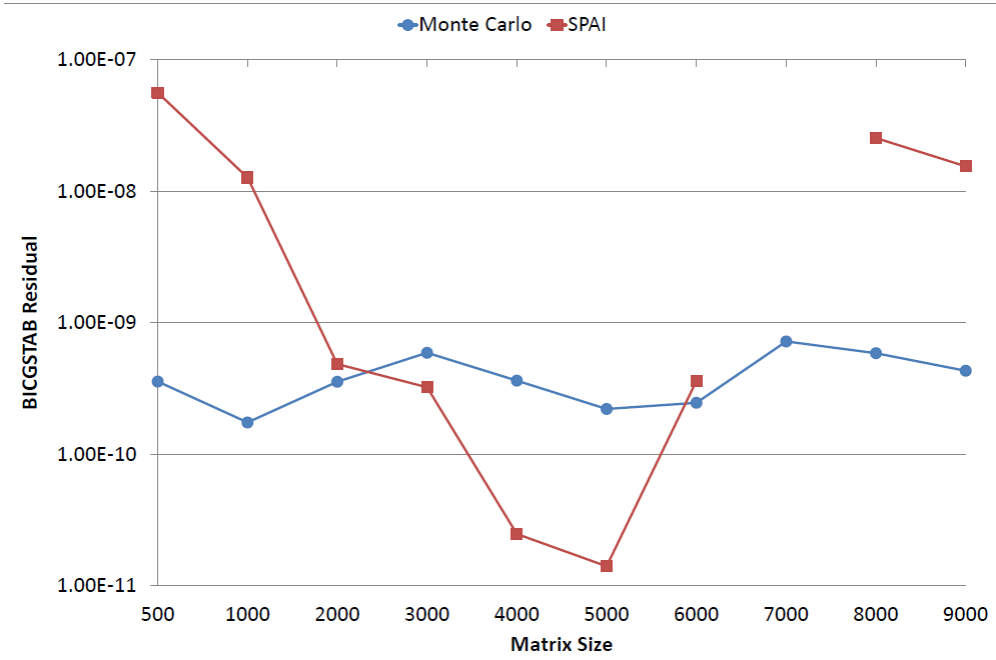
# MC & SPAI residuals

- Quality comparisons of the pre-conditioners
  - Used to solve System of Linear Algebraic Equations (SLAE)
  - Original input matrix, pre-conditioner, right hand side
  - Solved using BiCGSTAB implementation in SPAI



