Comparison of particle filters for nonlinear systems on massively parallel architectures

Claudio José Paz
cpaz@scdt.frc.utn.edu.ar
Research Center in Informatics for Engineering - UTN - FRC

Introduction

In robotics, there are dynamic systems in which it is necessary to estimate the system current state given information provided by measurements of available sensors. All these measurements contain different kinds of noise. Bayesian filtering is a recursive probabilistic method that estimates the system state using the system model and known sensor noise statistics.

Particle filter (PF) belongs to this kind of filters and it is able to work in systems with large nonlinearities. This algorithm is suitable to be used on parallel architectures, but it has major bottlenecks that need to be optimized. In this work different approaches to implement particle filters on massively parallel architectures are evaluated and the results of these filters in highly non-linear models are shown.

Methods

To evaluate the different algorithms a variant of the univariate nonstationary growth model presented in [1] was implemented as follows

$$x_{k+1} = 0.5x_k + \frac{25x_k}{1+x_k^2} + 8cos(1.2(k-1)) + w_k$$

$$y_k = \frac{x_k^3}{20} + v_k$$

The bootstrap PF approach [1] is the easiest to implement. It consists of three stages: Prediction, Update and Resampling. Between update and resampling, a weighted average of the particles is calculated to obtain the estimated state.

Prediction

$$x_k^i \leftarrow \mathbf{f}(x_{k-1}^i, k, w_k)$$

Update

$$z^i \leftarrow y_k - \mathbf{h}(x_k^i, v_k)$$

$$q_k^i \leftarrow q_{k-1}^i \frac{\exp\left(-\frac{z^{i^2}}{2\sigma_v^2}\right)}{\sqrt{2\pi\sigma_v^2}}$$

$$q_k^i \leftarrow \frac{q_k^i}{\sum q_k}$$

Weighted Average

$$x_k \leftarrow \sum x_k^i q^i$$

Resampling

$$u^i \leftarrow \mathcal{U}(0,1]$$

$$x_k^i \leftarrow x_{k-1}^m \text{ with } \sum_{j=0}^{m-1} q^j < u^i \leq \sum_{j=0}^{m} q^j$$

- Mayor bottleneck of particle filters
- Parallelization is not trivial

$c \leftarrow cumsum(q)$
for $i = 1 \rightarrow N$ **do**
  $u \leftarrow \mathcal{U}(0,1]$
  $l \leftarrow 0$
  while $u > c^l$ **do**
    $l++$
  end while
  $new\_p^i \leftarrow p^l$
end for

Three resampling variants were implemented for performance evaluation: Systematic [2], Shared Memory [3] and Residual Systematic Resampling [4] and all were compared with the bootstrap approach (BSR) for different amount of particles (up to 320K). Prediction and update stages were the same for all tests. Since, all implementations present very low estimation errors, in this work only time performance are shown. All algorithms were implemented on pyCUDA in a GeForce GTX560 board.

Results

Systematic Resampling (SR)

$c \leftarrow cumsum(q)$
$u \leftarrow \mathcal{U}(0, 1/N]$
for $i = 1 \rightarrow N$ **do**
  $\hat{u} \leftarrow u + N^{-1}(i-1)$
  $l \leftarrow 0$
  while $\hat{u} > c^l$ **do**
    $l++$
  end while
  $new\_p^i \leftarrow p^l$
end for

Shared Memory Resampling (SMR)

