Imperial College London

Parallel Resampling for Particle Filters on FPGAs

Shuanglong Liu, Grigorios Mingas, Christos-Savvas Bouganis
s.liu13@imperial.ac.uk

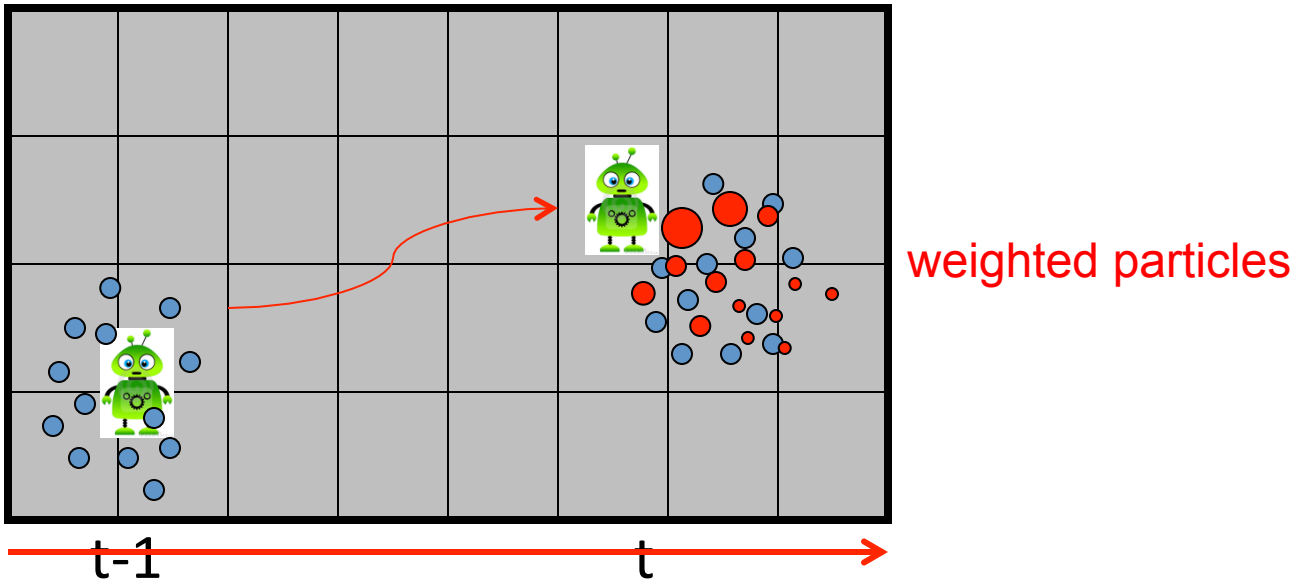
FPT 2014, Shanghai
12 Dec 2014

Motivation: Particle Filters

- To estimate a sequence of unknown variables given the observation variables.
- An approximation by a set of samples/particles.
- Powerful in estimation of non-Gaussian, nonlinear processes.

- Computationally intensive

Illustrative Example: Robot Localization



Given:

