Accelerating the Single Cluster PHD Filter with a GPU Implementation

Chee Sing Lee, José Franco, Jérémie Houssineau, Daniel Clark

***Abstract*—The SC-PHD filter is an algorithm which was designed to solve a class of multiple object estimation problems where it is necessary to estimate the state of a single-target parent process, in addition to estimating the state of a multi-object population which is conditioned on it. The filtering process usually employs a number of particles to represent the parent process, coupled each with a conditional PHD filter, which is computationally burdensome. In this article, an implementation is described which exploits the parallel nature of the filter to obtain considerable speed-up with the help of a GPU. Several considerations need to be taken into account to make efficient use of the GPU, and these are also described here.**

I. Introduction

Recent developments in multiple target tracking algorithms have spurred the creation of methods which use them as building blocks to solve more complex problems. For instance, filters based on the finite set statistics (FISST) framework [1] such as the PHD Filter [2] have been extended to estimate cluster processes [3], perform distributed sensor localization [4], parameter estimation [5], and simultaneous localization and mapping [6], [7], among other applications.

Augmenting the sophistication of the tracking systems, however, also tends to lead to a sharp increase in computational burden. However, interesting parallelization opportunities can often be exploited to reduce the overall running times of the algorithm. In particular, the Single Cluster PHD (SC-PHD) filter is suited for tracking problems where the observed targets' motion and evolution is dependent on some unobserved parent state. The use of this filter has been demonstrated in group target tracking [8], microscope drift estimation [9] and simultaneous localization and mapping [7].

Increasing the speed of inherently parallel programs can be achieved through the use of specialized processors, sometimes called accelerators. Graphics Processing Units (GPU) are an interesting type of accelerator since they specialize in carrying out massively parallel tasks. With increased computational needs spurred by more complex graphics intensive applications, the processing power of GPUs has quickly increased through the years, and there has been a surge of interest in exposing these capabilities for use in a wider range of applications. The CUDA framework is a recent effort by Nvidia corporation to simplify access to these capabilities in Nvidia graphics cards for general purpose computation.

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) Grant number EP/K014277/1 and the MOD University Defence Research Collaboration in Signal Processing.

The authors are with the School of Engineering & Physical Sciences, Heriot-Watt University, Edinburgh (e-mail: cslee@eia.udg.edu, {jf139,jh207,D.E.Clark}@hw.ac.uk)

The purpose of this article is to provide details on an implementation of the SC-PHD Filter on the CUDA framework. In Section II, the SC-PHD Filter is described. Section III talks about the particularities of the CUDA Framework and what considerations must be taken when programming for it, while Section IV details the implementation of this filter on this architecture. Sample results are shown in Section V, and finally the article concludes in Section VI.

II. The SC-PHD Filter

The PHD Filter is a multi-object estimation filter based on Finite Set Statistics (FISST). In this framework, the multi-object and measurement spaces are represented as random finite sets (RFS), which implies uncertainty both in the states of each of the targets being tracked and in the size of the population. The filter propagates the approximation related to the first moment, or intensity, of the multi-target Bayes filter, which is called the Probability Hypothesis Density (PHD). This is a function defined over the single-target state space whose integral over a particular region gives the expected number of targets in that region, and thus gives information not only on the number of objects in the population but also of their location.

An interesting class of multiple-target tracking problems deals with tracking a population whose state is conditional on a single-target random variable. This may be due to, for example, the state of the sensor that observes the population not being accurately known. The Single Cluster PHD Filter is an extension of the PHD Filter which is designed to estimate the state not only of the multiple object population through a conditional PHD $\tilde{D}_k(\mathbf{y}|\mathbf{x})$, but also of the parent process on which it is conditioned. The prediction equation for the SC-PHD filter is given by

$$D_{k|k-1}(\mathbf{x},\mathbf{y}) = \int s_{k-1}(\mathbf{x}')\pi_{k|k-1}(\mathbf{x}|\mathbf{x}')\tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x}')\,\mathrm{d}\mathbf{x}', \tag{1}$$

where s_{k-1} is the prior distribution of the parent state at time $k-1$, where $\pi_{k|k-1}$ is the Markov transition density of the parent process, and where $\tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x})$ is the predicted PHD of the daughter process:

$$\begin{aligned}\tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x}) &= \gamma_{k|k-1}(\mathbf{y}|\mathbf{x}) \\ &\quad + \int \tilde{D}_{k-1}(\mathbf{y}'|\mathbf{x})p_S(\mathbf{y}'|\mathbf{x})\tilde{\pi}_{k|k-1}(\mathbf{y}|\mathbf{y}',\mathbf{x})\ \mathrm{d}\mathbf{y}'\end{aligned}$$

with the following definitions:

$\gamma_{k|k-1}(\cdot|\mathbf{x})$ PHD for daughter birth process at time k, conditioned on parent state $\mathbf{x}$

$\tilde{D}_{k-1}(\cdot|\mathbf{x})$ prior PHD of the daughter state at time $k-1$, conditioned on parent state $\mathbf{x}$

$p_S(\cdot|\mathbf{x})$ object survival probability, conditioned on $\mathbf{x}$

$\tilde{\pi}_{k|k-1}(\cdot|\cdot,\mathbf{x})$ single-object Markov transition density for the daughter process, conditioned on $\mathbf{x}$.