# Non-Converging case Sparsity 0.5

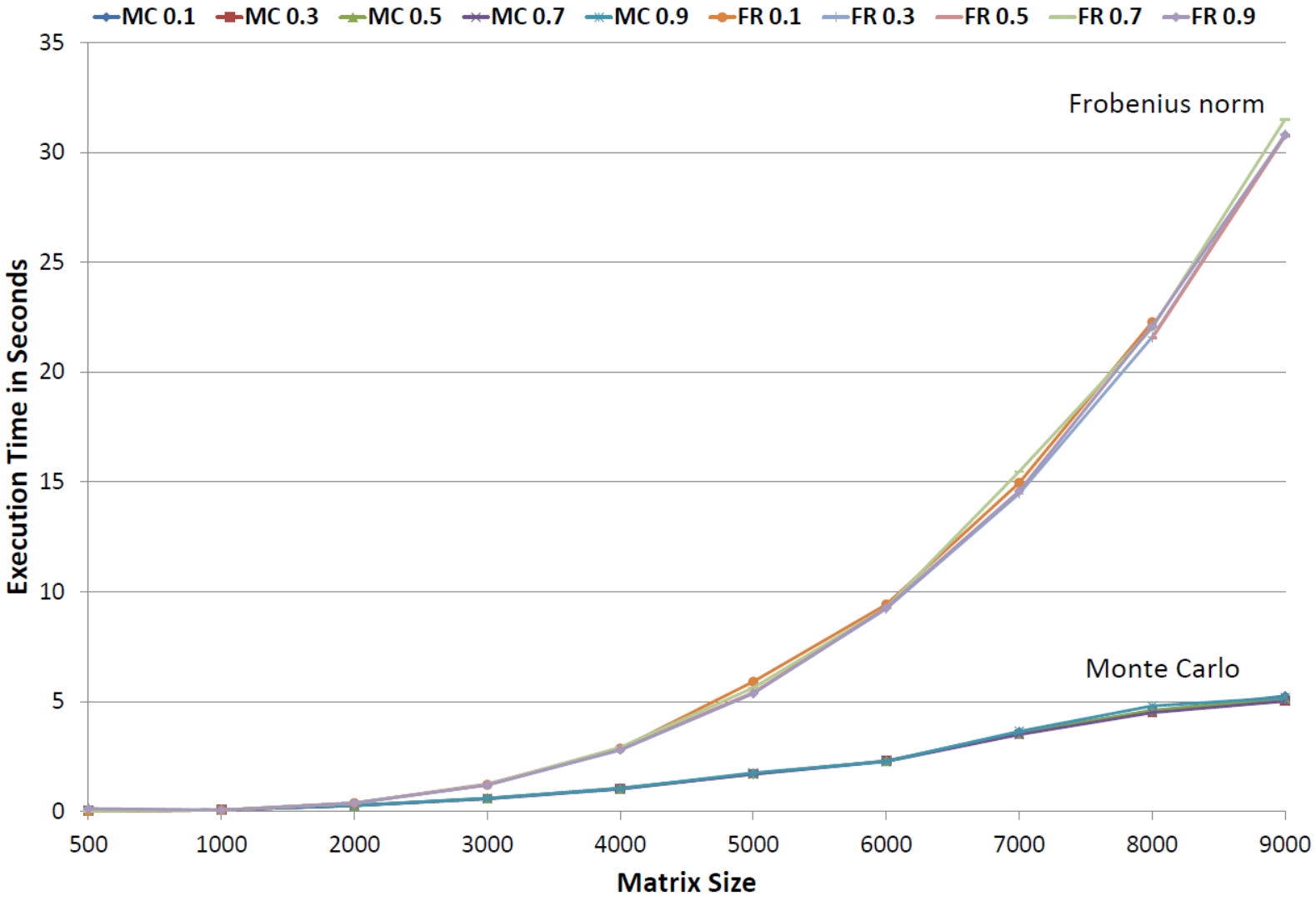


# Sparsity and computation

- SPAI computes the Frobenius norm of the input matrix
  - Workload depending on the size of the input matrix
- Monte Carlo algorithm uses Markov Chains
  - Independent of the size of the matrix
  - length and number of chains important
  - Original algorithm for dense matrices; extended to support general sparse cases
- Experiments have been run using various sparsity (10%-90%)

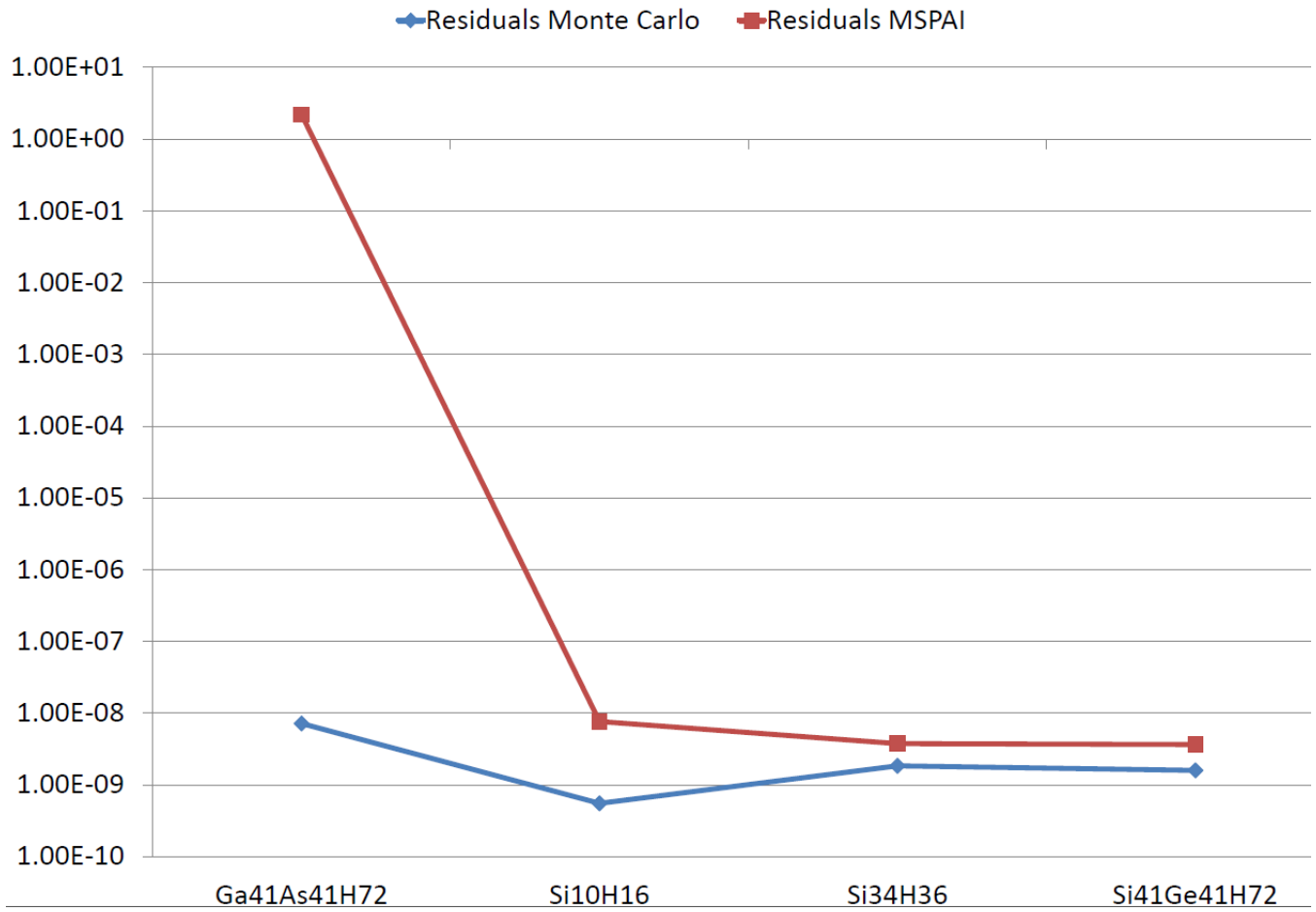
# Sparsity and computation cont.




