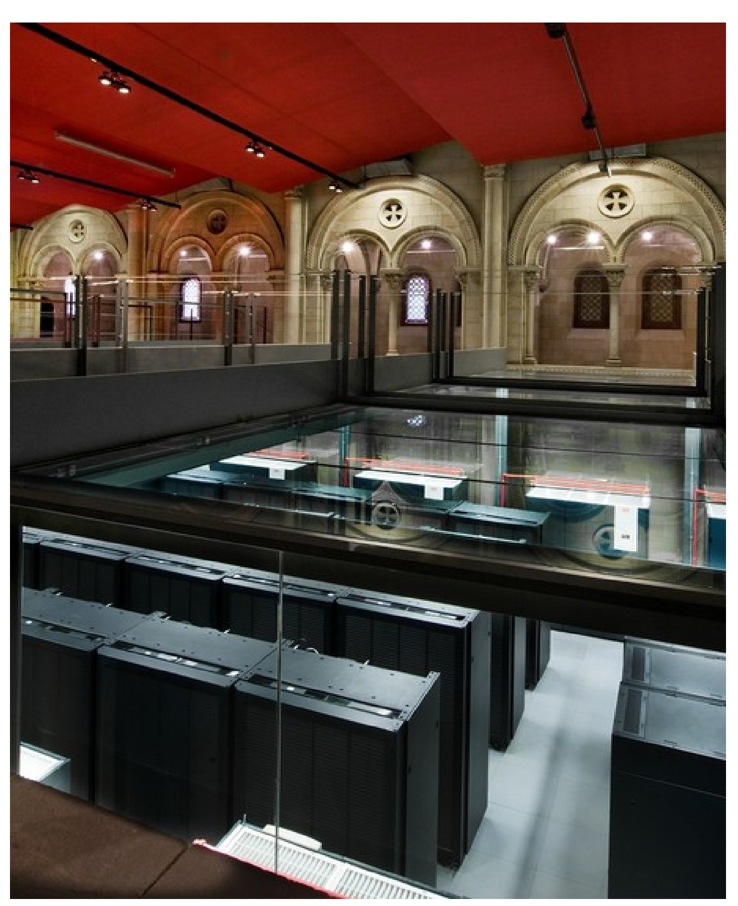
# Towards Advanced Hybrid Monte Carlo Methods for Linear Algebra for Extreme Scale Systems: Latest Advances and Results

Vassil Alexandrov (ICREA-BSC, ITESM)
Aneta Karaivanova (IICT)
Diego Davila (IBSC)
Anton Lebedev (UTubingen)
Oscar Esquevel(ITESM)

VERSAILLES 2017

# Overview

- Introduction -
- Needs and Motivation
- Overview - Monte Carlo Hybrid Methods
- Monte Carlo vs MSPAI
- Experimental results
- Conclusions



| Peak performance | 1,1 PFLOPS |
|---|---|
| Processor | 6.196 8-core Intel SandyBridge EP E5-2670/1600 20M 2.6GHz<br>84 Xeon Phi 5110 P |
| Memory | 100,8 TB |
| Disk | 2000 TB |
| Networks | Infiniband FDR10, GbE |
| OS | SUSE Linux ES |

*Infrastructure funded by:*

# Important Properties of Algorithms

- 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

# Challenges

To achieve excellent results scalability at all levels would be required:

- Mathematical models level
- Algorithmic level
- Systems level

# Monte Carlo Methods FOR LINEAR ALGEBRA

# 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 inde[...]ations ξi of ξ

$$\bar{\xi} = \frac{1}{N} \sum_{i=1}^{N} \xi_i$$

  - Then

$$\bar{\xi} \approx \alpha$$

  - And according to the Law of Large Numbers (LLN)

# Motivation: MC for Linear Algebra

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

# Motivation cont.

- Traditional direct Methods with dense matrices
  - Gaussian elimination
  - Gauss-Jordan
  - Both take O(n3) steps

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

# Monte Carlo Methods

- Fast stochastic approximation

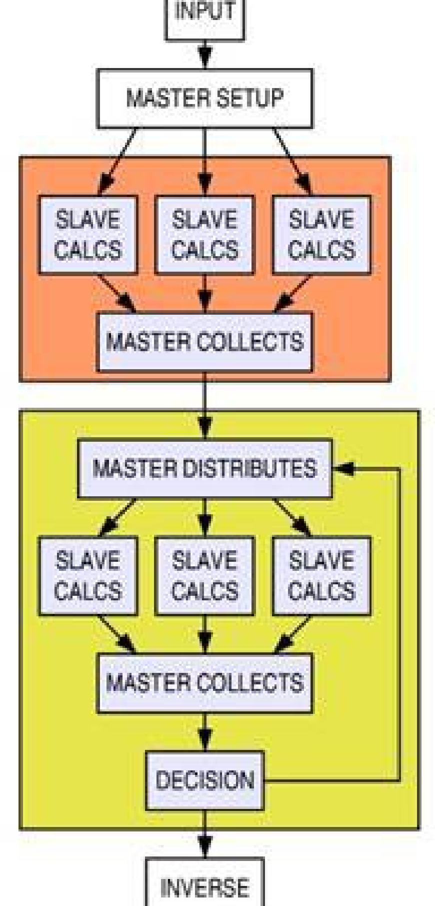
- Very efficient in finding a quick rough estimation
  - element or row of inverse matrix
  - component of solution vector

# Reason for using Monte Carlo

- O(NT) steps to find an element of the
  - inverse matrix B
  - 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 B
- 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 - B1
  - D has only the diagonal elements of ^B
  - B1 includes only off-diagonal elements
- Compute A = D-1B1
- 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 ^B-1

# Parallel Algorithm cont.

# 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 $r$)

# 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:

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

- Therefore

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

- 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:

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

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

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

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

- 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.

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

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

BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
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

# Experiments

## Selected test sets

- The University of Florida Sparse Matrix Collection
- Matrix Market
- Other applications

## Parameter and setting selection

- Computation of pre-conditioner to same accuracy
- Utilized in BiCGSTAB , GMRES or other solvers
- RHS generated from input matrix

## Monte Carlo approach without refinement filter

# 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.

MC 0.1 MC 0.3 MC 0.5 MC 0.7 MC 0.9 FR 0.1 FR 0.3 FR 0.5 FR 0.7 FR 0.9

Frobenius norm

Monte Carlo

Execution Time in Seconds

35
30
25
20
15
10
5
0

500 1000 2000 3000 4000 5000 6000 7000 8000 9000

Matrix Size

BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
MC vs MSPAI

# Test Matrices

| Matrix | Dimension | Non-zeros | Sparsity | Symmetry |
| --- | --- | --- | --- | --- |
| 1. Appu | 14,000 | 1,853,104 | 0.95% | non-symmetric |
| 2. Na5 | 5,832 | 305,630 | 0.46% | symmetric |
| 3. Nonsym_r5_a11 | 329,473 | 10,439,197 | 0.01% | non-symmetric |
| 4. Rdb2048 | 2, 048 | 12,032 | 0.29% | non-symmetric |
| 5. Sym_r3_a11 | 20, 928 | 588,601 | 0.13% | symmetric |
| 6. Sym_r4_a11 | 82, 817 | 2,598,173 | 0.04% | symmetric |

# Execution Time Breakdown

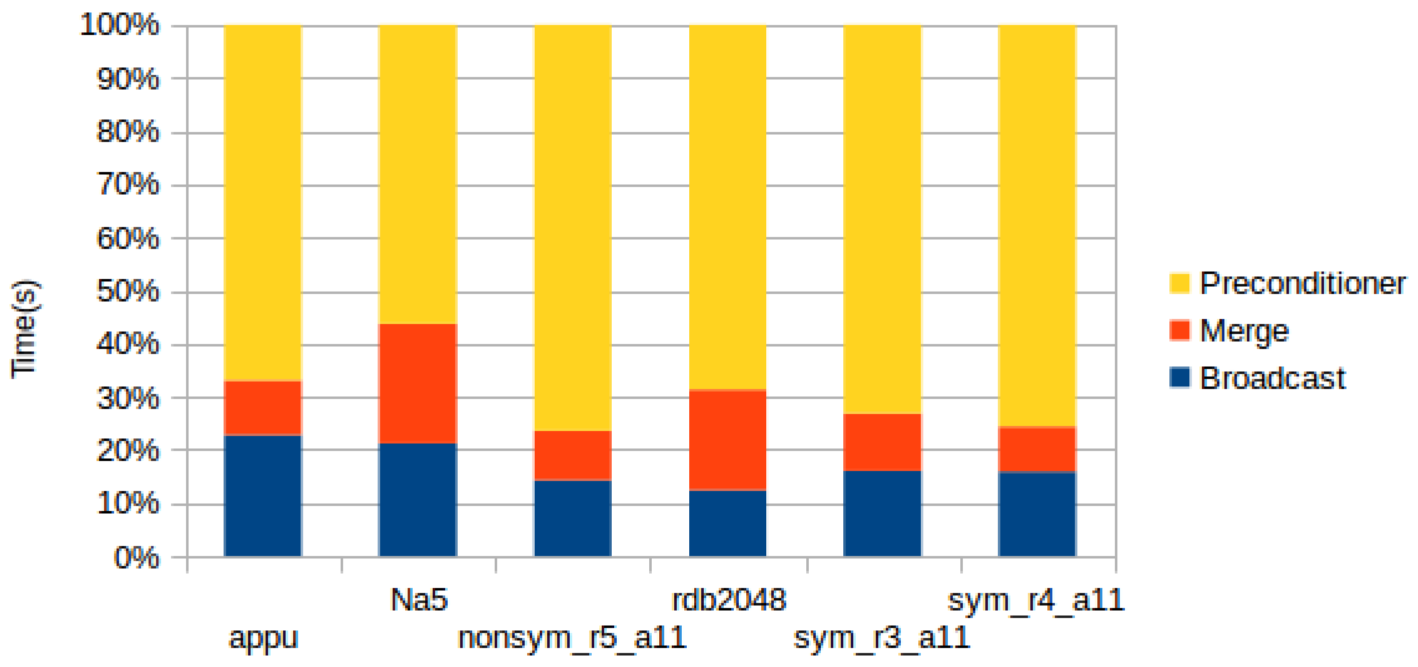



Figure 1. Execution time breakdown in a 16 cores execution.