The update equation, in turn, is given by

$$D_{k|k}(\mathbf{x},\mathbf{y}) = \frac{s_{k|k-1}(\mathbf{x})L_{\mathbf{Z}_k}(\mathbf{x})}{\int s_{k|k-1}(\mathbf{x}')L_{\mathbf{Z}_k}(\mathbf{x}')\ \mathrm{d}\mathbf{x}'}\tilde{D}_{k|k}(\mathbf{y}|\mathbf{x}),$$

where $s_{k|k-1}$ is the predicted PHD of the parent state, where $p_D(\cdot|\mathbf{x})$ is detection probability of the daughter process given a parent state $\mathbf{x}$, and where $L_{\mathbf{Z}_k}(\mathbf{x})$ the multi-object observation likelihood of the measurement set $\mathbf{Z}_k$ given a parent state $\mathbf{x}$, defined by

$$L_{\mathbf{Z}_k}(\mathbf{x}) = \exp\left(-\int p_D(\mathbf{y}|\mathbf{x})\tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x})\ \mathrm{d}\mathbf{y}\right)\prod_{z\in\mathbf{Z}_k}\eta_z(\mathbf{x}),$$

with

$$\eta_z(\mathbf{x}) = \kappa_k(z) + \int \tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x})p_D(\mathbf{y}|\mathbf{x})g(z|\mathbf{y},\mathbf{x})\ \mathrm{d}\mathbf{y},$$

where $g(\cdot|\cdot,\mathbf{x})$ is the single-object observation likelihood given the parent state $\mathbf{x}$ and where κ_k is the PHD of the measurement clutter process. The updated PHD $\tilde{D}_{k|k}(\cdot|\mathbf{x})$ of the daughter process is found to be

$$\tilde{D}_{k|k}(\mathbf{y}|\mathbf{x}) = \tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x})\cdot \left[(1-p_D(\mathbf{y}|\mathbf{x})) + \sum_{z\in\mathbf{Z}_k}\frac{g(z|\mathbf{y},\mathbf{x})p_D(\mathbf{y}|\mathbf{x})}{\eta_z(\mathbf{y}|\mathbf{x})}\right].$$

Practical implementations of either the PHD or SC-PHD filter require an appropriate representation for the intensity. In this article, the focus will be on accelerating a filter equipped with a particle representation for the distribution s_k of the parent process and a Gaussian mixture representation for the PHD of the daughter state. To reflect the conditional relationship between the parent and daughter processes, each particle in the parent distribution is associated with its own Gaussian Mixture (GM) representing the daughter PHD conditioned on that particle's trajectory. Then at any given timestep k, the parent and daughter processes can be stated as

$$s_k(\mathbf{x}) = \sum_{i=1}^{N_k}\zeta_k^{(i)}\delta(\mathbf{x}-\mathbf{x}_k^{(i)}) \tag{2}$$

$$D_k^{(i)}(\mathbf{y}|\mathbf{x}_k) = \sum_{j=1}^{J_k^{(i)}} w_k^{(i,j)}\mathcal{N}(\mathbf{y},\mu_k^{(i,j)},\mathbf{P}_k^{(i,j)}). \tag{3}$$

Here, N_k is the number of particles in the representation of the parent state, each with state $\mathbf{x}_k^{(i)}$ and weight $\zeta_k^{(i)}$, $J_k^{(i)}$ is the current number of components in the daughter Gaussian mixture, each component having weight $w_k^{(i,j)}$, mean $\mu_k^{(i,j)}$ and covariance $\mathbf{P}_k^{(i,j)}$. This representation is particularly convenient if the motion and measurement models are linear and Gaussian, or can be reasonably linearized as such, since it simplifies the obtention of the predicted and updated forms for the Gaussian mixture. To obtain the equation for prediction, equations (2) and (3) are substituted into the SC-PHD prediction equation (1), yielding

$$D_{k|k-1}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N_{k-1}}\zeta_{k-1}^{(i)}\pi_{k|k-1}(\mathbf{x}|\mathbf{x}_{k-1}^{(i)})\tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x}_{k-1}^{(i)}).$$

Next, the Markov transition density of the parent is approximated by drawing M samples from it:

$$\{\mathbf{x}^{(i,1)},\cdots,\mathbf{x}^{(i,M)}\} \sim \pi_{k|k-1}(\mathbf{x}|\mathbf{x}_{k-1}^{(i)})$$

$$D_{k|k-1}(\mathbf{x},\mathbf{y}) \approx \sum_{i=1}^{N_{k-1}}\sum_{j=1}^{M}\frac{\zeta^{(i)}}{M}\delta(\mathbf{x}-\mathbf{x}^{(i,j)})\tilde{D}_{k|k-1}(\mathbf{y}|\mathbf{x}_{k-1}^{(i)})$$