# EXTENSION TO MSPAI

# Residuals for Parsec data set

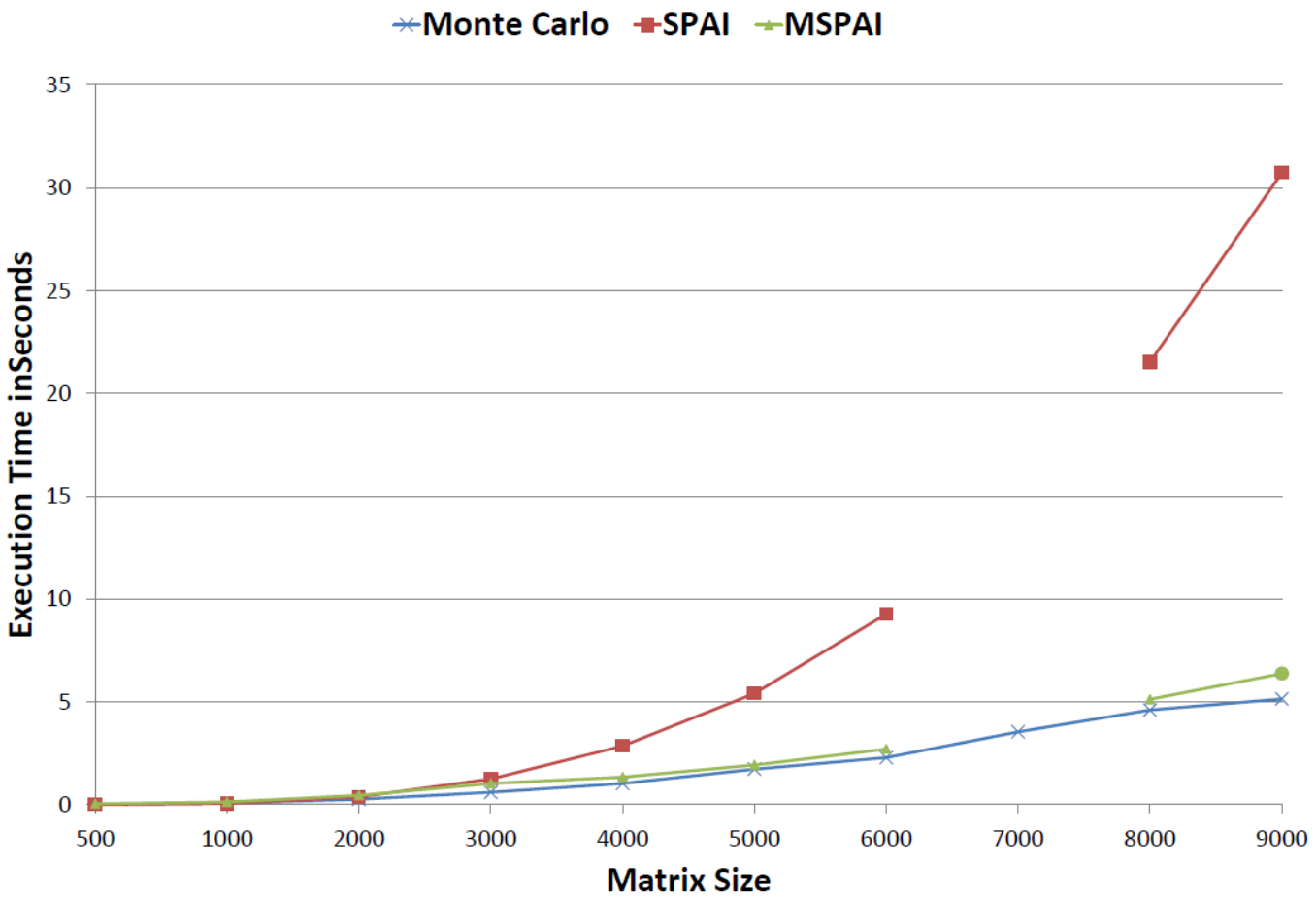
- Residuals within same order of magnitude
- Non-convergence for some types of matrices using SPAI




