EuroCC@Turkey
Parallel Computing on GPUs with CUDA

Dr. Özcan DÜLGER

Computer Engineering, Middle East Technical University

Computer Engineering, Artvin Coruh University

27 April 2022

EURO

TRUBA
Turkish Science e-Infrastructure

TÜBİTAK
ULAKBİM

Sabancı Üniversitesi

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

Contents

- Debugging and Profiling Performance
- Performance Optimization and Efficiency
- Some Libraries and Remaining Issues
- **CUDA Samples**

TESLA K40

Property	Value	Property	Value
Architecture	Kepler	Global Memory	11520 MB
Number of SMX	15	Shared Memory	49152 Byte
CUDA Core	2880	L2 Cache	1572864 Byte
Core Clock	745 MHz	Segment Size	128 Byte
Max. Thread / SMX	2048	Warp Size	32
Max. Thread / Block	1024	Max. Block / SMX	16

CUDA Samples

- Memory Coalescing Implementation of Metropolis Resampling
  - Coalesced variants of Metropolis Resampling
    - Metropolis-C1
    - Metropolis-C2
  - Global memory load transactions of Metropolis, Metropolis-C1 and Metropolis-C2
  - Compare their performance:
    - Bias and Mean Squared Error (MSE)
    - Target Tracking

  - Dülger, Ö., Oğuztüzün, H. and Demirekler, M. “Memory Coalescing Implementation of Metropolis Resampling on Graphics Processing Unit”, Journal of Signal Processing Systems vol.90, pp. 433–447 (2018)

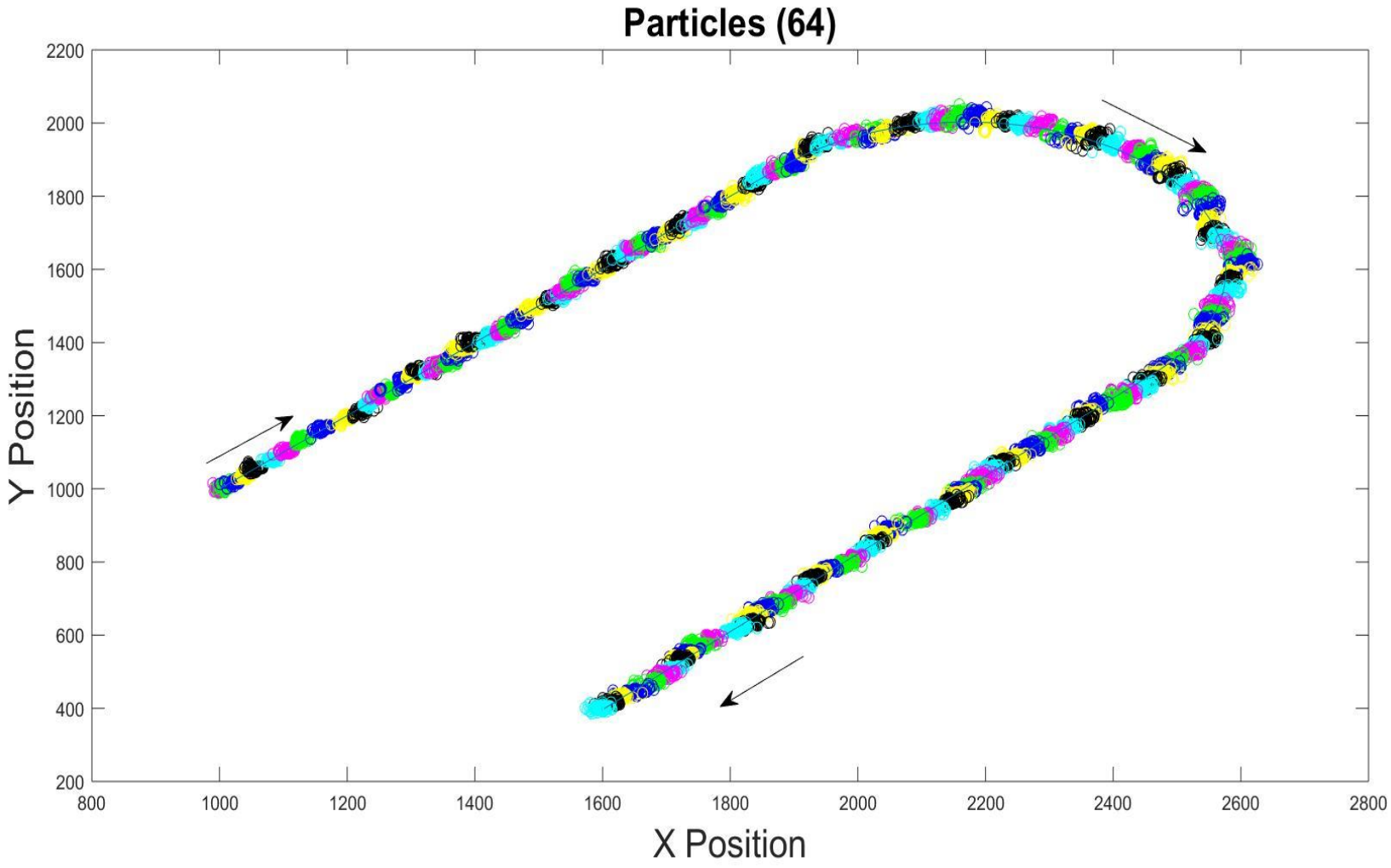
Outline

- Particle filter and resampling stage
- Metropolis resampling and non-coalesced global memory access problem
- Our proposed Metropolis-C1 and Metropolis-C2 resampling techniques
- Experimental results

