Implementation of a Multiple Frame Assignment Tracker in a CPU-GPU Integrated Environment

IMPLEMENTATION OF A MULTIPLE FRAME ASSIGNMENT TRACKER IN A CPU-GPU INTEGRATED ENVIRONMENT

BY

K. HERATHKUMAR, B.Tech.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2011) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Implementation of a Multiple Frame Assignment Tracker in a CPU-GPU Integrated Environment
AUTHOR:	K. Herathkumar B.Tech., (Computer Engineering) Dr MGR University, Chennai, India
SUPERVISOR:	Dr. T. Kirubarajan
NUMBER OF PAGES:	xiv, 56

Abstract

The Multi Frame Assignment (MFA) tracker solves the data association problem as a constrained optimization for fusing multiple sets of data to the tracks with an Interacting Multiple Model (IMM) estimator.

With the rapid development of parallel computing hardware such as GPU (Graphics Processor Unit) in recent years, GPGPU (General-Purpose computation on GPU) has become an important topic in scientific research applications. However, GPU might well be seen more as a cooperator than a rival to CPU. Therefore, exploiting the power of CPU and GPU in solving the MFA tracker algorithm based on CPU-GPU integrated computing environment is the focus of this thesis.

In this thesis, a parallel MFA algorithm implementation based on CPU-GPU integrated computing model to optimize performance is presented. The results show that the algorithm increases the average performance by 10 times compared with the traditional algorithm. Based on the results and current trends in parallel computing architecture, it is believed that efficient use of CPU-GPU integrated environment will become increasingly important to high-performance tracking applications.

Acknowledgements

I am especially grateful for the extensive and generous assistance of my supervisor, Professor T. Kirubarajan, who spent time with me each week to guide me through the thesis, provide funding and made numerous helpful suggestions to me.

I would like to thank Dr. R. Theramasa for his thorough reviews and perceptive comments, which greatly improved the quality of my thesis work. I would also like to thank Professors Aleksandar Jeremic and Alexandru Patriciu for being members of my committee. I appreciate all the time the members of my committee took to read this thesis and for providing their input and thoughts on the subject.

I would like to thank my family who provided endless encouragement and advise. This thesis is dedicated especially to my uncle who helped me a lot for my education.

Notation and Abbreviations

Abbreviations

MFA	Multi Frame Assignment
GPU	Graphical Processing Unit
GPGPU	General Purpose Graphical Processing Unit
GPS	Global Position System
CPU	Central Procession Unit
ML	Maximum Likelihood
MHT	Multiple Hypothesis Tracker
VP	Vertex Processor
FP	Fragment Processor
2D	Two Diamension
3D	Three Diamension
SIMD	Single Instruction Multiple Data
MIMD	Multi Instructions Multiple Data
SPMD	Single Program Multiple Data
API	Application Protocol Interface
PC	Personal Computer

HCDFG	Hierarchical Control Data Flow Graph
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
PDF	Probability Density Function
PHD	Probabilistic Hypothesis Density
RMSE	Root Mean Square Error

Notations

A	Area of surveillance region
$c_k(\mathbf{z})$	Distribution of false alarms
E_T	Total average received energy
$E(.)$	Expectations
f_c	Carrier frequency
$\mathbf{F}(.)$	Target motion model function
$g(.)$	Non-linear range measurement function
H_1	Target present hypothesis
H_0	No target present hypothesis
$\mathbf{H}$	Channel matrix
$I_o(.)$	Modified Bessel function of the second kind
k	Time step
l_{mn}	Likelihood ratio
L	Number of targets
L_p	Number of particles per target
M	Number of transmitters
M_m	Number of Monte Carlo runs
N	Number of receivers
N_p	Number of particles used in particle filter
$\hat{N}_t$	Estimated number of targets
p_{S+N}	Probability of signal plus noise measrment
p_N	Probability of noise only measrment

P_C	Probability of target survival
P_B	Probability of target birth
P_D	Probability of target death
P_d	Probability of detection
P_{fa}	Probability of false alarm
$\mathbf{Q}$	Covariance matrix
r	Range bin
$\mathbf{r}(t)$	Received signal
R	Total number of range bins
$\mathbf{s}(t)$	Transmitted signal
T	Sample time
$\mathbf{v}_k$	Process noise
V_{max}	Maximum target speed
V_{max}	Minimum target speed
$w(.)$	Weight of particle
$\mathbf{w}(t)$	Additive white Gaussian noise
$\mathbf{x}_k$	Target state vector
$\hat{\mathbf{x}}_k$	Estimated target state vector
γ	Complex amplitude measurement
ζ	Target scatterer
$\mathcal{R}$	Rayleigh distribution
σ_A	Rayleigh parameter
$\propto$	Proportional to
$\in$	Element of

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

In the past three decades tracking has been employed in a wide range of applications in both military and commercial systems. These applications include Global Position System (GPS), inertial navigation system, missile guidance and control, air traffic control, satellite orbit determination, maritime surveillance, fire control system, automobile navigation system and underwater target tracking systems. In each these cases, the computational requirement varies according to the number of targets present in the surveillance region.

Graphic Processing Units (GPU) have acquired programmability for general scientific applications in recent years. When such a generality of the GPU is emphasized, it is called General Purpose Graphic Processing Units (GPGPU) (J.D. Owens, D. Luebke, and N. Govindaraju, 2007). Dramatic improvements in computing speed have been made by appropriate use of GPUs in several fields. Hence, in order to enhance the efficiency of the target tracking algorithm, the advantage of GPU is utilized in this thesis.

With the current improvements in computer performance, the demand for target

tracking applications is ever growing. As a result, the processor development endues in two directions: general-purpose processor and special-purpose processor. One advantage of modern CPUs has been increased performance according to Moore's Law based on clock frequency and transistors (M. Ekman, F. Wang, and J. Nilsson, 2005). Transistors in CPU are not only responsible for the interpretation, implementation and completion of the various commands and arithmetic logic operation, but also for the control and coordination of the function of most parts of the computer. However, Modern Graphics Processor Unit (GPU) is specialized for compute-intensive, highly parallel computation. GPU has more transistors devoted to data processing rather than data caching and flow control. In recent years, GPU has seen rapid growth in performance, which has broken up the Moore's Law (J. Owens, D. Luebke, N. Govindaraju, M. Harris, J.Kruger, A. Lefohn, and T. Purcel, 2005). Since GPU has highly-efficient and flexible parallel programmable features, a growing number of researchers and business organizations have recognized some of the non-graphical rendering with GPU to implement the calculations, and hence have created a new field of study: GPGPU (D. Luebke, M. Harris, J. Kruger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, and A. Lefohn, 2004), the objective of GPGPU is to use GPU to implement more extensive scientific computing. GPGPU has been successfully used in algebra, fluid simulation, database applications, spectrum analysis, and other non-graphical applications (I. Buck, 2005). Extensive research done in the area indicates that GPU in solving compute-intensive problems has great advantage compared to CPU (C. Thompson, S. Hahn, and M. Oskin, 2002). However, due to the high intensive usage of GPU, it is impossible to complete all computing tasks solely on the GPU. Many control and serial instructions still need to be completed on the CPU.

