Tang *et al. EURASIP Journal on Wireless Communications and Networking* 2013, **2013**:38
http://jwcn.eurasipjournals.com/content/2013/1/38

EURASIP Journal on
Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Particle filter track-before-detect implementation on GPU

Xu Tang*, Jinzhou Su, Fangbin Zhao, Jian Zhou and Ping Wei

Abstract

Track-before-detect (TBD) based on the particle filter (PF) algorithm is known for its outstanding performance in detecting and tracking of weak targets. However, large amount of calculation leads to difficulty in real-time applications. To solve this problem, effective implementation of the PF-based TBD on the graphics processing units (GPU) is proposed in this article. By recasting the particles propagation process and weights calculating process on the parallel structure of GPU, the running time of this algorithm can greatly be reduced. Simulation results in the infrared scenario and the radar scenario are demonstrated to compare the implementation on two types of the GPU card with the CPU-only implementation.

Keywords: Track-before-detect, Particle filter, GPU

1. Introduction

Classical target detection and tracking is performed on the basis of pre-processed measurements, which are composed of the threshold output of the sensor. In this way, no effective integrations over time can be taken place and much information is lost. To avoid this problem, the track-before-detect (TBD) technique is developed to use directly the un-threshold or low threshold measurements of sensors to utilize the raw information. The TBD-based procedures jointly process more consecutive measurements, thus can increase the signal-to-noise ratio (SNR), and realize the detection and tracking of weak targets simultaneously.

The scenarios faced by TBD are almost nonlinear and non-Gaussian, so the particle filter (PF) [1] is a reasonable solution. The PF is a Monte Carlo simulation method and widely used in target tracking of linear or nonlinear dynamic systems [2,3]. Salmond and coauthors [4,5] first introduced the PF implementation of TBD (PFTBD) in infrared scenario. Then, Rutten et al. [6-8] proposed several improved PFTBD algorithms. Boers and Driessen [9] extended the work of PFTBD into the radar targets detection and the tracking application. PFTBD algorithms have demonstrated the improved track accuracy and the ability to follow the low SNR targets but at the price of an extreme increase of the computational complexity.

In recent years, the field programmable gate array (FPGA) and the graphics processing unit (GPU) are the most important architectures in parallel computing. As the rapid development of GPU technology, GPU is famous for its significant ability in parallel computing for both the graphic processing and the general-purpose computing. Moreover, the compute unified device architecture (CUDA) [10] is introduced to facilitate a hybrid utilization of GPU and central process unit (CPU) [11]. FPGA has been used to implement PFs, such as in [12,13]. However, with the increasing of number of particles, GPU is expected to obtain a better performance than FPGA. More specifically, the PF algorithms have been implemented on GPU [14-16], and achieve significant speedup ratio over the implementations on the traditional CPU fashion but with no losing the performance for its float point computation ability.

From the best of the authors' knowledge, no PFTBD algorithm implemented on GPU is given in the literature. In this article, we propose a novel implementation of PFTBD algorithm on GPU by the CUDA programming. Concerned with the difficulty of PFTBD beyond the PF, new scheme to dispatch the GPU resources for particles are developed and the programming about the likelihood ratio area are considered carefully.

* Correspondence: tangxu@uestc.edu.cn
Department of Electronic Engineering, University of Electronic Science and Technology of China, Chengdu, People's Republic of China

Simulations in both the infrared scenario and the radar scenario are given. Two types of the GPU card are utilized. The implementations of PFTBD algorithm on both of them achieve significant speedup over the CPU-only implementation. The initial version of this research first appeared in [17].

This article is organized as follows. Section 2 reviews the theory about PFTBD. In Section 3, we discuss the details about the parallel implementation of PFTBD on GPU and CUDA programming. The simulations results and discussions can be found in Section 4. Finally, we conclude this article in Section 5.

2. PFTBD theory

In this article, the single target recursive TBD algorithms are presented. The way to process raw measurement of sensor in TBD is different from the classical target tracking methods. The measurement model and the way of data processing vary with the sensor type. The models are set mathematically with a summary of the infrared scenario and the radar scenario. The simulations in Section 4 are based on these models.

2.1. Target model and measurement model of infrared sensor

Consider an infrared sensor that collects a sequence of two-dimensional images of the surveillance region, as in [7]. When the target presents, the state of the target is evolving as a constant velocity (CV) model. The time evolution model of the target used here is a linear Gaussian process

$$X_k = F\cdot \mathrm{X}_{\mathrm{k-1}} + \mathrm{Q}\cdot\mathrm{V}_{\mathrm{k}} \tag{1}$$

where $X_k = \left[x_k\ \bar{x}_k\ y_k\ \bar{y}_k\ I_k\right]^T$ is the state vector of the target. x_k and y_k are the positions of the target and $\bar{x}_k, \bar{y}_k$ are the velocity of the target. I_k is the returned unknown intensity from the target. The process noise V_k is the standard white Gaussian noise. A CV process model is used, which is defined by the transition matrix and the process noise covariance matrix

$$F = \begin{bmatrix} F_s & 0 & 0 \\ 0 & F_s & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad F_s = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix},$$
$$Q = \begin{bmatrix} Q_s & 0 & 0 \\ 0 & Q_s & 0 \\ 0 & 0 & q_2 T \end{bmatrix}, \quad Q_s = q_1 \begin{bmatrix} T^3/3 & T^2/2 \\ T^2/2 & T \end{bmatrix} \tag{2}$$

where T is the period of time between measurements, q_1 and q_2 denote the variance of the acceleration noise and the noise in target return intensity, respectively.

