GPU ACCELERATION OF THREAT MAP COMPUTATION AND APPLICATION TO SELECTION OF SONAR FIELD CONTROLS

Sergey Simakov, and Fiona Fletcher

Maritime Division, Defence Science and Technology Organisation, Edinburgh, Australia

ABSTRACT

The Threat Probability Density Map displays the outcomes of the search effort prior to detection and is a digital representation of the probability density function of location of the existing undetected threat. The Threat Map readily provides such diagnostics as the probabilities of the threat being present in different areas of interest and this information can be utilised in selection of sensor field controls. In this work we consider a GPU acceleration of the Monte-Carlo technique for Threat Map computation and discuss application to sonobuoys.

Index Terms— multistatic sonar, CUDA, GPU

1. INTRODUCTION

The Threat Probability Density Map, also known as the Threat Density Probability Map or simply a Threat Map, represents the combined effect of assumed target behavior and search history. Incze and Dasinger [1] considered the Threat Map in application to distributed multistatic sonar search in the Area Clearance and Area Denial tasks and demonstrated its utility for assessment of the state of the search effort. In the same work [1], they also discussed some potential use of the Threat Map in selection of sonar field[1] controls.

The computation of the Threat Map proposed in [1] employs the Monte-Carlo method combined with the Bayesian inference process. The appeal of this method is in the flexibility with which the threat behaviour and application of sonar fields can be handled. However, the Monte-Carlo approach is computationally demanding, so alternative techniques based on diffusion type modelling were also considered [2, 3, 4, 5].

In this work we revisit a variant of Monte-Carlo technique for Threat Map computation employed in [1], consider its acceleration using CUDA GPU and discuss application of such acceleration to concurrent selection of sonar field controls.

2. REPRESENTATIVE TASK AND DECISION SUPPORT TOOLS

We consider the following simplified version of the Area Clearance task discussed in [1].

A. Let A be a sizable domain in the ocean;

B. Suppose there is a sub-surface threat loitering in A;

C. Let A_0 be a sub-domain of A of some importance;

D. Consider the task of deploying a set of sonobuoys and maintaining their fields so as to keep the sub-domain A_0 clear of the undetected threat.

For simplicity we assume that both A and A_0 are square domains, the environment is range-independent, and the pattern in which the sonobuoys are to be deployed is fixed (see Fig. 1). In our example all sonobuoys are identical and each of them is assumed to be capable of acting as a source and a receiver. In order to accomplish the task (A to D) one has to select values for a range of parameters which define the fields of all possible bistatic and monostatic source-receiver pairs. We focus on the following field control parameters:

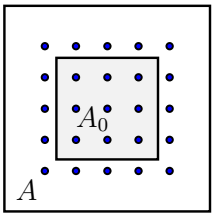

Fig. 1: Domain A, subdomain A_0, and the considered sonobuoy pattern.

- Separation distance ($\equiv$ distance between adjacent posts);
- Ping Plan ($\equiv$ sequencing of pinging sonobuoys);
- Ping Interval ($\equiv$ time between consecutive pings).

Before proceeding to selecting values for sonar field controls one has to give some consideration to how the quality of the selected values can be quantified and whether using separate criteria for selection of different field controls might be an option. The Coverage Area and the IPD map, which we briefly discuss in Section 2.1, are among more conventional decision support tools used in planning multistatic sonar fields [6].

2.1. IPD map and Coverage Area

The IPD map is a colour-coded display of the Instantaneous Probability of Detection (IPD) considered as a function of target location, and the Coverage Area is the area of the domain where the IPD exceeds a cut-off value set by the user. The IPD map and Coverage Area depend on the separation distance, so selection of the value for the separation distance can be based on maximising the Coverage area or designing the IPD map which meets the required criteria.

[1]The term *sonar field* is used here to refer to such ping-induced functions of threat location as Signal Excess (SE), Signal to Noise Ratio (SNR), or the Instantaneous Probability of Detection (IPD).