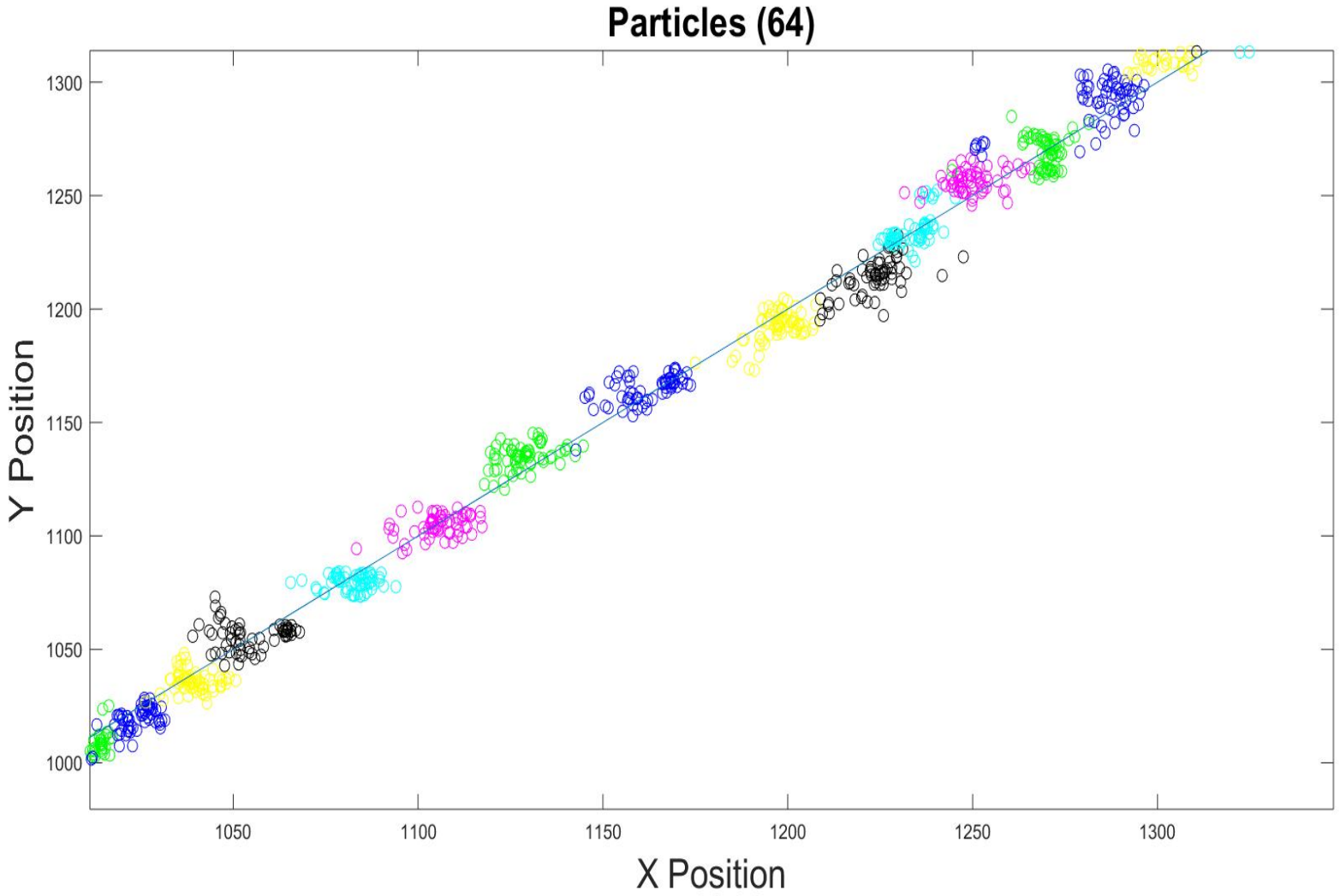
Particle Filter

- Serial Monte Carlo Estimation Method
  - System or measurement model is highly nonlinear
  - Uncertainties are large
- Posterior density function (PDF) approximated by
  - Randomly chosen particles
  - Along with their likelihood weights
- Due to have many particles, computational cost is high
  - GPUs offer promising solutions

Particle Filter



Particle Filter



Sampling Importance Resampling (SIR) Particle Filter

Algorithm 2.3 SIR Particle Filter ([31])

procedure $[\{x_k^i\}_{i=1}^N] = \text{SIR}(\{x_{k-1}^i\}_{i=1}^N, z_k)$

1: for $i = 1 : N$

$x_k^i \sim p(x_k|x_{k-1}^i)$: particle generation

$\tilde{w}_k^i = p(z_k|x_k^i)$: weight calculation

end for

2: $sum\tilde{w} = SUM[\{\tilde{w}_k^i\}_{i=1}^N]$

3: for $i = 1 : N$

$w_k^i = sum\tilde{w}^{-1} * \tilde{w}_k^i$

end for

4: $[\{x_k^i\}_{i=1}^N] = \text{RESAMPLE}\ [\{x_k^i, w_k^i\}_{i=1}^N]$

N = number of particles

i = index of the particle

k = index of the time step

x_k^i= state of the particle

$\tilde{w}_k^i$ = weight of the particle

w_k^i = normalized weight of the particle

z_k = measurement

x_k = true state of the track

$p(x_k|x_{k-1}^i)$ = transitional prior

$p(z_k|x_k^i)$ = likelihood function

Why Resample?

- As time progresses, the normalized weights of one particle becomes near to one, the others become nearly zero. Degeneracy problem!

- Effective solution is Resampling:
  - The particles with higher weights are replicated
  - Those with lower weights are eliminated

Resampling

$$ER(i) = \frac{N * \tilde{w}^i}{sum\tilde{w}}$$

(expected number of replication of ith particle after resampling)

- Ensures unbiased results
- Not possible:
  - Random number generators
  - Numerical issues

Sampling Importance Resampling (SIR) Particle Filter

procedure $[\{x_k^i\}_{i=1}^N] = \text{SIR}\ (\{x_{k-1}^i\}_{i=1}^N, z_k)$

1: foreach $i = 1 : N$

$x_k^i \sim p(x_k|x_{k-1}^i)$:particle generation

$\tilde{w}_k^i = p(z_k|x_k^i)$:weight calculation

→ Fully parallel. A thread for each particle

end foreach

2: $sum\tilde{w} = SUM[\{\tilde{w}_k^i\}_{i=1}^N]$ → Efficient parallel reduction

3: foreach $i = 1 : N$

$w_k^i = sum\tilde{w}^{-1} * \tilde{w}_k^i$ → Fully parallel. A thread for each weight

end foreach

$\hat{x_k} = SUM[\{x_k^i * w_k^i\}_{i=1}^N]$ → Efficient parallel reduction

4: $[\{x_k^i\}_{i=1}^N] = \text{RESAMPLE}\ [\{x_k^i, \tilde{w}_k^i\}_{i=1}^N]$

Systematic Resampling

Algorithm 3.1 Systematic Resampling [22]

1: **procedure** $[\{O^i\}_{i=1}^N] = \text{SYSTEMATIC}(\{\tilde{w}^i\}_{i=1}^N)$

2: $u \sim v[0\ 1)$

3: $C = \text{INCLUSIVE-PREFIX-SUM}(\tilde{w})$

4: foreach $i = 1 : N$

5: $\quad r^i = \frac{N * C^i}{C^N}$

6: $\quad O^i = min(N, \lfloor r^i + u \rfloor)$

7: end foreach

Cumulative summation of the weights

Less-readily parallelization

Each thread run in parallel

Cumulative number of replication

Numerical Instability problem

Lawrence M. Murray, Anthony Lee, and Pierre E. Jacob. Journal of Computational and Graphical Statistics, 2016.

Metropolis Resampling

Algorithm 3.3 Metropolis Resampling [22]

1: **procedure** $[\{xnew^i\}_{i=1}^N] = \text{METROPOLIS}(\{x^i, \tilde{w}^i\}_{i=1}^N, B)$
2: foreach $i = 1 : N$
3: $t = i$
4: for $m = 1 : B$
5: $u \sim v[0\ 1]$
6: $j \sim v\{1, .., N\}$
7: if $u \leq \tilde{w}^j / \tilde{w}^t$ then $t = j$ end if
8: end for
9: $xnew^i = x^t$
10: end foreach