The variable $E_k \in \{e, \bar{e}\}$ denotes the existence or non-existence of the target and evolves according to a two-state Markov chain. The transitional probability matrix is defined by $\prod_{ij} = \begin{pmatrix} 1-P_b & P_b \\ P_d & 1-P_d \end{pmatrix}$, where $P(E_k = 1| E_{k-1} = 0) = P_b$ is the probability of target birth and $P(E_k = 0|E_{k-1} = 1) = P_d$ is the probability of target disappearance.

The measurement z_k at each time is a two-dimensional intensity image of the interested region consisting of the $n \times m$ resolution cells. The measurement of each cell $z_k^{(i,j)}$ with $i = 1,\ldots, n,\ j = 1,\ldots, m$ is as

$$z_k^{(i,j)} = \begin{cases} h^{(i,j)}(X_k) + W_k^{(i,j)}, & E_k = e \\ W_k^{(i,j)}, & E_k = \bar{e} \end{cases} \tag{3}$$

where $h^{(i,j)}(X_k)$ is the intensity of the target in the cell (i,j). $h^{(i,j)}(X_k)$ is also the spread reflection form of target and is defined for each cell by

$$h^{(i,j)}(X_k) = \frac{\Delta_x \Delta_y I_k}{2\pi \sum^2} \exp\left(-\frac{(x_k - i\Delta_x)^2 + (y_k - i\Delta_y)^2}{2\sum^2} \right) \tag{4}$$

where Δ_x and Δ_y denote the size of a resolution cell in each dimension. The parameter $\sum$ represents the extent of blurring. Then the likelihood function during the presence and absence of the target, respectively, in each cell can be written as

$$\begin{cases} p(z_k^{(i,j)}|X_k, E_k = 1) = \dfrac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\dfrac{\left[z_k^{(i,j)} - h^{(i,j)}(X_k)\right]^2}{2\sigma^2} \right) \\ p(z_k^{(i,j)}|X_k, E_k = 0) = \dfrac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\dfrac{\left[z_k^{(i,j)}\right]^2}{2\sigma^2} \right) \end{cases} \tag{5}$$

Therefore, the likelihood ratio for cell (i, j) is giving as

$$\ell(z_k^{(i,j)}|X_k, E_k) = \begin{cases} \dfrac{p(z_k^{(i,j)}|X_k, E_k = e)}{p(z_k^{(i,j)}|E_k = \bar{e})}, & E_k = e \\ 1, & E_k = \bar{e} \end{cases} \tag{6}$$

where

$$\ell(z_k|X_k, E_k = 1) \approx \prod_{i \in C_x(X_k)} \prod_{j \in C_y(X_k)} \exp\left(\frac{-h^{(i,j)}(X_k)\left[h^{(i,j)}(X_k) - 2z_k^{(i,j)}\right]}{2\sigma^2} \right) \tag{7}$$

and $C_x(X_k)$, $C_y(X_k)$ are the index sets of cells that are affected by the target in the x and y dimensions, named as the likelihood ratio area. The size of them is determined by the application parameter, such as the resolution of the observation area and the intensity of the target. The bigger likelihood ratio area, the more latent target information can be utilized.

The measurement noise $W_k^{(i,j)}$ in each cell is assumed as the independent white Gaussian distribution with zero mean and variance σ^2. The SNR for the target is defined by

$$\mathrm{SNR} = 10\log\left[\left(I_k \Delta_x \Delta_y / 2\pi\Sigma^2\right)/\sigma\right]^2 (\mathrm{dB}) \tag{8}$$

2.2. Target model and measurement model of radar sensor

Different from infrared sensor, the raw measurement data of radar sensor are always based on range-Doppler-bearing of the signal, as in [9]. The time evolution model of the target used here is also a linear Gaussian process:

$$X_k = F \cdot \mathrm{X_{k-1}} + \mathrm{Q} \cdot \mathrm{V_k} \tag{9}$$

where $X_k = \left[x_k\, \bar{x}_k\, y_k\, \bar{y}_k\right]^T$ is the state vector of the target. x_k and y_k are the positions of the target and $\bar{x}_k, \bar{y}_k$ are the velocity of the target. If we make T as the update time, then the transition matrix and the process noise covariance matrix can be defined as

$$F = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$
$$Q = \begin{bmatrix} 1/2(a_{\max x}/3)T^2 & 0 \\ 0 & 1/2\left(a_{\max y}/3\right)T^2 \\ 1/2(a_{\max x}/3)T & 0 \\ 0 & 1/2\left(a_{\max y}/3\right)T \end{bmatrix} \tag{10}$$

where $a_{\max x}$, $a_{\max y}$ is the maximum accelerations and the process noise V_k is the standard white Gaussian noise.