Measurement-driven births are used, as described in [10], so the incorporation of new targets into the daughter PHD is delayed until the update step. Therefore, the predicted daughter PHD describes only persisting daughter objects propagated forward in time:

$$\tilde{D}_{S,k|k-1}(\mathbf{y}|\mathbf{x}_{k-1}^{(i)}) = p_S(\mathbf{y}|\mathbf{x}_{k-1}^{(i)})\sum_{j=1}^{J_{S,k|k-1}} w_{S,k|k-1}^{(j)}\mathcal{N}\big(\mathbf{y},\mu_{S,k|k-1}^{(j)},\mathbf{P}_{S,k|k-1}^{(j)}\big|\mathbf{x}_{k-1}^{(i)}\big)$$

$$\begin{aligned} w_{S,k|k-1}^{(j)} &= w_{k-1}^{(j)} \\ \mu_{S,k|k-1}^{(j)} &= f(\mu_{k-1}^{(j)}) \\ \mathbf{P}_{S,k|k-1}^{(j)} &= \mathbf{F}_k\mathbf{P}_{k-1}^{(j)}\mathbf{F}_k^T + \mathbf{W}_k\mathbf{Q}_k\mathbf{W}_k^T \end{aligned}$$

where $f(\mu)$ is the (possibly non-linear) motion model describing the motion of daughter objects and $\mathbf{F}$ and $\mathbf{W}$ are the Jacobians of the motion model with respect to the daughter feature and process noise respectively. The predicted joint PHD would now consist of $N_{k|k-1} = M \times N_{k-1}$ particles corresponding to the parent state, and the same number of Gaussian mixtures representing the conditional daughter PHDs. This means the number of particles would grow geometrically with every prediction, but when the particles are resampled, only a number of samples equal to the number of original particles is drawn, to make it feasible to iterate for as many time steps as necessary.

For the update step, each conditional daughter PHD is individually updated with the received measurements:

$$\begin{aligned} \tilde{D}_{k|k}(\mathbf{y}|\mathbf{x}) = (1-p_D(\mathbf{y}|\mathbf{x}))&\sum_{j=1}^{J_{k|k-1}} w_{k|k-1}^{(j)}\mathcal{N}\big(\mathbf{y};\mu_{k|k-1}^{(j)},P_{k|k-1}^{(j)}\big) \\ &+ \sum_{z\in\mathbf{Z}_k}\sum_{j=1}^{J_{k|k-1}} w_{k|k}^{(j)}\mathcal{N}\big(\mathbf{y};\mu_{k|k}^{(j)}(z),\mathbf{P}_{k|k}^{(j)}(z)\big) \\ &+ \sum_{z\in\mathbf{Z}_k}\frac{w_0^\gamma}{\eta_z(\mathbf{y}|\mathbf{x})}\mathcal{N}\big(\mathbf{y};z^*(\mathbf{x}),\mathbf{R}^*\big), \end{aligned}$$

where

$$\eta_z(\mathbf{y}|\mathbf{x}) = \kappa(z) + p_D(\cdot|\mathbf{x}) \sum_{l=1}^{J_{k|k-1}} g(z|\mathbf{y},\mathbf{x}) w_{k|k-1}^{(j)} + w_0^{\gamma}$$
$$w_{k|k}^{(j)} = w_{k|k-1}^{(j)} p_D(\cdot|\mathbf{x}) g(z|\mathbf{y},\mathbf{x}) w_{k|k-1}^{(j)} \eta_z(\mathbf{y}|\mathbf{x})^{-1}$$
$$\mu_{k|k}^{(j)}(z) = \mu_{k|k-1}^{(j)} + \mathbf{K}(z - h(\mu_{k|k-1}^{(j)}, \mathbf{x}))$$
$$\mathbf{P}_{k|k}^{(j)}(z) = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}^{(j)}$$
$$\mathbf{K}_k = \mathbf{P}\mathbf{H}_k^T \mathbf{S}_k^{-1}$$
$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k.$$