Motion model: $p(x_t \mid x_{t-1})$ x_t : position of the robot at time step t

Sensory model: $p(y_t \mid x_t)$ y_t : observed distance from the sensor at t

Visualization of Particle Filter

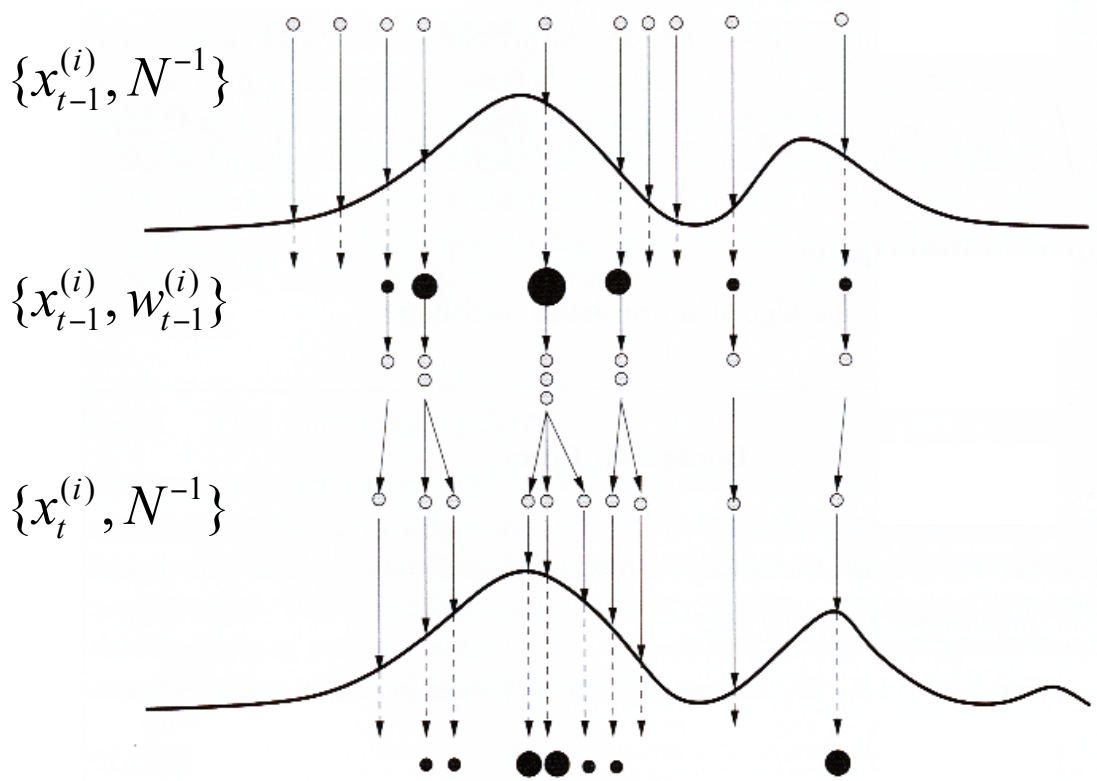
Sample unweighted particles

Compute importance weights

Resample

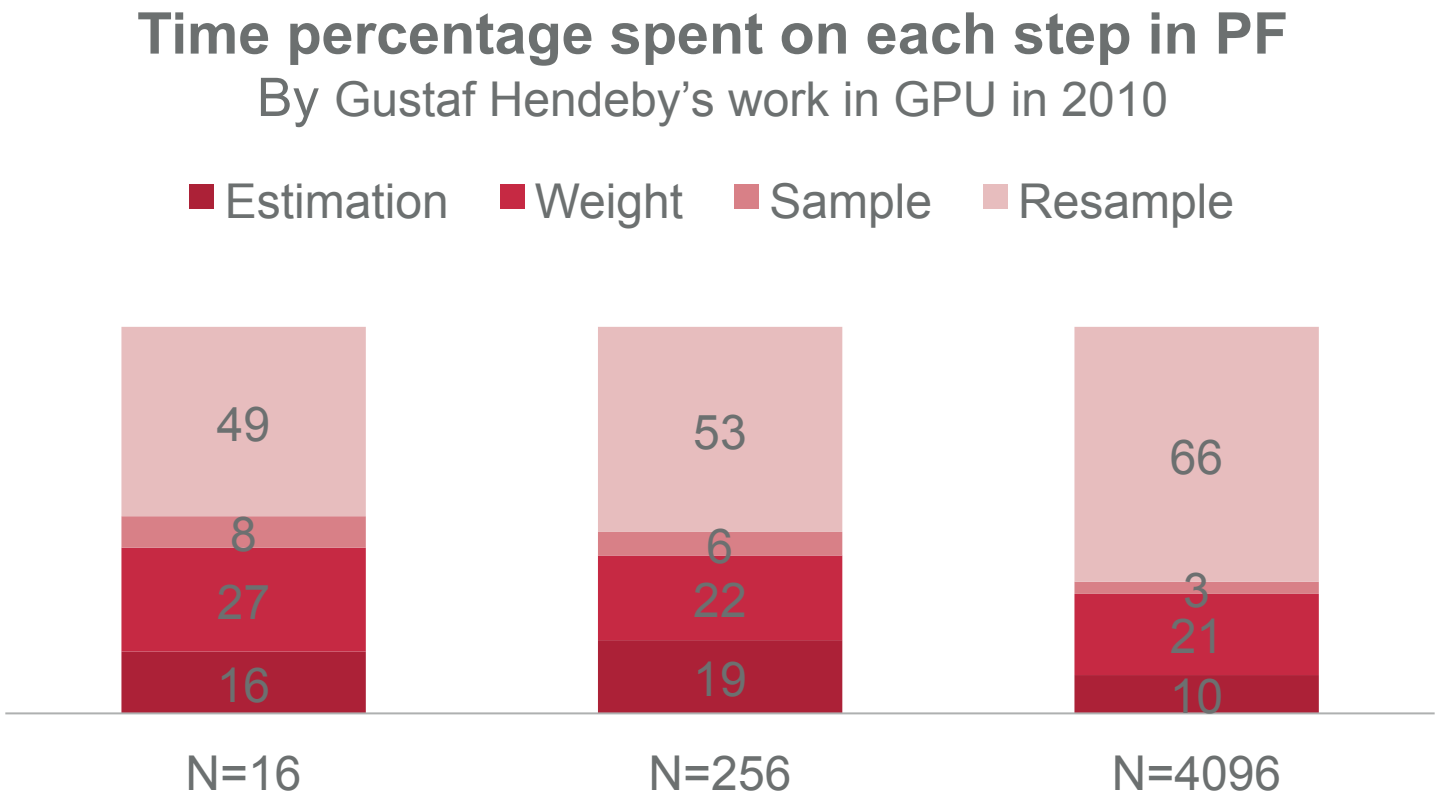
Predict $p(x_t|y_{1:t})$

i = 1 ,…, N=10 particles



From "*Sequential Monte Carlo methods in practice*", Springer, 2001

Motivation: Resampling



- Localization accuracy largely depends on resampling result;

Contribution

Systematic Resampling (ISR)

Residual Systematic (RSR)

Metropolis Resampling

Rejection Resampling

- parallel architectures for each of them;
- Speed up: 1.7x-49x compared to GPU design;
- Memory access strategy for parallel Metropolis and rejection resampling;

Outline

- Motivation
- Introduction & Implementation
  - Resampling methods
  - Parallel Architectures
- Results
- Conclusions

Resampling

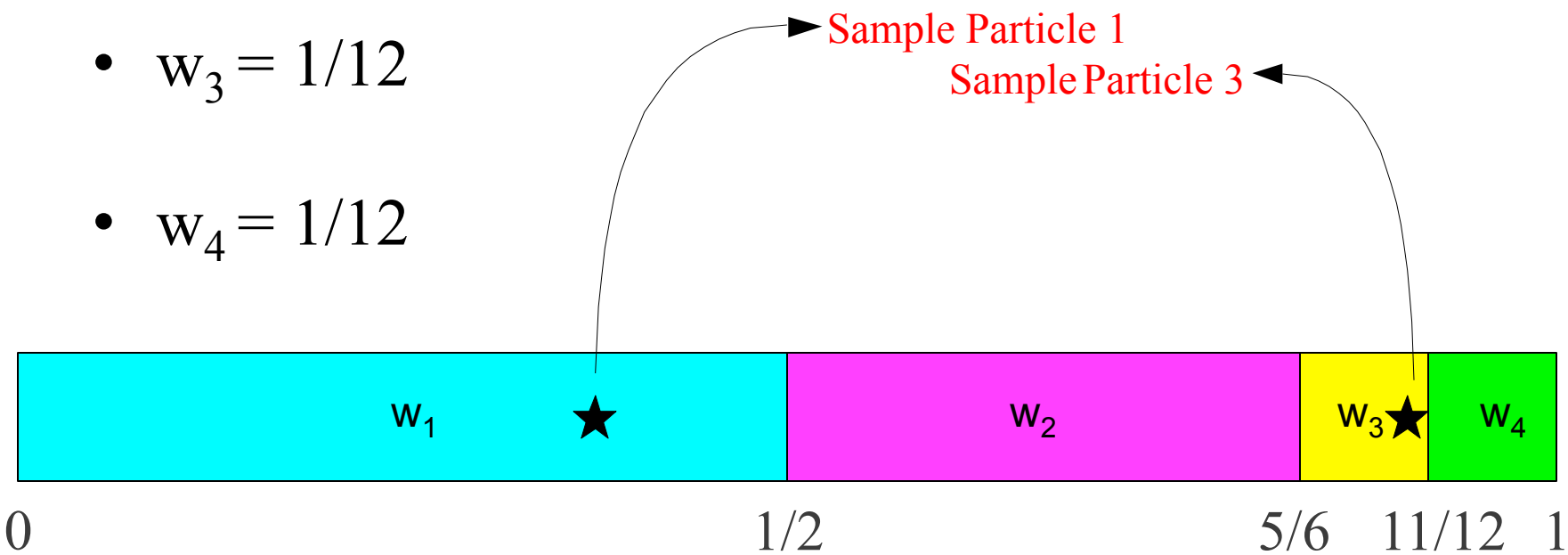
- **input**: A Set of N weighted particles $\{x_i, w_i\}$.
- **output**: N Random samples, where the probability of sampling particle x_i is given by weight w_i.