GPU might well be seen more as a cooperator than a rival to CPU. To utilize GPU for general purpose computations is to exploit the power of CPU and GPU in solving generic problems based on collaborative and heterogeneous computing environment, so that it provides a better solution to solve general computing problems. Generally, traditional parallel task scheduling algorithms are used to schedule task to the first idle processing unit (C.H. Pin and H.L. Sheng, 2001). However, the differences of the heterogeneous processors in performance and bandwidth should be analyzed. In CPU and GPU heterogeneous environment, partitioning and scheduling tasks depend on four characteristics:

- Computing model of task adapted to the CPU or GPU programming model
- Reasonable amount of computing resources
- Task of keeping data within the capacity of GPU memory
- Bottleneck of data transfer efficiency between CPU and GPU

In this thesis, the main focus is on CPU-GPU integrated computing model, which aims at optimizing the performance of parallel data task scheduling. In terms of gaining the power of General Purpose Graphical Process Unit (GPGPU) parallel computing, the Multi Frame Assignment (MFA) tracker algorithm in multiple target tracking is implemented using NVIDIA GeForce 285GTX and compare with the traditional data parallel task scheduling algorithm. The results show that the algorithm increased average performance by 10 times faster than the traditional algorithm implementation on the CPU only environment. On the basis of this result, the conclusion of this thesis can be further extended to the multi-processor CPU collaborative with GPU for heterogeneous computing environment .

The rest of the thesis is organized as follows: The subsequent section provides

background of MFA Tracker and GPU programming model in order to facilitate and understanding of this thesis. In Section 2, an explanation of MFA tracker and its requirement for massive parallel computing requirement is provided, In Section 3, the background on GPGPU and CUDA framework for data parallelization is given. In Section 4, the bottleneck of CPU-GPU integrated computing is analyzed. Then the CPU-GPU integrated computing model and the data parallelization and scheduling algorithm are also described. Section 5 presents the experiment results with the Multi Frame Assignment algorithm, and shows that the performance of traditional sequential MFA algorithm, which is implemented in a CPU only environment, is better than the data parallelization MFA algorithm, which is implemented in the CPU-GPU integrated environment. Conclusions and suggestions for future work are presented in Section 6.

Chapter 2

MULTIFRAME ASSIGNMENT TRACKER FOR MULTITARGET TRACKING

2.1 Target Tracking

The process of inferring the value of a quantity of interest from indirect, inaccurate and uncertain observations is called estimation. The tracking process can be described as the task of estimating the state of a target at the current time and at any point in the future. The estimation of the current state of a dynamic system from noisy data is called filtering and estimating the future state is called prediction (Y. Bar-Shalom, X. Li and T. Kirubarajan, 2001). In addition to the estimates, the tracking system should produce some measure of the accuracy of the state estimates.

2.2 Multitarget Tracking

A track is a symbolic representation of a target moving through an area of interest. In the tracking system, a track is represented by a filter state that gets updated on each new measurement. Figure 2.1 illustrates the basic elements of a conventional Multiple Target Tracking (MTT) system. A signal processing unit converts the signals from the sensor to measurements, which become the input data to the MTT system. The incoming measurements are used for the track maintenance.

Track maintenance refers to the functions of track initiation, confirmation and deletion (Y. Bar-Shalom and X. R. Li, 1995). Observations not assigned to existing tracks can initiate new tentative tracks. A tentative track becomes confirmed when the number and the quality of the measurements included in that track satisfy the confirmation criteria. Similarly, a track that is not updated becomes degraded, and it must be deleted if not updated within some reasonable interval. Gating tests evaluate which possible measurement-to-track pairings are reasonable and a more detailed association technique is used to determine final pairings. After the inclusion of new observations, tracks are predicted ahead to the arrival time for the next set of observations. Gates are placed around these predicted positions and the processing cycle repeats.

If the true measurement conditioned on the past is normally (Gaussian) distributed with its Probability Density Function (PDF), it is given by

$$p(z_{k+1}|Z_k) = N[z_{k+1}; \hat{z}_{k+1}|k, S(k+1)] \tag{2.1}$$

where z_{k+1} is the measurement at time $k+1$, $Z_k = [z_1, z_2, ..., z_k]$, $\hat{z}_{k+1}|k$ is the predicted

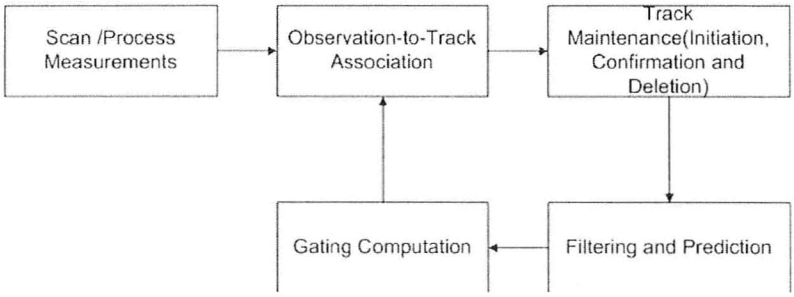

Figure 2.1: Multitarget tracking system

(mean) measurement at time $k+1$ and $S(k+1)$ is the measurement prediction covariance, then the true measurement will be in the following region

$$V(k+1,\gamma) = \{z : [z..z_{k+1}|k]S(k+1)^{-}1[z..\hat{z}_{k+1|k}] < \gamma\} \tag{2.2}$$

with the probability determined by the gate threshold γ.

The region defined by (2.2) is called gate or validation region (V) or association region. The region is also known as the ellipse (or ellipsoid) of probability concentration: the region of minimum volume that contains a given probability mass. The validation procedure limits the region in the measurement space where the information processor looks to find the measurement from the target of interest. Measurements outside the validation region can be ignored since they are too far from the predicted location and very unlikely to have originated from the target of interest. It can so happen that more than one measurement is found in the validation region.

Figures 2.2 and 2.3 illustrate the gating for two well-separated and closely-spaced targets, respectively. In the Figures 2.1 and 2.2 $\bullet$ indicates the expected measurement and the $\star$ indicates the received measurement (R. Tharmarasa, T. Kirubarajan, M.L. Hernandez and A. Sinha, 2007).

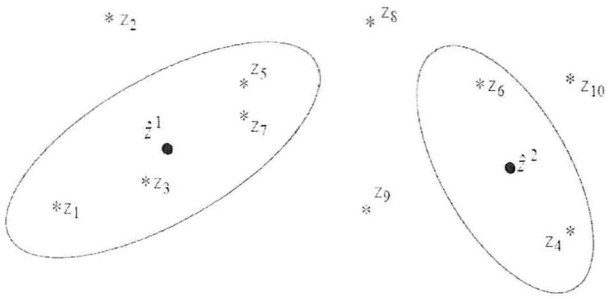

Figure 2.2: Well-seperated targets' validated regions

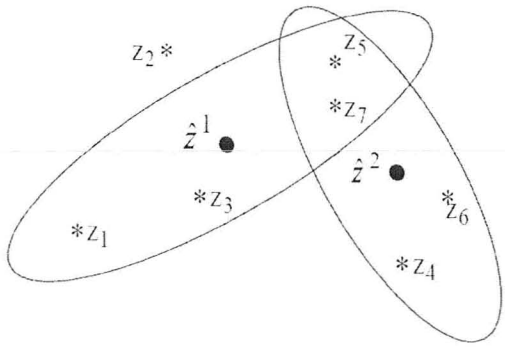

Figure 2.3: Closely-spaced targets' validated regions

Any measurement falling inside the validation region is called a validated measurement. In Figure 2.3: Validation regions of two closely spaced targets and the measurements inside those validation regions are shown (Y. Bar-Shalom and X. R. Li, 1995). More than one measurement is validated in Figure 2.2; there is an association uncertainty. That is, there is ambiguity as to which, if any, of the validated measurements is target originated, assuming that at most one measurements is target generated.