At each discrete time k, the measurement z_k is the reflected power of target. z_k is based on the presence of the target and is defined by

$$z_k = \begin{cases} h(X_k) + W_k, & E_k = e \\ W_k, & E_k = \bar{e} \end{cases} \tag{11}$$

In this article, the measurements are modeled as power levels in range-Doppler-bearing $N_r \times N_d \times N_b$ sensor cells. Thus, z_k can be defined by $z_k = \{z_k^{(i,j,l)} : i = 1,\ldots,N_r,\ \ j = 1,\ldots,N_d,\ \ l = 1,\ldots,N_b\}$. The power measurements per range-Doppler-bearing cell can be defined as $z_k^{(i,j,l)} = |z_{A,k}^{(i,j,l)}|^2$, where $z_{A,k}^{(i,j,l)}$ represents the complex amplitude data of the target, which is

$$z_{A,k}^{(i,j,l)} = \begin{cases} A_k h_A^{(i,j,l)}(X_k) + n_k, & E_k = e \\ n_k, & E_k = \bar{e} \end{cases} \tag{12}$$

where A_k is the complex amplitude and $A_k = \widetilde{A}_k e^{i\phi_k}, \phi_k \in (0, 2\pi)$. n_k is complex Gaussian noise defined by $n_k = n_{Ik} + in_{Qk}$, where n_{Ik} and n_{Qk} are independent, zero mean white Gaussian noise with variance σ_n^2. They are related to W_k as $W_k = |n_{Ik} + n_{Qk}|^2$. $h_A^{(i,j,l)}(X_k)$ is the reflection form that is defined for every range-Doppler-bearing cell by

$$h_A^{(i,j,l)}(X_k) = \exp\left(-\frac{(r_i - r_k)^2}{2R}L_r - \frac{\left(d_j - d_k\right)^2}{2D}L_d - \frac{(b_l - b_k)^2}{2B}L_b\right) \tag{13}$$

where $i = 1,\ldots,N_r$, $j = 1,\ldots,N_d$, and $l = 1,\ldots,N_b$ with

$$\begin{cases} r_k = \sqrt{x_k^2 + y_k^2} \\ d_k = \dfrac{1}{\sqrt{x_k^2 + y_k^2}}\left(x_k \bar{x}_k + y_k \bar{y}_k\right). \\ b_k = \arctan(y_k / x_k) \end{cases} \tag{14}$$

L_r, L_d, and L_b are constants of power losses. R, D, and B are related to the size of a range, a Doppler, and a bearing cell. In summary, the power in every range-Doppler-bearing measurement cell can be defined as

$$\begin{aligned} z_k^{(i,j,l)} &= \left|z_{A,k}^{(i,j,l)}\right|^2 \\ &= \begin{cases} \left|A_k h_A^{(i,j,l)}(X_k) + n_{Ik} + in_{Qk}\right|^2 & E_k = e \\ \left|n_{Ik} + in_{Qk}\right|^2 & E_k = \bar{e} \end{cases} \end{aligned} \tag{15}$$

These measurements are suppose to be exponentially distributed [10],

$$p(z_k^{(i,j,l)} | X_k, E_k) = \frac{1}{\mu_0^{(i,j,l)}} \exp\left(-\frac{z_k^{(i,j,l)}}{\mu_0^{(i,j,l)}}\right) \tag{16}$$

where

$$\begin{aligned}\mu_0^{(i,j,l)} &= E_{n_{Ik},n_{Qk}}\left[z_k^{(i,j,l)}\right]\\ &= E_{n_{Ik},n_{Qk}}\left[\begin{cases}\left|\widetilde{A}_k e^{i\varphi_k} h_A^{(i,j,l)}(X_k)+n_{Ik}+in_{Qk}\right|^2\\ \left|n_{Ik}+in_{Qk}\right|^2\end{cases}\right]\\ &= \begin{cases}\widetilde{A}_k{}^2\left(h_A^{(i,j,l)}(X_k)\right)^2+2\sigma_n^2\\ \sigma_n^2\end{cases}\\ &= \begin{cases}Ph_P^{(i,j,l)}(X_k)+2\sigma_n^2 & E_k=e\\ \sigma_n^2 & E_k=\bar{e}\end{cases}\end{aligned} \tag{17}$$

with

$$\begin{aligned}h_P^{(i,j,l)}(X_k) &= \left(h_A^{(i,j,l)}(X_k)\right)^2\\ &= \exp\left\{-\frac{(r_i-r_k)^2}{R}L_r-\frac{(d_j-d_k)^2}{D}L_d\right.\\ &\quad\left.-\frac{(b_l-b_k)^2}{B}L_b\right\}\end{aligned} \tag{18}$$

which generalizes the power of the target in every range-Doppler-bearing cell.

The process of the likelihood ratio in the measurement model of radar sensor is the same with that in the measurement model of infrared sensor of (6).

The SNR for the radar model is defined by

$$\mathrm{SNR} = 10\log\left(P/2\sigma^2\right)[\mathrm{dB}] \tag{19}$$

2.3. PF solution for TBD

Different from PF, the *posteriori* filtering distribution is calculated with a mixture of two parts of particles.

(1) One part is the birth particles $\left\{X_k^{(b)i},\widetilde{w}_k^{(b)i}\right\}$, which did not existed in the previous time and are sampled from proposal distribution, when $E_{k-1}^{(b)i}=\bar{e}$, but $E_k^{(b)i}=e$.
(2) The other part is the continuing particles $\left\{X_k^{(c)i},\widetilde{w}_k^{(c)i}\right\}$, which keep existence and are sampled from state transition probability density, when $E_{k-1}^{(c)i}=e$, but $E_k^{(c)i}=e$.