Particles	Weights	Replication
1	$w_1 = 1/2$	2
2	$w_2 = 1/3$	1
3	$w_3 = 1/12$	1
4	$w_4 = 1/12$	0

- Two classes of resampling methods:
  - Direct Sampling
  - Monte Carlo Sampling

Direct Resampling

- $w_1 = 1/2$
- $w_2 = 1/3$
- $w_3 = 1/12$
- $w_4 = 1/12$



Cumulative Distribute Function (CDF) of the weight w_i

Parallel Systematic Resampling

- Parallel Systematic Resampling method 1: RSR

systematic resampling:

1. $u_1 \sim U[0,1)$, sum=sum(w)
2. **for** j = 1:N

   o_j = **floor**($N*w_j/sum+u_j$)

   $u_j \sim f(u_{j-1}, w_j)$

   end

Parallel systematic resampling:

1. $u_1 \sim U[0,1)$, sum_1=sum($w_{1:N/2}$)
2. $u_{N/2+1} \sim f(sum_1, u_1)$
3. **for** j = 1:N/2

   o_j = **floor**($N*w_j/sum+u_j$)

   $u_j \sim f(u_{j-1}, w_j)$

   end

   for j = N/2+1:N

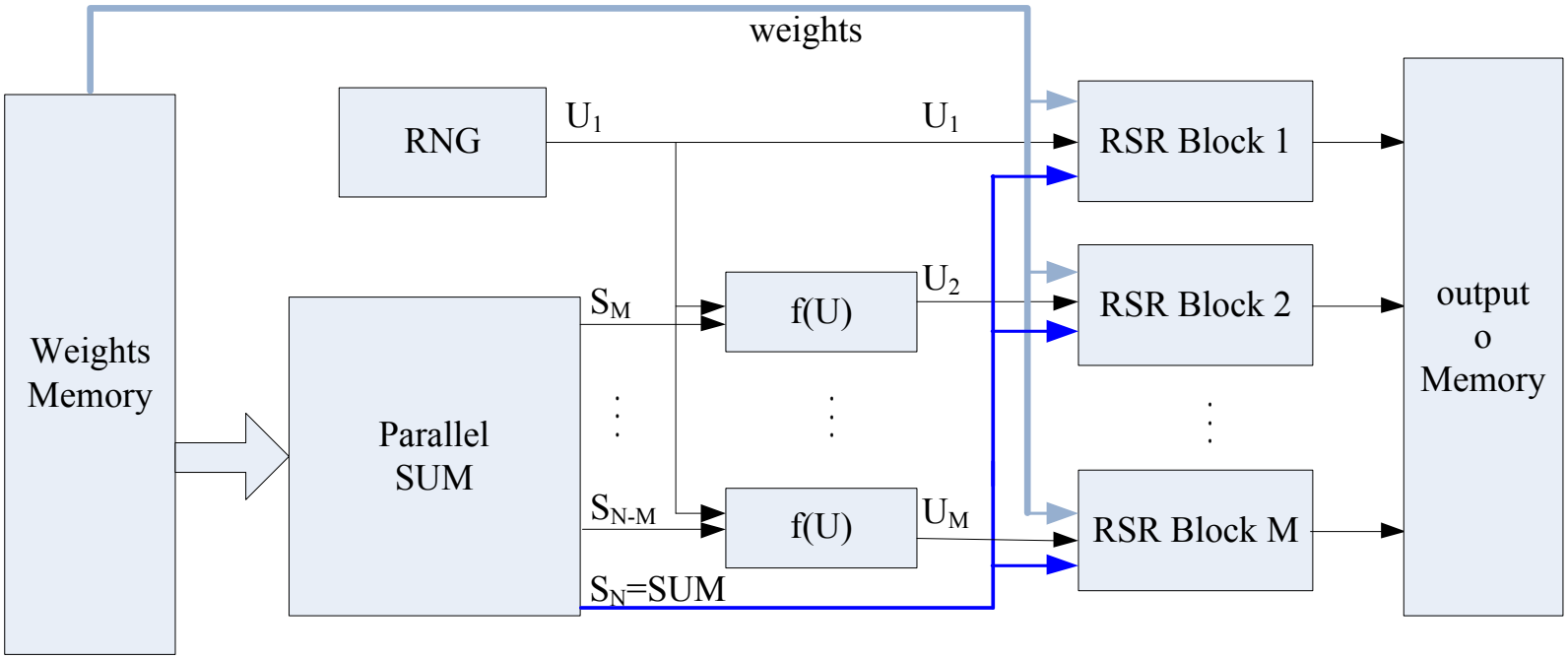
   o_j = **floor**($N*w_j/sum+u_j$)

   $u_j \sim f(u_{j-1}, w_j)$

   end

Parallel Architectures

- Residual Systematic Resampling (RSR)



Parallel Systematic Resampling

- Parallel Systematic Resampling method 2: ISR

systematic resampling:

1. $u_1 \sim U[0,1)$, sum=sum(w)
2. **for** j = 1:N

   o_j = **floor(N*w_j/sum+u_j)**

   $u_j \sim f(u_{j-1}, w_j)$

   end

Parallel systematic resampling:

1. Generate cumulative sum of w:

   c = cumsum(w) ;
2. Generate cumulative sum of o:

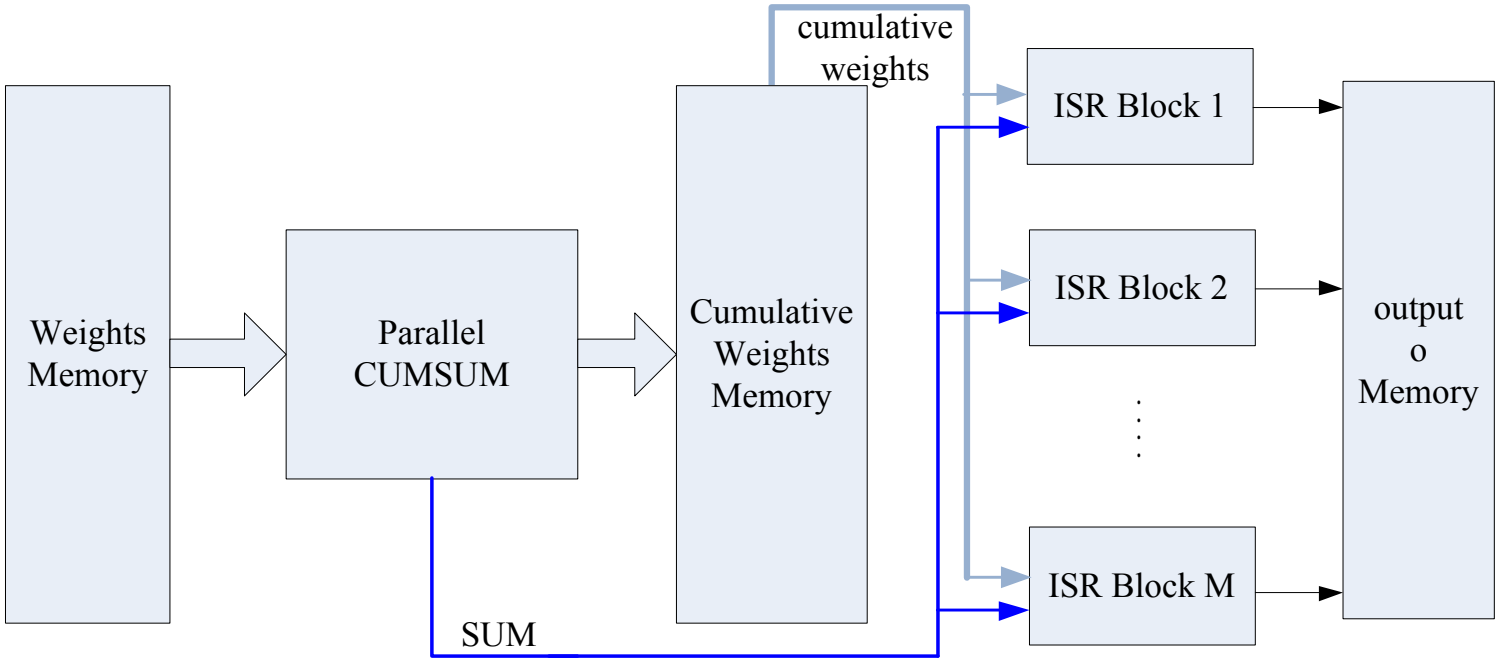
   foreach j = 1:N

   O_j = floor(N*c_j/sum)

   end

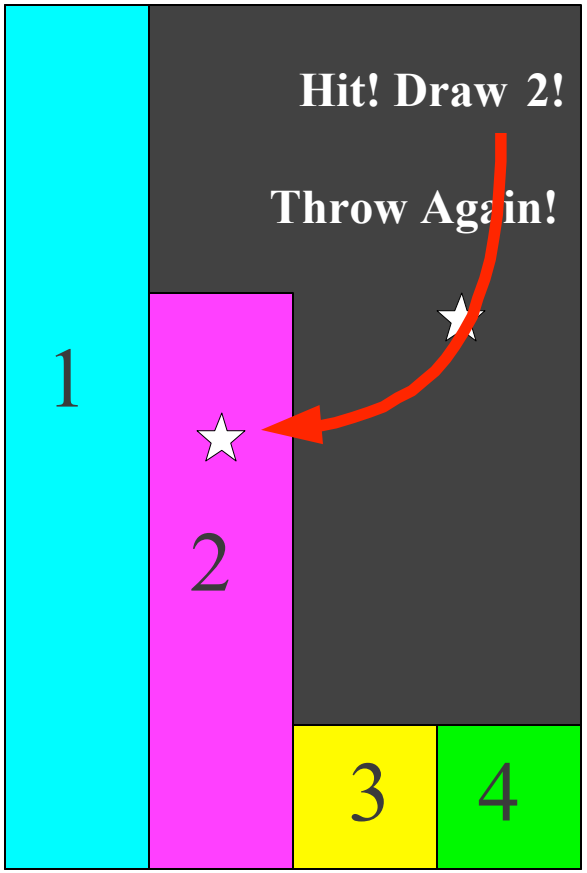
Parallel Architectures

- Improved Systematic Resampling (ISR)



Monte Carlo Resampling

- $w_1 = 1/2$
- $w_2 = 1/3$
- $w_3 = 1/12$
- $w_4 = 1/12$



Throw darts to the dartboard

Hit rate depends on the variance of the weights!

How to Parallelize Resampling

- Parallel Monte Carlo Resampling:
  - One dartboard and M darts;
  - Throw the darts together each time;

- Problem: Each dart needs to go any place of the board;
- Our Solution: A random permutation generator (RPG) is used to reallocate the weights to each parallel resampling block;