In tracking multiple targets, the basic idea is the same as in single target tracking provided that the targets are well-separated (Figure 2.2). However, if the targets are closely spaced, which causes the corresponding validation regions to overlap (Figure 2.3), the validation region for a particular target may also contain detections from nearby targets as well as clutter detections. Hence, there is a need for a data association technique in order to resolve the measurement origin uncertainty. In this thesis, in order to simplify the analysis, it is assumed that the targets are well-separated so that no measurement-origin uncertainty with respect to the targets.

2.2.1 Data association

Data Association is an algorithm designed to handle measurement origin uncertainty by determining which track a measurement should belong to. It is also called measurement to track association . Several methods have been discussed in the literature with the issue of target initiation, existing target update and track termination (Y. Bar-Shalom, X. Li and T. Kirubarajan, 2001). Among the strategies available to solve data association problems, Multiple Hypothesis Tracker (MHT) attempts to

keep track of all possible association hypotheses over time. MHT can be computationally complex as the number of association hypotheses grows exponentially over time. The nearest neighbor method associates each target with the closest measurement in the target space. This procedure has the shortcoming of pruning away many feasible hypotheses. A more appealing Joint Probabilistic Data Association Filter (JPDAF) uses a gating procedure to prune away infeasible hypotheses.

The problem of tracking multiple targets in clutter considers the situation where there are possibly several measurements in the validation region of each target. The set of validated measurements consists of:

- the correct measurement (if detected and falls within the gate)
- the undesired measurements: false alarms

Then the problem of data association is that of associating the measurements in each validation region with the corresponding track (target). The simplest possible approach Nearest Neighbor (NN), is to use the measurement nearest to the predicted measurement as if it were the correct one. An alternative approach, Strongest Neighbor (SN), is to select the strongest measurement among the validated ones (Y. Bar-Shalom, X. Li and T. Kirubarajan, 2001).

Since any of the validated measurements could have originated from the target, this suggests that all the measurements from the validation region should be used in some fashion. A Bayesian approach, called Probabilistic Data Association (PDA), associates probabilistically all the neighbors to the target of interest (Y. Bar-Shalom, X. Li and T. Kirubarajan, 2001). This is the standard technique used for data association in conjunction with the Kalman filter or the extended Kalman filter. The Kalman filter can be applied only if the models are linear and measurement and

process noises are independent and white Gaussian (Y. Bar-Shalom, T. Kirubarajan and C.Gokberk, 2005).

2.2.2 Filtering Algorithms

In order to analyze and make inferences about a dynamic system, at least two models are required: first, a model describing the evolution of the state with time (the system model), second, a model relating the noisy measurements to the state (the measurement model).

$$x_{k+1} = f_k(x_k) + n_k \tag{2.3}$$

$$z_k = h_k(x_k) + w_k \tag{2.4}$$

where x_k is the state of the target and z_k is the measurement vector at revisit time k. It is assumed that these models are available. The probabilistic state-space formulation and the requirement for the update of information on receipt of new measurements are ideally suited for the Bayesian approach. This provides a rigorous general framework for dynamic state estimation problems.

In the Bayesian approach to dynamic state estimation, one attempts to construct the posterior probability density function of the state based on all available information, including the set of received measurements. Since this PDF contains all available statistical information, it can be considered to be the complete solution to the estimation problem. In principle, an optimal estimate of the state may be obtained from the PDF. A recursive filtering approach means that the received data can

be processed sequentially rather than as a batch so that it is not necessary to store the complete data set nor to reprocess existing data if a new measurement becomes available. Such a filter consists of essentially two stages: prediction and update. The prediction stage uses the system model to predict the state PDF forward from one revisit time to the next. Since the state is usually subject to unknown disturbances, prediction generally translates, deforms and spreads the state PDF.

Suppose that the required PDF $p(x_k|Z_k)$ at revisit time k is available, where $Z_k = [z_1, z_2, ..., z_k]$. The prediction stage involves using the system model (2.3) to obtain the prior PDF of the state at time $k+1$ and is given by

$$p(x_k + 1|Z_k) = \int p(x_{k+1}|x_k)p(x_k|Z_k)\mathrm{d}xk \tag{2.5}$$

The update operation uses the latest measurement to modify the prediction PDF. At revisit time k, a measurement z_k becomes available and will be used to update the prior via Baye's rule:

$$p(x_{k+1}|Z_{k+1}) = \frac{p(z_{k+1}|x_{k+1})p(x_{k+1}|Z_k)}{p(z_{k+1}|Z_k)} \tag{2.6}$$

In the above, the likelihood function $p(z_{k+1}|x_{k+1})$ is defined by the measurement model (2.4).

The above recursive propagation of the posterior density is only a conceptual solution in that, in general, it cannot be determined analytically. Solutions do exist but in a restrictive set of cases.

2.2.3 Kalman Filter

The Kalman filter assumes that the state and measurement models are linear and the initial state error and all the noises entering the system are Gaussian and, hence, parameterized by a mean and covariance (M. S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, 2002). Under the above assumptions, if $p(x_k|Z_k)$ is Gaussian, it can be proved that $p(x_{k+1}|Z_{k+1})$ is also Gaussian.

Then, the state and measurement equations are given by

$$x_{k+1} = F_k x_k + n_k \tag{2.7}$$

$$z_k = H_k x_k + w_k \tag{2.8}$$

If F_k and H_k are known matrices, $v_k \sim N(0, \Gamma_k)$ and $w_k \sim (0, \Sigma_k)$, the Kalman filter algorithm can then be viewed as the following recursive relationship (M. S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, 2002).

$$p(x_k|Z_k) = N(x_k; m_k|k, P_k|k) \tag{2.9}$$

$$p(x_{k+1}|Z_k) = N(x_{k+1}; m_{k+1}|k, P_{k+1}|k) \tag{2.10}$$

$$p(x_{k+1}|Z_k) = N(x_{k+1}; m_{k+1}|k, P_{k+1}|k) \tag{2.11}$$

where

$$m_{k+1}|k = F_{k+1} m_k|k \tag{2.12}$$

$$P_{k+1}|k = \Gamma_k + F_{k+1}P_k|k{F_{k+1}}^T \tag{2.13}$$

$$m_{k+1}|k+1 = m_{k+1}|k + K_{k+1}(z_{k+1} - H_{k+1}m_{k+1}|k) \tag{2.14}$$

$$P_{k+1}|k+1 = P_{k+1}|k - K_{k+1}H_{k+1}P_{k+1}|k \tag{2.15}$$

with

$$S_{k+1} = H_{k+1}P_{k+1}|k{H_{k+1}}^T + \Sigma_{k+1} \tag{2.16}$$

$$K_{k+1} = P_{k+1}|k^T_{H_{k+1}}S^-1_{k+1} \tag{2.17}$$

In the above, $N(x; m, P)$ is a Gaussian density with argument x, mean m and covariance P.

This is the optimal solution to the tracking problem if the above assumptions hold. The implication is that no algorithm can perform better than a Kalman filter in this linear Gaussian environment.

In many situations of interest the assumptions made above do not hold. Hence the Kalman filter cannot be used as described above, and approximations are necessary.

2.2.4 Extended Kalman Filter

If the functions in the (2.3) and (2.4) are nonlinear, then a local linearization of the equations may be a sufficient description of the nonlinearity. Local linearization of the above functions are

$$\hat{F_k} = \frac{df_k(x)}{dx} \tag{2.18}$$

$$\hat{H}_k = \frac{dh_k(x)}{dx} \tag{2.19}$$

The Extended Kalman Filter (EKF) is based on that $p(x_k|Z_k)$ is approximated by a Gaussian. Then all the equations of the Kalman filter can be used with this approximation and the linearized functions (M. S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, 2002).