Each thread run in parallel

Trade off between speed and bias

Non-coalesced global memory access problem

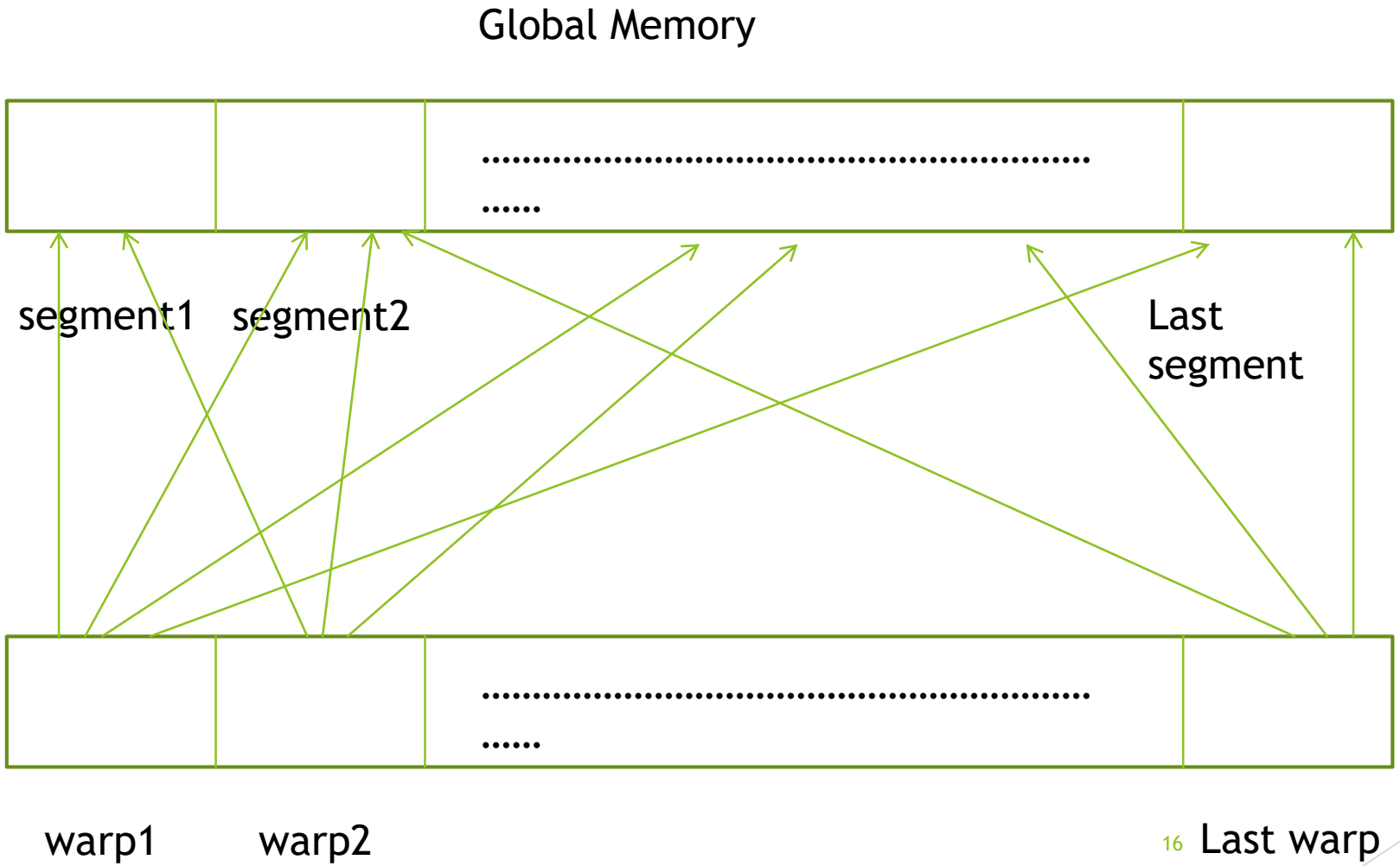
Only ratio of two weights

Lawrence M. Murray, Anthony Lee, and Pierre E. Jacob. Journal of Computational and Graphical Statistics, 2016.

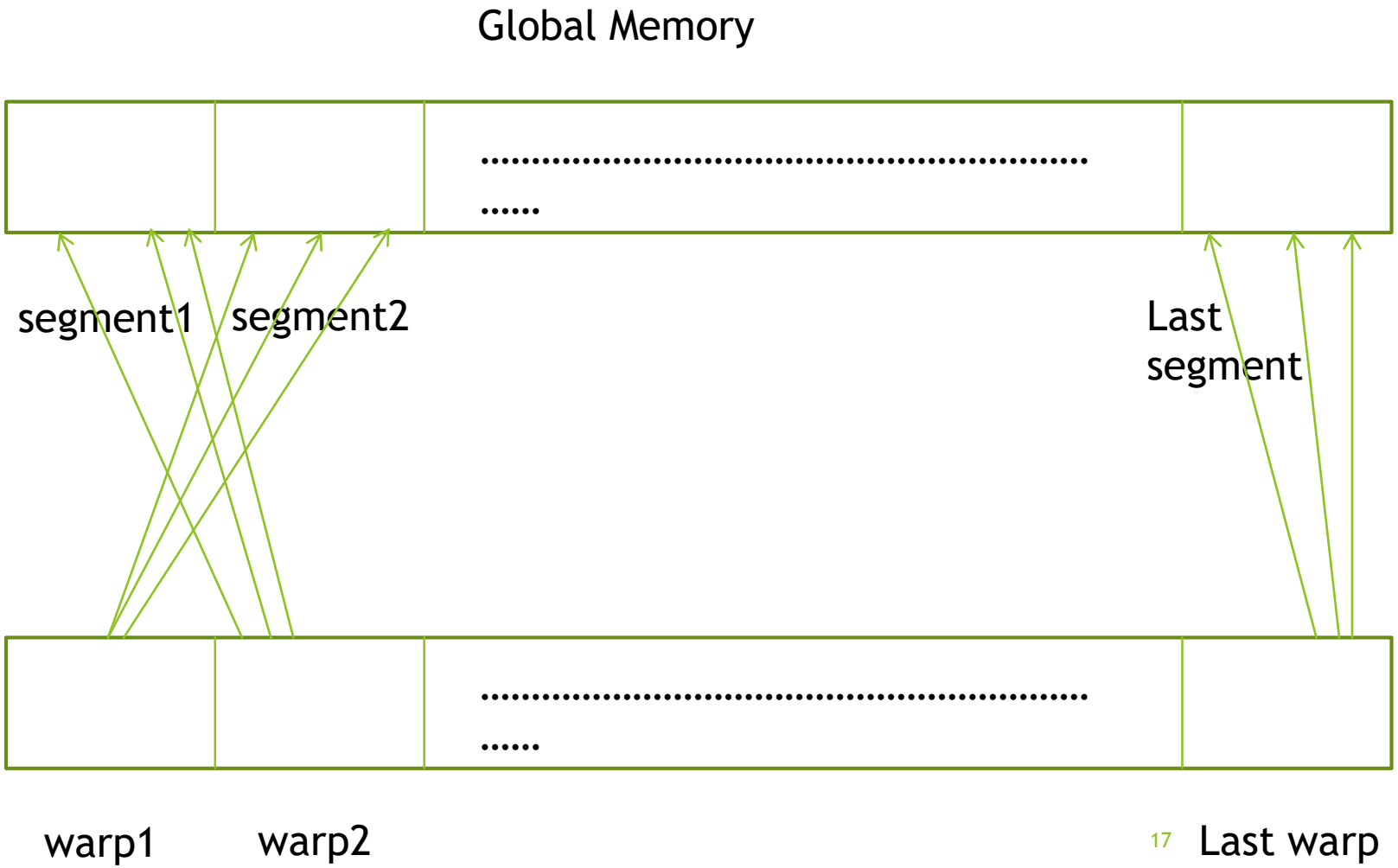
Global Memory

- Operations performed segment by segment
- Warp:
  - Is an execution unit consists of 32 threads
  - They run the same instruction concurrently
  - They are physically related to each other
  - Reading data from different segments causes serialization