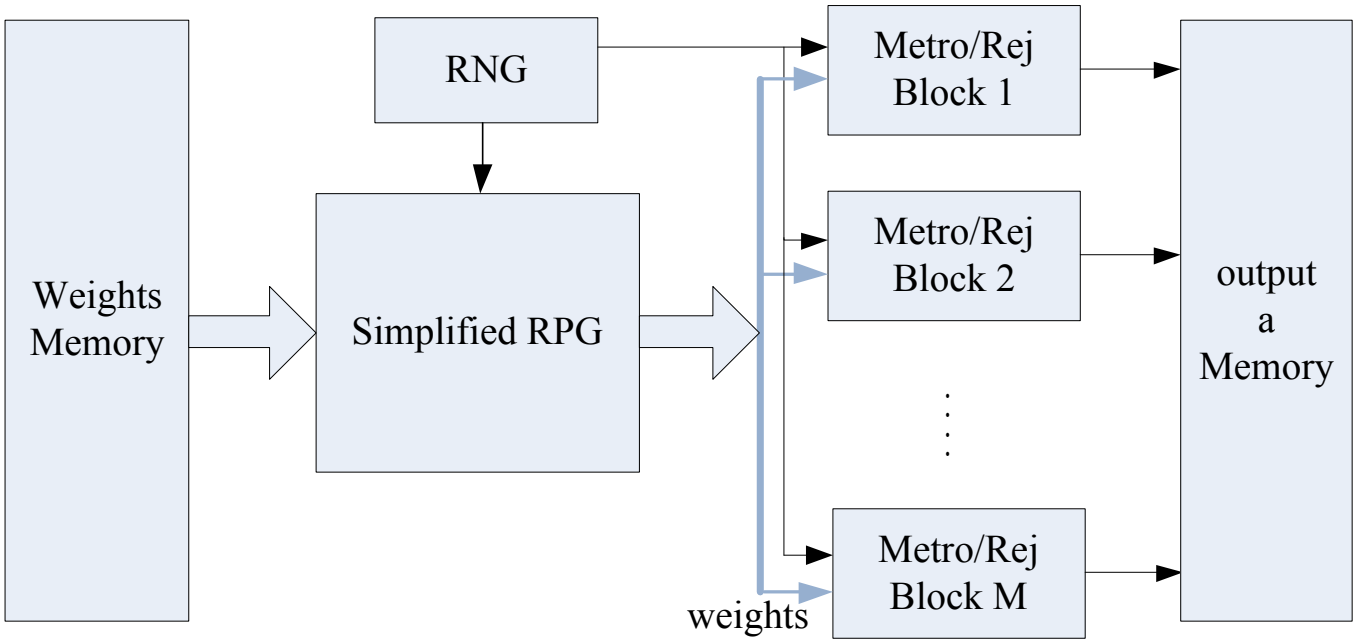
Parallel Architectures

- The optimized RPG for Metropolis & Rejection Resampling

The simplified RPG in FPGAs	An example for M=4
1: Initialization: Q ~ permutation of the numbers 0:M-1 in binary; ($M\log_2 M$ bits in total) 2: RNG: i ~ U{0, 1, … ,$M\log_2 M$-1}; 3: Out = Cyclic shifter (Q, i); 4: Output: Out ~ one random permutation of 0:M-1 as a binary representation.	1: Initialization: Q = {0 1 2 3} ~ 00_01_10_11 2: i ~ U{0, 1, … ,7} = 3; 3: Out = Cyclic shifter (Q, 3); 4: Output: Out = 11_01_10_00 ~ {3 1 2 0}