# Execution Time Breakdown

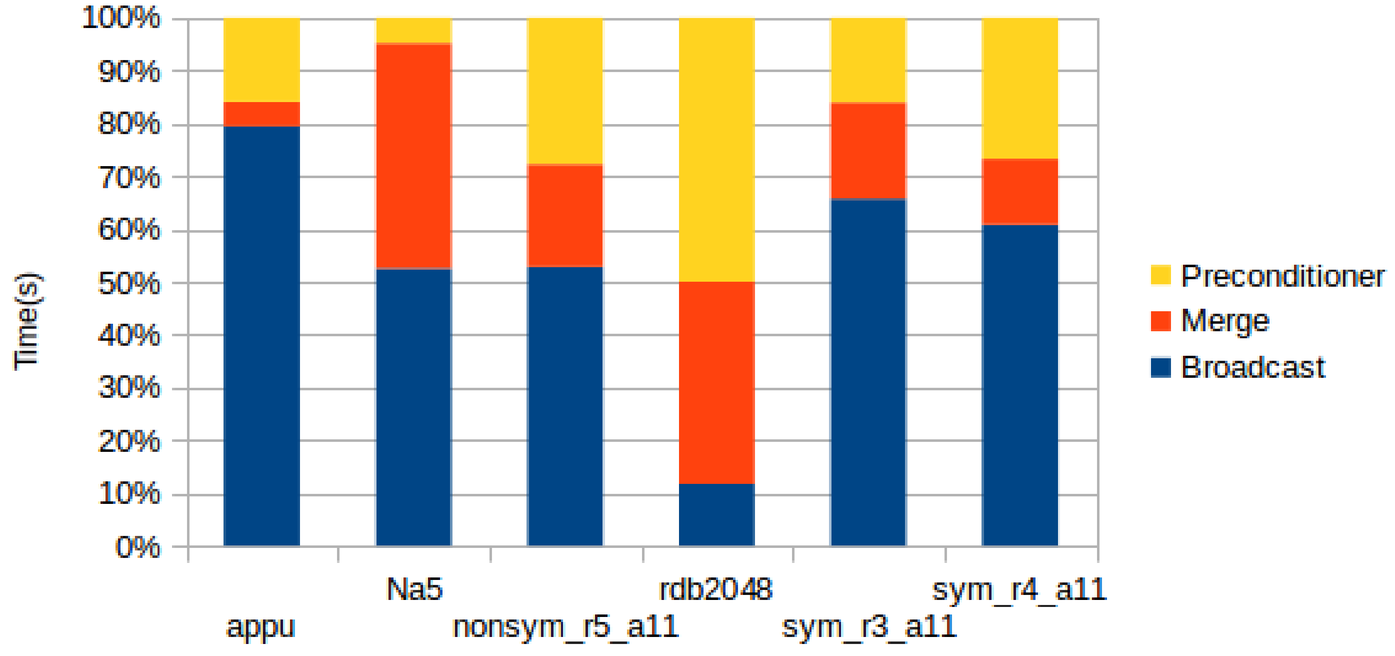



Figure 2. Execution time breakdown in a 256 cores execution.

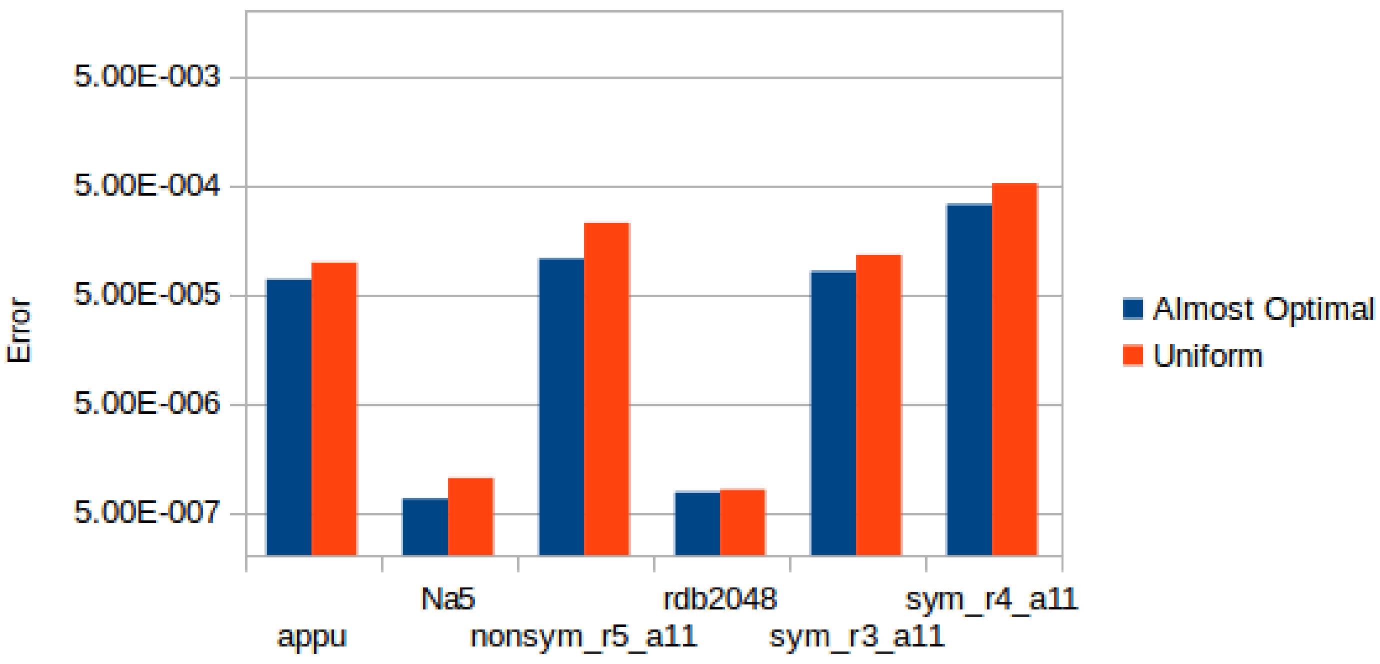



Figure 6. Error calculation when using Uniform and Almost Optimal distributions with 16 cores.

# Employing Mixed MPI/OpenMP version

Master MPI process

Broadcast

Worker MPI process
( 1 process per node)

Work partition among openMP threads

OpenMP thread
(4 threads per node)

Internal OpenMP merge

External MPI merge

# Mixed MPI/OpenMP

nlpkkt120_v9.2
nlpkkt120_v10.0

Figure 8. Scalability comparison for the two-step broadcast for a relatively big matrix (3.5M x 3.5M)

# MC vs MSPAI

appu

Time(s)

1000
100
10
1
0.1
0.01
0.001
0.0001
0.00001

16
32
64
128
256
512

Num. cores

MSPAI
MC_v9.2
MC_nocomm

Figure 9. Scalability comparison MSPAI and MC for matrix appu

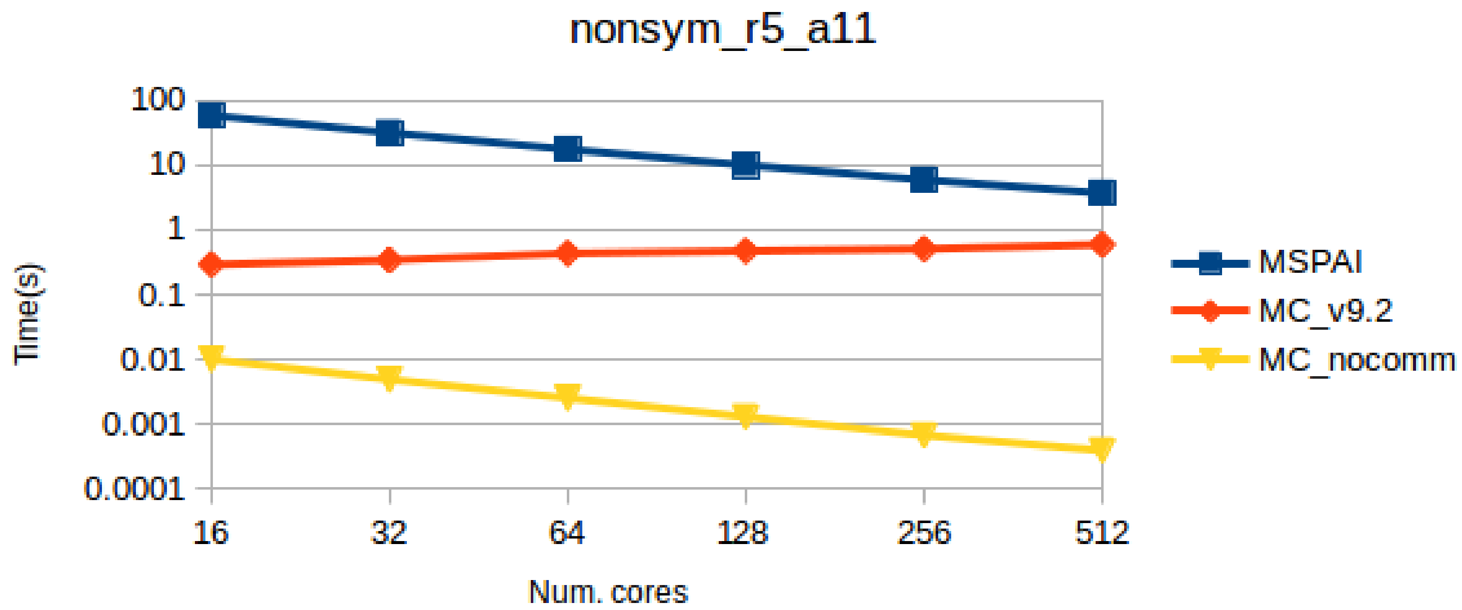



Figure 10. Scalability comparison MSPAI and MC for matrix non-sym r5 a11.