University of Florida Sparse Matrix Collection
Parsec data set

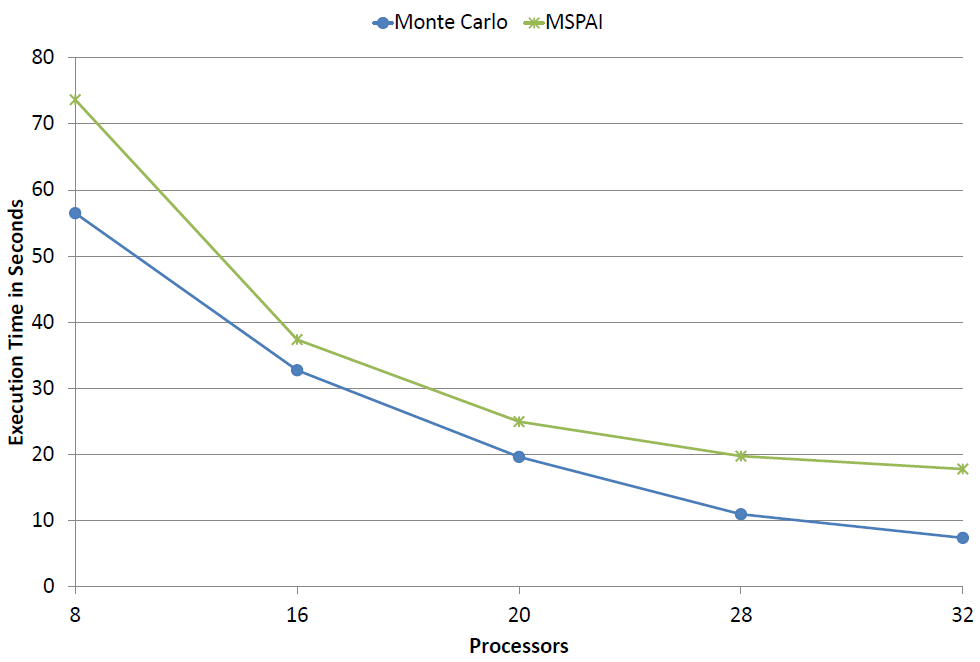
# Convergence observations

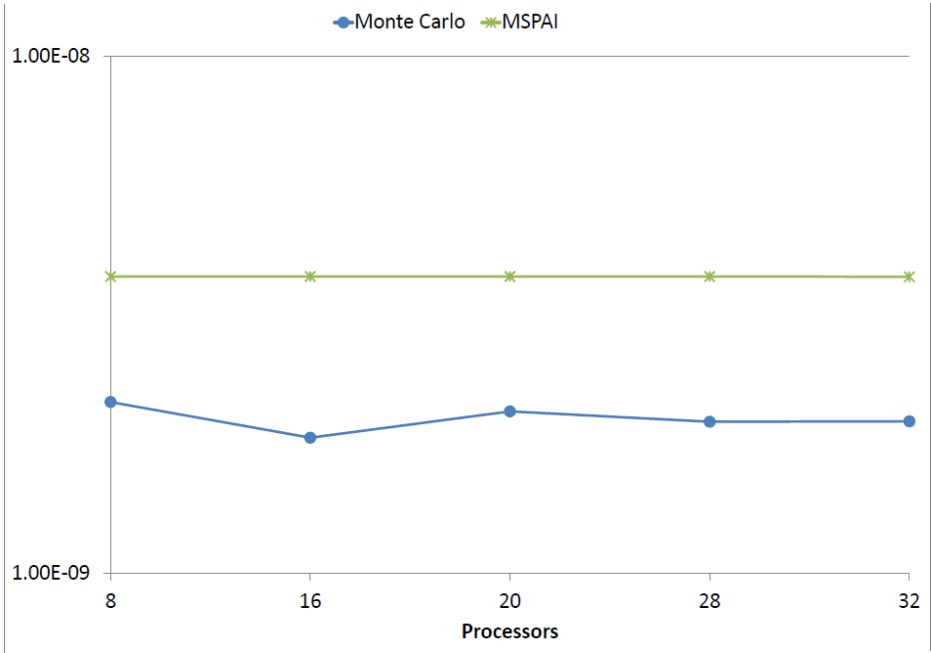
- Non-convergence for some types of matrices using SPAI
- Monte Carlo converges in observed cases



# Si34H36

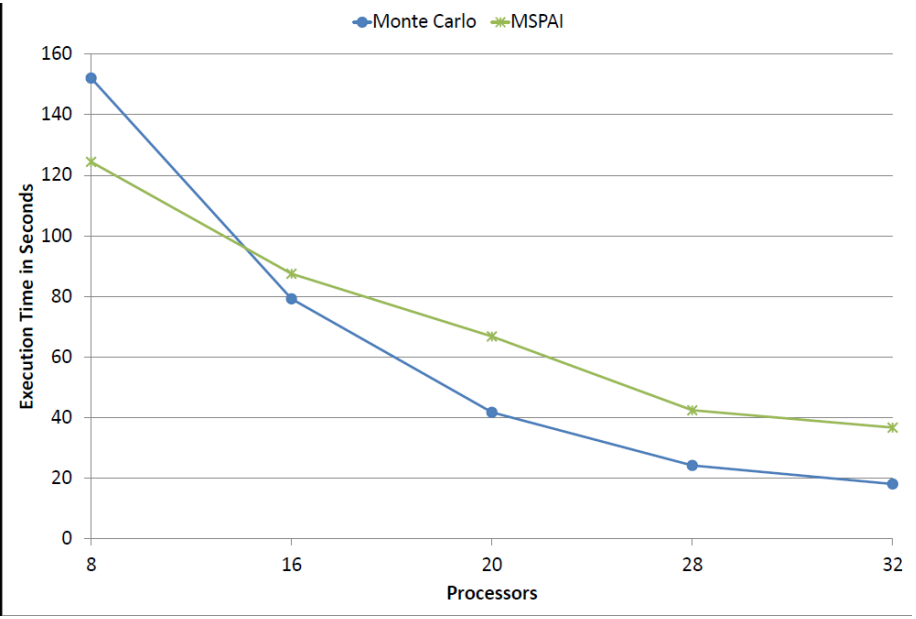
- 97,569 x 97,569
- 5,156,379 nonzeros
- Real symmetric