If the true density is substantially non-Gaussian, then a Gaussian can never describe it well. In such cases, particle filters will yield an improvement in performance in comparison to the EKF.

2.3 Multiframe Assignment

2.3.1 Overview

The Multi Frame Assignment (MFA) algorithm associates the latest $S-1$ scans of measurements (frames) to the tracks, i.e., when the frame at scan k is received, the association is performed between the track list at scan $k-S+1$ and measurements at scans $(k-S+s)_{s=2}^{S}$, The MFA is often implemented as a sliding window (Y. Bar-Shalom, T. Kirubarajan and C.Gokberk, 2005). After performing the association at kth scan, tracks are updated only with the measurements from the $(k-S+2)$th scan. When the next frame at scan $k+1$ is received, the window is advanced to cover the frames at scans $(k-S+s)_{s=3}^{S+1}$ and association is performed between these frames and the tracks at $(k-S+2)$.

Observe that at scan k the MFA assignment algorithm makes a hard decision to only a single frame of data, i.e, after the association at scan k the tracks are updated

with only the frame at scan $k-S+2$. The association between the tracks at $k-S+1$ and the rest of the frames is only tentative (i.e., soft decisions), and can be changed in the subsequent processing in light of the new measurements. This soft decision capability gives the MFA advantage over the 2-D assignment, where the tracks are associated and updated with the latest frame. The price paid for this advantage is the delayed decision making and increased computational cost (R. Tharmarasa, S. Sutharsan and T. Kirubarajan, 2009).

The MFA algorithm formulates the data association as a constrained global optimization problem. The objective is to minimize the total assignment cost of associating sequences of measurements to tracks. The minimization of total assignment cost of associating the measurements to the tracks is gained by parallelizing the MFA algorithm in multilevel, and implementation of MFA algorithm in the General Purpose Graphical Processing Unit (GPGPU).

2.3.2 2-D Assignment

The data association yields decisions as to which of the received measurements should be used to update each track. In assignment, the data association is formulated as a constrained optimization problem, where the cost function to be minimized is a combined negative Log-Likelihood Ratio (LLR) evaluated using the results from the state estimator.

The assignment between the list of tracks and the latest list (scan/frame) of measurements is formulated as a discrete optimization (matching) problem to maximize a dimensionless global likelihood ratio of the measurements-to-tracks assignment. The likelihoods are obtained from the state estimator such as the Kalman filter or the

Interacting Multiple Model (IMM) estimator (Y. Bar-Shalom and X. R. Li, 1995), for the target kinematic state and the classifier outputs for the target types.

In 2-D assignment the measurements from the scan list $M(k)$ are matched (or deemed to have come from) the tracks in list $T(k-1)$ by formulating the matching as a constrained global optimization problem. The optimization is carried out to minimize the cost of associating (or not associating) the measurements to tracks.

2.3.3 S-D Assignment

In 2-D assignment only the latest scan is used and information about target evolution through multiple scans is lost. Also it is not possible to change an association later in light of subsequent measurements. A data association algorithm may perform better when a number of past scans are used. This corresponds to multidimensional assignment for data association.

In S-D assignment, which is the optimization-based Multiple Hypothesis Tracking (MHT) with a sliding window the latest $S-1$ scans of measurements are associated with the established track list (from time $k-S+1$, where k is the current time, i.e., with a sliding window of time depth $S-1$) in order to update the tracks.

Chapter 3

GPGPU PROGRAMMING

3.1 What is GPGPU?

A recent trend in computer architecture is the move from traditional, single-core processors to multi-core processors and further to many-core or massively multi-core processors. This is primarily due to the difficulties in making individual processors faster, while it is possible to provide more processing power by putting more cores onto a single die. The result of this trend is that computational problems which can take advantage of multiple threads can see significant linear speedup with the number of cores available (Oxford University, 2010).

Since GPUs have been independently developed to perform data-parallel computation using multiple cores, the scientific computing community is keen to take advantage of this technology by performing some computation on GPUs that has traditionally been done on CPUs. This is referred to as General-Purpose computation on GPUs (GPGPU) (F. Wu, M. Cabral and J.Brazelton, 2010).

3.2 GPU Architecture

From a traditional standpoint of graphics processing, GPU is designed to process graphics made up of geometric primitives such as points, line, and triangles. Modern graphics pipeline shown in Figure 3.1 is mainly composed of Vertex Processor (VP), Rasterizer and Fragment Processor (FP).

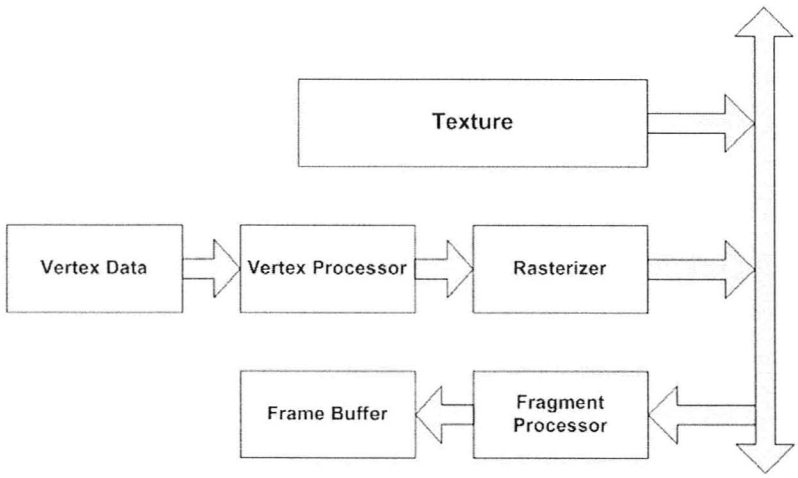

Figure 3.1: Programmable graphics pipeline

GPUs operate according to the standardized graphics pipeline (see Figure 3.1), which is implemented at hardware level (C. Thompson, S. Hahn, and M. Oskin, 2002). This pipeline, which defines how the graphics should be processed, is highly optimized for the typical graphics application, i.e., displaying 3D objects.

The vertex processor receives vertices, i.e., corners of the geometrical objects to display, and transform and project them to determine how the objects should be shown on the screen. All vertices are processed independently and as much in parallel as there are pipelines available. In the rasterizer it is determined what fragments, or potential pixels, the geometrical shapes may result in, and the fragments are passed

on to the fragment processor. The fragments are then processed independently and as much in parallel as there are pipelines available, and the resulting color of the pixels is stored in the frame buffer before being shown on the screen.

At the hardware level the graphics pipeline is implemented using a number of processors, each having multiple pipelines performing the same instruction on different data. That is, GPUs are Single-Instruction, Multiple-Data (SIMD) processors, and each processing pipeline can be thought of as a parallel sub-processor.

From an alternative point of view, GPU can be seen as a streaming processor containing arrays of VPs and FPs operating in parallel (NVIDIA Corporation, 2010). When the programmer specifies a shader program which is called kernel and a data stream, GPU maps this data onto the available processors to compute the result. Current GPU has Multiple-Instruction, Multiple-Data (MIMD) VPs and Single-Program, Multiple-Data (SPMD) FPs. GPU execute batches of fragment threads in SIMD execution style (one thread per pixel). Both VPs and FPs are highly computationally capable (I. Buck, 2005).