# MC vs MSPAI

Figure 11. Scalability comparison MSPAI and MC for matrix rdb2048.

# Preconditioner calculation

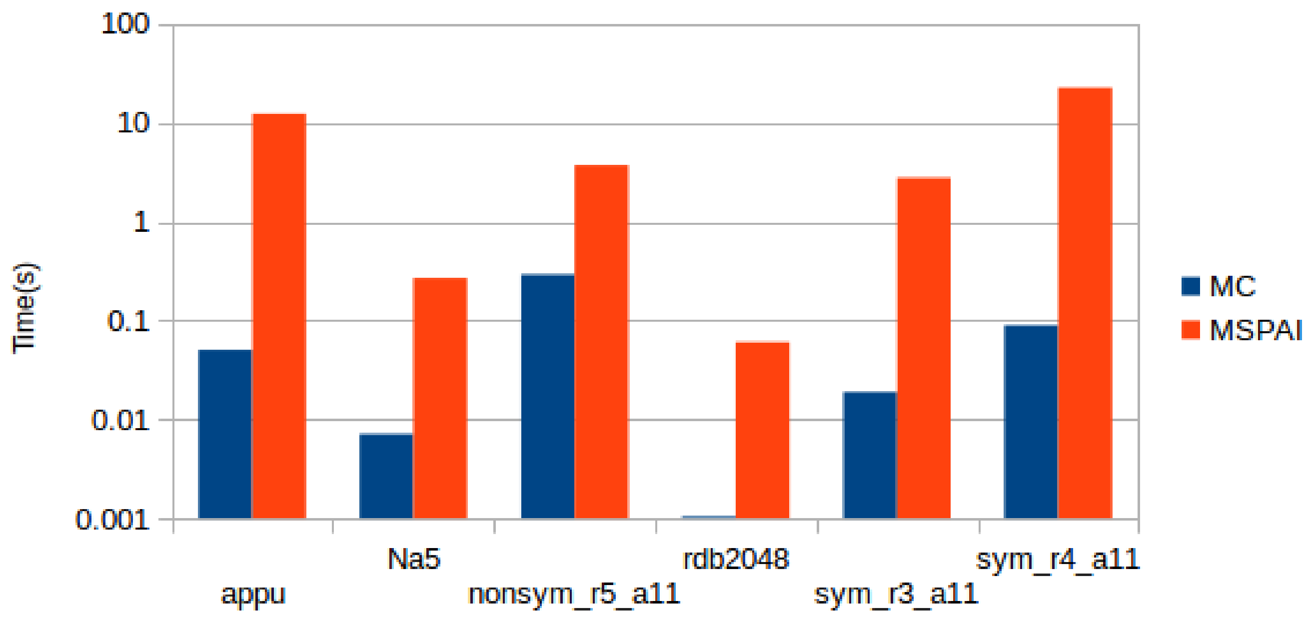



Figure 12. Fastest execution time achieved during the preconditioner calculation.

# GMRES timing

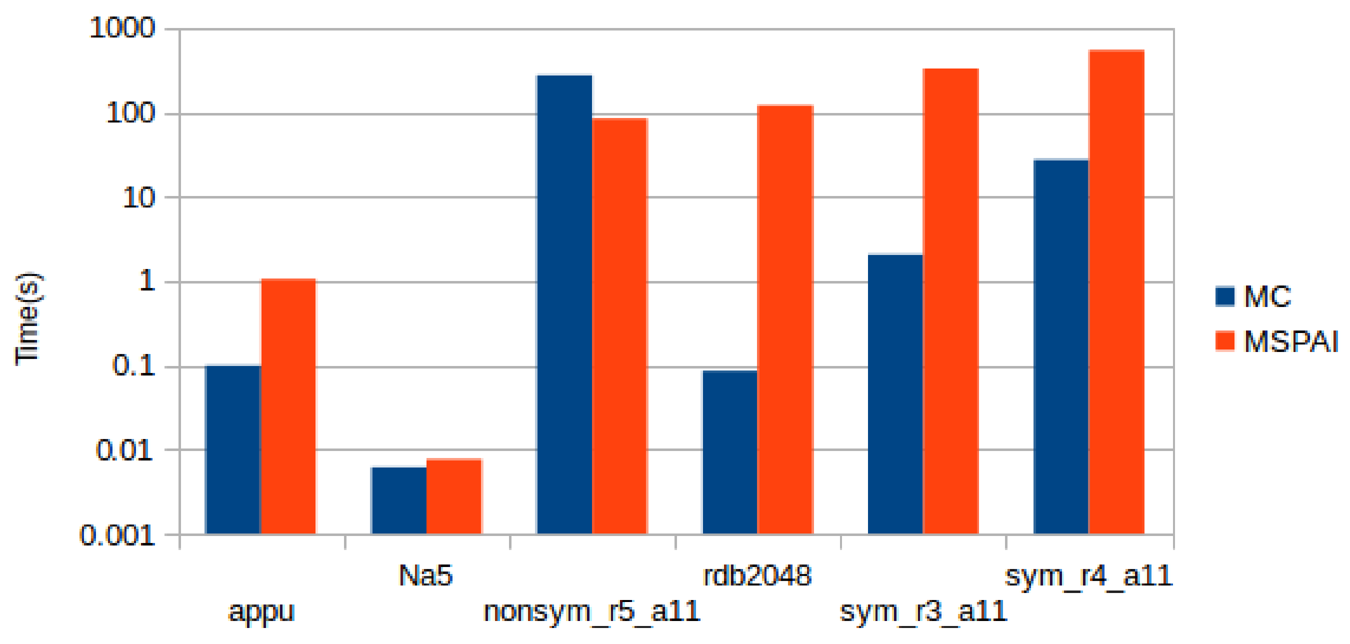



Figure 13. Time required by the solver to find the solution for the preconditioned system.

# Total time

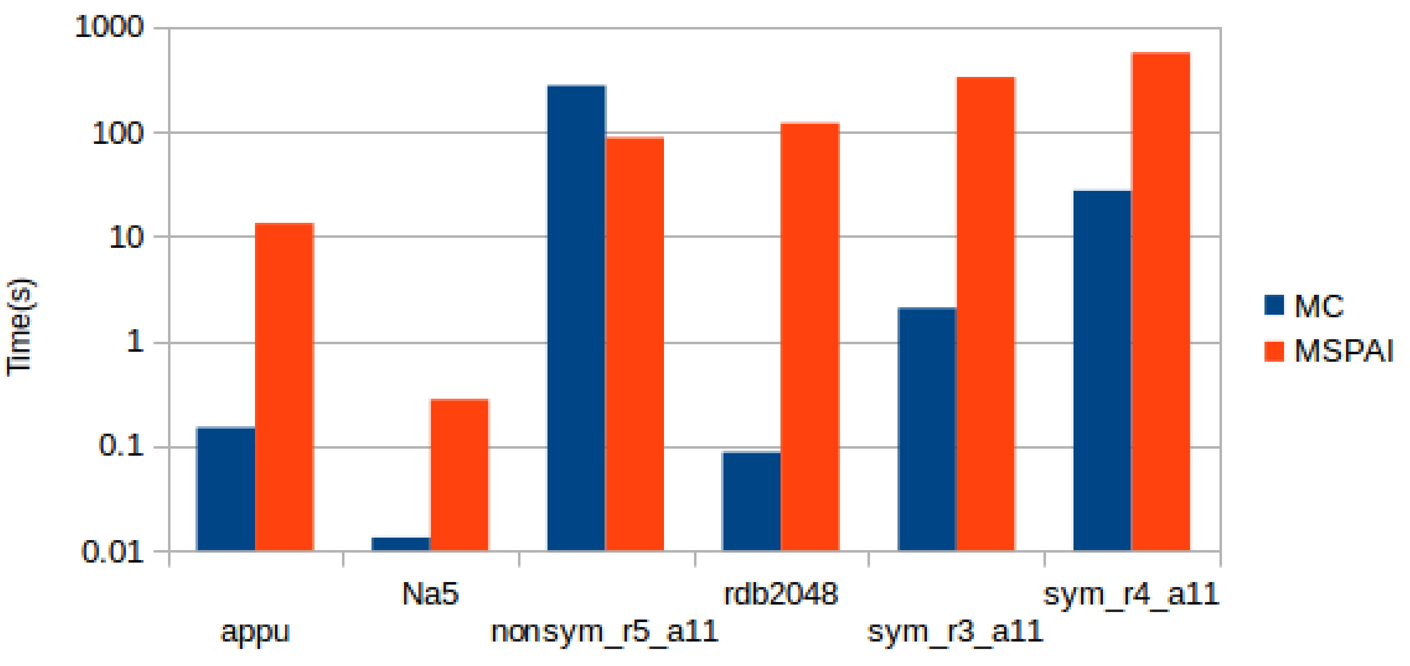



Figure 14. Total time = Preconditioner construction time + Solver execution time.