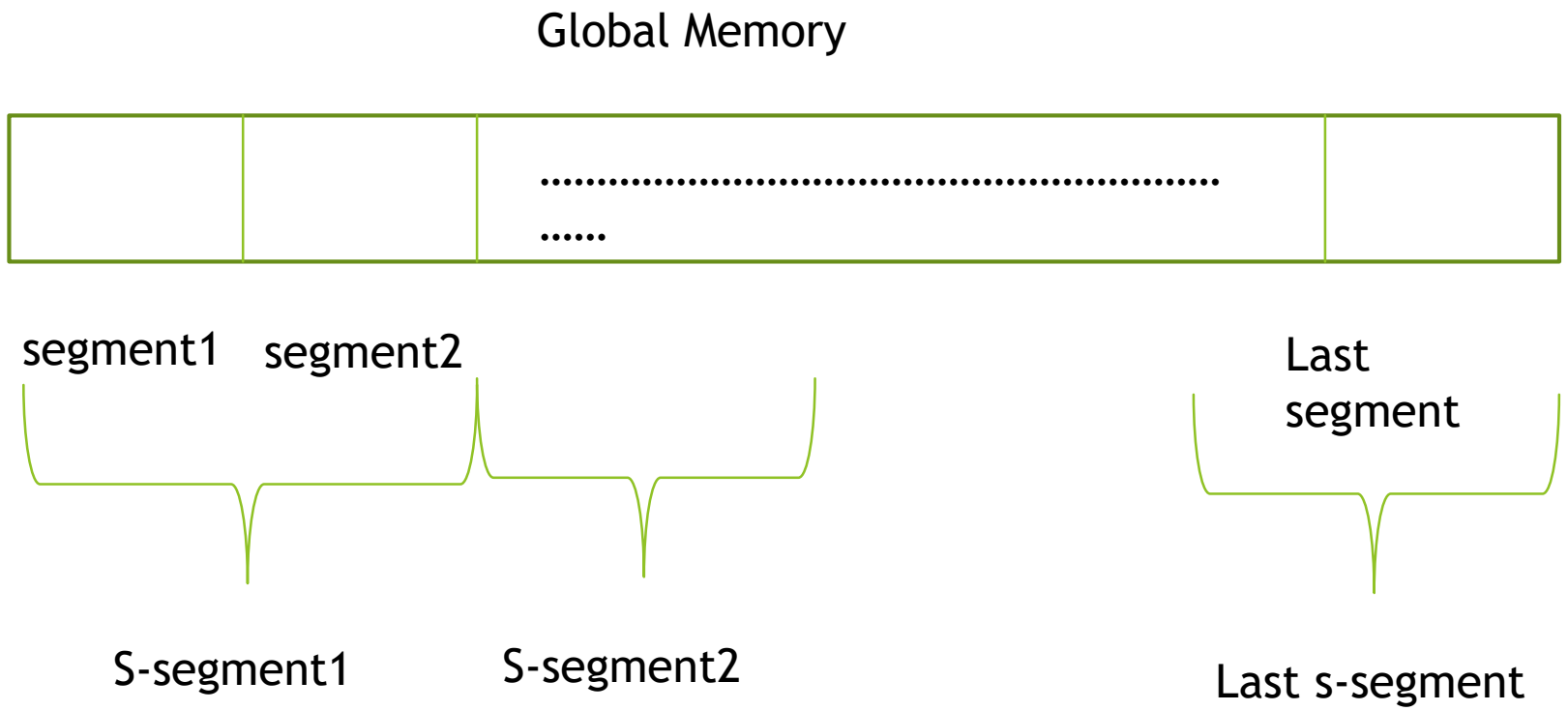
Metropolis Access Pattern



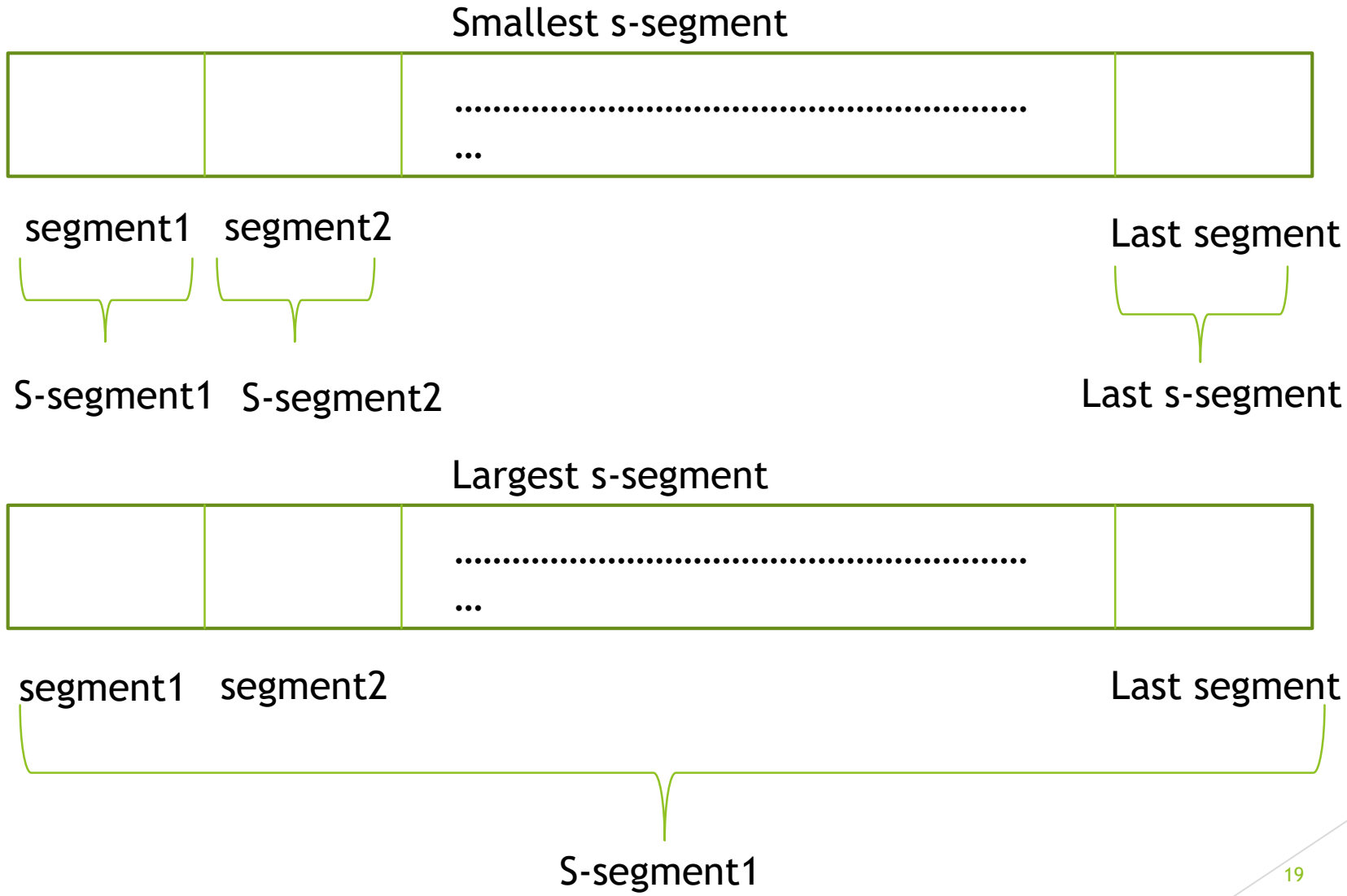
Our Approach



S-segment


Global Memory
segment1
segment2
Last segment
S-segment1
S-segment2
Last s-segment

S-segment


Smallest s-segment
segment1
segment2
Last segment
S-segment1
S-segment2
Last s-segment
Largest s-segment
segment1
segment2
Last segment
S-segment1

Metropolis-C1

Algorithm 5.1 Metropolis-C1 Resampling

1: **procedure** $[\{xnew^i\}_{i=1}^N]$ = METROPOLIS-C1$(\{x^i, \tilde{w}^i\}_{i=1}^N, B, SC, DC)$
2: foreach $i = 1 : N$
3: $t = i$
4: $s \sim v\{1, .., SC\}$
5: for $m = 1 : B$
6: $u \sim v[0\ 1]$
7: $j \sim v\{(s-1) * DC + 1, \ldots, s * DC\}$
8: if $u \leq \tilde{w}^j / \tilde{w}^t$ then $t = j$ end if
9: end for
10: $xnew^i = x^t$
11: end foreach

Selects an s-segment only once

Selects within this s-segment

Metropolis-C2

Algorithm 5.2 Metropolis-C2 Resampling

1: **procedure** $[\{xnew^i\}_{i=1}^N]$ = METROPOLIS-C2($\{x^i, \tilde{w}^i\}_{i=1}^N, B, SC, DC$)
2: foreach $i = 1 : N$
3: $t = i$
4: for $m = 1 : B$
5: $u \sim v[0\ 1]$
6: $s \sim v\{1, .., SC\}$
7: $j \sim v\{(s-1) * DC + 1, \ldots, s * DC\}$
8: if $u \leq \tilde{w}^j / \tilde{w}^t$ then $t = j$ end if
9: end for
10: $xnew^i = x^t$
11: end foreach

Selects an s-segment at each iteration

Selects within this s-segment

Experimental Environment

$$\tilde{w}^i = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(\tilde{x}^i - y)^2\right)$$

- We take y=4, and we create16 different weight sets and run them 256 times and get their average to obtain the output
- CUDA profiler:
  - gld_transactions_per_request metric
  - Average number of global memory load transactions performed for each global memory load
    - Count the number of transactions in each load operations of all warps. The number of transaction is between 1 and 32.
    - Get the average of total number of transactions