Decision support tools based only on the IPD map and the Coverage Area do not allow us to capture temporal effects, which limits their use in selection of the Ping Plan and Ping Interval. Furthermore, even when using these tools for selection of the Separation Distance, care must be exercised, especially in the scenarios exhibiting strong dependence on the bistatic target orientation and using calculations which involve the Bistatic Target Strength or Doppler-sensitive waveforms [1]. An example provided in Fig. 2 illustrates this.

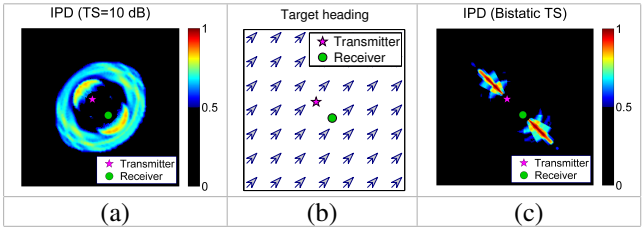

Fig. 2. IPD maps for constant and bistatic Target Strengths.

The scenario for which the IPD maps in Fig. 2 (a) and (c) were obtained is displayed in Fig. 2 (b): we consider a target, which can be anywhere in the considered area and is moving in the North-East direction. Signal Excess components were generated using a Gaussian ray bundle model [7]. The IPD map in Fig. 2 (a) was obtained using a fixed value TS_0 of the Target Strength, whereas the calculation of the IPD map in Fig. 2 (c) used a bistatic Target Strength. These two IPD maps are clearly very different, and this difference cannot be eliminated through a selection of TS_0.

3. THREAT MAP AND ITS COMPUTATION

Unlike the IPD map, the Threat Map considered in [1] can capture temporal effects and it readily handles aspect-dependent scenarios with one (as in Fig. 2) or more possible target headings at each target location. The Threat Map is a digital representation of the probability density function of location of the existing threat given it has not been detected. Integration of the Threat Map over the entire domain A (see Fig. 1) gives 1, while its integration over A_0 gives the probability $P_0(t)$ that the undetected target is present in A_0.

In this section we revisit the Monte-Carlo algorithm for computation of the Threat Map and discuss its acceleration using CUDA GPU computation.

3.1. Monte-Carlo method for Threat Map computation

Threat Map computation using a Monte-Carlo method based on Bayesian inference was proposed in [1]. Fig. 3 provides a schematic and some nomenclature used in the algorithm. Further comments are provided below:

- In this method we launch a large number of virtual targets or particles (represented as red dots in Fig. 3). These virtual targets are possible realisations of the actual target.

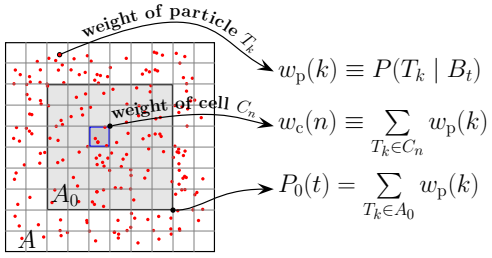

Fig. 3. Monte-Carlo method of Threat Map computation.

- Conditional probabilities $P(T_k \mid B_t)$ are used to assign weights $w_{\mathrm{p}}(k)$ to each of the particles ($k = 1, \ldots, K$), where B_t is the event that the target has not been detected by time t and T_k is the event that the actual target has been realised as virtual target k. It is assumed that $P(T_k) = 1/K$. Bayes theorem is used to transform the weights $w_{\mathrm{p}}(k)$ to the form

$$w_{\mathrm{p}}(k) = P(B_t \mid T_k) \Big/ \sum_{k=1}^{K} P(B_t \mid T_k) \qquad (1)$$