# Low discrepancy (quasirandom) sequences

- The quasirandom sequences are deterministic sequences constructed to be as uniformly distributed as mathematically possible (and, as a consequence, to ensure better convergence for the integration)
- The uniformity is measured in terms of discrepancy which is defined in the following way: For a sequence with N points in $[0,1]^s$ define

$$R_N(J) = \frac{1}{N}\#\{x_n \in J\} - vol(J) \text{ for every } J \subset [0,1]^s$$

$$D_N^* = \sup_{E^*} |R_N(J)|,$$

$E^*$ - the set of all rectangles with a vertex in zero.

- A s-dimensional sequence is called quasirandom if

$$D_N^* \leq c(\log N)^s N^{-1}$$

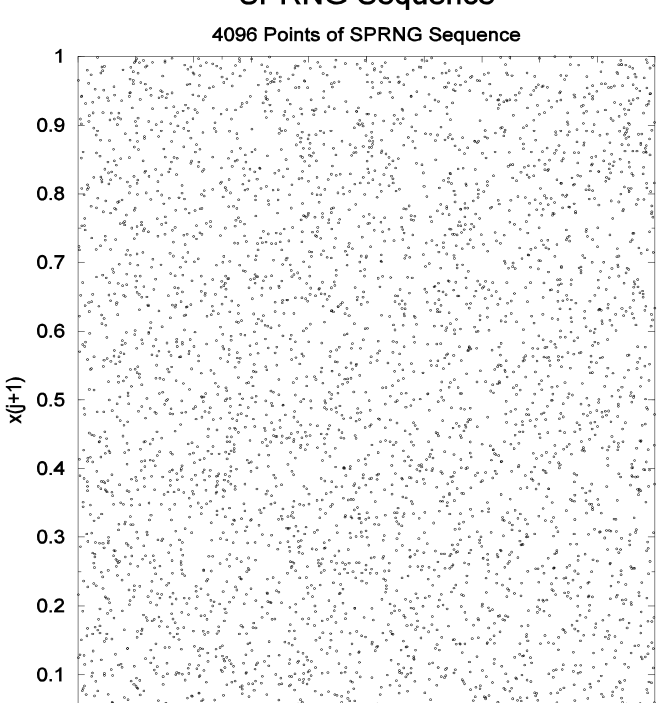
- Koksma-Hlawka inequality (for integration):

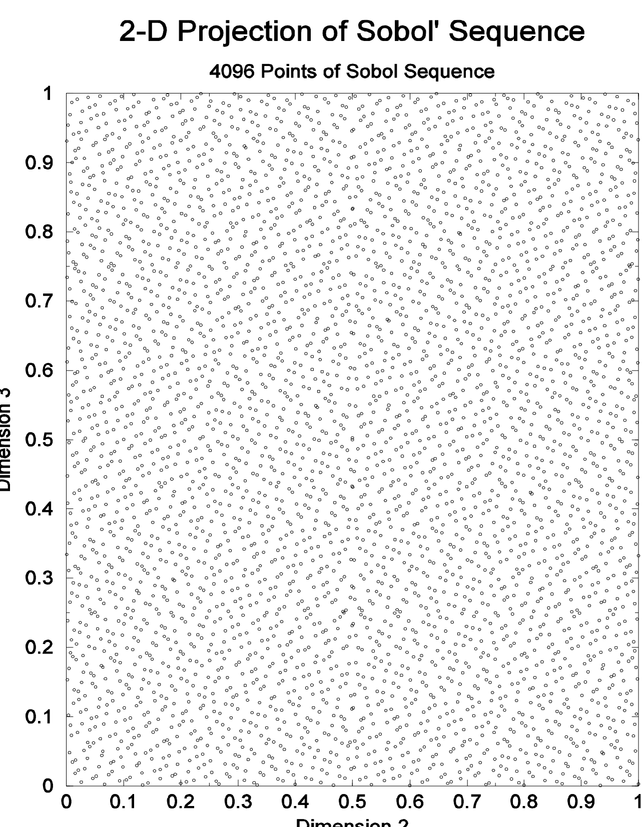
$$\varepsilon[f] \leq V[f]\, D_N^*$$

(where $V[f]$ is the variation in the sense of Hardy-Kraus)

- The order of the error is $O((\log N)^s N^{-1})$

# PRNs and QRNs



# Some facts

- Discrepancy of real random numbers:

  D*N = O(N-1/2 (log log N)-1/2)

- Klaus F. Roth (Fields medal 1958) proved the following lower bound for star discrepancy of N points in s dimensions:

  D*N ≥O(N-1 (log N)(s-1)/2)

- Sequences (indefinite length) and point sets have different “best” discrepancies:
  - Sequence: D*N ≤ O(N-1 (log N)s-1)
  - Point set: D*N ≤ O(N-1 (log N)s-2)

# Most often used sequences (Halton Sequence)

- Let n be an integer presented in base p. The p-ary radical inverse function is defined as

$$\phi_p(n) \equiv \frac{b_0}{p} + \frac{b_1}{p^2} + \dots + \frac{b_m}{p^{m+1}}$$

where $p$ is prime and $b_i$ comes from

$$n = b_0 + b_1 p + \dots + b_m p^m,$$

with $0 \le b_i < p$

- An s-dimensional Halton sequence is defined as:

$$(\phi_{p_1}(n), \phi_{p_2}(n), \dots, \phi_{p_s}(n))$$

with $p_1$ $p_2$ …., $p_s$ being relatively prime, and usually the first $s$ primes

# Most often used sequences

- In our computations we have used scrambled modified Halton sequence [Atanassov 2003]:

  xn(i) = ∑j=0m imod (aj(i)kij+1 + bj(i),pi) pi–j-1

  (scramblers bj(i), modifiers ki in [0, pi – 1] )

# Most often used sequences (Sobol)

- Sobol sequence (1967) $\{x_n = (x_n(1), x_n(2), \ldots, x_n(s))\}$
- The j-th coordinate of the n-th point of s-dimensional ***Sobol sequence*** $x_n = (x_n(1), x_n(2), \ldots, x_n(s))$ is generated through the recursion:

$$x_n(j) = b_1 v_1(j) \otimes b_2 v_2(j) \otimes \ldots b_w v_w(j)$$

where $v_i(j)$ is i-direction number for dimension j, and $\otimes$ is bit-by-bit exclusive-or operation ($b_i$ are the coefficients of representation of n in base 2)

- How to determine $v_i(j)$ : for each dimension a different primitive polynomial is chosen and its coefficients are used to define:

# Quasirandom Sequences and their scrambling

- Unfortunately, the coordinates of the quasirandom sequence points  in high dimensions show correlations. A possible solution to this problem is the ***scrambling***.
- The purpose of scrambling:
  - To improve 2-D projections and the quality of quasirandom sequences in general
  - To provide practical method to obtain error estimates for QMC
  - To provide simple and unified way to generate quasirandom numbers for parallel computing environments
  - To provide more choices of QRN sequences with better (often optimal) quality to be used in QMC applications

# Scrambling techniques

- Scrambling was first proposed by Cranley and Patterson (1979) who took lattice points and randomized them by adding random shifts to the sequences. Later, ***Owen*** (1998, 2002, 2003) and ***Tezuka*** (2002) independently developed two powerful scrambling methods through permutations
- Although many other methods have been proposed, most of them are modified or simplified Owen or Tezuka schemes (*Braaten and Weller, Atanassov, Matousek, Chi and Mascagni, Warnock*, etc.)
- There are two basic scrambling methods:
  - Randomized shifting
  - Digital permutations

  (Permuting the order of points within the sequence)
- The problem with Owen scrambling is its computational complexity

# Scrambling

- *Digital permutations*: Let $(x^{(1)}_n, x^{(2)}_n, \ldots, x^{(s)}_n)$ be any quasirandom point in $[0, 1)^s$, and $(z^{(1)}_n, z^{(2)}_n, \ldots, z^{(s)}_n)$ is its scrambled version. Suppose each $x^{(j)}_n$ has a b-ary representation $x^{(j)}_n = 0.\, x^{(j)}_{n1}\, x^{(j)}_{n2} \ldots x^{(j)}_{nK}, \ldots$ with K defining the number of digits to be scrambled. Then