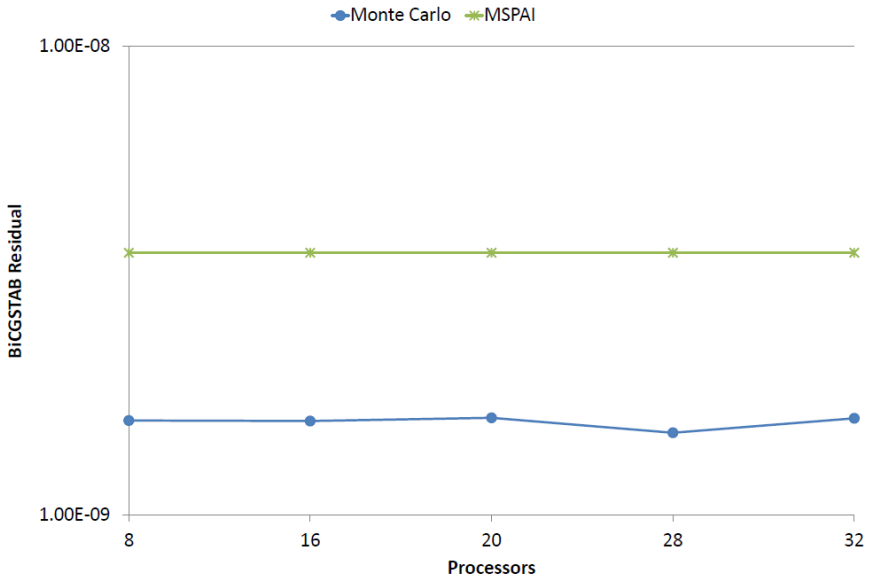


# Si41Ge41H72

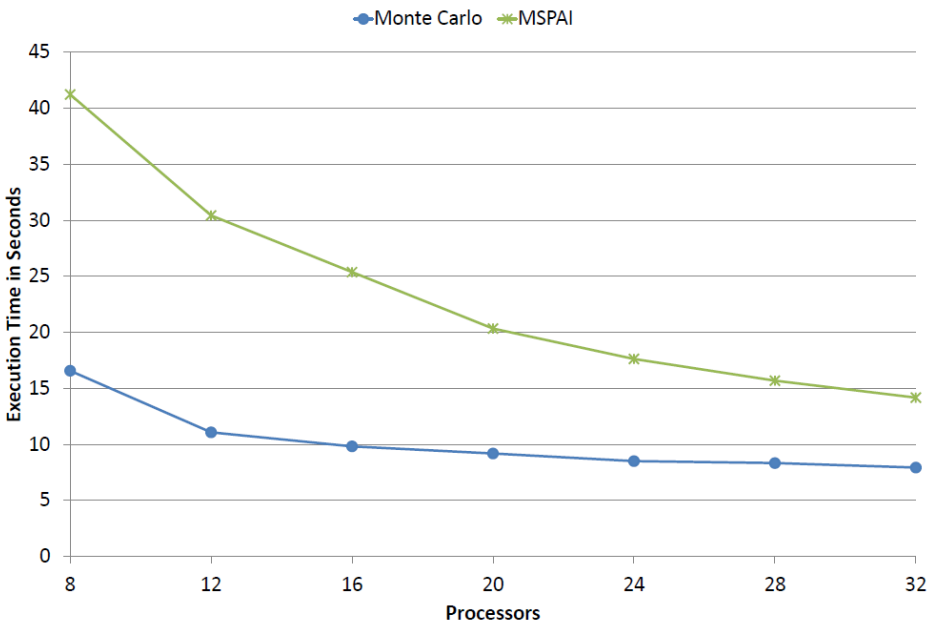
- 185,639 x 185,639
- 15,011,265 nonzeros
- Real symmetric