  and the multistatic IPD history which particle k has been exposed to during all pings before time t is used to compute $P(B_t \mid T_k)$.
- The entire area A is covered with a grid having a sufficiently small cell size. In Fig. 3 we use a linear index n ($n = 1, \ldots, N$) for cell enumeration.
- The weight of cell C_n is calculated as the total weight of all particles which are currently inside C_n. Note that the notation T_k is also used in Fig. 3 as a shorthand for virtual target k. Threat PDF in the cell is obtained as the ratio of cell weight to the cell area.
- Similarly, the probability of presence of the undetected target in A_0 is obtained by calculating the total weight of all particles inside A_0.
- In the process of Threat Map computation, particle weights $\{w_{\mathrm{p}}(k)\}_{k=1,\ldots,K}$ are updated upon every sonar transmission. Changes in cell weights $\{w_{\mathrm{c}}(n)\}_{n=1,\ldots,N}$ are driven by particle motion and changes in particle weights.

A pseudo-code for Threat Map computation is provided below.

Key variables:

- `X`, `Y`, `Vx`, `Vy`: particle coordinates and velocity components (length K vectors);
- `Wp` and `Ip`: particle weights and particle cell indexes (length K vectors);
- `Wc`: cell weights (length N vector).

Single step update:

A. Update `X`, `Y`, `Vx`, `Vy`

B. Use `X`,`Y` and grid parameters to update `Ip`

C. Update `Wp` if there is a transmission at this time-step

D. Compute `Wc` from {`Wp, Ip`} using the process:

```
for k=1:K
  Wc(Ip(k))=Wc(Ip(k))+Wp(k);
end
```

3.2. GPU computation of Threat Map

Each particle in Threat Map computation has to be considered separately, so, when the number of particles is large, the sequential processing is rather slow. In order to accelerate the computation we turned to parallel processing using CUDA GPU [8]. CUDA is a parallel computing platform and a programming model which allows one to program Graphics Processing Units (GPUs) using high-level languages such as C/C++, or FORTRAN [8, 9, 10]. CUDA C has a C-style syntax, and CUDA functions, called kernels, are called from a CPU (host) code, while their execution occurs on the GPU (device). The invoked kernel is executed by every launched thread. Typically a thread is designed to operate on a portion of data, and the assignment of the thread to the data subset is facilitated through the use of built-in variables `threadIdx`, `blockIdx`, `blockDim` and `gridDim`. Kernel invocation syntax allows such execution configuration parameters to be passed as the "number" of threads per block, "number" of blocks and the amount of shared memory allocated for each block. The hardware and software infrastructure required for development and execution of CUDA GPU applications is summarised in [11].

We developed our GPU application for Threat Map computation as a set of CUDA kernels callable from MATLAB® using its CUDA GPU interface accessible through the Parallel Computing Toolbox™ [12]. The key utilities in this interface are the `gpuArray` and `CUDAKernel` classes. Objects of the `gpuArray` class allow deployment of logical and fundamental numerical arrays onto GPU, while `CUDAKernel` objects provide access to functionality of kernels developed in CUDA and pre-compiled into the PTX form [13]. `CUDAKernel` objects are evaluated using method `feval` with execution configuration parameters passed implicitly as object properties.

We defined kernels for propagating particles (Step A of the pseudo-code in Section 3.1), updating their weights `Wp` (Step C of the pseudo-code), and updating cell weights `Wc` (Step D of the pseudo-code). Note that the process in Step D is a histogram type calculation, which in a multi-threaded case can result in a racing condition when two or more threads attempt to access and change data at the same address simultaneously. Such racing conditions are prevented through the use of atomic operations [8, 9].

We deploy the key variables listed in Section 3.1 as `gpuArray` data. A small subset of data is deployed as constant memory for a faster access. Our application uses `curand` library [14] to generate random numbers in the device code. To call `curand` library functions we have to define global device data of type `curandState`. Since such data type cannot be deployed using `gpuArray` objects, separate kernels have to be used to perform the associated initialisation and clean-up steps.