With measurement driven births, the terms $z^*(\mathbf{x})$ and $\mathbf{R}^*$ are the mean and covariance of the birth term generated from measurement z. The term $z^*(\mathbf{x}) = h^{-1}(z, \mathbf{x})$ is chosen such that $h(z^*(\mathbf{x}), \mathbf{x}) = z$, while $\mathbf{R}^*$ can be chosen to be $\mathbf{R}^*_{(\cdot)}(\mathbf{x}) = \mathbf{J}^* \mathbf{R}_{(\cdot)} \mathbf{J}^{*T}$ if the Jacobian $\mathbf{J}$ of the inverse measurement function h^{-1} is known. After updating the daughter Gaussian Mixtures, the weights of the parent particles are updated using their multi-object likelihoods $L_{\mathbf{Z}_k}(\mathbf{x}_{k|k-1}^{(i)})$:

$$\zeta_k^{(i)} = \frac{L_{\mathbf{Z}_k}(\mathbf{x}_{k|k-1}^{(i)}) \zeta_{k|k-1}^{(i)}}{\sum_{l=0}^{N_{k|k-1}} L_{\mathbf{Z}_k}(\mathbf{x}_{k|k-1}^{(l)}) \zeta_{k|k-1}^{(l)}},$$

$$L_{\mathbf{Z}_k}(\mathbf{x}_{k|k-1}^{(i)}) = \exp\left(\sum_{j=0}^{J_{k|k-1}^{(i)}} w_{k|k-1}^{(j)} \right) \prod_{z \in \mathbf{Z}_k} \eta_z(\mathbf{y}_{k|k-1}^{(i)} | \mathbf{x}_{k|k-1}^{(i)}).$$

III. The CUDA Programming Model

Effective parallel programming with CUDA requires an understanding of the capabilities and limitations of the GPU as a computational platform. A CUDA application will generally contain portions which are executed in parallel on the GPU (device code), and portions which are executed serially by the CPU (host code), with program flow alternating between host code and device code. Device code is executed in parallel by a *grid* of hundreds or thousands of threads on the GPU, which is subdivided into thread *blocks*. Each thread within the grid has access to index values denoting the particular thread block in which it is located, and its location within that block. These indices are used to differentiate behavior among threads, for example, accessing different portions of the input and output data arrays. Every thread receives the same set of instructions in the form of a special C function called a *kernel*. In practice, CUDA kernels are not written to execute an entire algorithm on the GPU, but rather to perform smaller, computationally demanding portions of the algorithm that have been previously singled out through profiling. Ideal candidates for GPU parallelization are procedures in which very large arrays are processed, with every array element being operated upon with an identical set of computationally expensive instructions. In other words, rather than looping serially through an array with a `for` or `while` loop, a GPU kernel would execute all iterations of the loop in parallel, with a single thread handling each iteration.

The most important factor affecting the performance of a CUDA application is memory bandwidth. Prior to execution of a kernel, the input data must be transferred from the host to the device, and the output data must be transferred from the device back to the host following kernel execution. The time to perform these operations is very long relative to the time to execute computational tasks, so one must ensure that the computational gains from parallization outweigh the penalties incurred by memory operations. An effective CUDA application will strike a balance between processing small data arrays in serial on the CPU and processing large arrays in parallel on the GPU. There are many more design considerations for optimizing the performance of a CUDA kernel, which are outlined in the CUDA toolkit documentation [11].

IV. Parallel Implementation

The Gaussian mixture SC-PHD filter propagates the parent state as a collection of particles, with each particle being linked to a daughter Gaussian mixture PHD conditioned on the state and trajectory of that particle. These pseudocode listings will make use of the following abbreviations:

$$D_k(\mathbf{x}, \mathbf{y}) \doteq \{\mathbf{x}^{(i)}, \omega^{(i)}, \tilde{D}_k^{(i)}(\mathbf{y}|\mathbf{x})\}_{i=1}^{N_k}$$
$$\tilde{D}_k^{(i)}(\mathbf{y}|\mathbf{x}) \doteq \{\mu^{(j)}, \mathbf{P}^{(j)}, w^{(j)}\}_{j=1}^{J^{(i)}}$$

Each of the subroutines listed here accepts as input and returns as output a single parent particle and its corresponding PHD. Serial procedures are marked as **Functions** and parallel procedures meant to be ran on the GPU are denoted **Kernels**. For numerical stability, it is recommended that weights and likelihoods be replaced with their log-equivalents, and that the corresponding computations be modified accordingly.

The source code for this GPU implementation of the SC-PHD filter is freely available under the Apache 2.0 license at the following address:

https://github.com/cheesinglee/cuda-PHDSLAM

The top-level filter iteration should look familiar to those experienced with other Bayesian filter methods such as the EKF. The update step is split into two subroutines: `PreUpdate` computes auxilliary terms which are needed for the computation of the SC-PHD update and `Update` implements the actual update equations. In this work the `PruneAndMerge` subroutine was implemented as described in [12], but an alternative Gaussian mixture reduction algorithm can be substituted at the user's discretion. The same may be said about the `Resample` subroutine for resampling the parent particles.