Metropolis-C1

Algorithm 5.1 Metropolis-C1 Resampling

1: **procedure** $[\{xnew^i\}_{i=1}^N]$ = METROPOLIS-C1($\{x^i, \tilde{w}^i\}_{i=1}^N, B, SC, DC$)
2: foreach $i = 1 : N$
3: $t = i$
4: $s \sim v\{1, .., SC\}$
5: for $m = 1 : B$
6: $u \sim v[0\ 1]$
7: $j \sim v\{(s-1) * DC + 1, \ldots, s * DC\}$
8: if $u \leq \tilde{w}^j / \tilde{w}^t$ then $t = j$ end if
9: end for
10: $xnew^i = x^t$
11: end foreach

Selects an s-segment only once

Selects within this s-segment

- S-segment size (SS) is 128 KB and 2048 KB for C1 and C2
  - In 128 KB: $\tilde{w}^j$ is loaded in one transaction for C1 and C2.
  - In 2048 KB: $\tilde{w}^j$ is loaded at most 16 transactions for C1 and C2.
  - In 128 KB: $\tilde{w}^t$ is loaded at most 2 transactions for C1 and at most 32 transactions for C2.
  - In 2048 KB: $\tilde{w}^t$ is loaded at most 17 transactions for C1 and at most 32 transactions for C2.

Metropolis-C2

Algorithm 5.2 Metropolis-C2 Resampling

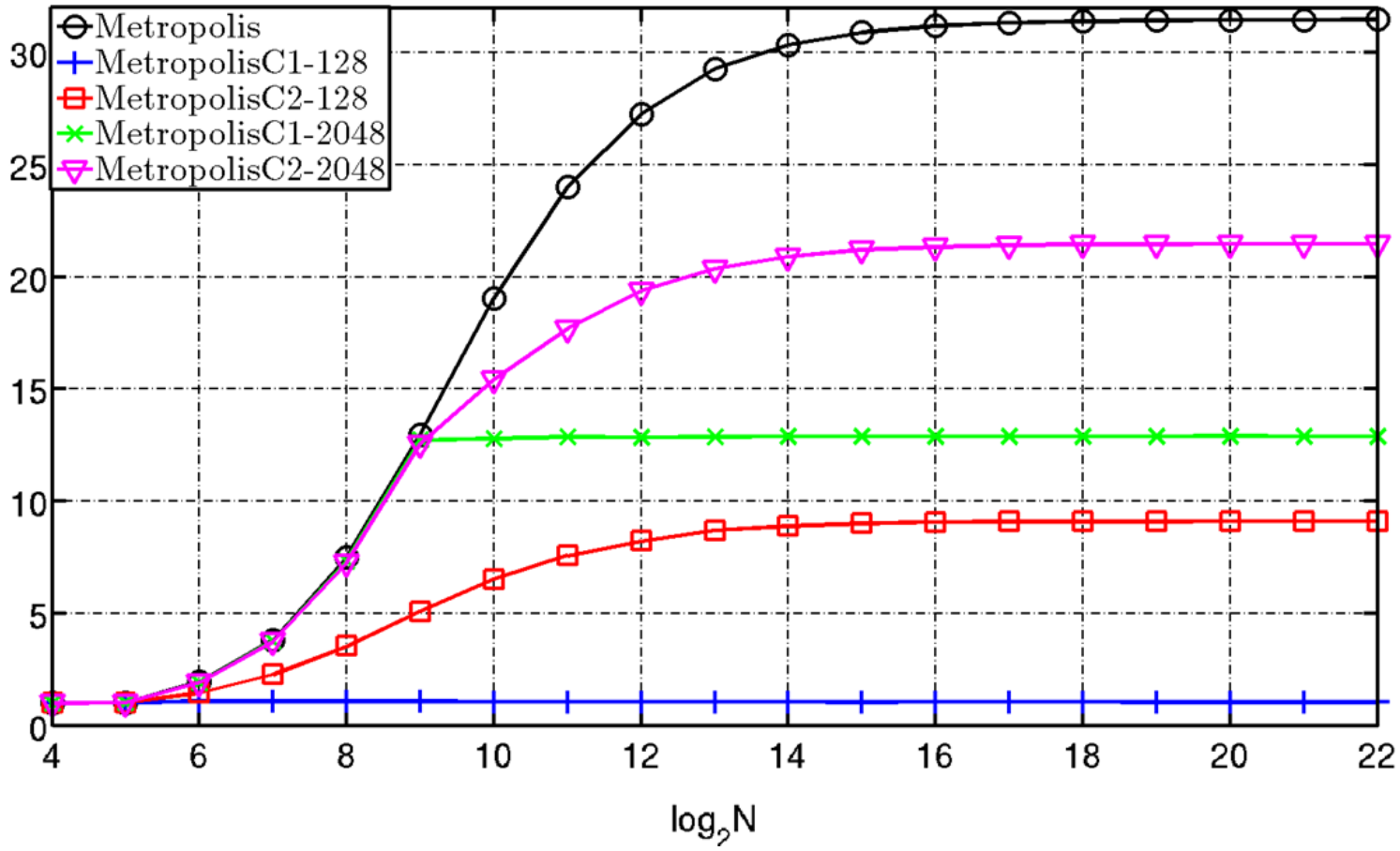
1: **procedure** $[\{xnew^i\}_{i=1}^N]$ = METROPOLIS-C2($\{x^i, \tilde{w}^i\}_{i=1}^N, B, SC, DC$)
2: foreach $i = 1 : N$
3: $t = i$
4: for $m = 1 : B$
5: $u \sim v[0\ 1]$
6: $s \sim v\{1, .., SC\}$
7: $j \sim v\{(s-1) * DC + 1, \ldots, s * DC\}$
8: if $u \leq \tilde{w}^j / \tilde{w}^t$ then $t = j$ end if
9: end for
10: $xnew^i = x^t$
11: end foreach

Selects an s-segment at each iteration

Selects within this s-segment

- S-segment size (SS) is 128 KB and 2048 KB for C1 and C2
  - In 128 KB: $\tilde{w}^j$ is loaded in one transaction for C1 and C2.
  - In 2048 KB: $\tilde{w}^j$ is loaded at most 16 transactions for C1 and C2.
  - In 128 KB: $\tilde{w}^t$ is loaded at most 2 transactions for C1 and at most 32 transactions for C2.
  - In 2048 KB: $\tilde{w}^t$ is loaded at most 17 transactions for C1 and at most 32 transactions for C2.