We also used `gpuArray` objects to deploy precomputed Signal Excess (SE) data components consisting of the Target Echo data obtained for a standard target with Target Strength of 0 dB and Reverberation Level data. Because the GPU memory is somewhat limited we have to be mindful about how much data is deployed on to the GPU. In the bistatic range-independent case considered in our application, the Signal Excess is a function of four variables, viz. range, azimuth, separation distance and depth. We reduced the dimensionality and hence the size of SE data that had to be deployed to GPU by taking advantage of the monostatic reduction for Target Echo computation and using the fact that the Reverberation Level in the considered range-independent environment is a nearly constant function of depth.

Implementing the components for Threat Map computation as CUDA kernels allowed us to achieve a significant improvement in the speed of execution. For example, tests run on our CPU/GPU configuration (Intel® Core™ i5/GeForce GT 740M) showed that IPD computation by a CUDA kernel is 30-40 times faster than by a well-vectorised MATLAB code expected to run at the same speed as its C equivalent.

3.3. Sample depletion and resampling

In the process of Threat Map computation we have to perform resampling in order to address an issue similar to what is known in particle filtering as *sample depletion* [15] or *degeneracy problem* [16] when very few particles have large weights, while weights of the rest of the particles become very small. In our case this results in an instability of the associated section of the plot of $P_0(t)$ due to reduction of the number of particles effectively used in computation to only those with high weights.

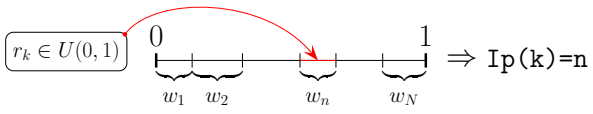

Fig. 4. Resampling procedure.

Resampling produces a set of K equally-weighted particles $\{\mathbf{x}_\mathrm{p}(k),\, w_\mathrm{p}(k) = 1/K\}$ spread over the cells of the Threat Map so as to represent a cell-wise constant PDF obtained from the cell weights $\{w_n \equiv w_\mathrm{c}(n)\}$. First we use the inversion method [17, Ch. 3] to determine the index `Ip(k)` of the cell (see Fig. 4), then we obtain $\mathbf{x}_\mathrm{p}(k)$ by sampling from a uniform distribution defined in the respective cell. The resampling has to be carried out before the degeneracy occurs – we can choose to resample every fixed number of pings or upon reaching a threshold set for the effective number of samples $\hat{N}_\mathrm{eff} \equiv 1/\sum_{k=1}^{K} w_\mathrm{p}^2(k)$ [16, Sec. 3.2].

3.4. Effects of kernel smoothing

The obtained Threat Map is a cell-wise constant function of target location. We can obtain a continuous form of the Threat Map using Kernel Smoothing [18]. If bandwidth parameters employed in this procedure are small in comparison with the size of A_0, then the difference between plots of $P_0(t)$ obtained using a cell-wise constant and smoothed Threat Map is small (see Fig. 5), although the Maps themselves may be visibly different (cf. Fig. 5 (a) and Fig. 5 (b)). The Kernel Smoothing in the computation summarised in Fig. 5 was carried out using KDE Toolbox developed by A. Ihler [19]. In this example, we used Gaussian kernels with the bandwidth parameter set to a value twice as large as the cell size. Kernel smoothing requires a significant computational effort. Fig. 5 shows that this step is not essential for computation of $P_0(t)$.