In the GPU implementation, the joint PHD consists of one array of particle states representing the parent, and another array of Gaussians representing all conditional daughter PHDs concatenated together. Typically, the parent state consists of several hundred particles, each represented by a vector of less than 10 dimensions. The daughter array is several orders of magnitued larger. Each daughter PHD consists of several hundred Gaussian terms, each comprised of a scalar weight, mean vector, and covariance matrix. In general, operations

Function SC-PHD($D_{k-1}, \mathbf{Z}_k$)

Input: Prior joint state estimate, current measurements
Output: Posterior joint state estimate, measurements
for $i = 1 \dots N_{k-1}$ **do**
  $D^{(i)}_{k|k-1}$ =`Predict` $(D^{(i)}_{k-1}(\mathbf{x}, \mathbf{y}))$
  $\hat{D}^{(i)}_{k|k-1}$ =`PreUpdate` $(D^{(i)}_{k|k-1}(\mathbf{x}, \mathbf{y}), \mathbf{Z}_k)$
  $[\tilde{D}^{(i)}_{k|k}, L_{\mathbf{Z}_k}(\mathbf{X})]$ = `Update` $(\tilde{D}^{(i)}_{k|k-1}, \hat{D}^{(i)}_{k|k-1}, \mathbf{Z}_k)$
  $\omega^{(i)} = \omega^{(i)} \times L_{\mathbf{Z}_k}(\mathbf{X})$
  $\tilde{D}^{(i)}_{k|k}$ = `PruneAndMerge` $(\tilde{D}^{(i)}_{k|k})$ `// [12, Table II]`
end
`// Particle resampling according to [13, Algorithm 2]`
$\mathbf{x}^{(1\dots N_k)}$ = `Resample` $(\mathbf{x}^{(1\dots N_k)}, \omega^{(1\dots N_k)}_k)$
$\omega^{(1\dots N_k)} = 1/K$
return $D_k = \{\mathbf{x}^{(i)}, \omega^{(i)}, \tilde{D}^{(i)}_k(\mathbf{y}|\mathbf{x})\}_{i=1}^{N_k}$

Function Predict(D_{k-1})

Input: Prior parent state particle and conditional daughter PHD
Output: Predicted sensor state and object PHD
`// Sample sensor transition density`
for $n = 1 \dots M$ **do**
  $\mathbf{x}^{(n)}_{k|k-1} \sim \pi(\tilde{\mathbf{x}}|\mathbf{x})$
  $\omega^{(n)}_{k|k-1} \leftarrow \omega$
  `// Predict daughter PHD`
  forall the $\{\mu^{(j)}_{k-1}, \mathbf{P}^{(j)}_{k-1}, w^{(j)}_{k-1}\}$ *in* $\tilde{D}_{k-1}$ **do**
    $w^{(j)}_{k|k-1} = w^{(j)}_{k-1}$
    $\mu^{(j)}_{k|k-1} = f(\mu^{(j)}_{k-1})$
    $\mathbf{P}^{(j)}_{k|k-1} = \mathbf{F}^{(j)}\mathbf{P}^{(j)}_{k-1}\mathbf{F}^{(j),T} + \mathbf{W}^{(j)}\mathbf{Q}^{(j)}\mathbf{W}^{(j),T}$
  end
  $\tilde{D}^{(n)}_{k|k-1} = \{\mu^{(j)}_{k|k-1}, \mathbf{P}^{(j)}_{k|k-1}\}_{j=1}^{J}$
end
return $D_{k|k-1} = \{\mathbf{x}^{(n)}_{k|k-1}, \omega^{(n)}_{k|k-1}, \tilde{D}^{(n)}_{k|k-1}\}_{n=1}^{M}$

Function CUDAPredict(priorParent, priorDaughter)

```
predictedParent = hostarray(sizeOf(priorParent))
for i = 0, ..., nParticles do
    processNoise = makeNoise()
    predictedParent[i] =
    computeParentMotion(priorParent[i],processNoise)
end
predictedDaughter = devicearray(sizeOf(priorDaughter))
noiseArray = makeNoise()
daughterPredictKernel
(priorDaughter,predictedDaughter,noiseArray)
```

involving the daughter PHD array will be carried out on the GPU, while operations involving only the parent array will be carried serially with the CPU, as the parent array is too small to overcome the memory performance bottleneck. By parallelizing the filter, the `for` loop in the above pseudocode is eliminated.

A. Thread grid dimensioning and indexing

The threads related to the daughter Gaussian mixture of a single particle need to share information in order to perform the update, so we assign one thread block per parent particle. This raises two design challenges which must be dealt with: 1) thread blocks must be uniform in size, but the different conditional daughter PHDs will differ in size, and, 2) the number of Gaussian terms in a single conditional daughter may exceed the maximum allowable thread block size. To address both issues, an array is passed to the GPU where each element indicates the number of terms in the corresponding GM. A prefix sum of this array yields an indexing offset pointing to the beginning of the appropriate Gaussian mixture in the concatenated array. Moreover, knowing the number of Gaussians permits the construction of a `for` loop which processes the input data `blockDim` threads at a time.