Experimental Results



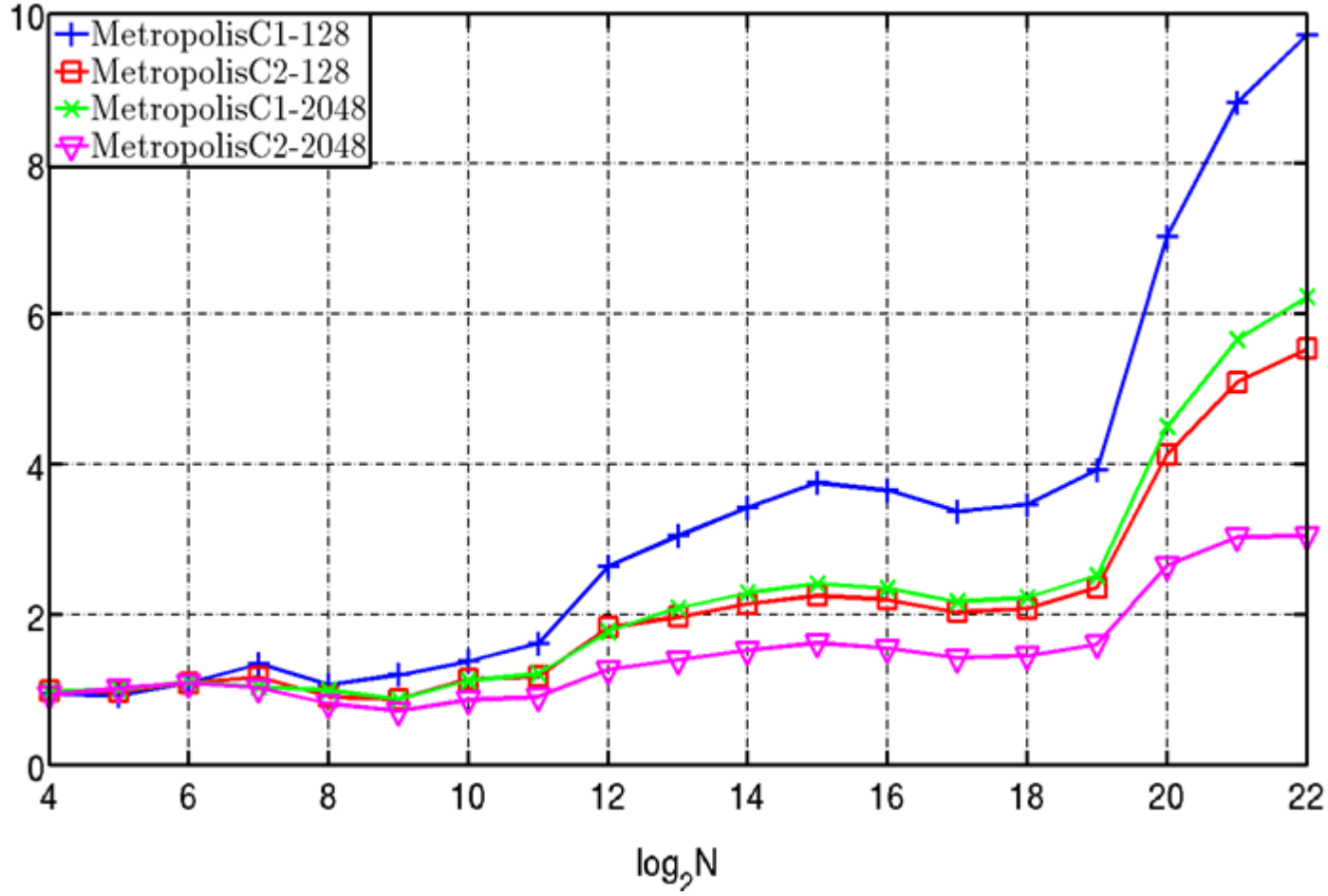
Techn.	$\tilde{w}^j$	$\tilde{w}^t$
C1-128	1	2
C2-128	1	32
C1-2048	16	17
C2-2048	16	32

- X-axis is the number of particles and Y-axis is the result of gld_transactions_per_request metric

Experimental Results

- Non-coalesced access patterns have direct effects on the execution time, but not only the one
- Hardware resources and limitations:
  - Size of L1 and L2 caches
  - Number of cores
  - Maximum number of threads that can be active at a time
- Number of particles

Experimental Results



- X-axis is the number of particles and Y-axis is the speedup

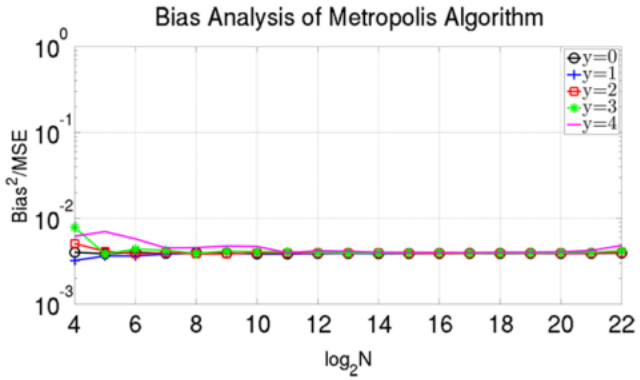
L1 Cache Usage

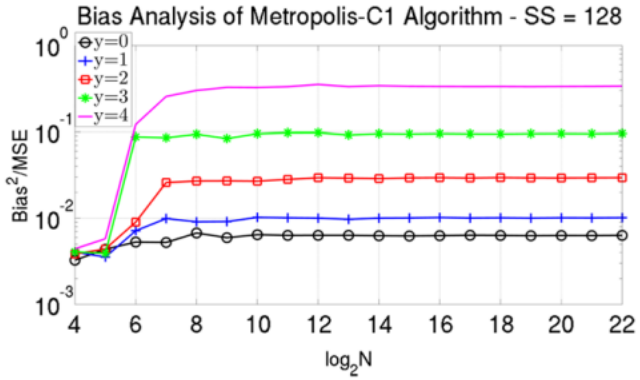
Table 4 L1 cache global loads hit ratio (percent) of the resampling algorithms.

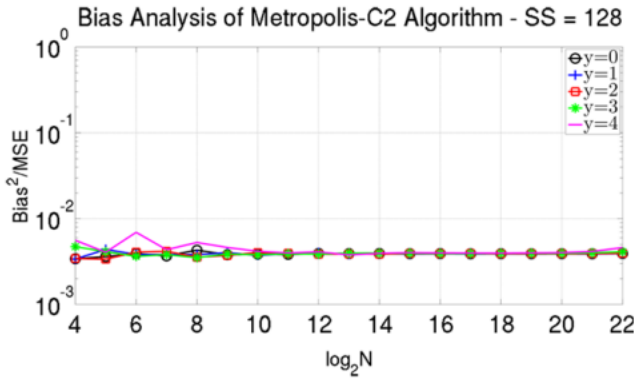
Resampling methods	Cache size	Number of particles (N)								
		1024	2048	4096	8192	16,384	32,768	65,536	262,144	1,048,576
M	16 KB	99.72	99.80	99.85	51.01	26.29	13.61	7.07	1.80	0.45
M	32 KB	99.73	99.82	99.84	99.87	51.79	26.41	13.49	3.44	0.91
M	48 KB	99.73	99.81	99.85	99.87	77.52	39.65	20.20	5.15	1.36
C1-SS = 128	16 KB	95.64	96.36	96.61	96.62	96.75	96.82	96.80	96.81	96.80
C1-SS = 128	32 KB	94.98	96.13	96.44	96.55	96.77	96.82	96.82	96.85	96.84
C1-SS = 128	48 KB	95.54	96.32	96.63	96.60	96.75	96.79	96.84	96.86	96.85
C1-SS = 2048	16 KB	99.51	99.56	99.59	74.79	54.75	43.99	38.57	34.51	33.49
C1-SS = 2048	32 KB	99.48	99.57	99.61	99.63	92.66	85.11	80.25	76.18	75.09
C1-SS = 2048	48 KB	99.51	99.55	99.62	99.63	99.47	98.28	96.94	95.49	95.05
C2-SS = 128	16 KB	98.92	99.11	99.27	65.33	36.82	20.80	12.38	5.96	4.30
C2-SS = 128	32 KB	98.89	99.19	99.30	99.38	81.46	62.07	49.21	38.33	35.54
C2-SS = 128	48 KB	98.88	99.13	99.32	99.37	97.15	88.58	80.01	70.98	68.55
C2-SS = 2048	16 KB	99.54	99.64	99.71	53.34	27.42	13.85	7.04	1.83	0.51
C2-SS = 2048	32 KB	99.53	99.65	99.69	99.72	57.79	30.60	16.12	4.99	2.41
C2-SS = 2048	48 KB	99.49	99.64	99.71	99.74	85.90	51.70	30.05	11.33	7.20