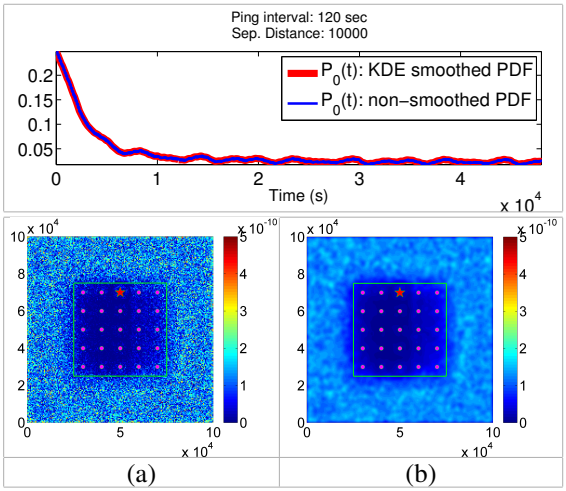

Fig. 5. Effects of kernel density smoothing on the probability of presence $P_0(t)$ and the Threat Map. Displays: (a) cell-wise constant Threat Map, and (b) smoothed Threat Map.

4. FIELD CONTROL SELECTION AND DYNAMIC PING SEQUENCING

The ability of applied sonar fields to produce the probability of presence $P_0(t)$ which settles down at a steady state value $\bar{P}_0$ depends on the considered scenario, the selected ping plan and the values used for the separation distance d and ping interval T_p. For example, if a target uniformly distributed in A is stationary and its target strength is isotropic, then the probability of undetected presence just after ping M is

$$P_0(t_M) = \int_{A_0} \eta_M(\mathbf{x}) dA / \int_A \eta_M(\mathbf{x}) dA$$

where $\eta_M(\mathbf{x}) = \prod_{m=1}^{M}(1 - \mathrm{IPD}_m(\mathbf{x}))$, t_M is the time of the ping transmission, and $\mathrm{IPD}_m(\mathbf{x})$ is the single-ping multistatic probability of detection induced by the source transmitting at time t_m. Because $\eta_M(\mathbf{x})$ is positive and decreasing with M, it converges as $t_M \to \infty$ and so does $P_0(t_M)$.

Use of $\bar{P}_0$ as a decision support tool was proposed in [1]. In our field control selection task, we first fix a Ping Plan and obtain d and T_p by minimising the steady state value $\bar{P}_0(d, T_\mathrm{p})$. Generally, $\bar{P}_0(d, T_\mathrm{p})$ tends to decrease as T_p decreases, which reduces the task of minimising $\bar{P}_0(d, T_\mathrm{p})$ to minimising $\bar{P}_0(d, \bar{T}_\mathrm{p})$ with respect to single variable d, where $\bar{T}_\mathrm{p}$ is the smallest value affordable for the ping interval. The best ping plan can be selected from a finite pre-defined set of plans using the smallest of the obtained minima.

Ping sequencing can also be formed dynamically. One possible strategy, which often has performance similar to a predefined sequential selection [3], is choosing the post for the next transmission randomly. Intelligent ping sequencing using the probability of threat presence is discussed in [3]. Following [3] one can employ the look-ahead technique in which projected values $P_0(t; u)$ of the probability of presence are first obtained for each candidate post u, and the sonobuoy at post $u_0 = \mathrm{argmin}_u P_0(t; u)$ is selected to be the next source for pinging at time t. Fig. 6 illustrates a ping sequencing technique which is computationally less expensive. In this technique we partition A_0 into zones Z_u (see Fig. 6 (a)) and assign them to the respective posts. Selection of post u_0 for pinging at time t is carried out by comparing the total particle weights $p_u(t-)$ in different zones just before the transmission and choosing $u_0 = \mathrm{argmax}_u p_u(t-)$. This selection is motivated by a crude model in which the effect of transmission by source u is emulated by reduction of $p_u(t)$ to zero. Within this model, the look-ahead and zoning techniques produce the same ping sequences. Plots in Fig. 6 (b) show that these two methods exhibit a comparable performance even when the effect of sonar transmission is evaluated using proper IPD computation.