B. Prediction

The prediction stage involves propagating both the parent distribution and daughter PHDs forward in time. The parent prediction is performed serially while the daughter PHD is predicted in parallel on the GPU. Implementing a per-thread random number generator would be prohibitively complicated, so the random noise samples are generated by the host, and passed to the kernel as input arguments.

C. Pre-Update

In the `PreUpdate` subroutine, single-object measurement likelihoods for each measurement and each term within the daughter Gaussian mixture are computed. The Gaussian mixture terms are also updated with the measurements via a Kalman update.

Let $nPredict$ be the total number of terms in the concatenated predicted daughter PHD, and $nMeasure$ be the number of received measurements. The preupdate will compute $nPreupdate = nPredict \times nMeasure$ single-object likelihoods and updated Gaussians.

D. Update

The remainder of the SC-PHD update is performed by the `Update` subroutine. Here, the predicted and preupdated daugther PHD terms, along with measurement-driven birth terms are concatenated and reweighted to form the final updated PHD.

The updated daughter PHD will be the preupdate array concatenated with $nPredict$ terms representing missed detection terms, and $nMeasure \times nParticles$ terms representing measurement-driven birth. Within each thread block, shared memory is used to compute weight normalization terms and the multi-object likelihood value to re-weight the parent particle.

V. Results

To analyze the performance gains that can be obtained using the GPU implementation of the filter, a simple tracking scenario was generated consisting on targets moving in a plane according to Brownian motion. The sensor that observes this population also moves according to Brownian motion, and thus

Kernel daughterPredictKernel(prior,noise,predictedArray)

for $j = 0$,*nThreads,2nThreads,...,nFeatures* **do**
  idx = tid + j
  predictedArray[idx] =
  `computeMotion` (prior[idx],noise[idx])
end

Function PreUpdate($D_{k|k-1}, \mathbf{Z}_k$)

Input: Predicted sensor particle and multi-object PHD, measurements
Output: Pre-update terms for SC-PHD daughter update
forall the $\{\mu^{(j)}, \mathbf{P}^{(j)}, w^{(j)}\}$ *in* $D_{k|k-1}$ **do**
  $\hat{z} = h^{-1}(\mathbf{X}, \mu^{(j)})$ `// Predicted measurement`
  $\mathbf{S} = \mathbf{H}\mathbf{P}^{(j)}\mathbf{H}^T + \mathbf{R}$ `// Innovation covariance`
  $\mathbf{K} = \mathbf{P}^{(j)}\mathbf{H}^T\mathbf{S}^{-1}$ `// Kalman Gain`
  $\mathbf{P}^{(j|\cdot)} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}$ `// Updated covariance`
  for $i = 1 \dots |Z|$ **do**
    `// single-object likelihood`
    $p(z^{(i)}|\mu^{(j)}, y) = \mathcal{N}\big(z^{(i)}; \hat{z}, \mathbf{S}\big)$
    $\mu^{(j|i)} = \mu + \mathbf{K}(z - \hat{z})$ `// Updated mean`
  end
end
$\hat{D}_{o|s} = \{\mu^{(j|i)}, \mathbf{P}^{(j|i)}\}_{\substack{j=1\dots J \\ i=1\dots|Z|}}$
return $p(\cdot|\cdot, \mathbf{x}), \hat{D}_{k|k-1}$

the acquired measurements exhibit random drift, which also needs to be estimated. Let the parent state be defined by a bias vector $\mathbf{X} = [x, y]$. Given noise parameters σ_x, σ_y for the parent motion model, the parent particles are predicted as follows;

$$\mathbf{X}_{k|k-1} = \mathbf{X}_{k-1} + \mathbf{v} \qquad \mathbf{v} \sim \mathcal{N}\big(\mathbf{X}; \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}\sigma_x^2 & 0\\ 0 & \sigma_y^2\end{smallmatrix}\right]\big)$$

Concurrently with the parent particles, the terms in the daughter Gaussian mixtures are predicted using a Brownian motion model with noise parameters σ_r, σ_s. The predicted mean and covariance are computed as follows:

$$\mu_{k|k-1} = \mu_{k-1} \qquad \mathbf{P}_{k|k-1} = P_{k|k-1} + \left[\begin{smallmatrix}\sigma_r^2 & 0\\ 0 & \sigma_s^2\end{smallmatrix}\right]$$

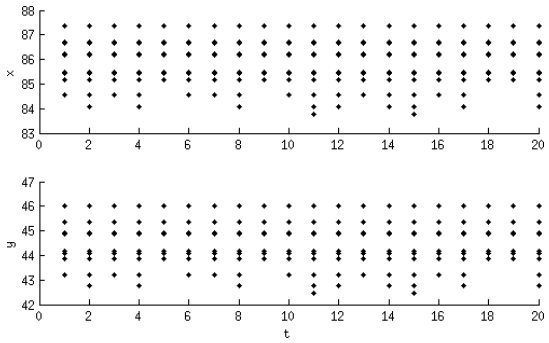

Fig. 1. Simulated measurements, with x versus time above and y versus time below.