The algorithm routine of PFTBD is given as follows [6]:

Sample a set of N_b birth particles from the proposal density $X_k^{(b)i}\sim q_b(X_k|E_k=e,E_{k-1}=\bar{e},z_k)$ and calculate the unnormalized weights of birth particles from the likelihood ratio:

$$\widetilde{w}_k^{(b)i} = \frac{l(z_k|X_k^{(b)i},E_k^{(b)i}=e)p(X_k^{(b)i}|E_k^{(b)i}=e,E_{k-1}^{(b)i}=\bar{e})}{N_b q\left(X_k^{(b)i}\middle|E_k^{(b)i}=e,E_{k-1}^{(b)i}=\bar{e},z_k\right)}, \tag{20}$$

where $q_b\left(X_k^{(b)i}\middle|E_k^{(b)i}=e,E_{k-1}^{(b)i}=\bar{e}\right)$ is the prior density of the target.

Sample a set of N_c continuing particles from state transition probability density $X_k^{(c)i}\sim q_c(X_k|E_k=e,E_{k-1}=e,z_k)$. The unnormalized weights of continuing particles are given as

$$\widetilde{w}_k^{(c)i} = \frac{1}{N_c}l\left(z_k\middle|X_k^{(c)i},E_k^{(b)i}=e\right) \tag{21}$$

where $q_c(X_k|E_k=e,E_{k-1}=e,z_k)$ is the state transition density of the target.

Calculate the probability of existence about the target according to the unnormalized weights of particles

$$\hat{P}_k = \frac{\widetilde{M}_b+\widetilde{M}_c}{\widetilde{M}_b+\widetilde{M}_c+P_d\hat{P}_{k-1}+[1-P_b]\left[1-\hat{P}_{k-1}\right]} \tag{22}$$

with $\widetilde{M}_b = P_b\left[1-\hat{P}_{k-1}\right]\sum_{i=1}^{N_b}\widetilde{w}_k^{(b)i},\ \widetilde{M}_c=[1-P_d]\hat{P}_{k-1}\sum_{i=1}^{N_c}\widetilde{w}_k^{(c)i}.$

Normalize the weights of particles as $w_k^{(b)i}=\frac{P_b\left[1-\hat{P}_{k-1}\right]}{\widetilde{M}_b+\widetilde{M}_c}\widetilde{w}_k^{(b)i}$, $w_k^{(c)i}=\frac{[1-P_b]\hat{P}_{k-1}}{\widetilde{M}_b+\widetilde{M}_c}\widetilde{w}_k^{(c)i}$ and resample N_b+N_c particles down to N_c particles $\{X_k^i, 1/N_c\}$. Give the estimate of the target state at time k and calculate the root mean square error (RMSE) of location error by: $L_{\mathrm{RMSE}}=\sum_{i=1}^{n}\sqrt{\left(X_k^i-\hat{X}_k^i\right)^2}/N_c$.

The main difference of PFTBD from the general PF in both the infrared scenario and the radar scenario is that a product of cell's intensity in the observation area is needed in the calculation of particle weight, as in (6). Moreover, this operation is the main body that contributes the high time complexity of PFTBD. Suppose that the time complexity in weight process of PF is $O(m)$ with m particles. Then in PFTBD, the time complexity of weight process is $O(m\times n^2\times n)$ for a sequential algorithm with $n\times n$ cells and m particles. Thus, some efficient parallel implementations should be introduced to relief this overhead.

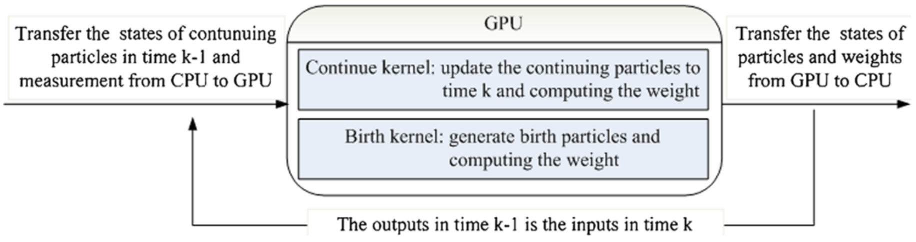
Transfer the states of contuning particles in time k-1 and measurement from CPU to GPU
GPU
Continue kernel: update the continuing particles to time k and computing the weight
Birth kernel: generate birth particles and computing the weight
Transfer the states of particles and weights from GPU to CPU
The outputs in time k-1 is the inputs in time k

Figure 1 The process of computing states and weights of particles on GPU.

3. The implementation of PFTBD on GPU

3.1. Parallel processing on CUDA

In the modern GPUs, there are hundreds of processor cores, which are named as the stream multiprocessor (SM). Each SM contains many scalars stream processors (SP) and can perform the same instructions simultaneously. CUDA is a general purpose parallel computing architecture that makes GPUs to solve complex problems in a more efficient way than on a CPU. In CUDA programming, GPU can be responsible for the parallel computationally intensive parts and CPU can accomplish the other parts. On GPU, each task schedule unit, named as the kernel, is performed in the thread on the SP. The threads are organized into the block that is performed on the SM [18]. Threads can communicate with the other threads in the same block by using the shared memory efficiently. Moreover, two thumb rules should be noted: (1) Overhead data transferring between the GPU and the CPU should be avoided. (2) Access data in the shared memory is much cheaper than in the global memory of GPU [18].