Parallel Architectures

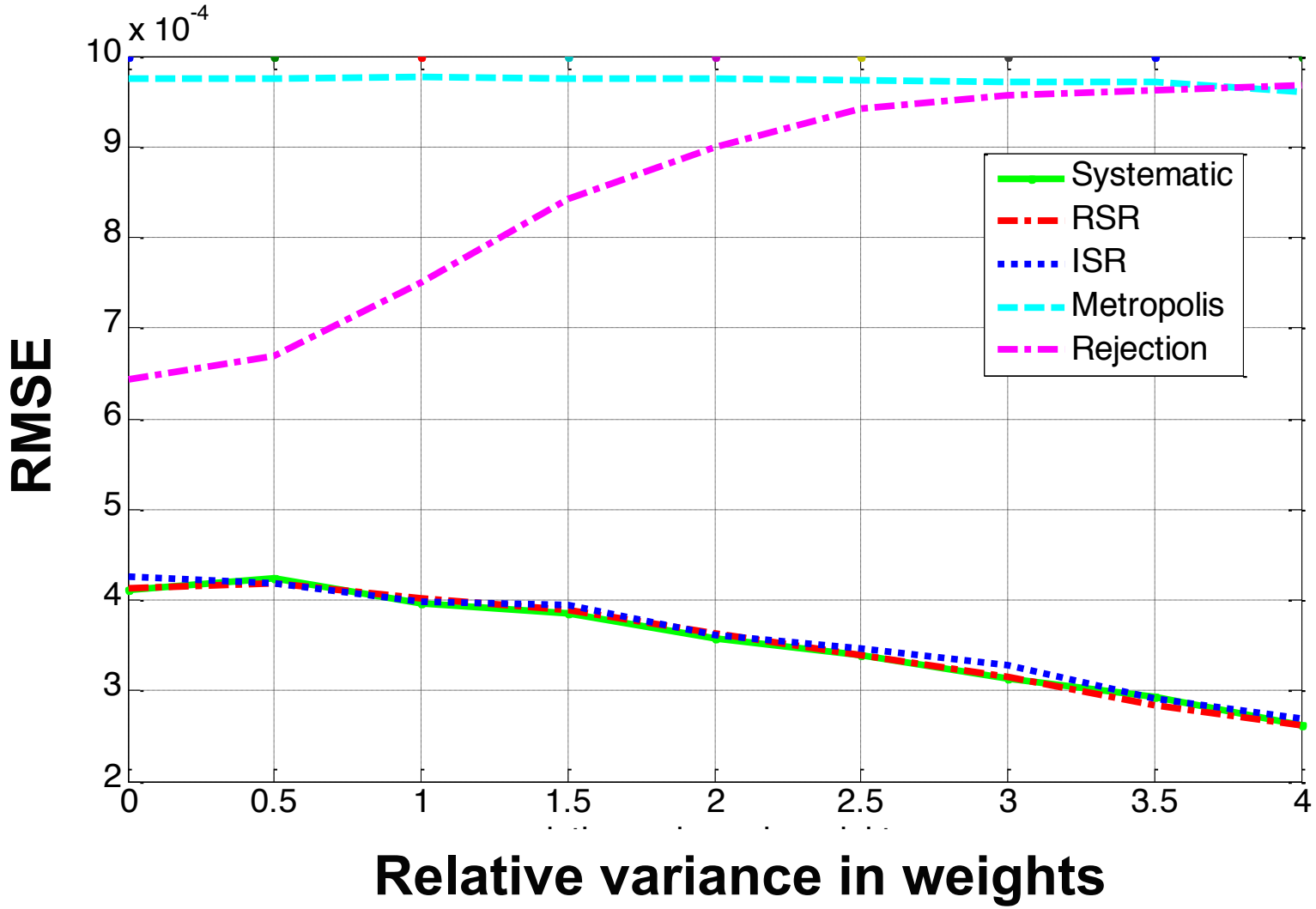
- Metropolis & Rejection Resampling



Outline

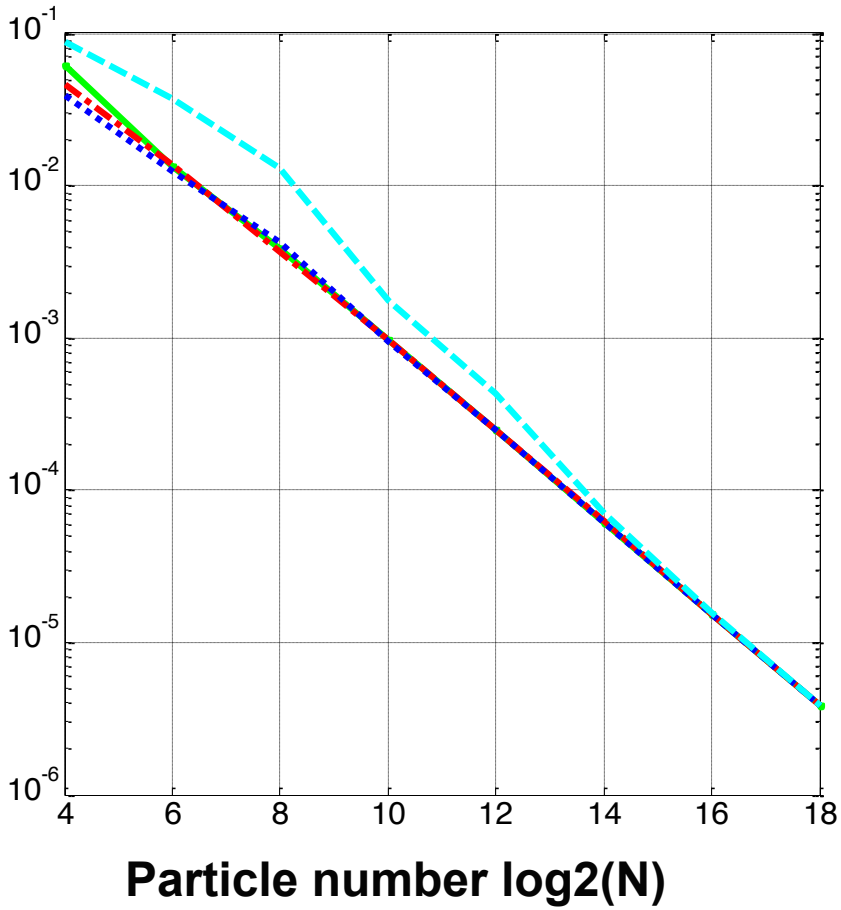
- Motivation
- Introduction & Implementation
- Results
  - Resampling quality
  - Performance on the execution time
- Conclusions

Resampling Quality (RMSE)

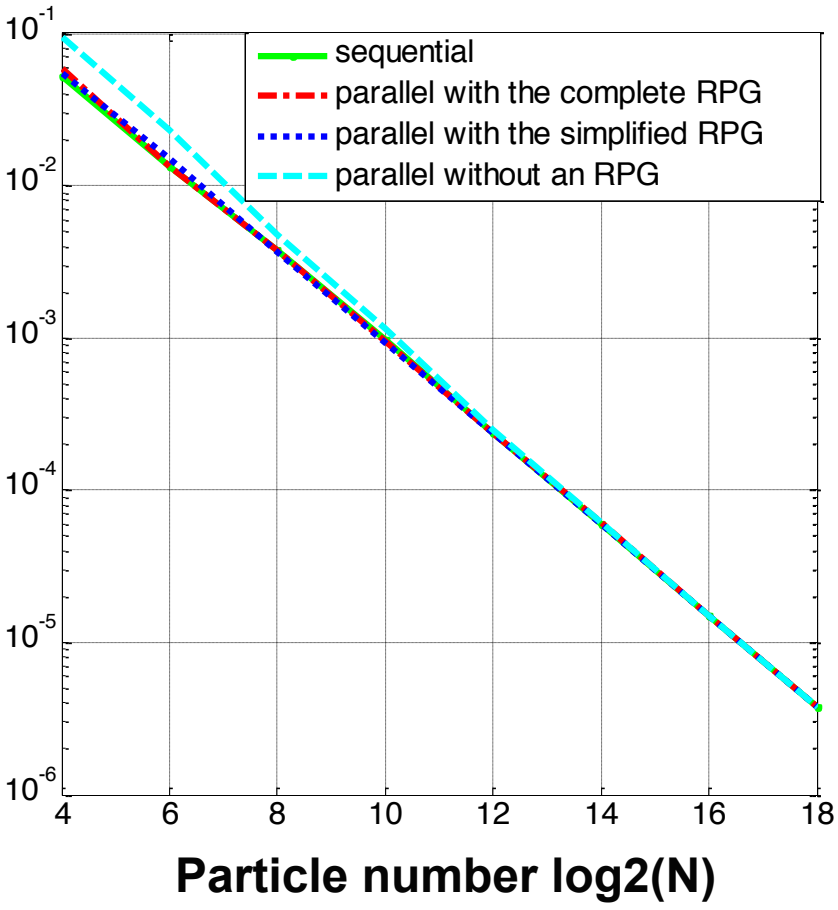


Test results of memory access strategies for parallel Metropolis and Rejection implementations