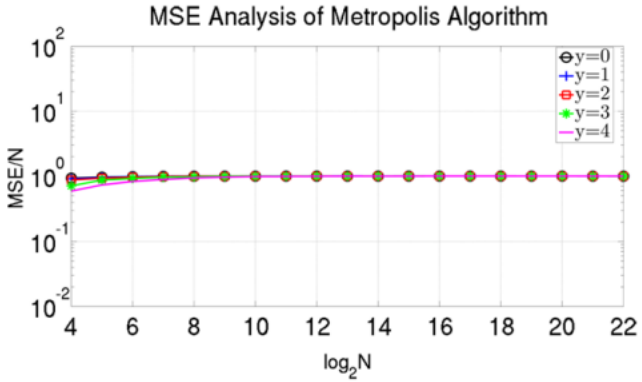
The values in the cells are the ratios of the cache hit in L1 cache for the global loads inside the CUDA kernel of the resampling algorithms. There are also two pieces of information given in the cells of the table. The numbers written normally mean that the algorithm runs faster compared to its no-cache setting. The underlined numbers mean that the algorithm runs slower compared to its no-cache setting

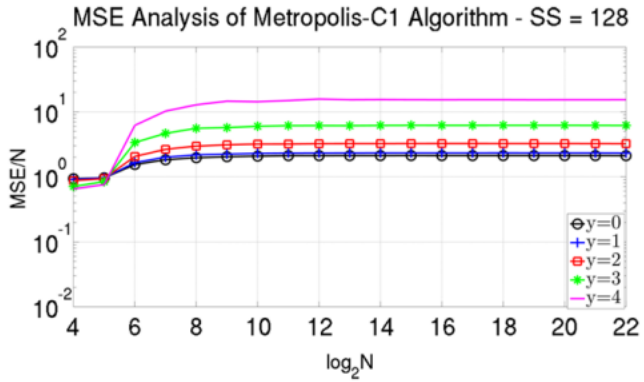
Bias and MSE Results

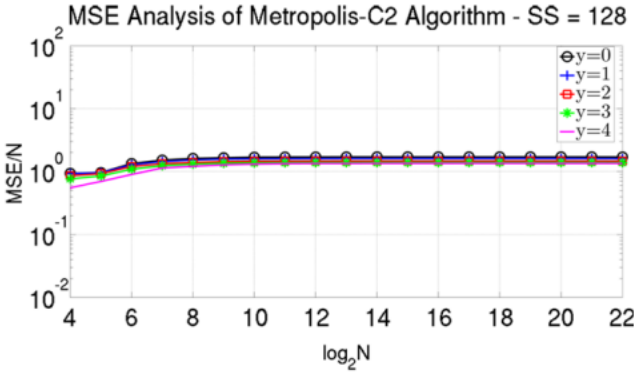




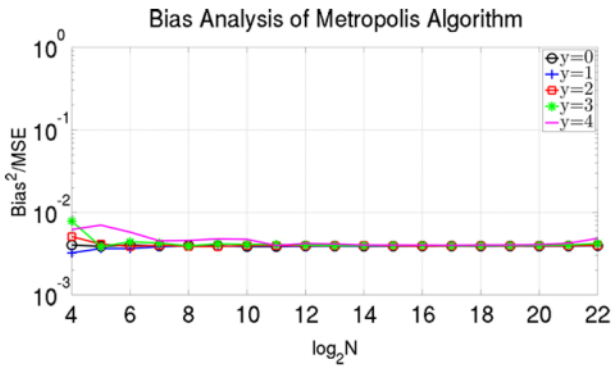


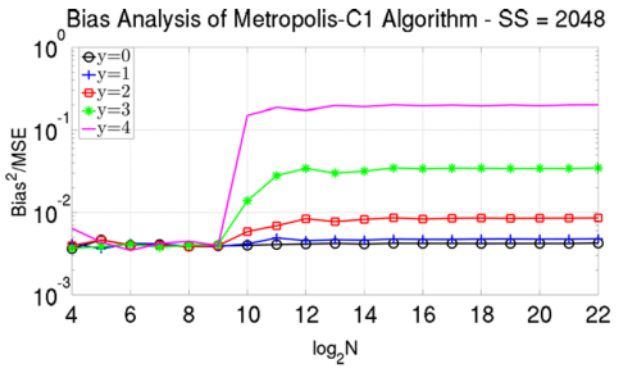


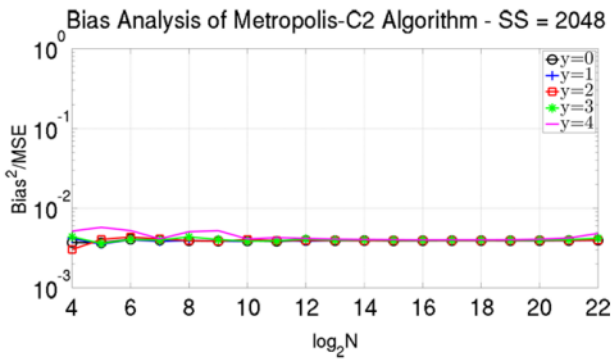


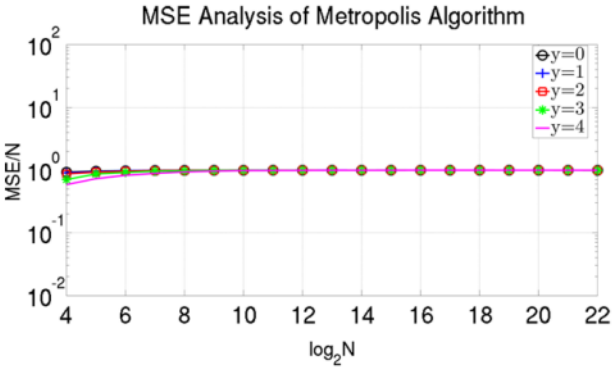


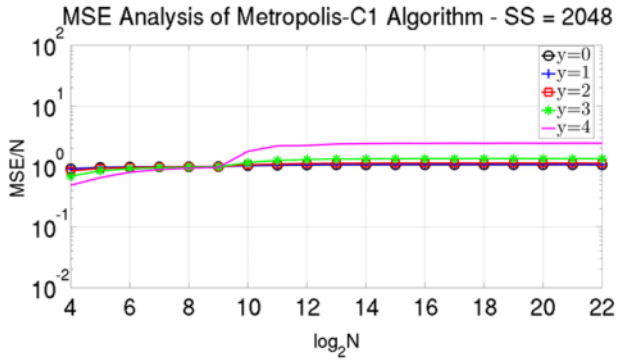
Bias and MSE Results

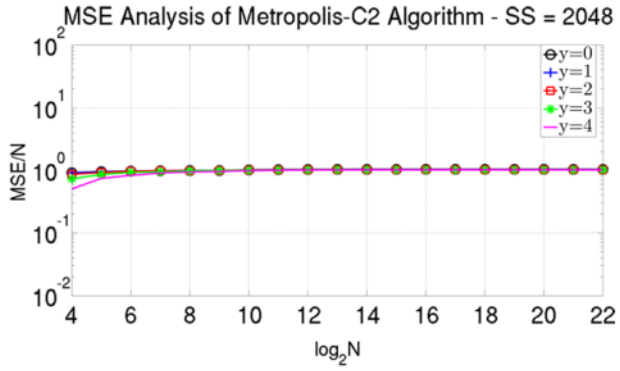






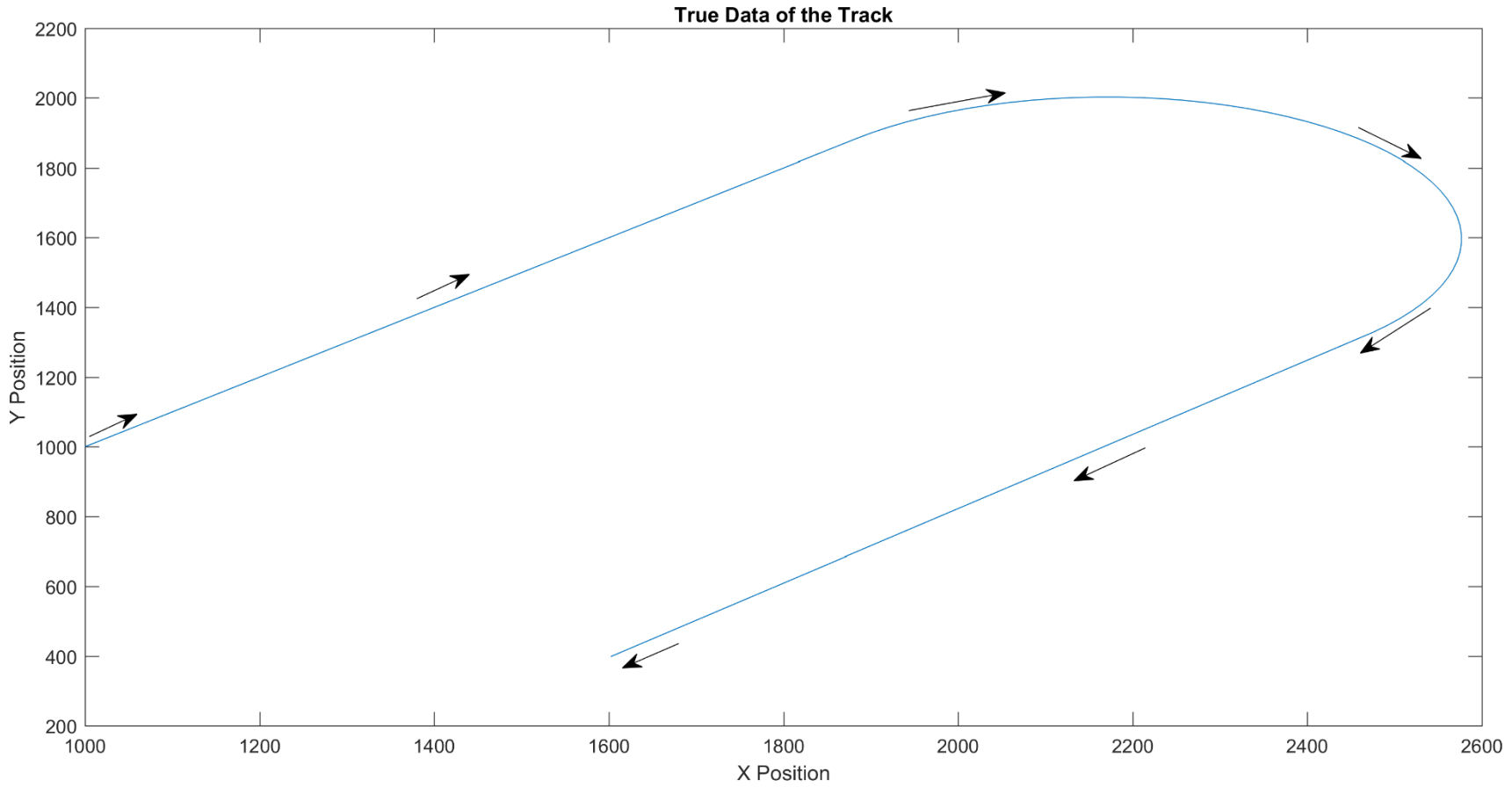






- As s-segment size increases, both C1 and C2 approach to Metropolis. When SS is equal to $4xN$, both C1 and C2 become same with Metropolis
- Hence, with s-segment size we provide a trade-off between speed and quality

Target Tracking with Radar



State of the target:

- x position
- y position
- velocity in x position
- velocity in y position

Target Tracking with Radar

Nearly Constant Velocity Model

State at time k

Gaussian Process Noise

$$x_k = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * x_{k-1} + \begin{bmatrix} \frac{T^2}{2} & 0 \\ 0 & \frac{T^2}{2} \\ T & 0 \\ 0 & T \end{bmatrix} * q_k$$

Measurement at time k

Range

Gaussian Measurement Noise

$$y_k = \begin{bmatrix} \sqrt{p_x^2 + p_y^2} \\ \arctan(\frac{p_y}{p_x}) \end{bmatrix} + v_k$$

Bearing

Execution Time Ratios of Each Stage

$log_2 N = 14$	Metropolis	C1-SS=128	C1-SS=2048	C2-SS=128	C2-SS=2048
Stage1	9.53	15.05	12.80	13.37	11.15
Stage2	4.45	6.80	5.76	6.01	5.01
Stage3	14.98	22.99	19.45	20.33	16.94
Stage4	71.04	55.16	61.98	60.29	66.90
Total Exec. Time	0.062	0.041	0.049	0.047	0.056
$log_2 N = 16$	Metropolis	C1-SS=128	C1-SS=2048	C2-SS=128	C2-SS=2048
Stage1	9.37	19.16	14.12	14.79	10.90
Stage2	1.80	3.69	2.72	2.84	2.10
Stage3	6.50	13.29	9.78	10.25	7.56
Stage4	82.33	63.86	73.38	72.12	79.45
Total Exec. Time	0.179	0.088	0.119	0.114	0.154
$log_2 N = 18$	Metropolis	C1-SS=128	C1-SS=2048	C2-SS=128	C2-SS=2048