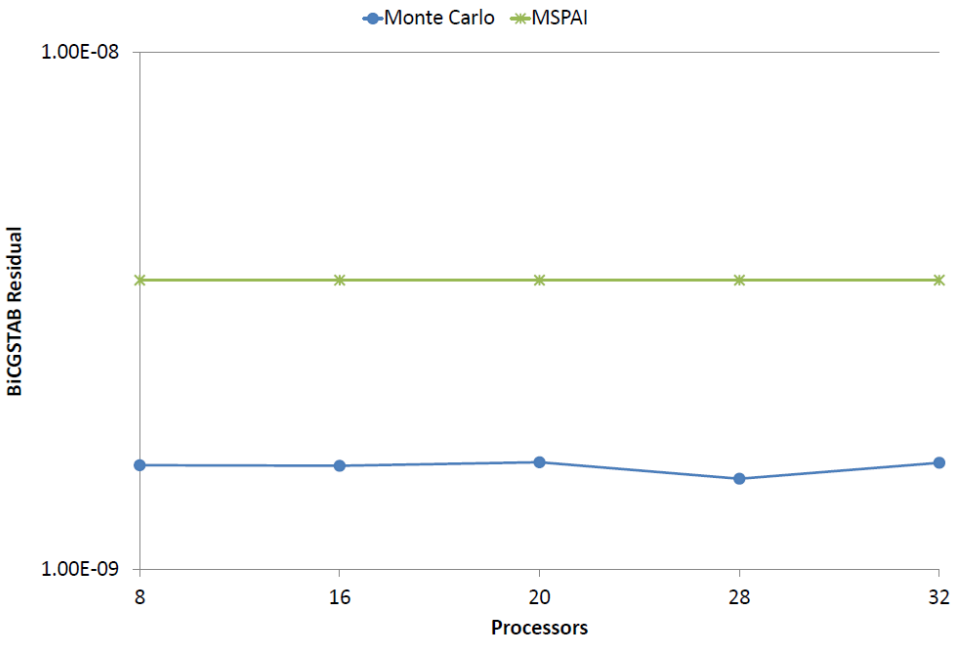


# Si10H16

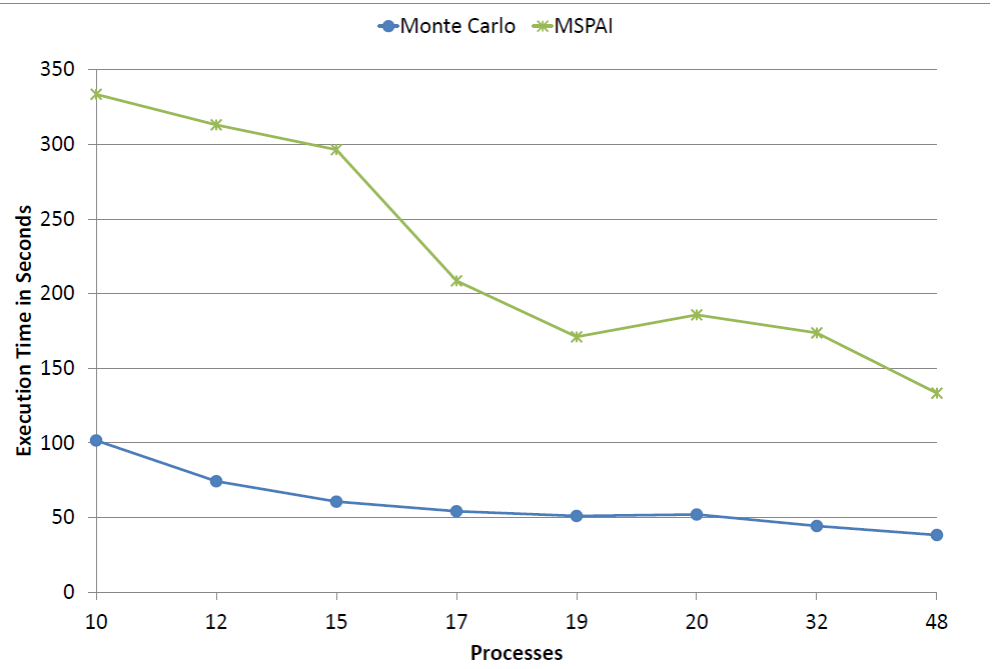
- 17,077 x 17,077
- 875,923 nonzeros
- Real symmetric