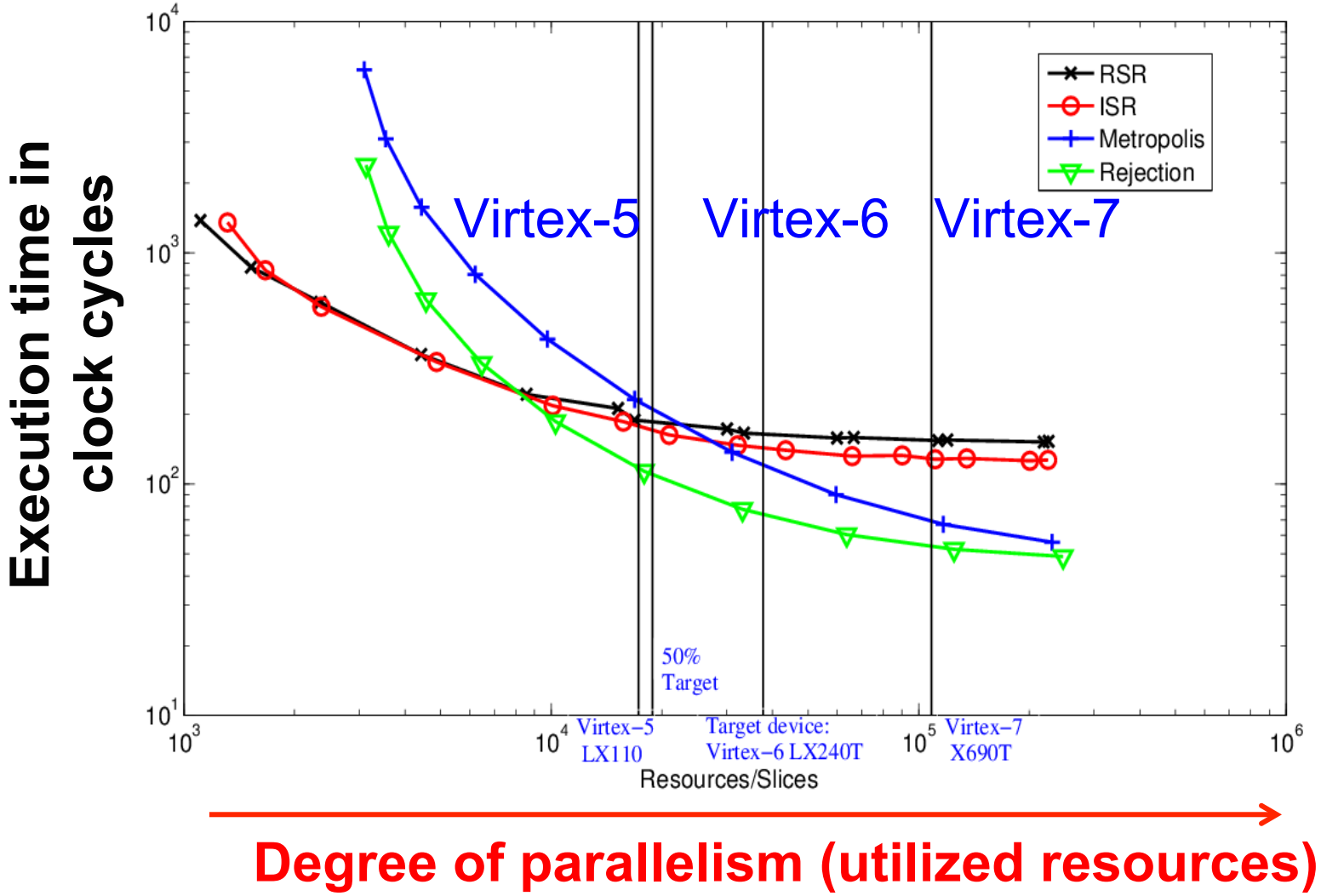
RMSE of Metropolis



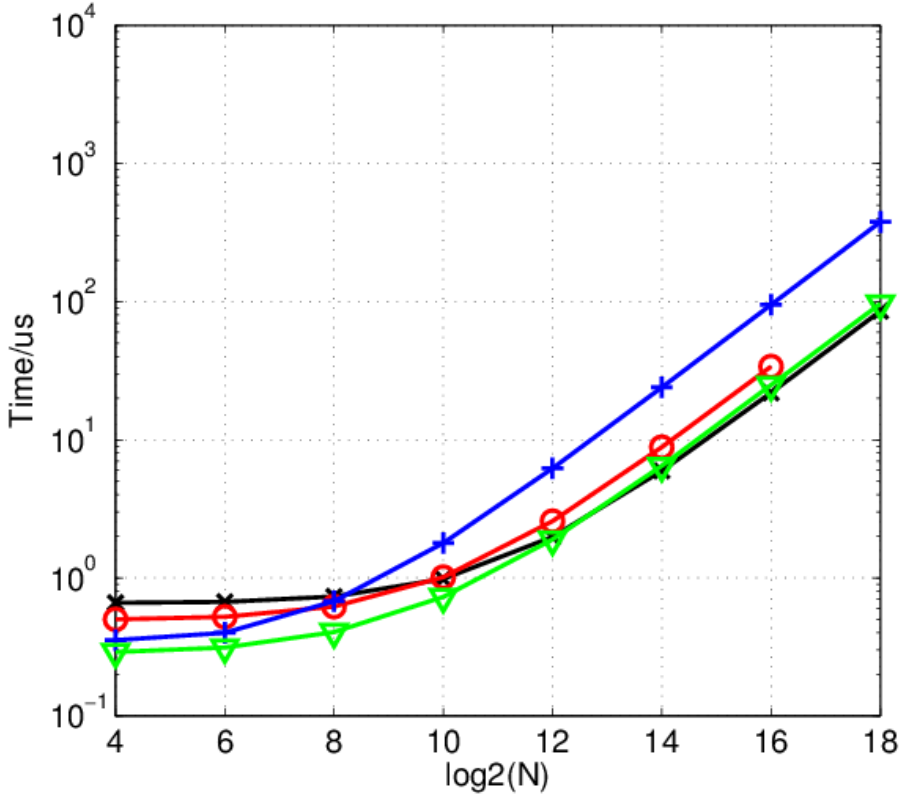
RMSE of Rejection

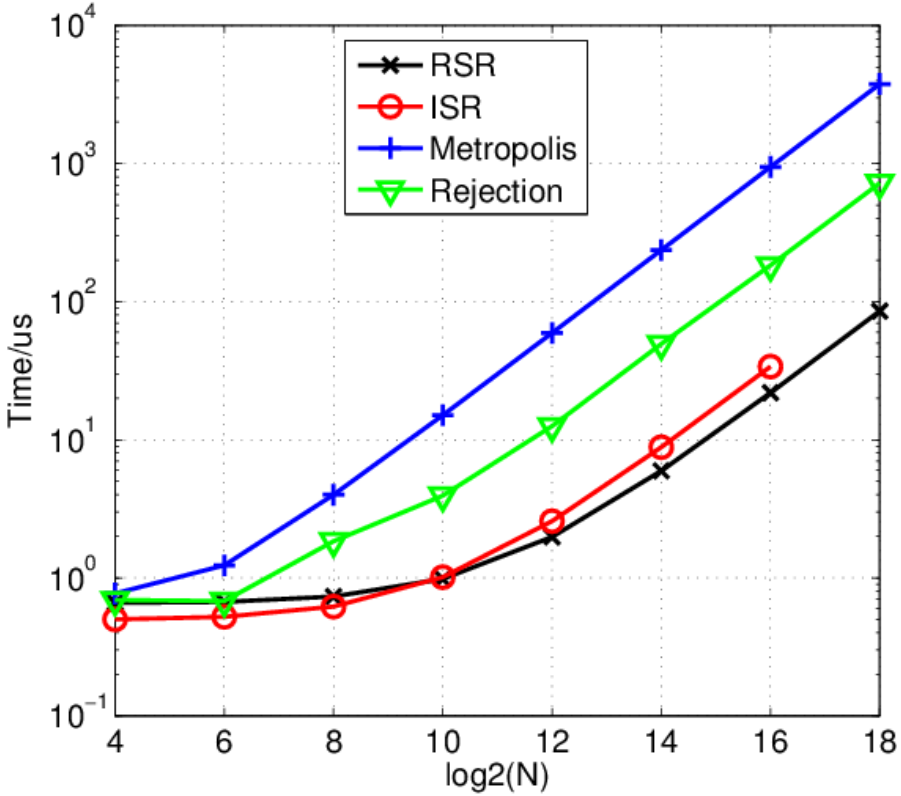


Execution Times vs Resources



Execution Times vs the particle numbers N

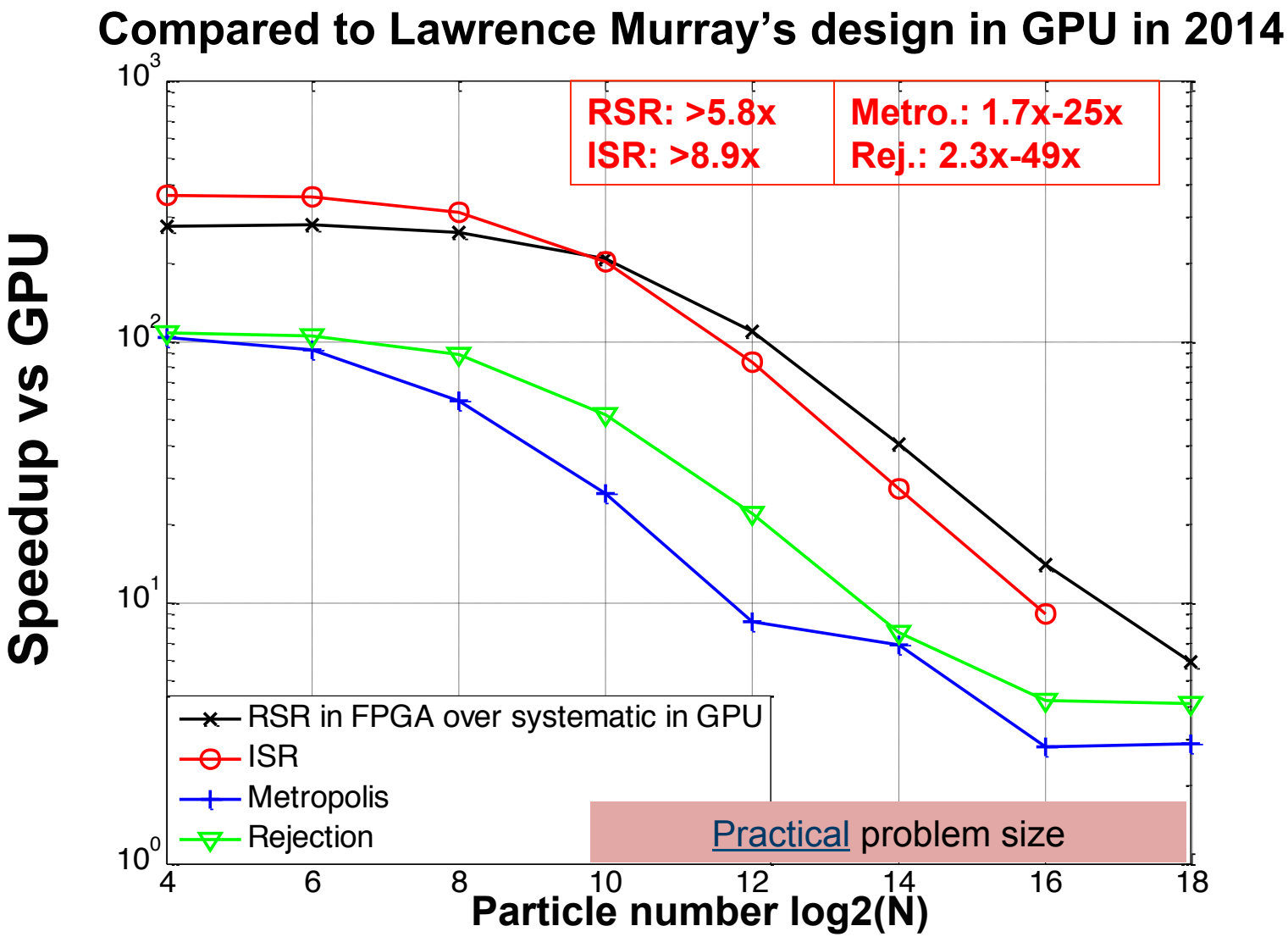




at small variance of the weights

large variance of the weights

Speedup of FPGA implementation vs GPU (NVIDIA K20)



Conclusions

- Our FPGA resamplers achieve 1.7x – 49x speedups vs GPU implementation for the four algorithms;
- Monte Carlo resampling should only be considered at small variance in weights and at small particle numbers;
- The RPG proposed here are confirmed to be necessary for parallel Metro/Rej Architectures, in order to not effect the resampling results.

Thanks

Q & A