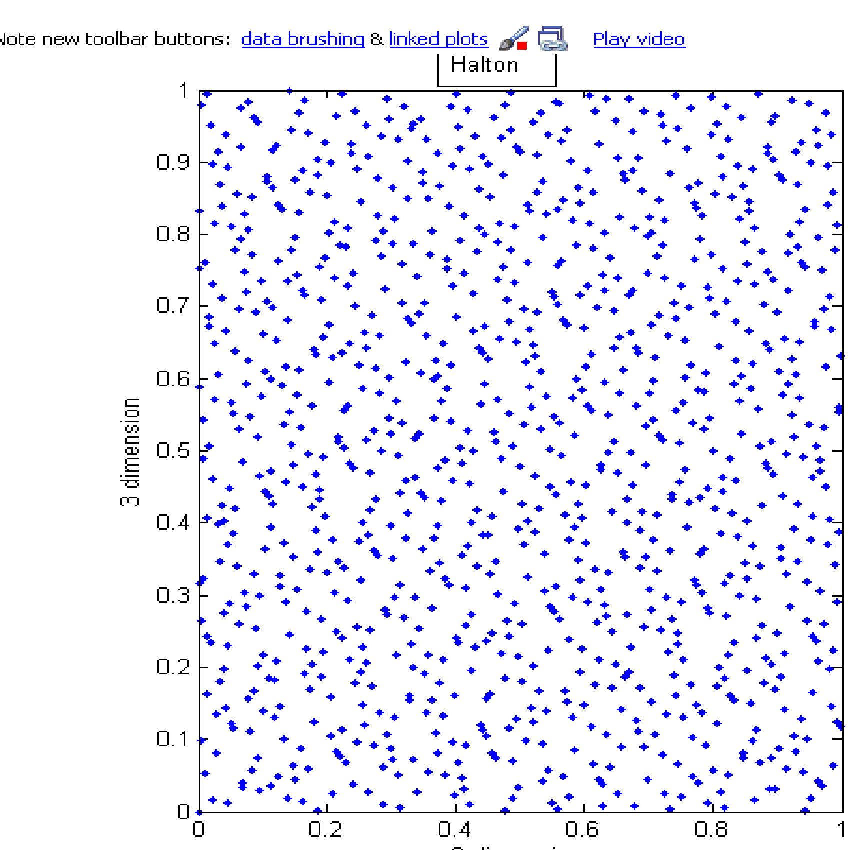
$$z^{(j)}_n = \sigma(x^{(j)}_n), \text{ where } \sigma=\{\Phi_1, \ldots, \Phi_K\}$$ и $\Phi_i$, is a uniformly chosen permutation of the digits $\{0,1,\ldots,b-1\}$.

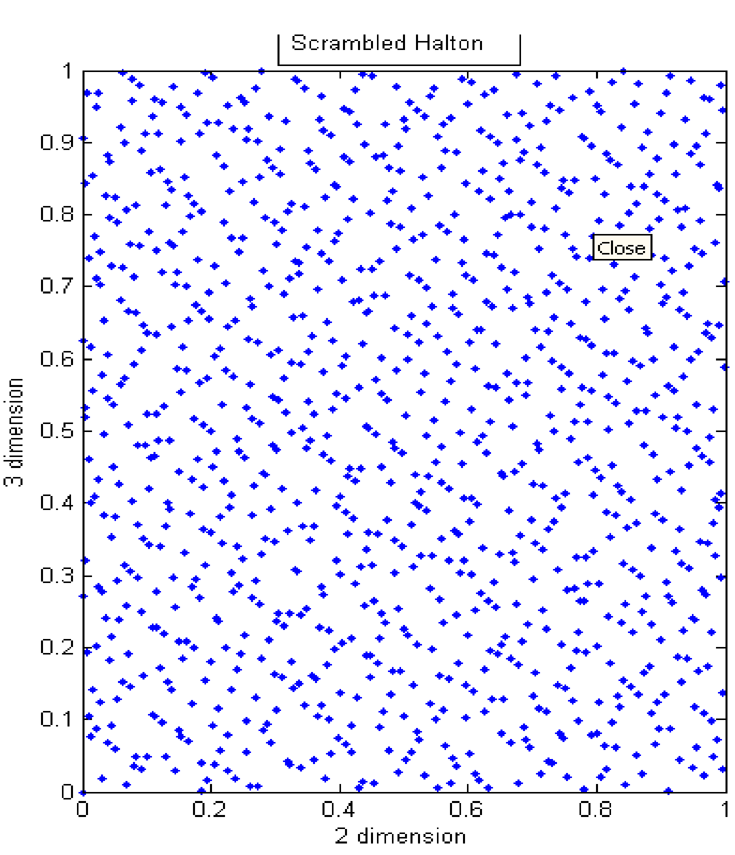
- *Randomized shifting* has the form

$$z_n = x_n + r \pmod 1,$$

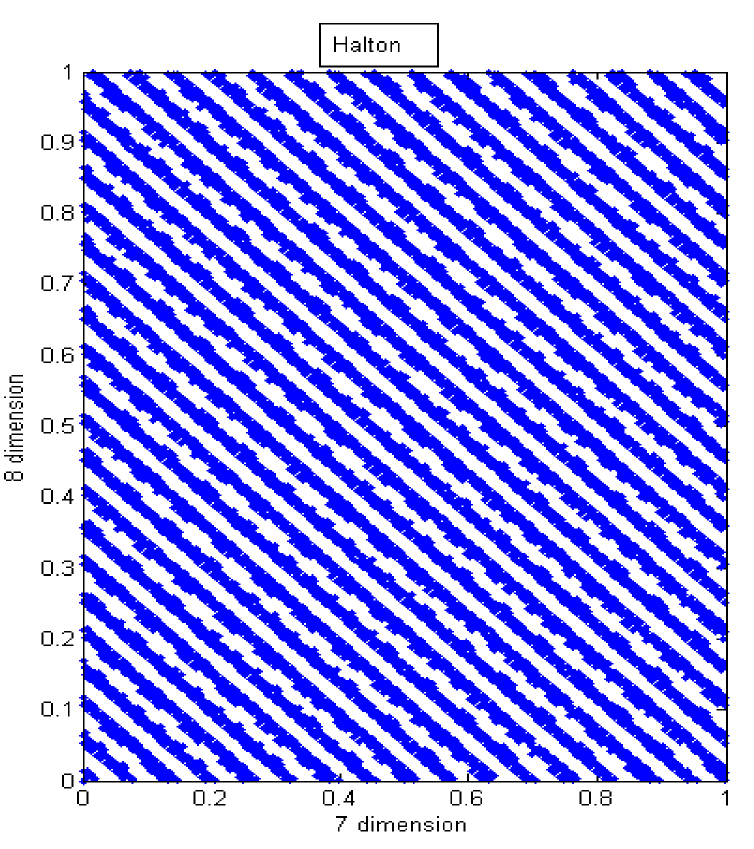
where $x_n$ is any quasirandom number in $[0, 1)^s$ and $r$ is a single s-dimensional pseudorandom number.

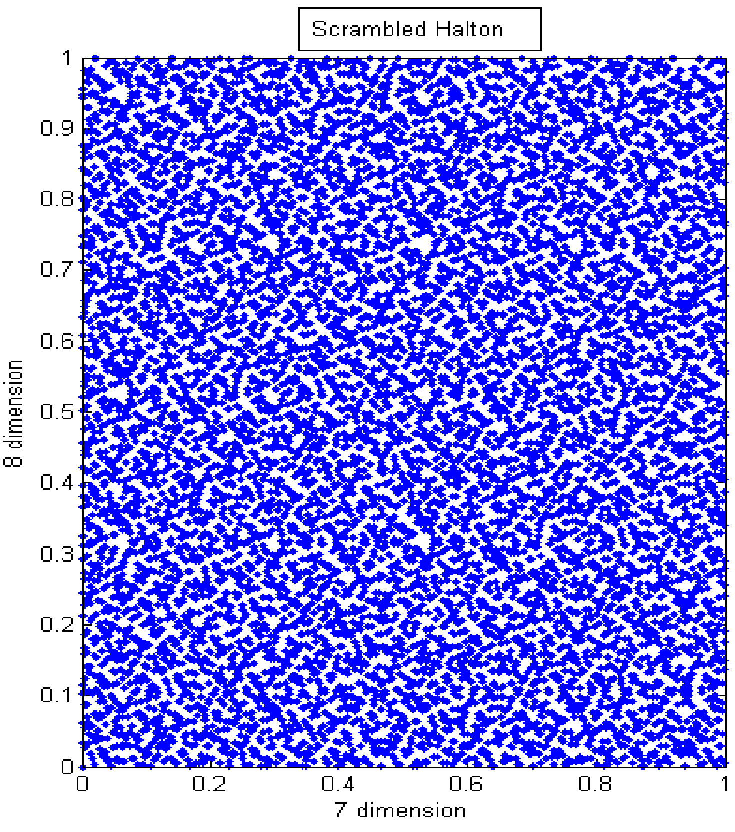
# Two-dimensional projection of Halton sequence and scrambled Halton sequence (dimension 3)