3.2. PFTBD on GPU

Obviously, in both implementations of PF and PFTBD, the particles propagation process and weights computing process have the high computational cost but with high concentration of parallelizability. Considering the implementation of PF on GPU, both processes above can be realized in one kernel because there are regular operations in individual threads. However, in PFTBD, different from the particles propagation process in PF, there are two kinds of particles, $X_k^{(c)i}$ and $X_k^{(b)i}$. The way to get the states of this two kinds of particles is different, as the difference of $q_b\left(X_k^{(b)i}\middle|E_k^{(b)i}=e,E_{k-1}^{(b)i}=\bar{e}\right)$ and $q_c(X_k|E_k = e, E_{k-1} = e, z_k)$ in Section 2.3. Besides that, there are product operations among threads in the calculation of particle weight which is also different from PF. Therefore, we schedule PFTBD with two CUDA kernels named as the birth kernel and the continue kernel, respectively, onto GPU. The birth kernel calculates the state and weight of birth particles. The continue kernel do the same calculations with continue particles.

The input data of both kernels, which is transferred from the CPU memory to the GPU global memory, are the measurement data in the current time step. For continue kernel, the state of continuing particles in the previous time step is also needed. GPU blocks and threads are allocated according to the number and state of particles. The state of continuing particles $X_k^{(c)i}$ updates by the prior density of target. Meanwhile, the state of birth particles $X_k^{(b)i}$ samples from the proposal density $q(\bullet)$ with uniform distribution on GPU. The noise with Gaussian distribution is generated on GPU by the CUDA library functions.

After obtained the states of the particles, both kernels calculate the weights of the particles. During the process of weights computing in (20) and (21), the states of particles are needed. On this point, the process of getting the states of particles and weights computing are combined into one kernel to overcome excessive data transmission between CPU and GPU. This part of kernel can be designed as various forms according to different size of $C_x(X_k)$ and $C_y(X_k)$ in (7). The number of blocks is equal to the number of particles and the size of threads is equal to the size of likelihood ratio area cells. In our implementation, depending on the size of surveillance region, we extend $C_x(X_k)$ and $C_y(X_k)$ from all the sets of cell indices to part. According to this approach, we can extend the application background

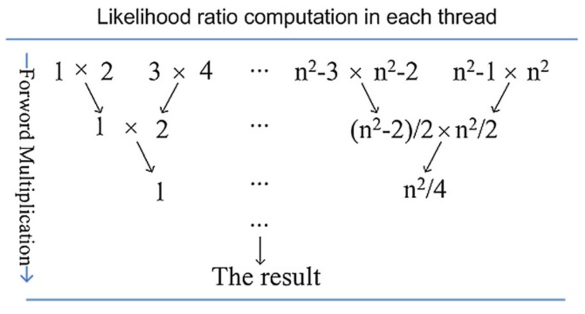

Figure 2 Implementation of multiplication in the shared memory.

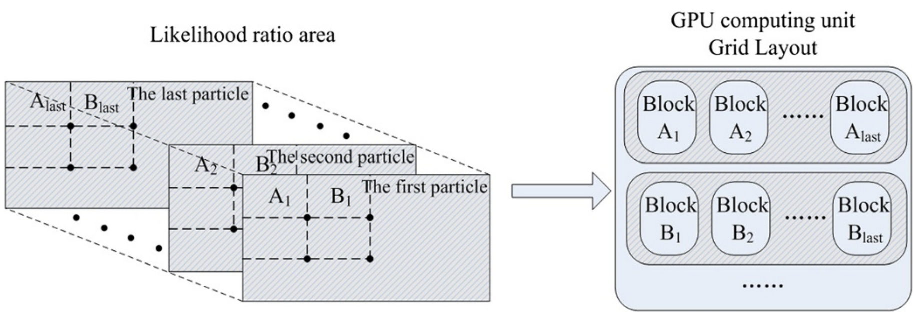

Figure 3 The programming on GPU for the likelihood ratio area of large scene.

from small scenes to large scenes. This problem is made a farther discussion in Section 3.3.

Figure 1 shows that both kernels need current measurements z_k as the inputs. Meanwhile, to update the continuing particles, the previous states of continuing particles $X_{k-1}^{(c)i}$ are also needed. After computing state and weight of particles on GPU, the state and weights of both parts of particles, as the outputs, are transferred back to CPU.

Other operations, such as calculating the probability of detection, resampling, and estimating the state of the target which needs interaction for all state of particles and their weights cannot be implemented in parallel, are arranged on CPU.

3.3. Likelihood ratio area programming

The likelihood ratio function is a multiplication over all the contributions of likelihood ratio area cells. For a scale of n^2 array of likelihood ratio area cells with m particles are used, the computing of weight will entail m blocks and resulting n^2 threads in each block. The value of n^2 should be smaller than the maximum number of threads limited by the hardware. Under this condition, the calculating of likelihood ratio in each cell can be parallelized in every thread, but the process of product cannot be parallelized.

In order to alleviate the time complexity caused by the multiplication, the reduction algorithm is adopted in the shared memory of each block to do the product as illustrated in Figure 2. In this way, running time of the multiplication can be reduced significantly. As a result, the time complexity of weight process in PFTBD could be $O(\log_2 n)$ comparing to $O(n)$ without using the shared memory.