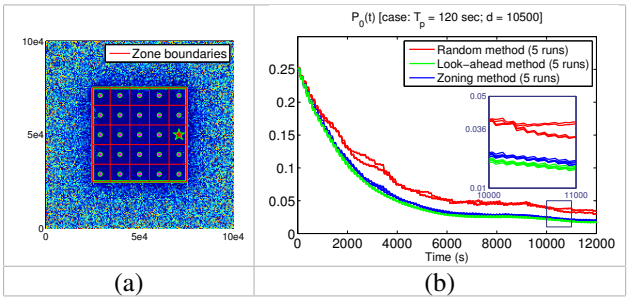

Fig. 6. Partitioning of A_0 into zones (a) and comparison of performance of different ping sequencing methods (b).

5. CONCLUSION

In this work we revisited some aspects of Monte Carlo method of Threat Map computation and showed how it can be accelerated with CUDA GPUs. Considered tests and examples indicate that such an acceleration creates further opportunities for using the Threat Map in selection of sonar field controls and dynamic ping sequencing.

6. REFERENCES

[1] B. I. Incze and S. B. Dasinger, "A Bayesian method for managing uncertainties relating to distributed multistatic sensor search," in *Information Fusion, 2006 9th International Conference on*, July 2006.

[2] D. W. Krout, M. A. El-Sharkawi, W. L. J. Fox, and M. U. Hazen, "Intelligent ping sequencing for multistatic sonar systems," in *Information Fusion, 2006 9th International Conference on*, July 2006.

[3] D. W. Krout, W. L. J. Fox, and M. A. El-Sharkawi, "Probability of target presence for multistatic sonar ping sequencing," *Oceanic Engineering, IEEE Journal of*, vol. 34, no. 4, pp. 603–609, Oct 2009.

[4] D. W. Krout, G. M. Anderson, E. Hanusa, and B. D. Jones, "Threat modeling for sensor optimization," in *Oceans - San Diego, 2013*, Sept 2013.

[5] D. W. Krout and T. Powers, "Sensor management for multistatics," in *17th International Conference on Information Fusion*, July 2014.

[6] A. R. Washburn, "A multistatic sonobuoy theory," Tech. Rep. NPS-OR-10-005, Operations Research Department, Naval Postgraduate School, Monterey, CA 93943, August 2010.

[7] H. Weinberg and R. E. Keenan, "Gaussian ray bundles for modeling high-frequency propagation loss under shallow-water conditions," *The Journal of the Acoustical Society of America*, vol. 100, no. 3, pp. 1421–1431, 1996.

[8] "CUDA C Programming Guide: Design Guide," NVIDIA, PG-02829-001_v6.5, Aug 2014, Available at http://docs.nvidia.com/cuda/.

[9] J. Sanders and E. Kandrot, *CUDA by Example: an Introduction to General-Purpose GPU Programming*, NVIDIA/Addison-Wesley, 2011.

[10] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: a Hands-on Approach*, NVIDIA/Elsevier/Morgan Kaufmann, 2010.

[11] "NVIDIA CUDA Getting Started Guide for Microsoft Windows: Installation and Verification on Windows," NVIDIA, DU-05349-001_v6.5, Aug 2014, Available at http://docs.nvidia.com/cuda/.

[12] "Parallel Computing Toolbox User Guide (R2014a)," MathWorks, March 2014.

[13] "CUDA Compiler Driver NVCC: Reference Guide," NVIDIA, TRM-06721-001_v6.5, Aug 2014, Available at http://docs.nvidia.com/cuda/.

[14] "CURAND Library: Programming Guide," NVIDIA, PG-05328-050_v6.5, Aug 2014, Available at http://docs.nvidia.com/cuda/.

[15] A. T. Ihler, *Inference in Sensor Networks: Graphical Models and Particle Methods*, Ph.D. thesis, Massachusetts Institute of Technology, 2005.

[16] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House, 2004.

[17] L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, 1986.

[18] M. P. Wand and M. C. Jones, *Kernel Smoothing*, Chapman & Hall/CRC, 1995.

[19] "KDE Toolbox for MATLAB," Available at http://www.ics.uci.edu/˜ihler/code/kde.html, 2003.