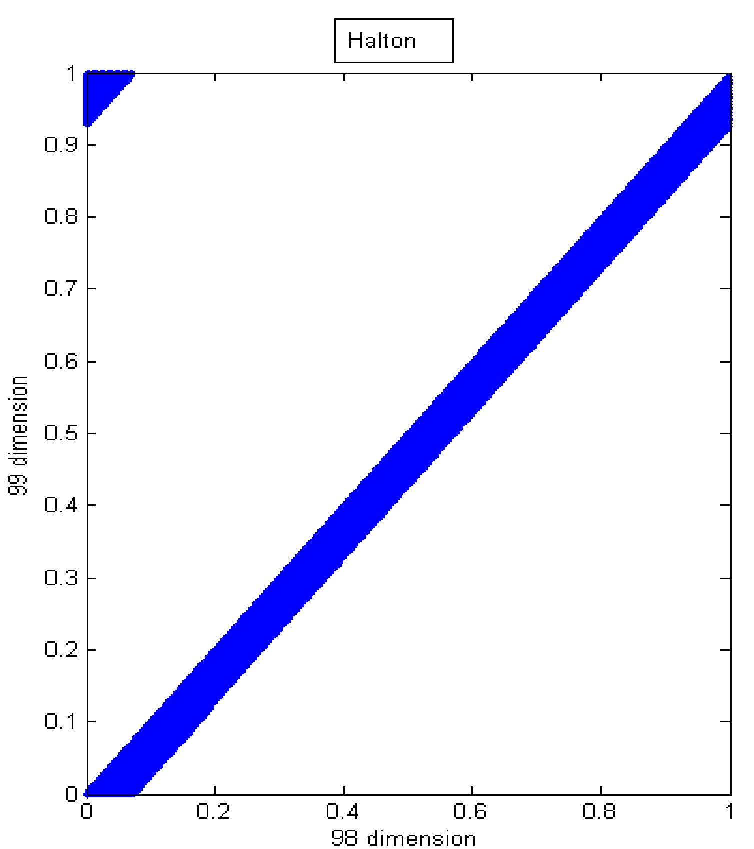



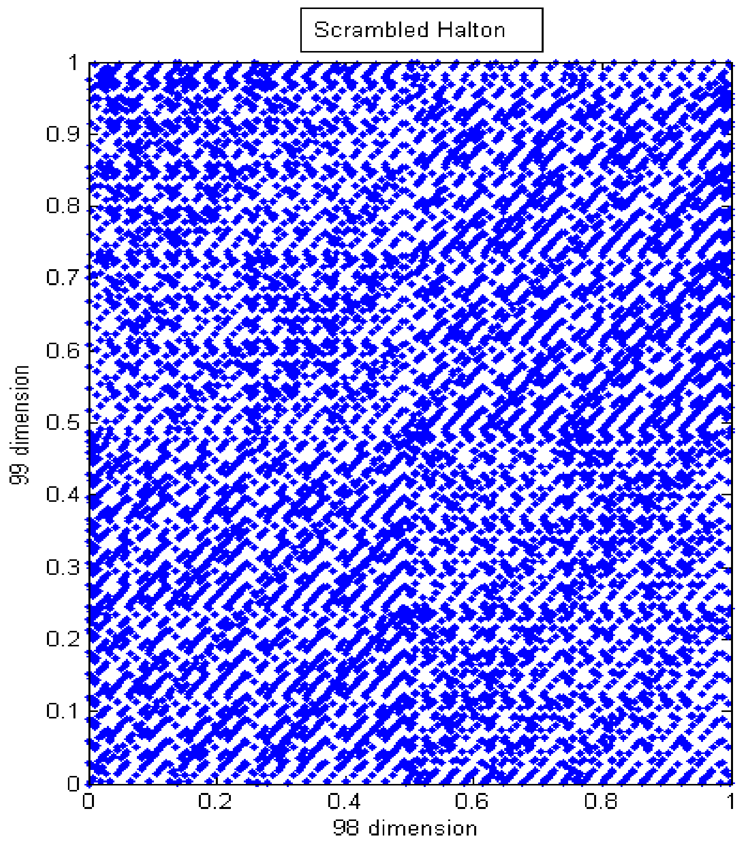
# Two-dimensional projection of Halton sequence and scrambled Halton sequence (dimension 8)



# Two-dimensional projection of Halton sequence and scrambled Halton sequence (dimension 99)



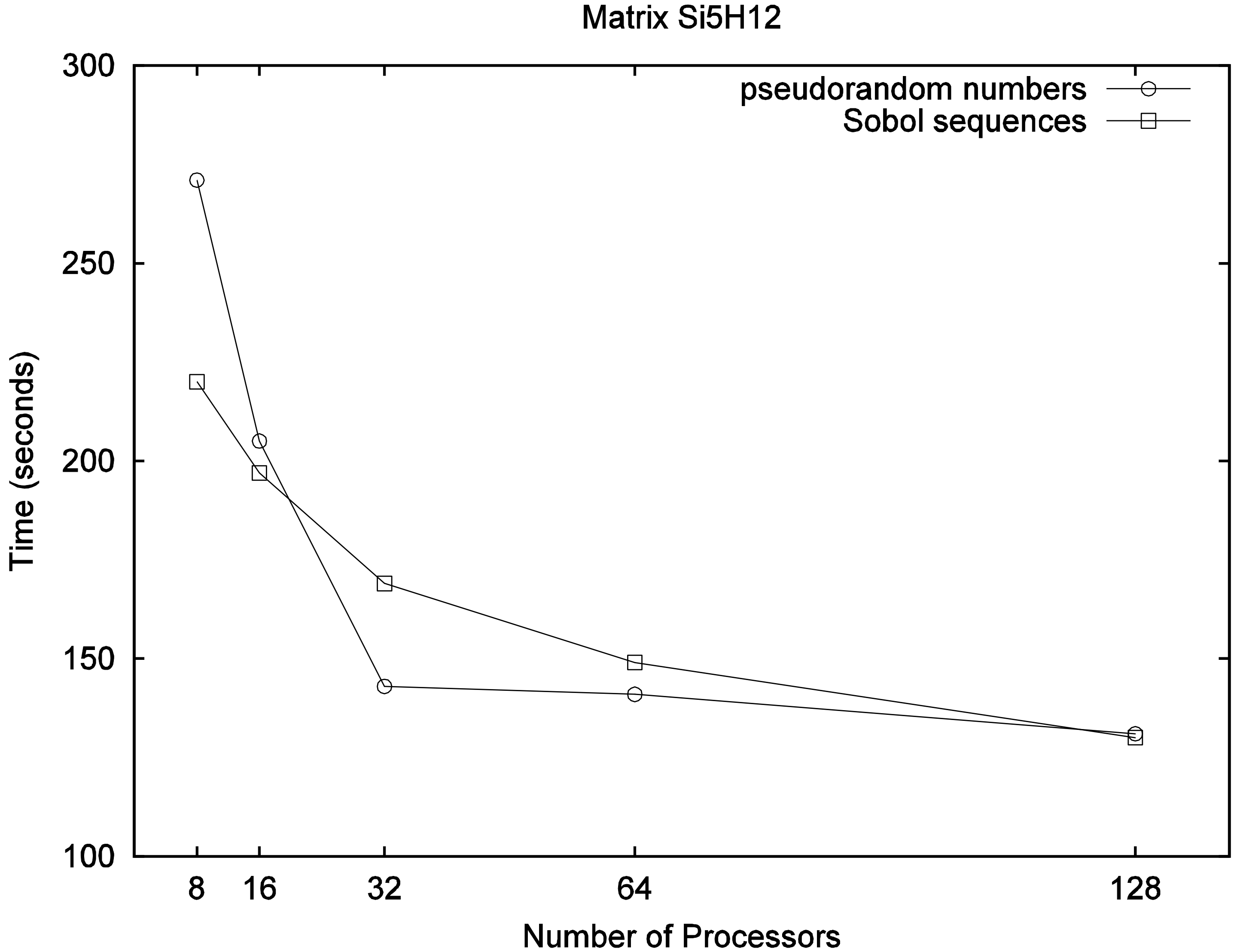
# Error estimate for QMC

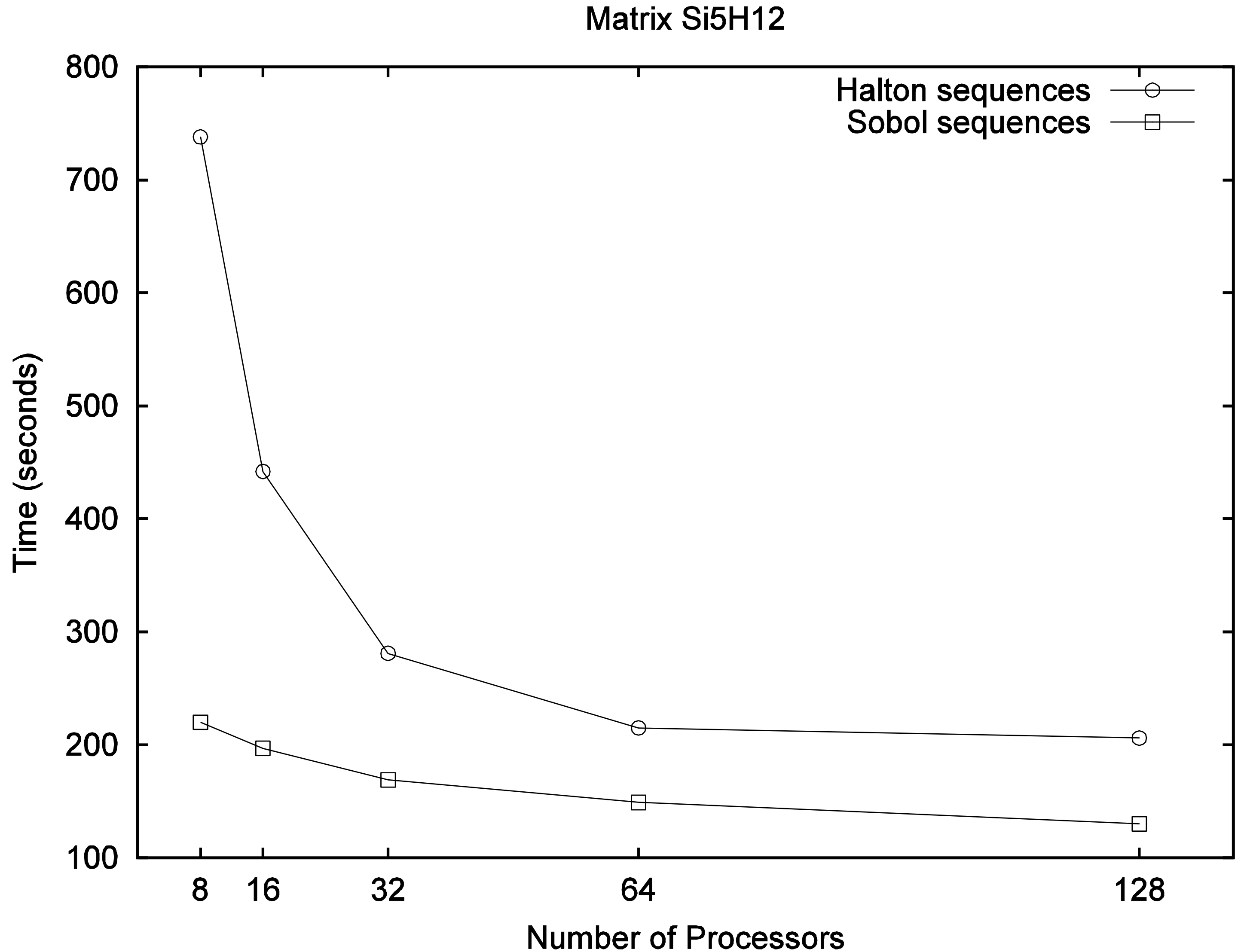
- Scrambling provides a practical method to obtain error estimates for QMC based by treating each scrambled sequence as a different and independent random sample from a family of randomly scrambled quasirandom numbers, thus allowing standard (Gaussian) confidence intervals to be considered.