Kernel preupdateKernel(predictedDaughterArray, predictedParentArray,measurements,offsets,likelihoods, preupdateDaughter)

parent = predictedParentArray[blockIdx.x] ; predicted = predictedDaughterArray[threadIdx.x]
z = measurements[threadIdx.y]
preupdate.weight = computeSingleObjectLikelihood(z,predicted,parent)
preupdate.mean,preupdate.cov = KalmanUpdate(z,predicted,parent) ;
preupdateDaughter[threadIdx.x,threadIdx.y] = preupdate

Function Update($D_{k|k-1}, \hat{D}_{k|k-1}, p_{\mathbf{z}|\mu,\mathbf{x}}$)

Input: Predicted multi-object PHD and pre-update terms
Output: Updated multi-object PHD
forall the $\mu^{(j)}, \mathbf{P}^{(j)}, w^{(j)}$ *in* $D_{k|k-1}$ **do**
  `// Non-detection terms`
  $\mu_{nd}^{(j)} = \mu^{(j)}; \quad \mathbf{P}_{nd}^{(j)} = \mathbf{P}^{(j)}; \quad w_{nd} = w^{(j)}$
  `// Detection terms`
  for $i = 1 \dots |Z|$ **do**
    $\mu_d^{(j|i)} = \mu^{(j|i)}; \quad \mathbf{P}_d^{(j|i)} = \mathbf{P}^{(j|i)};$ `// from` $\hat{D}_{k|k-1}$
    $w_d^{(j|i)} = \tilde{w}^{(j)} p_D p_{z|s,o}(z_i|\tilde{\mu}^{(j)}, y)$
  end
  `// Measurement-derived birth terms`
  for $i = 1 \dots |\mathbf{Z}_k|$ **do**
    $\mu_0^{(i)} = h^{-1}(y, z_i); \quad \mathbf{P}_0^{(i)} = \mathbf{R}^*; \quad w_0^{(i)} = w_0$
  end
  `// Normalize weights`
  `// compute multi-object likelihood`
  $\tilde{N} = \sum_{j=1}^{J} \tilde{w}^{(j)}$
  $L_{\mathbf{Z}_k}(\mathbf{X}) = \exp(\tilde{N})$
  for $i = 1 \dots |Z|$ **do**
    $\eta_{z_i} = \kappa(z_i) + \sum_{j=1}^{J} w_d^{(j|i)} + 2w_0$
    for $j = 1 \dots J$ **do**
      $w_d^{(j|i)} /= \eta_{z_i}$
    end
    $w_0^{(i)} /= \eta_{z_i}$
    $L_{\mathbf{Z}_k}(\mathbf{X}) = L_{\mathbf{Z}_k}(\mathbf{X}) \times \eta_{z_i}$
  end
end
`// Concatenate terms`
$\mu_{k|k} = [\mu_{nd}^{(1\dots J)}, \mu_d^{(1\dots J_{k|k-1}|1\dots|\mathbf{Z}_k|)}, \mu_0^{(1\dots|\mathbf{Z}_k|)}]$
$\mathbf{P}_{k|k} = [\mathbf{P}_{nd}^{(1\dots J)}, \mathbf{P}_d^{(1\dots J_{k|k-1}|1\dots|\mathbf{Z}_k|)}, \mathbf{P}_0^{(1\dots|\mathbf{Z}_k|)}]$
$w_{k|k} = [w_{nd}^{(1\dots J)}, w_d^{(1\dots J_{k|k-1}+1\dots|\mathbf{Z}_k|)}, w_0^{(1\dots|\mathbf{Z}_k|)}]$
return $\{\mu_{k|k}, \mathbf{P}_{k|k}, w_{k|k}, L_{\mathbf{Z}_k}(\mathbf{X})\}$

The measurement model is simply an identity function on the daughter state, biased by the parent particle state.

$$\hat{z}(\mu|\mathbf{X}) = \mu_{k|k-1} + \mathbf{X}_{k|k-1}$$

The conditional single object measurement likelihood is computed by evaluating the following multi-variate Gaussian:

$$p(z|\mu, \mathbf{X}) = \mathcal{N}\big(z; \hat{z}, \mathbf{S}\big)$$
$$\mathbf{S} = P_{k|k-1} + \mathbf{R}$$

where $\mathbf{R}$ is the measurement noise covariance matrix. This model is similar to the simultaneous protein tracking and