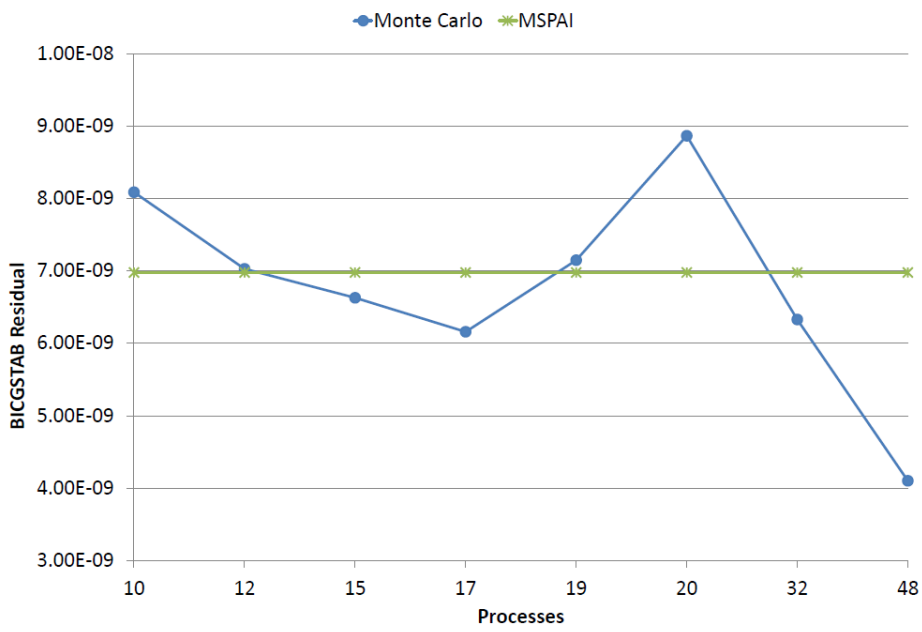


# Appu

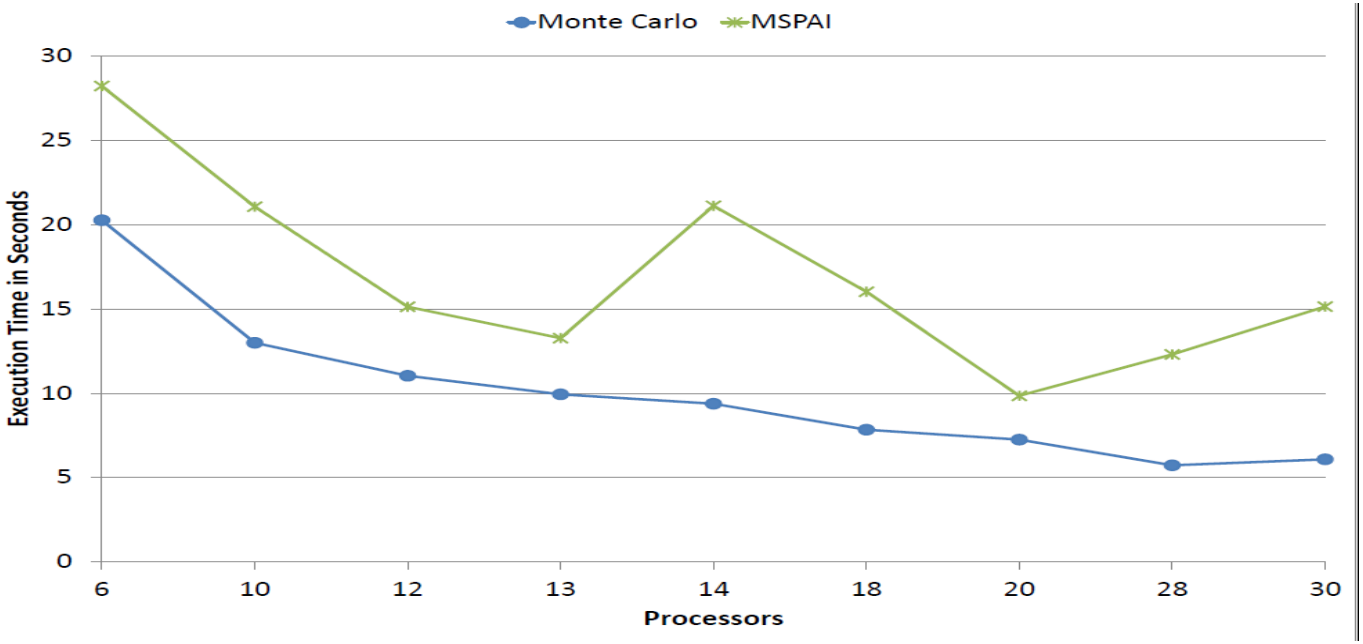
- 14,000 x 14,000
- 1,853,104 nonzeros
- real nonsymmetric



# PSMIGR 3

- 3140 x 3140
- 543,160 nonzeros
- real nonsymmetric
- non diagonally dominant




# OPTIMIZATION

# Example: Parallel Regularized Multiple-Criteria Linear Programming

$$\min_z \frac{1}{2} w^\top H w + \frac{1}{2} \xi^{(1)\top} Q \xi^{(1)} + \frac{1}{2} b^2 + C e^\top \xi^{(1)} - D e^\top \xi^{(2)},$$

$$\text{s.t.} \; (w \cdot x_i) + (\xi_i^{(1)} - \xi_i^{(2)}) = b, \quad \text{for } \{i | y_i = 1\},$$

$$(w \cdot x_i) - (\xi_i^{(1)} - \xi_i^{(2)}) = b, \quad \text{for } \{i | y_i = -1\},$$

$$\xi^{(1)}, \xi^{(2)} \geq 0,$$

**Zhiquan Qi, Vassil Alexandrov,, Yong Shi, Yingjie Tian, Parallel Regularized Multiple-Criteria Linear Programming, to appear in Procedia Computer Science 2014, Proc. of ITQM, May 2014.**

# Example: Parallel Regularized Multiple-Criteria Linear Programming

- Convex problem, through the dual:

$$\min_{\alpha,\ \xi^{(1)}} \frac{1}{2}\alpha^{\top}(K(A,A^{\top})+ee^{\top})\alpha+\frac{1}{2}\xi^{(1)\top}Q\xi^{(1)},$$
$$\text{s.t. } -Q\xi^{(1)}-Ce\le E\alpha\le -De,$$