Stage1	7.53	20.98	13.33	13.73	9.37
Stage2	0.62	1.72	1.09	1.12	0.76
Stage3	2.69	7.50	4.76	4.90	3.35
Stage4	89.17	69.80	80.82	80.24	86.52
Total Exec. Time	0.741	0.266	0.419	0.406	0.595
$log_2 N = 20$	Metropolis	C1-SS=128	C1-SS=2048	C2-SS=128	C2-SS=2048
Stage1	4.63	21.62	12.18	12.39	7.72
Stage2	0.21	1.00	0.56	0.57	0.36
Stage3	1.14	5.31	2.98	3.04	1.89
Stage4	94.02	72.08	84.27	84.01	90.03
Total Exec. Time	4.545	0.973	1.728	1.699	2.725
$log_2 N = 22$	Metropolis	C1-SS=128	C1-SS=2048	C2-SS=128	C2-SS=2048
Stage1	3.24	21.66	11.87	11.55	6.79
Stage2	0.12	0.79	0.44	0.42	0.25
Stage3	0.70	4.67	2.56	2.49	1.46
Stage4	95.94	72.87	85.14	85.54	91.50
Total Exec. Time	25.403	3.799	6.935	7.124	12.120

- The values in the table are the ratios of execution times of each stage to the total execution time

RMSE Results

Number of particles	16384	65536	262144	1048576	4194304
Metropolis	7.62567	7.60410	7.60047	7.59979	7.59980
C1-SS=128	7.76200	7.74972	7.74671	7.74626	7.74616
C1-SS=2048	7.62207	7.60041	7.59731	7.59536	7.59557
C2-SS=128	7.62645	7.60640	7.60141	7.60017	7.59955
C2-SS=2048	7.62468	7.60559	7.60067	7.59972	7.59956

RMSE = Root Mean Squared Error

Error = $\sqrt{(true_x - estimated_x)^2 + (true_y - estimated_y)^2)}$

References

- Ristic, B., Arulampalam, S. and Gordon, N., Beyond the Kalman Filter: Particle Filters for Tracking Applications. Artech House, 2004.
- Murray, L. M., Lee, A. and Jacob, P. E., "Parallel resampling in the particle filter." Journal of Computational and Graphical Statistics, 25(3):789–805, 2016.
- Gong, P., Basciftci, Y. O. and Ozguner, F., "A parallel resampling algorithm for particle filtering on shared-memory architectures." In Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), 2012 IEEE 26th International, pages 1477–1483. IEEE, 2012.
- Ortega, J. M., Numerical Analysis: A Second Course. Academic Press, New York, 1972.
- Dülger, Ö., Oğuztüzün, H. and Demirekler, M., "Memory coalescing implementation of Metropolis resampling on graphics processing unit.“ Journal of Signal Processing Systems, 90(3):433–447, 2018.
- Ö. Dülger and H. Oguztüzün, "Non-coalesced access patterns of global memory load transactions in metropolis resampling implemented on graphics processing unit," 2018 26th Signal Processing and Communications Applications Conference (SIU), 2018