Kernel PHDUpdate(predictedGaussians,offsets,measurements)

```
[idxIn,idxOut,nPredict,blockOffset] =
computeIndices(offsets,threadIdx,blockIdx)
predicted = predictedGaussians[idxIn]
i = idxOut - blockOffset
if i >nPredict then non-detection
    weight = predicted.weight
    updatedGaussians[idxOut] = predicted
else if nPredict <= i <(nPredict*(1 + nMeasurements)) then
detection
    m = floor((i - nPredict)/nPredict)
    updated = preUpdate(predicted,measurements[m]) weight =
    updated.weight updatedGaussians[idxOut] = updated
else if i >= (nPredict*(1 + nMeasurements)) then birth
    m = i - (nPredict*(1 + nMeasurements))
    birth = computeBirth(measurements[m])
end
sharedmem[threadIdx] = weight
normalizer = sum(sharedmem)
updatedGaussians[idxOut] /=normalizer
```

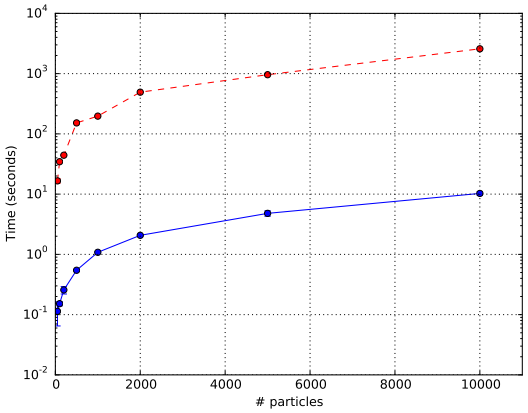

Fig. 2. Execution time of the GM-SCPHD filter. The dashed line indicates the running time for the serial implementation, and the solid the parallel one. The ordinate is the logarithm of the running time.

microscope drift estimation problem presented in [9].

For simulation inputs, a 20 time step scenario was generated for which 10 Monte Carlo runs were executed with varying parent particle counts: $N = \{50, 100, 200, 500, 1000, 2000, 5000, 10000\}$. The test data can be visualized in Figure 1. The timing results are shown in Figure 2. These results show that while the GPU implementation is significantly faster than the serial implementation, the execution time still increases proportionally to the number of parent particles. Further profiling revealed that the majority of the computation time occurs within the Gaussian mixture reduction subroutine. Because it is an inherently sequential operation, the GM reduction was left as a serial subroutine to be run on the CPU. As the CPU loops over each parent particle to reduce its corresponding Gaussian mixture, it is to be expected that time would increase with a higher number of particles.

VI. Conclusions

The SC-PHD filter is a useful tool when solving a wide class of estimation problems that consist on jointly tracking a parent single-target state and a daughter multi-object state that is conditioned on the parent. Although this filter has been widely shown to perform well when solving this problem, serial implementations are slow since they do not exploit the inherent parallelism offered by this filter. In this article, a parallel version of the filter was detailed, and an implementation exploiting the high-performance capabilities of GPUs was described. This implementation was shown to perform well on simulated data, providing considerable speed-up compared to the serial version of the same algorithm.

References

[1] R. P. Mahler, *Statistical multisource-multitarget information fusion*. Artech House ˆ eBoston Boston, 2007, vol. 685.

[2] ——, "Multitarget bayes filtering via first-order multitarget moments," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 39, no. 4, pp. 1152–1178, 2003.

[3] A. Swain and D. Clark, "Bayesian estimation of the intensity for independent cluster point processes: An analytic solution," *Procedia Environmental Sciences*, vol. 7, pp. 56–61, 2011.

[4] M. Uney, B. Mulgrew, and D. Clark, *Cooperative sensor localisation in distributed fusion networks by exploiting non-cooperative targets*. ICASSP, 2014.

[5] B. Ristic, D. E. Clark, and N. Gordon, "Calibration of multi-target tracking algorithms using non-cooperative targets," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 3, pp. 390–398, 2013.

[6] C. S. Lee, D. Clark, and J. Salvi, "Slam with single cluster phd filters," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 2096–2101.

[7] C. S. Lee, D. E. Clark, and J. Salvi, "Slam with dynamic targets via single-cluster phd filtering," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 3, pp. 543–552, 2013.

[8] A. Swain and D. E. Clark, "First-moment filters for spatial independent cluster processes," in *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2010, pp. 76 970I–76 970I.

[9] J. Franco, J. Houssineau, D. Clark, and C. Rickman, "Simultaneous tracking of multiple particles and sensor position estimation in fluorescence microscopy images," in *Control, Automation and Information Sciences (ICCAIS), 2013 International Conference on*. IEEE, 2013, pp. 122–127.

[10] J. Houssineau and D. Laneuville, "Phd filter with diffuse spatial prior on the birth process with applications to gm-phd filter," in *Information Fusion (FUSION), 2010 13th Conference on*. IEEE, 2010, pp. 1–8.

[11] Nvidia Corporation, "CUDA toolkit documentation," 2014. [Online]. Available: http://docs.nvidia.com/cuda/

[12] B.-N. Vo and W.-K. Ma, "The gaussian mixture probability hypothesis density filter," *Signal Processing, IEEE Transactions on*, vol. 54, no. 11, pp. 4091–4104, 2006.

[13] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 174–188, 2002.