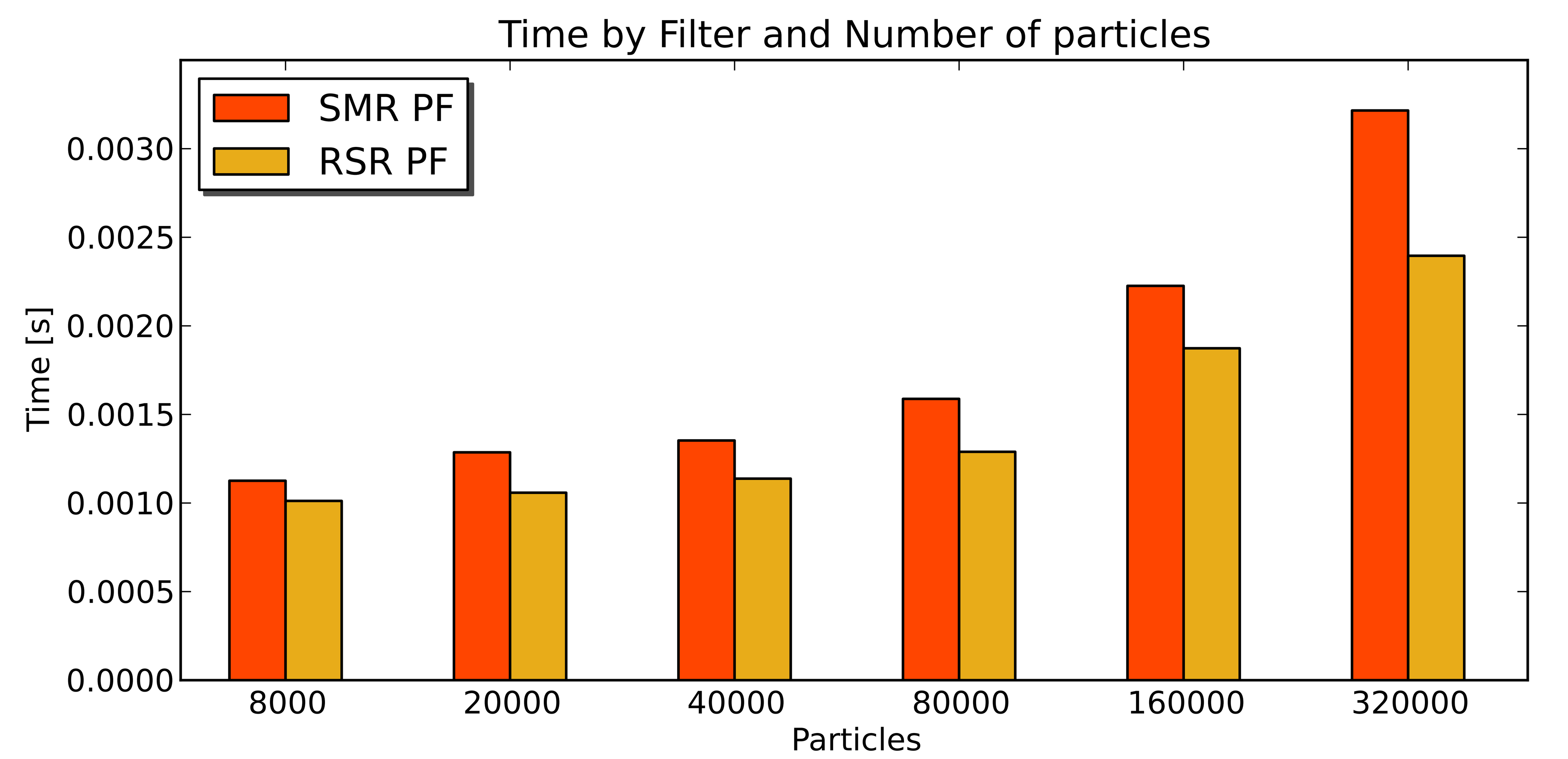
$c \leftarrow cumsum(q)$
$u \leftarrow \mathcal{U}(0, 1/N]$
for $i = 1 \rightarrow N$ **do**
  $b \leftarrow c^i - u$
  $l \leftarrow b * N + 2$
  $r \leftarrow (b + q^i) * N + 1$
  for $j = l \rightarrow r$ **do**
    $new\_p^j \leftarrow p^i$
  end for
end for

Residual Systematic Resampling (RSR)

$c \leftarrow cumsum(q)$
$u \leftarrow \mathcal{U}(0, 1/N]$
for $i = 1 \rightarrow N$ **do**
  $l \leftarrow i - 1$
  if $l >= 0$ **then**
    $u \leftarrow u + l/N - c^l$
  end if
  $l \leftarrow (c^i - u) * N + 1$
  $new\_p^i \leftarrow p^l$
end for

The particle filters were tested using different amount of particles (4K, 8K, 20K, 40K, 80K, 160K and 320K). All implemented methods outperform the bootstrap approach. The fastest was the Residual Systematic Resampling (RSR) which was able to work in real-time with incoming data up to 400Hz.

Filter / N	4K	8K	20K	40K	80K	160K	320K
BSR	0.0020s	0.0033s	0.0109s	0.0592s	0.3256s	0.9051s	2.2223s
SR	0.0014s	0.0018s	0.0032s	0.0055s	0.01073s	0.0204	0.0388s
SMR	0.0010	0.0011s	0.0012s	0.0013s	0.0017s	0.0023s	0.0030s
RSR	0.0009s	0.0010s	0.0011s	0.0012s	0.00132s	0.0018s	0.0024s



Conclusion

Before GPUs, solving real-time problems with particle filters was unsuitable, because they need a big amount of particles for achieve asymptotic results. Nowadays, low-cost GPU allows the implementation of this kind of filters.

Three different resampling algorithms were implemented and tested on GPU using pyCUDA and all of them showed better performance than the parallel version of the bootstrap particle filter. Residual systematic resampling is the fastest implementation. Residual systematic and shared memory resampling are suitables for real-time application even with a big amount of particles.

In future work, the implemented resampling methods will be tested with real problems in real-time applications.

Acknowledgment

This work was partially funded by the project "Autonomous Vehicle Guidance Fusing Low-cost GPS and other Sensors", PICT-PRH-2009-0136.

References

[1] Gordon, Salmond, Smithdt, *Novel approach to nonlinear/non-Gaussian Bayesian state estimation*. IEEE Radar and Signal Processing, 1993
[2] Arulampalam, Maskell, Gordon, Clapp, *A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking*, IEEE Signal Processing, 2002
[3] Gong, Basciftci, Ozguner, *A Parallel Resampling Algorithm for Particle Filtering on Shared-Memory Architectures*, IEEE Parallel and Distrib. Processing, 2012
[4] Bolic, Djuric, Hong, *New resampling algorithms for particle filters*, IEEE Acoustics, Speech, and Signal Processing, 2003