- Reformulate the problem into global optimization one

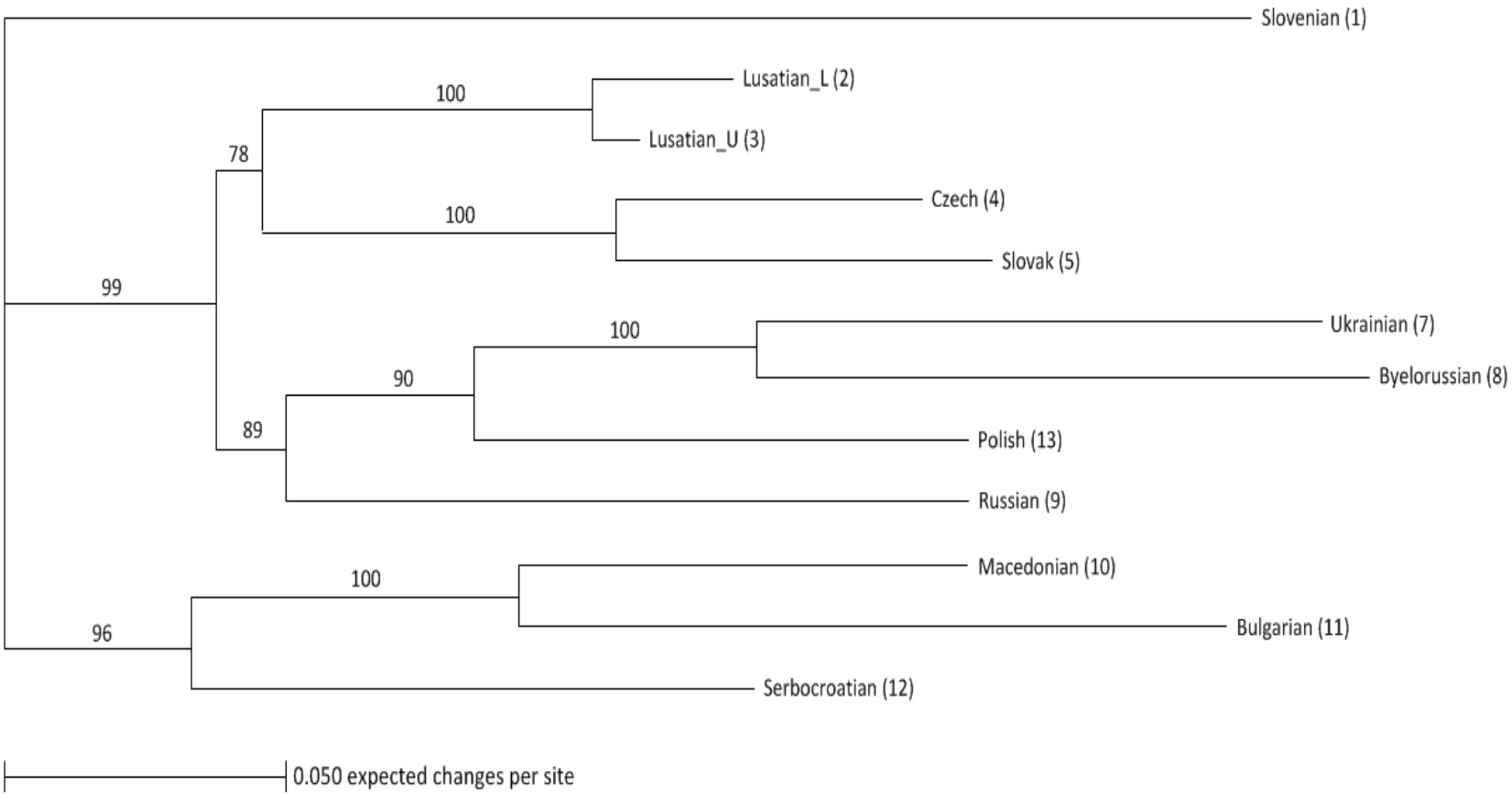
$$\min_{\pi} f(\pi) = \frac{1}{2}\pi^{\top}\Lambda\pi+\lambda^{\top}\max\{G\pi-Ce,0\}^2 + \mu^{\top}\max\{H\pi+De,0\}^2,$$

- Parallelize efficiently:
  - by dividing the variables
  - by dividing the area to subareas

**Zhiquan Qi, Vassil Alexandrov,, Yong Shi, Yingjie Tian, Parallel Regularized Multiple-Criteria Linear Programming, to appear in Procedia Computer Science 2014, Proc. of ITQM, May 2014.**


# ANALYSING LANGUAGES

# Analysing Slavonic Languages




# MULTI-LEVEL, MULTISCALE METHODS

# Example: Tackling multiscale problems using multilevel Monte Carlo methods and algorithms

Approach:

- Stochastic approximation of the overall problem
- Local refinement with deterministic methods


# CONCLUSIONS

# Conclusions

Properties/Challenges:

- Enable minimum communication, achieved through data localization and replication.
- Increased precision, is achieved by adding extra computations using the obtained solution so far without restart.
- Fault-Tolerance, is achieved by adding extra computations without restart.
- They are naturally resilient, resilience is achieved by replacing the part of the lost or incorrect computations with additional computations without restart.

# Conclusions

Scalability at all levels is needed:

- Mathematical model level
- Algorithmic level
- Systems level.