- QMC error for Markov chain based problems:

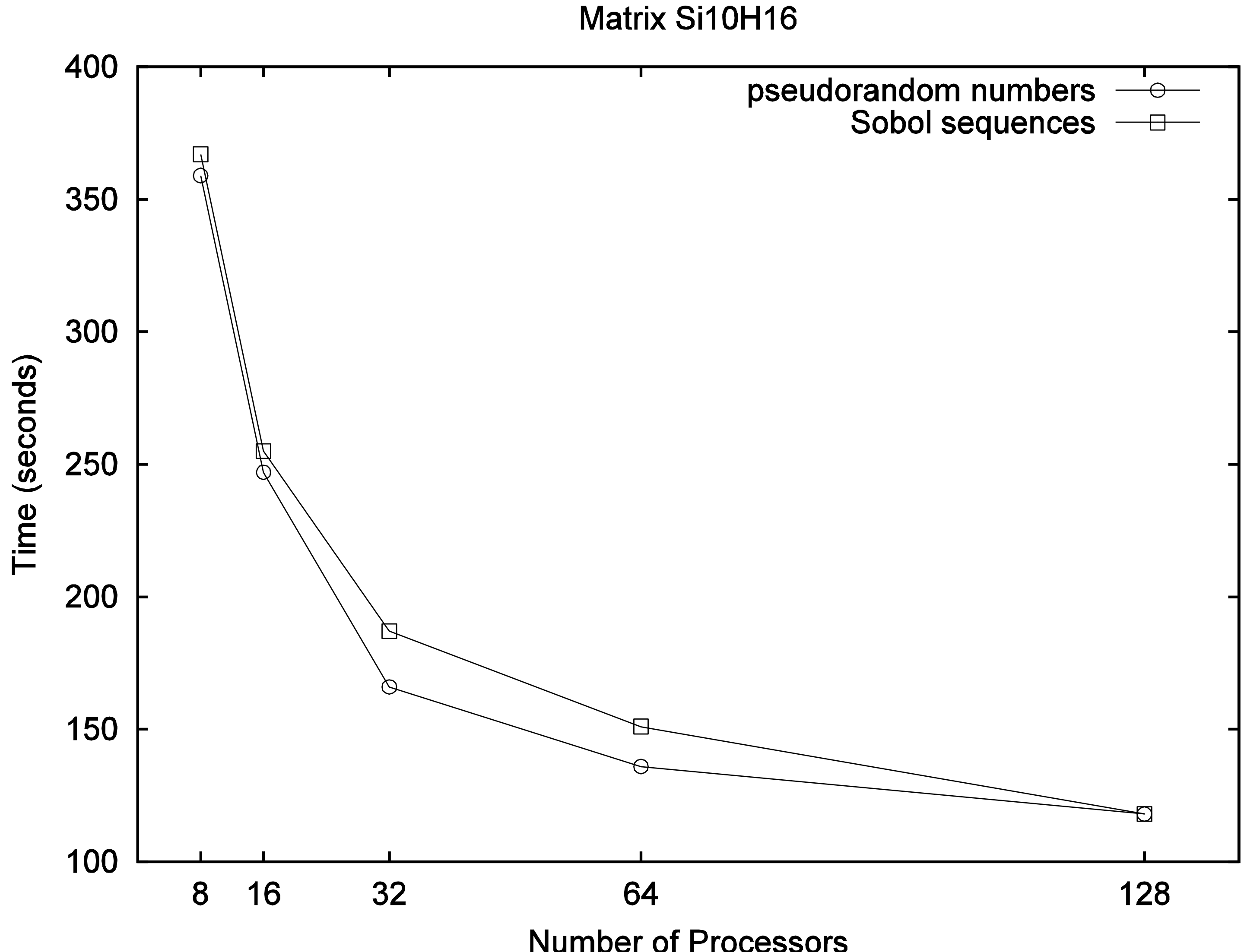
$$\delta N(\zeta(Q')) \leq V(\zeta \circ \Gamma-1).(D^*N(Q))$$

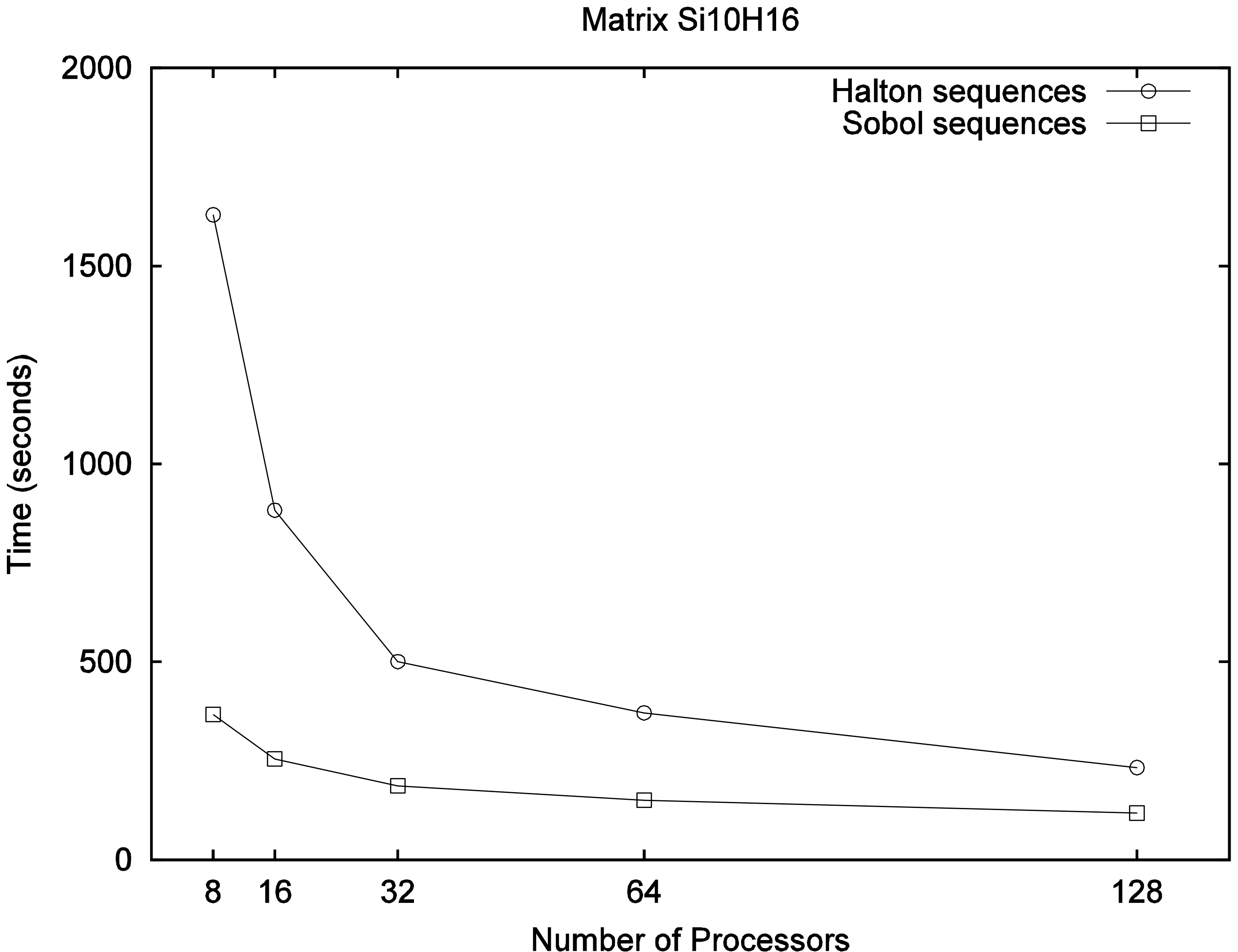
where Q = {γi} is a sequence of vectors in [0,1)sT, Q' = {ωi} is a sequence of quasirandom walks generated from Q using the mapping Γ.