For larger application scenarios, such as in radar application, the likelihood ratio area includes a 3D array of dimension $R \times D \times B$, which always exceeds the maximum available threads in one block. Thus, some strategies must be applied to resolve the massive parallelism in large scene. According to the number of threads in one block, the likelihood area is divided into small areas as illustrated in Figure 3. The multiplications of each area A_1, A_2,. . .,A_{last} are calculated on GPU at the same time and other areas are sequentially calculated. Then the result of the likelihood ratio area is the product of all the small areas.

Obviously, the operations in Figure 3 are complex. From algorithm aspect, Torstensson and Trieb [19] have made a research on different size of likelihood ratio areas in radar application. Its scheme is to use small likelihood ratio areas to obtain the tradeoff between the performance and the extremely high computational cost. In the GPU implementation, we can follow the idea of [19] to sidestep the complex operations discussed above. More specifically, by using likelihood ratio areas with sizes that are just lower than the block, we can obtain better performance but with little computational cost increase. The simulations on different size of likelihood ratio areas are given in Section 4.2.

4. Simulation results

4.1. Simulations in infrared scenario

The simulation in infrared scenario is based on the model in [7]. The length of observation time is 30 and a target presents from frames 7 to 21. The observation area is divided into $n \times m = 20 \times 20$ cells and the cell size is $\Delta x = \Delta = 1$. The probability of birth and death is

Table 1 Benchmark systems

	System 1	System 2	System 3
Software	Visual studio 2010 professional with CUDA 4.1 SDK		MATLAB 2010a
Hardware	Nvidia GeForce GT9500	Nvidia GeForce 240GT	Pentium(R) Dual-Core E5800 @ 3.20 GHz
	32 cores @ 550 MHz	96 cores @ 550 MHz	
	16.0 GB/s GDDR2	54.4 GB/s GDDR3	

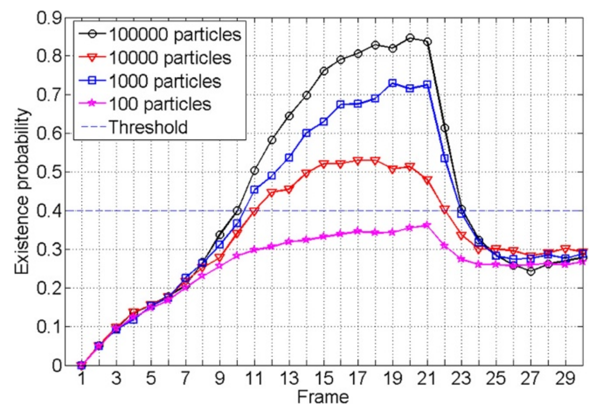

Figure 4 The existence probability with different number of particles.

set as P_b = 0.05 and P_d = 0.05. The initial state of the target is X_7 = $[4.2\ 0.45\ 7.2\ 0.25\ 20]^T$. The SNR is 3 dB. More information about the parameters can be seen in [7]. Various numbers of particles are adopted with each 100 Monte Carlo trials. To verify the effect of different implementation, simulations are performed on three systems, which are given in Table 1.

4.1.1. The performances with different numbers of particle

The performances of the existence probability and the location error on System 2 with different number of particles are compared in Figures 4 and 5, respectively.

Figure 4 shows that with the increase in the number of particles, the probability of detection improves significantly. When the number of particle is 100, the existence probability is always below the detection threshold, so the target cannot be detected. However, with 1,00,000 particles, not only the target can be detected faster, but also the detection probability is increased rapidly. With the time accumulation, when the target appears, the detection probability can eventually reach more than 0.8. Therefore, the number of particles is one of the key factors of the detection performance in PFTBD. From the other side, Figure 5 shows that the location error decreases efficiently with the increase in the number of particles. Note that both the results are consistent with the theory algorithm and have faint difference with the results of System 3, which are not given here for simplicity.

4.1.2. The speedup ratio of GPU to CPU

For larger likelihood region, the parallel computing is more complex and hence GPU achieves higher efficiency than CPU. As the likelihood region reducing, the speedup ratio of GPU to CPU decreases but the processing time on the GPU is still faster than the CPU. A further analysis of the time spent in different part of the algorithm by the hybrid implementation of GPU and CPU is shown in Figure 6.

Figure 6 shows that most of the time is spent on the two kernel: Birth particle(·) and Continue particle(·). Eighty percentage of executive time is cost on GPU. It means that GPU has fully been utilized.

The running time on GT9500, 240GT, and CPU, respectively, in different number of particles is given in Figure 7.

From Figure 7, we can find that with the growing number of particles, the speedup ratio between GPUs and CPU improves significantly. Moreover, Figure 7 shows that the speedup ratio of 240GT is quadruple than GT9500. It is consistent with the specifications in Table 1 that 240GT has the number of CUDA cores triple than that in GT9500 and the memory interface width is much larger than that in GT9500.