Much research on GPGPU has been presented. GPU is a large amount of programmable floating-point horsepower that can be exploited for compute-intensive applications completely unrelated to graphics processing. But GPU is hardly a computational panacea. Its arithmetic power results from a highly specialized architecture (C. Thompson, S. Hahn, and M. Oskin, 2002). Today, with the rapid development of GPU, many programming challenges and limitations of architecture have been solved by hardware design and programming techniques. A new issue associated with the usage of GPU for general-purpose computation is how to integrate with CPU to achieve better performance.

3.2.1 GPGPU cards

GPGPU cards provide an abundance of computational cores (240 on a Nvidia GTX 280) associated with a limited amount of memory and memory bandwidth. Each core can compute three single precision floating point operations per clock for a total maximum performance of 10^9 floating point operation per second (TFLOPS). This is roughly 20 times more computational power than an Intel i7 CPU. Each GPGPU card can have from 512MB to 4GB of memory with a bandwidth of 140GB/sec. The bandwidth available per core is 140(GB/sec) / 240 cores = 0.58GB/sec/core. An iCore7 CPU can access up to 64 (GB/sec) / 8 cores = 8GB/sec/core. Each GPGPU core has access to 13.8 less data per second than an Intel i7 CPU (NVIDIA Corporation, 2010). For this reason, algorithms that are more computationally intensive than memory intensive are particularly suited for GPGPU cards. It is important to note that, in this thesis, MFA algorithms is a memory intensive algorithm due to the high capacity of data involved. So that MFA algorithm is rewritten to minimize memory transfer and maximize computational loads.

3.2.2 GPGPU Memory Model

GPGPU cards have six different types of memory, each with its own characteristics: global (large, read/write, no cached, slow), texture (large, only read, cached), constant (small, only read, cached), registry (small, read/write, fast), local (like global), and shared (small, read/write, fast, temporary). Shared memory and registry are employed at runtime to store the temporary information for threads (N. Govindaraju, S. Larsen,J. Gray and D. Manocha , 2007).

3.3 CUDA Framework

3.3.1 CUDA programming environment

High-level programming languages are prepared for parallel programming in GPU. In the case of NVIDIA's GPU products, C-like language called CUDA is prepared. CUDA employs a programming model called single program multiple data stream (SPMD). This is a direct reflection of the limitation such that a warp has to execute a single operation. A sample pseudo code is given below. In this pseudo code, the behavior of threads in GPU are defined. First, data used in GPU code is transferred by call cudaMemcpy(). Then copies of kernal foo() are spawned by a special calling syntax foo<<<...>>>(), where the number of copies is specified in the bracket. While GPU works, CPU does some computation or simply waits by calling cudaStreamSynchronize(). Finally, the host PC brings back the result of GPU's computation by calling again cudaMemcpy().

Threads are grouped into a block and blocks are further grouped into a grid. A rough image of task distribution is the following: a block is assigned to a multiprocessor, it is divided into warps, and then warp is processed by thread processors. Here, note that threads in a block are guaranteed to be executed synchronously, whereas blocks are not. If one has to synchronize among blocks after some task, they should prepare a GPU code that ends at the task and a CPU code that waits all tasks of all blocks, by calling cudaThreadsSynchronize().

```
//GPU(device) code
float (*g)[N];
_global_ void foo(){
    // who am i?
    int tid=blockDim.x * blockIdx.x + threadIdx.x;
    int Nthreads=blockDim.x * gridDim.x;
    assert(Nthreads==N);
    // do my own task
    for(i=0; i < M; i++ ){
        g[i][tid]=g[i][tid] + ...;
        // do special task, if I am a special thread
        if(tid==0){...}
 }
//CPU(host machine) code
float c[M][N];
int main(int argc, char** argv){
    cudaMalloc(&g, sizeof(float)*N*M);
    cudaMemcpy(g,c,sizeof(float)*N*M, cudaMemcpyHostToDevice);
    nblocks = N*M/32;
    nthreads=32;
    foo<<<nblocks, nthreads>>>();
    cudaStreamSynchronize(0);
    cudaMemcpy(c,g,sizeof(float)*N*M, cudaMemcpyDeviceToHost);
}
```

3.3.2 Data Parallelism

A data-parallel computation is where computation has been parallelized by distributing the data amongst computing nodes. It can be contrasted with a task-parallel computation, in which the distribution of computing tasks is emphasized as opposed to the data. One framework that is used to accomplish data-parallelism is Single Instruction, Multiple Data (SIMD), in which multiple processors execute the same instructions on different pieces of data. This is the architecture used in GPUs, since it allows flow control computation to be shared amongst processors and thus allows more of the hardware to be devoted to instruction execution.

Not all computation must be completely parallelizable. Although typically every thread will run identical functions, the functions themselves can condition on thread identifiers and data so that different instructions are executed in some threads. However, in SIMD architectures this leads to a performance hit since computation only occurs in parallel when the same instructions are being performed (J. Owens, D. Luebke, N. Govindaraju, M. Harris, J.Kruger, A. Lefohn, and T. Purcel, 2005).

Chapter 4

MFA TRACKER IMPLEMENTATION IN CPU-GPU INTEGRATED ENVIRONMENT

4.1 Overview

To implement MFA algorithm based on CPU-GPU integrated computation, tasks partition can be divided into two categories: computing tasks and communicating tasks.

• Computing tasks run in CPU or GPU whereby several instructions act on data, and then retrieve computing results.

• Communicating tasks are responsible for controling the data input and output of computing tasks.

MFA Algorithm can be divided into multiple sub-tasks, and these sub-tasks are implemented through a certain processing flow. The structure of sub-tasks consists of data and operation. Therefore, sub-tasks are mapped to CPU implementation or GPU implementation by scheduling sub-tasks to different hardware. Hierarchical Control Data Flow Graph [HCDFG] can be used to describe computing model of GPU-CPU heterogeneous environment. HCDFG allows nesting of the traditional data flow hierarchically (K. N. Levitt and W. T. Kautz, 1972). Especially, it is not only a good description of multi-tasks in an algorithm, but can also optimize performance of scheduling fine-grained sub-tasks between CPU and GPU. As shown in Figure 4.1, and computing tasks are operating nodes, and communicating tasks are transmitting nodes, and edges between nodes describe direction of data flow (L. Wang, Y.Z Huang, X. Chen, and C.Y. Zhang, 2008).

In order to exploit task parallelism on coarse grain decomposition, MFA algorithm can be divided into many sections, e.g., segments S1 to S6. Therefore, implementation of each segment is mapped to CPU or GPU.

For simplicity, the case of one CPU and one GPU is evaluated. The execution time T_{tot} is defined in (4.1). The execution time of CPU is denoted as T_{cpu}, the execution time of GPU as T_{gpu}, and the communication time between CPU and GPU as T_{com}. Then

$$T_{tot} = T_{cpu} + T_{gpu} + T_{com} \tag{4.1}$$

From (4.1) (L. Wang, Y.Z Huang, X. Chen, and C.Y. Zhang, 2008), the following

essential characteristics are identified and associated with how to minimize T_{tot} to achieve maximized performance while we are implementing the MFA algorithm in CPU-GPU integrated environment:

- Communication time between CPU and GPU, which might be a bottleneck of CPU-GPU integrated computing environment.
- The paradigm of mapping sub-tasks of MFA tracker to CPU implementation or GPU implementation by scheduling sub-tasks to different hardware.