Matrix Si5H12
pseudorandom numbers
Sobol sequences
Time (seconds)
300
250
200
150
100
8
16
32
64
128
Number of Processors



Matrix Si5H12
Halton sequences
Sobol sequences
Time (seconds)
800
700
600
500
400
300
200
100
8
16
32
64
128
Number of Processors


Matrix Si10H16
400
350
300
250
200
150
100
Time (seconds)
pseudorandom numbers
Sobol sequences
8
16
32
64
128
Number of Processors


Matrix Si10H16
Halton sequences
Sobol sequences
Time (seconds)
2000
1500
1000
500
0
8
16
32
64
128
Number of Processors

# MC vs QMC

Precision ep 0.5

Seconds

0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0

16
32
64

Processors

MC appu
Halton appu
MC Na5
Halton Na5

Figure 15. MC and QMC preconditioners execution time

# MC vs QMC

Precision ep 0.1

Seconds: 0,25 · 0,2 · 0,15 · 0,1 · 0,05 · 0

Processors: 16 · 32 · 64

MC appu · Halton appu · MC Na5 · Halton Na5

Figure 16. MC and QMC  preconditioners execution time

# MC vs QMC

| Matrix | QMC-Halton, eps=0.5 | MC, eps=0.5 | QMC-Halton, eps=0.1 | textbfMC, eps=0.1 |
|---|---|---|---|---|
| Appu | 0.135902 | 0.139417 | 1.62762 | 2.78041 |
| bcsstm13 | 106.616 | 107.129 | 119.986 | 120.103 |
| Na5 | 0.012513 | 0.013384 | 0.027767 | 0.025393 |
| Rdb2048 | 0.158109 | 0.197112 | 0.17864 | 0.170368 |
| Si10H16 | 0.0312071 | 0.0299697 | 0.4368090 | 0.4941460 |

Figure 17. MC and QMC solver times

# Further Improvements

## Discarding elements of the matrix

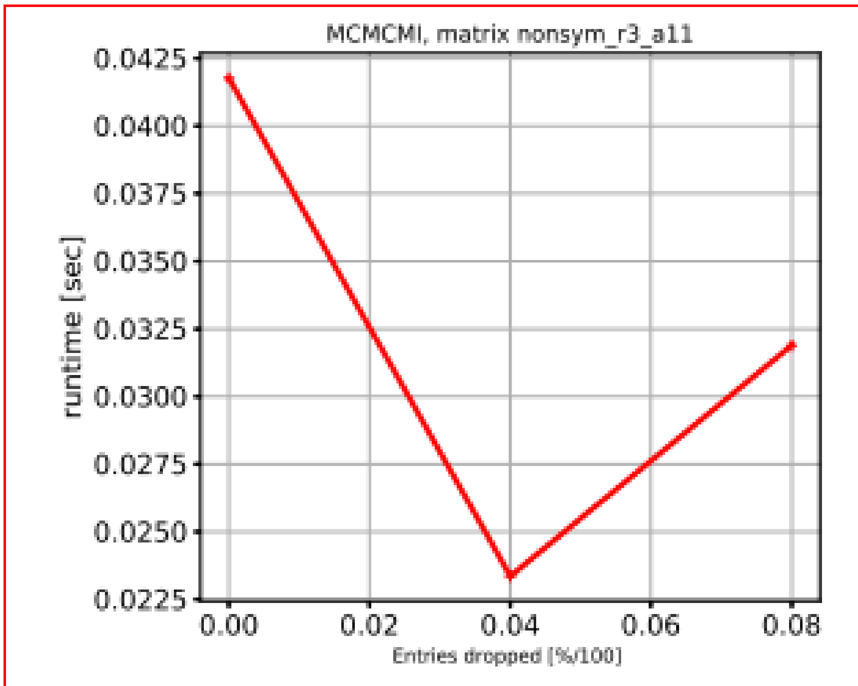



Figure 3. Execution time of the preconditioner computation.

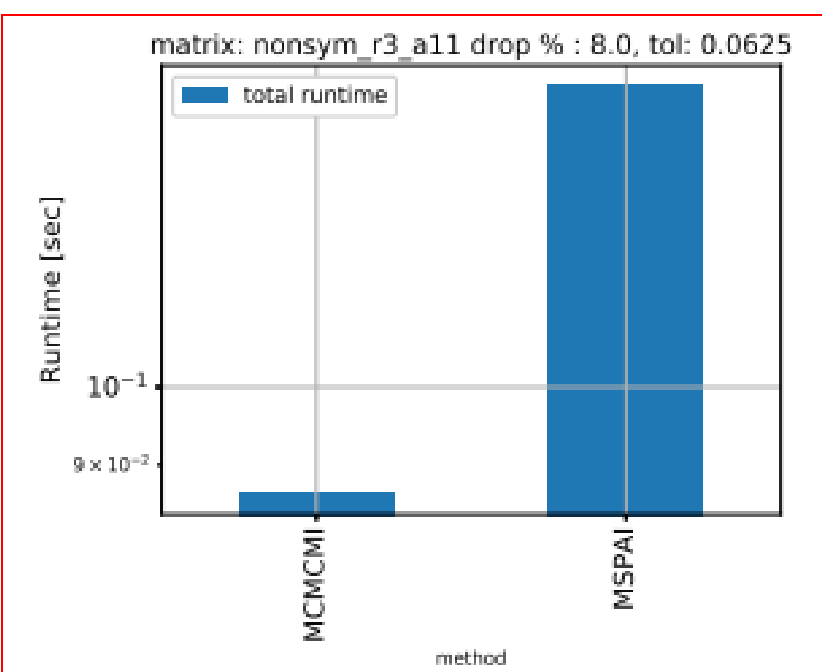



Figure 2. Total time = preconditioner construction time + solver execution time.

# Further Improvements

Preconditioner computation, N=[512]

runtime [sec]

$10^2$
$10^1$
$10^0$
$10^{-1}$

MCMCMI
MSPAI

ID1_2_P3_7_stiffness
rdb2048
sym_r6_a11
nonsym_r3_a11

# Further Improvements

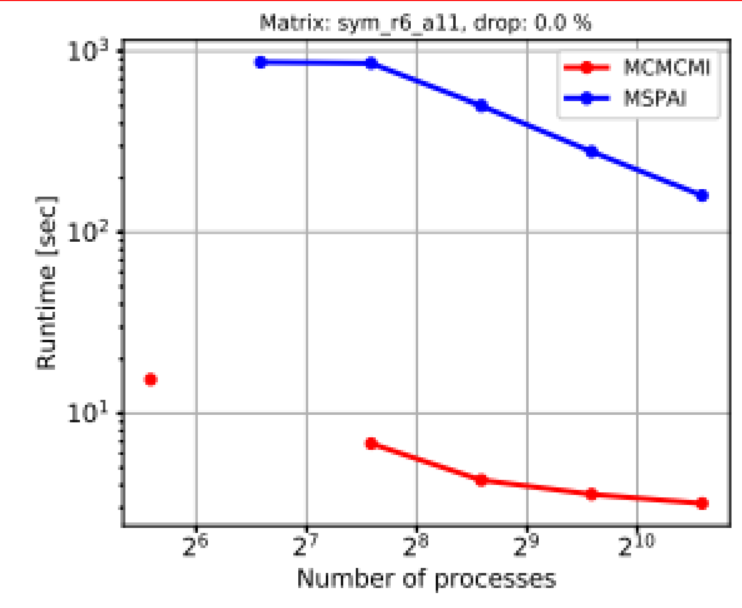



Figure 7. Execution time of the preconditioner computation for varying number of *processes*. Transition probabilities are computed by the master process and broadcast to the workers.

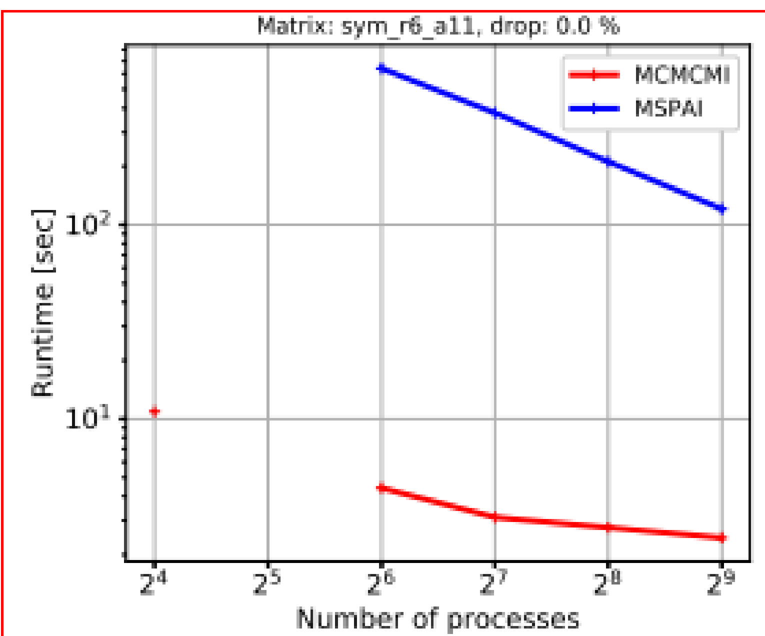



Figure 8. Execution time of the preconditioner computation for varying number of *processes* with transition probabilities not being broadcast.

# Further Work

- MC stochastic projection approach
- GPU based implementations
- Further experiments and comparisons

# Conclusions

# Conclusions and Future Work

- MC and QMC provide good quality preconditioners
- Need to enhance the reuse of sub-chains in longer Markov Chains
- quasi-Monte Carlo and MC deliver the same quality preconditioners

www.bsc.es

Barcelona Supercomputing Center
*Centro Nacional de Supercomputación*

# Questions ?

vassil.alexandrov@bsc.es

http://www.bsc.es/computer-sciences/extreme-computing