4.2. The simulation in radar scenario

The simulation in radar scenario is based on the model in [19]. Length of observation time is 30 and target presents from frames 7 to 21. Initially, the range and Doppler cells of particles are uniformly distributed between [85, 90]km, [−0.22, −0.10]km/s in x direction, and

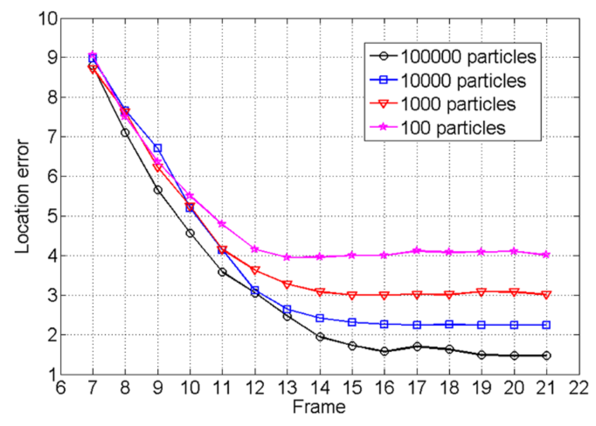

Figure 5 The location error with different number of particles.

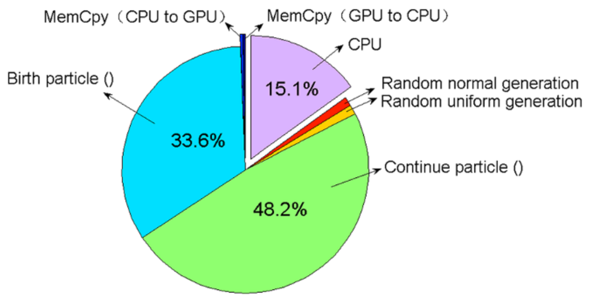

Figure 6 Relative time spent in different parts of GPU and CPU implementation.

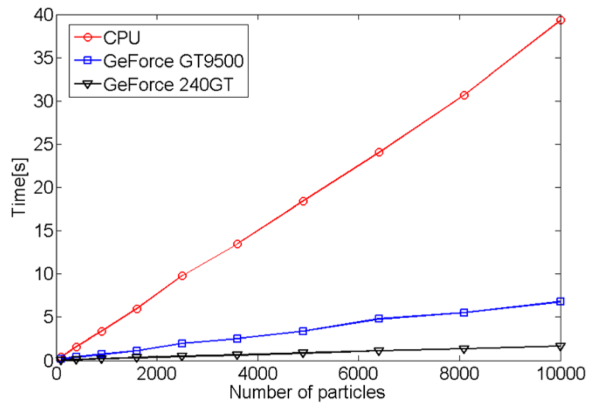

Figure 7 Time comparison between CPU and GPU with different numbers of particles.

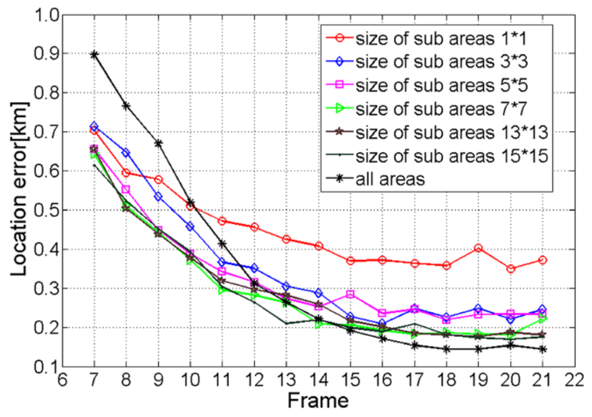

Figure 9 The location error with different size of sub areas in radar scenario.

[−0.1, 0.1]km, [−0.10, 0.10]km/s in y direction. The measurements are consisting of $N_r \times N_d \times N_b = 50 \times 16 \times 1$ sensor cells in each time. The initial state of the target is $X_7 = [89.6\ 0.2\ 0\ 0]^T$. The SNR is 3 dB. The number of both the birth particles and continue particles is 10,000. More information about the parameters can be seen in [19]. In this simulation, the same benchmark systems with Section 4.1 are used.

From the algorithm routine in Section 2, we know that there are no remarkable differences in the implementation of PFTBD on GPU between infrared scenario and radar scenario. Here, we emphasize the results for different size of likelihood ratio areas. The probability of detection and the location error with various sizes of likelihood ratio areas on System 2 are given in Figures 8 and 9, respectively.

Figures 8 and 9 show that with the increasing in the size of sub areas, the probability of detection improves and the location error decreases. The simulation results in the size of sub areas 3 × 3 performs much better than in sub areas of 1 × 1. However, when the size of sub areas extends larger, the existence probability has not improved significantly. From Table 2, we can see that the computational cost is greatly increased as the sub areas increasing in System 3. Nevertheless, in System 2, the increase in running time is so mild for they are all processed in the share memory. Moreover, when the size of sub areas is bigger than 5 × 5, the advantage of the parallelism in GPU can be seen. To our consideration, in the GPU-implemented PFTBD, the size of likelihood areas should be adjusted not only by the application scenario, but also by the amount of share memory on specified hardware.

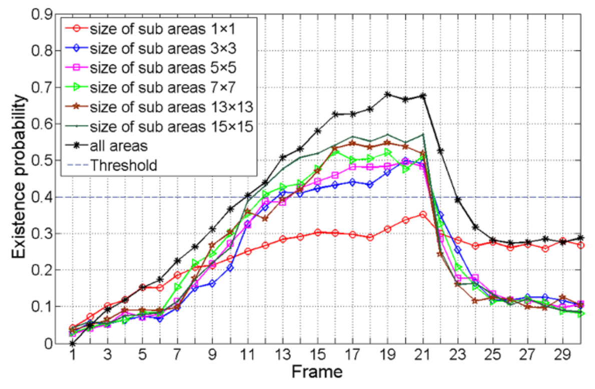

Figure 8 The existence probability with different size of sub areas in radar scenario.