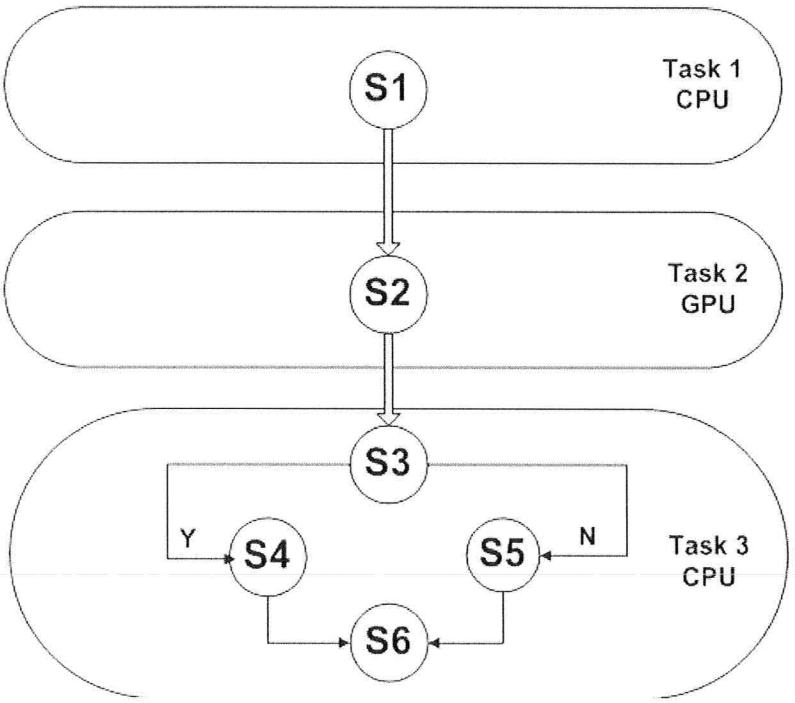

Figure 4.1: Paradigm of task partition in CPU-GPU intergrated environment

4.1.1 Bottleneck of GPU-CPU intergrated computing environment

From the implementation point of view of the CPU-GPU integration, CPU is considered as a host processor that can both distribute tasks and execute tasks, while GPU can only execute tasks (NVIDIA Corporation, 2010). In order to analyze the performance bottlenecks of CPU-GPU integrated computing, the data communication speed of upload-bandwidth (transfer results of GPU to CPU memory) and download-bandwidth (transfer data of CPU to GPU memory) are examined to investigate the data transfers performance and data communication bottleneck in the CPU-GPU integrated computing environment while we implement such MFA algorithm with massive amount of data processing.

In order to experiment the data upload and download speed between host (CPU) and device (GPU), massive amounts of data from target measurement sets are generated and used for analyzing the data communication bottleneck. The large capacity of target measurement sets with different capacity of data transferred from CPU to GPU as well as GPU to CPU. The time taken for the data transfer from CPU to GPU as well as GPU to CPU is calculated. Table 4.1 shows the data obtained from this bandwidth test experiment. Two different graphs Figure 4.2 and Figure 4.3 are plotted using the Table 4.1.

As shown in Figure 4.2 and Figure 4.3, experiment results of bandwidth are much lower than the theoretical bandwidth of PCI-Express 16x (4GB/s). Therefore, data transfers efficiency between CPU and GPU is a bottleneck and it is more of a bottleneck than the hardwares capacity for computation in CPU-GPU integrated environment. For the purpose of finding the data communication bottleneck experiments

Table 4.1: CPU to GPU Bandwidth Test

Dataset(MB)	Upload Speed(Mb/s)	Download Speed(Mb/s)
1.08	275.71	1220.58
2.16	346.08	2020.78
3.24	661.67	2720.58
3.84	768.97	3181.62
4.44	754.88	2150.23
5.04	721.89	2380.83
5.64	987.26	2348.13
6.24	930.4	2440.44

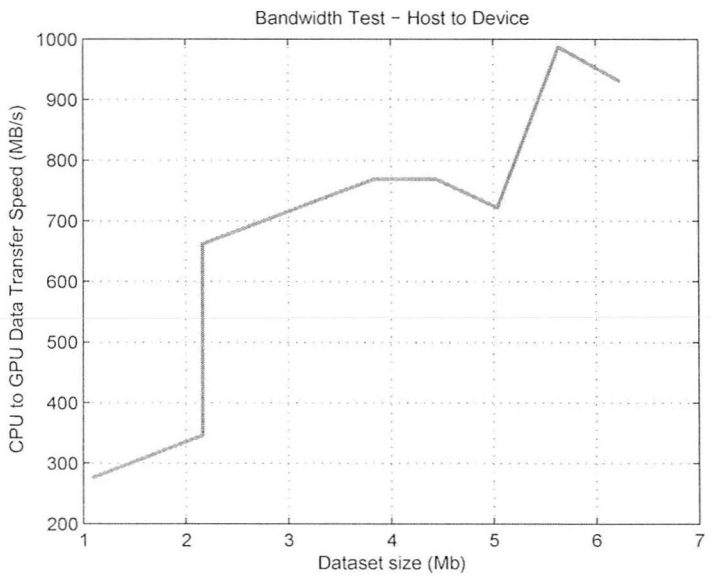

Figure 4.2: Data transfer rate from host (CPU) to device (GPU)

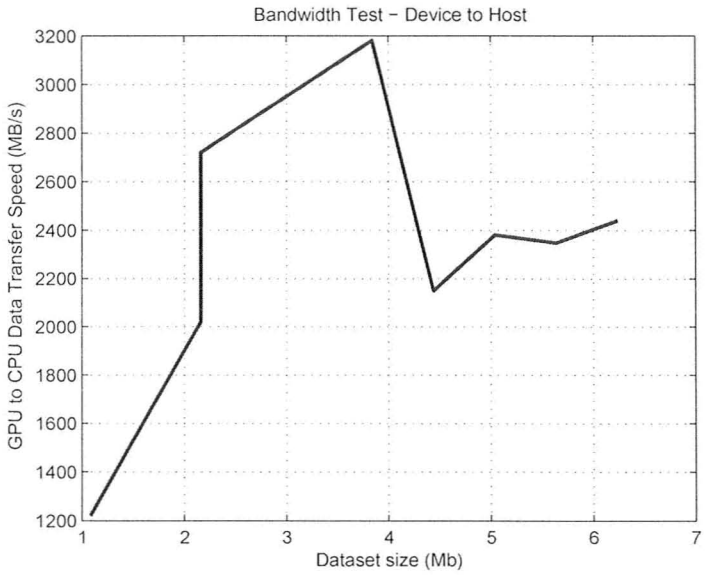

Figure 4.3: Data transfer rate from device (GPU) to host (CPU)

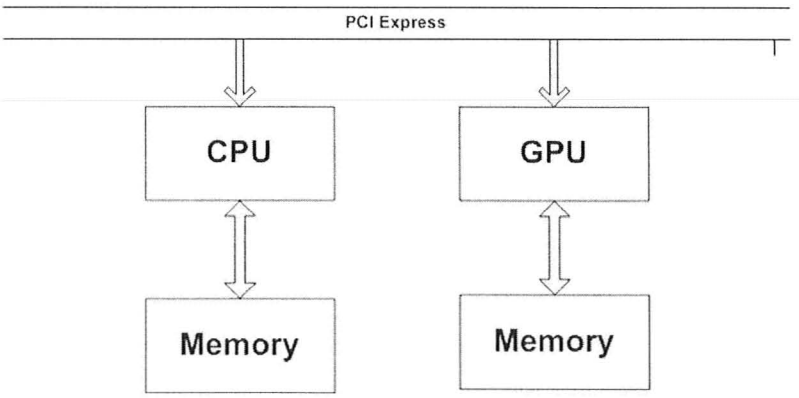

Figure 4.4: CPU-GPU intergrated computing model.

presented in this work, NVIDIA GeForce 285GTX as graphics card (GPU Device) is used. This card has 240 graphics processors clocked at 1476MHz, 1GB GDDR3 video memory clocked at 1242MHz and the data transfers between CPU and GPU device occur through the PCI-Express 16x interface, which is the connecting bridge in the PC and the host (CPU) and device (GPU) as shown in the Figure 4.4. This processor includes 5 VPs and 12 FPs. As for the computer systems, Intel Pentium D 820 based system with 2GB RAM hardware specification has been used.

4.1.2 Parallel task optimization (PTO) algorithm

Parallel task optimizing algorithm of CPU-GPU integrated environment indicates relationship among computing capability of processor, communicating cost, and size of data sets. At first, in this work, three assumptions are made as the basis of parallel task optimization of parallel processing in CPU-GPU integrated environment, and then an algorithm (L. Wang, Y.Z Huang, X. Chen, and C.Y. Zhang, 2008) is given in this section later.

The computing capability of processor is denoted as P_{cpu}, communicating or data transferring cost of GPU (communication or data transferring between host CPU and device GPU as Tr_{gpu}, communicating cost of CPU (communication between host CPU and main memory) as Tr_{cpu}, size of data sets as d. the following four mathematical assumptions are made in order to optimize the time taken for the data communication between the host CPU and the device GPU (L. Wang, Y.Z Huang, X. Chen, and C.Y. Zhang, 2008):

$$d \propto Tr_{gpu} \tag{4.2}$$

$$d \propto Tr_{cpu} \tag{4.3}$$

$$\frac{\Delta(P)}{\Delta d} < \frac{\Delta Tr_{cpu}}{\Delta d} < \frac{\Delta Tr_{gpu}}{\Delta d} \tag{4.4}$$

$$Tr^{d}_{cpu} \ll Tr^{d}_{gpu} \tag{4.5}$$

As far as the above mentioned assumptions are considered, there are two major characteristics. First, when the size of data sets changes, the communication or data transferring cost will be considered as a major factor of performance. Second, the GPU and CPU both have independent local memory. Communication can be divided into three parts: communication between GPU and video memory, CPU and main memory, and interface between CPU and GPU. As described in Section 4.1.1, data transfers efficiency between CPU and GPU is a bottleneck. Therefore, communication cost between host (CPU) and device (GPU) is far more important than other costs (L. Wang, Y.Z Huang, X. Chen, and C.Y. Zhang, 2008).

On the basis of the concept mentioned above, an optimization algorithm is developed for parallel task optimization which is implemented in MFA tracker algorithm of parallel data processing in CPU-GPU integrated environment.The parallel task optimization (PTO) steps are follows (L. Wang, Y.Z Huang, X. Chen, and C.Y. Zhang, 2008):

Step 1: If the time taken to the traffic or communication time Tr is ignored, the executed time of GPU and CPU could be measured separately, which is $T_{gpu}(P_i)$ and $T_{cpu}(P_i)$ under some calculational measures by analyzing the different measures of the

task. Supposing $T_{gpu}(P_i) < T_{cpu}(P_i)$ is right when i is located in the interval of $[m, n]$, GPU will execute this task, otherwise CPU will do.

Step 2: By analyzing the different calculational measures of Pi and corresponding traffic Tr_i of the task, under some calculational measures, the executed time of GPU and CPU is obtained, which is $T_{gpu}(Pi) + T_{gpu}(Tr_i$ and $T_{cpu}(P_i) + T_{cpu}(Tr_i)$, if the following equation (4.6) is right when i is located in the interval of $[m, n]$, GPU will execute this task, otherwise CPU execute the task.

$$T_{gpu}(P_i) + T_{gpu}(Tr_i) < T_{cpu}(P_i) + T_{cpu}(Tr_i) \tag{4.6}$$

Step 3: Based on step 2, the parallel task scheduling is further optimized, that code segments can be divided into subtasks $TaskS_1, TaskS_2, ..., TaskS_n$. Under the premise that the size of data sets is the same both on CPU and GPU for each subtask, the compute-intensive subtask will be executed by GPU.

4.2 Parallelization of MFA Tracker Tasks

In order to resolve the computational complexity issues in the MFA tracker algorithm, the multilevel parallelization in MFA tracker is introduced, in order to implement the tracker in the parallel computation general purpose GPU systems. Multilevel parallelization enables many independent and highly parallelizable tasks to be executed concurrently, including: 1) multiple frame assignment problems via a parallelization of the partitioning task, and 2) the numerous gating tests, state estimates, covariance calculations, and likelihood function evaluations (used as cost coefficients in the multiframe assignment(MFA) problem) via a parallelization of the data association

interface task (R.L. Popp, K.R. Pattipati, Y. Bar-Shalom, and R.R. Gassner, 1998).

4.2.1 Parallelization of data association task

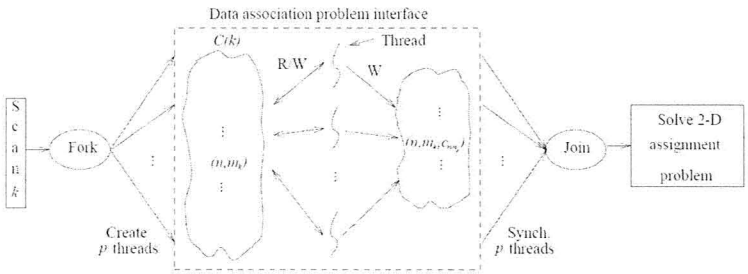

Figure 4.5: IMM/multiframe shared memory parallelizaion

Even though it has been a historical and widely held belief that the most computationally intensive aspect of multitarget tracking has been the task of solving the data association problem, contrary to conventional wisdom, the interface to the data association problem also comprises a significant fraction of the workload. Consequently, as illustrated in Fig. 4.5 (R.L. Popp, K.R. Pattipati, Y. Bar-Shalom, and R.R. Gassner, 1998), in m-best multiframe, in particular, based on the supervisor/worker model, a supervisor thread initially forks a specified number of worker threads, say p, to process the set of candidate associations, i.e., C(k). Once forked, the supervisor awaits processing of $C(k)$ to be completed by the p worker threads via a joint operation. Worker threads, asynchronously and in parallel, process a specified number of candidate associations per serialized critical section access across mutually exclusive track and measurement data. The processing of a candidate association primarily consists of computing the numerous independent gating tests (which consists of a coarse maximum velocity gating test and a fine Kalman filter elliptical gating test),

state estimates, covariance calculations, and likelihood function evaluations used as assignment cost coefficients in the 2-D assignment problem. Since the processing cost corresponding to each candidate association is not uniform (depends on the results of gating), dynamic scheduling of candidate associations across threads is employed. In this way, because candidate associations are dynamically scheduled, maximum load balancing is achieved.