5. Conclusions

In this article, we propose an efficient implementation of PFTBD algorithm on GPU by CUDA programming. Since the parallel part of the PFTBD algorithm bears the main computation, the running time of the GPU-implemented PFTBD algorithm is greatly reduced by effectively dealing with the particles and the likelihood ratio computations. The implementations are tested on two types of GPU card for the infrared scenario and the radar scenario. As a result, the performance of the GPU-implemented PFTBD algorithm can significantly be improved by employing much more particles in GPU than in CPU.

Table 2 Running time for various sub areas compared between systems 2 and 3

Condition	Sub areas					
	1 × 1	3 × 3	5 × 5	7 × 7	13 × 13	15 × 15
System 3 time(s)	3.422	6.070	10.025	16.171	41.322	50.853
System 2 time (s)	10.343	10.436	10.578	10.976	15.972	20.446

Competing interests

The authors declare that they have no competing interests.

Acknowledgments

This study was supported by the Fundamental Research Funds for the Central Universities of China (ZYGX2011J012).

Received: 11 December 2012 Accepted: 8 January 2013
Published: 19 February 2013

References

1. B Ristic, S Arulampalam, N Gordon, *Beyond the Kalman Filter: Particle Filters, for Tracking Applications* (Artech House, Boston, 2004)
2. DJ Salmond, D Fisher, NJ Gordon, Tracking in the presence of spurious objects and clutter, in *Proceedings of SPIE, Signal and Data Processing of Small Targets, vol. 3373* (, Farnborough, Hants, UK, 1998), pp. 460–747
3. MS Arulampalam, S Maskell, N Gordon, T Clapp, A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. IEEE Trans Signal Process **50**(2), 174–188 (2002). doi:10.1109/78.978374
4. DJ Salmond, H Birch, A particle filter for track-before-detect, in *Proceedings of the American Control Conference*. vol. 5 (Arlington, VA, USA, 2001), pp. 3755–3760. doi:10.1109/ACC.2001.946220
5. M Rollason, D Salmond, A particle filter for track-before-detect of a target with unknown amplitude, in *Proceedings of the IEEE International Seminar on Target Tracking Algorithms and Applications*. vol. 1 (QinetiQ, Farnborough, UK, 2001), pp. 14-1–14-4. doi:10.1049/ic:20010240
6. MG Rutten, NJ Gordon, S Maskell, Efficient Particle based track before detect in Rayleigh noise, in *Proceedings of SPIE, Signal and Data Processing of Small Targets, vol. 5428* (Orlando, FL, 2004), pp. 509–519
7. MG Rutten, B Ristic, NJ Gredon, A comparison of particle filters for recursive track-before-detect, in *Proceedings of the 8th International Conference on Information Fusion*. vol. 1 (Piscataway, 2005), pp. 169–175. doi:10.1109/ICIF.2005.1591851
8. MG Rutten, NJ Gordon, S Maskell, Recursive track-before-detect with target amplitude fluctuations. IEE Proc Radar Sonar Navigat **152**, 345–352 (2005). doi:10.1049/ip-rsn:20045041
9. Y Boers, JN Driessen, Multitarget particle filter track before detect application. IEE Proc Radar Sonar Navigat **151**, 351–357 (2004). doi:10.1049/ip-rsn:20040841
10. The resource for CUDA developers (2010), 2010. http://www.nvidia.com/object/cuda_home.html
11. Z Shu, C Yanli, *GPU Computing for High Performance-CUDA* (Beijing, China, 2009)
12. M Bolic, PM Djuric, S Hong, Resampling algorithms and architectures for distributed particle filters. IEEE Trans Signal Process **53**(7), 2442–2450 (2005)
13. M Bolic, A Athalye, S Hong, PM Djuric, Study of algorithmic and architectural characteristics of Gaussian particle filters. J Signal Process Syst **61**, 205–218 (2009)
14. C Lenz, G Panin, A Knoll, A GPU-accelerated particle filter with pixel-level likelihood, in *International Workshop on Vision, Modeling and Visualization (VMV)* (Konstanz, Germany, 2008), pp. 235–241
15. G Hendeby, J Hol, R Karlsson, F Gustafsson, A graphics processing unit implementation of the particle filter, in *Proceedings of the 15th European Statistical Signal Processing* (Poznan, Poland, 2007), pp. 1639–1643
16. L Peihua, An efficient particle filter–based tracking method using graphics processing unit (GPU). J Signal Process Syst **68**, 317–332 (2012)
17. T Xu, S Jinzhou, Z Fangbin, Particle filter track-before-detect implementation on GPU, in *Proceedings of the International Conference on Communications, Signal Processing, and Systems (CSPS)* (Beijing, China, 2012), pp. 16–18
18. NVIDIA, *CUDA (Compute Unified Device Architecture) C programming guide 4.1*, 2011. https://developer.nvidia.com/cuda-downloads
19. J Torstensson, M Trieb, *Particle Filtering for Track Before Detect Applications* (University of Linkoping, Sweden, 2005)

doi:10.1186/1687-1499-2013-38
Cite this article as: Tang *et al.*: **Particle filter track-before-detect implementation on GPU.** *EURASIP Journal on Wireless Communications and Networking* 2013 **2013**:38.