4.2.2 Parallelization of cost calculation task

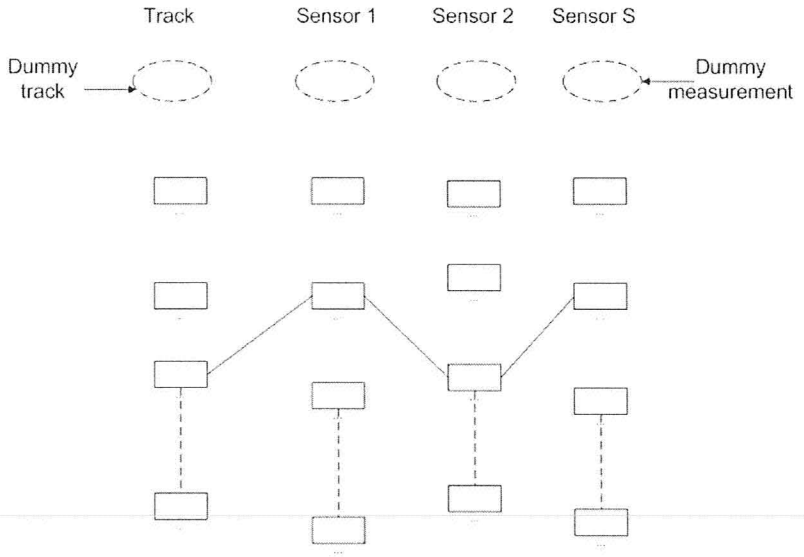

Figure 4.6: Assignment tree for $(S+1)-D$

In multiframe assignment, measurements from multiple (say, S) sensors are matched to the tracks in the track list. That is, elements of $(S+1)$ lists are matched together through a optimization algorithm to solve the data association. The number of lists in the assignment determines the dimension (D) of the optimization problem. This assignment can be done in two ways. The first one, called S-D + 2-D assignment, is a

two-step approach in which measurements from all the sensors are grouped together first and then the grouped measurements are associated to the tracks (R. Tharmarasa, S. Sutharsan and T. Kirubarajan, 2009). The second approach called $(S+1)-D$ assignment is a one-step algorithm in which measurements are directly assigned to the tracks. In the $(S+1)-D$ formulation, if S is equal to one then it will be same as 2-D. In this tracker,$(S+1)-D$ assignment is used since it does not require a maximum likelihood (ML) estimate, which is used in measurement-to-measurement association in $S-D$. A sample assignment tree for $(S+1)-D$ is shown in Figure 4.5.

The cost of assigning measurements $i_1, i_2, ..., i_2$ to track t is given by

$$c_{ti_1i_2...,i_S} = -\log\left(\frac{p(Zi_1i_2...,i_S|X_t)}{p(Zi_1i_2...,i_S|\emptyset)}\right) \tag{4.7}$$

$$p(Zi_1i_2...,i_S|X_t) = \prod_{s=1}^{S}(1-P_D)^{1-u(i_s)}(P_D p(z_{i_s}|X_t))^{u(i_s)} \tag{4.8}$$

$$p(Zi_1i_2...,i_S|\emptyset) = \prod_{s=1}^{S}\frac{1}{\psi_s} \tag{4.9}$$

$$u_{(i_s)} = \begin{cases} 0 & i_s = 0 \\ 1 & \text{otherwise} \end{cases} \tag{4.10}$$

where $u_{(i_s)}$ takes value 0 if $i_s = 0$ and 1 otherwise, PD is the probability of detection and s is the volume of the measurement space of sensor s.

The most likely hypothesis is determined by the $(S+1)-D$ assignment formulation by solving the following constrained optimization (R. Tharmarasa, S. Sutharsan and T. Kirubarajan, 2009):

$$\min_{x_{ti_1i_2\ldots i_s}} \sum_{t=0}^{T} \sum_{i_1=0}^{N_1} \sum_{i_2=0}^{N_2} \ldots \sum_{i_s=0}^{N_s} c_{ti_1i_2\ldots i_s} x_{ti_1i_2\ldots i_s} \tag{4.11}$$

subject to

$$\sum_{i_1=0}^{N_1} \ldots \sum_{i_s=0}^{N_s} x_{ti_1i_2\ldots i_s} = 1, t = 1, \ldots, T \tag{4.12}$$

$$\sum_{t=0}^{T} \sum_{i_2=0}^{N_2} \ldots \sum_{i_s=0}^{N_s} x_{ti_1i_2\ldots i_s} = 1, j = 1, \ldots, N_1 \tag{4.13}$$

$$\sum_{t=0}^{T} \sum_{i_1=0}^{N_1} \ldots \sum_{i_{s-1}=0}^{N_{s-1}} x_{ti_1i_2\ldots i_s} = 1, j = 1, \ldots, N_s \tag{4.14}$$

where $x_{ti_1i_2\ldots i_s}$ can be given by

$$x_{ti_1i_2\ldots i_s} = \begin{cases} 1 & Z_{i_1i_2\ldots i_s} \text{is from target t} \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

In the above formulations, $i_s = 0$ indicates the dummy measurement, $t = 0$ indicates the dummy track, T is the total number of tracks and N_s is the number of measurements from sensor s.

According to the above mentioned cost calculation in (4.7) and the optimization in 4.11 are very high computational tasks. Also, the data such as measurement and track lists used in this task computation are very high compute-intensive. Hence when performing the cost calculation operation in the single CPU computing environment, it takes massive amount of time for the completion of particular computation. By implementing this cost calculation task in CPU-GPU integrated environment, high computational efficiency is gained using the PTO optimization algorithm as demonstrated in the simulation.

Computational load of the $(S+1)-D$ assignment can be reduced by approximately decomposing and assigning multiple threads for the individual $(S+1)D$ assignment into S individual 2-D assignments. That is, each sensor's measurements are associated to the tracks separately.

4.2.3 When to Use the GPU for Data Association?

To perform the data association in the GPU, estimates of all the tracks and all the measurements must be transferred to the GPU. The track information contains estimated state and corresponding covariance for each IMM mode. Hence, the size of a track information is

$$B_t = (n_x M + n_x^2 M) \times 64 \tag{4.16}$$

where n_x is the size of the state and M is the number of IMM modes. In the above, 64-bits are used for a double value.

For each measurement, following information must be transferred:

- time
- measurements
- variances
- SNR
- PFA
- sensor state

Hence, the size of the measurement information is

$$B_m = (1 + n_z + n_z + 1 + 1 + n_s)64 \tag{4.17}$$

where n_z is the size of the measurement and n_s is the size of the sensor state.

The communication load for n_t targets and n_{m_k} measurements is

$$B(n_t, n_{m_k}) = n_t B_t + n_{m_k} B_m \tag{4.18}$$

Then, communication time will be

$$T_c(n_t, n_{m_k}) = B(n_t, n_{m_k})/r_t \tag{4.19}$$

where r_t is the data transfer rate between the CPU and the GPU.

The computation time of the data association in the CPU will depend on the programming language, for example, implementing in MATLAB might take few hundreds times than that of C implementation. Hence, it is hard to quantify the computation time. A lookup table may be created from experiments.

If the data association computation time for n_t targets and n_{m_k} measurements in the CPU is $T^{DA}_{CPU}(n_t, n_{m_k})$, the corresponding computation time in the GPU will be

$$T^{DA}_{GPU}(n_t, n_{m_k}) \approx T^{DA}_{CPU}(n_t, n_{m_k})/f_g \tag{4.20}$$

where f_g is the improvement factor of the GPU.

Then, it is beneficial to perform the data association in the GPU if and only if

$$T^{DA}_{CPU}(n_t, n_{m_k}) < T^{DA}_{CPU}(n_t, n_{m_k})/f_g + T_c(n_t, n_{m_k}) \quad (4.21)$$

$$T^{DA}_{CPU}(n_t, n_{m_k})\left(1 + \frac{1}{f_g}\right) < T_c(n_t, n_{m_k}) \quad (4.22)$$

$$T^{DA}_{CPU}(n_t, n_{m_k}) < \frac{T_c(n_t, n_{m_k})f_g}{1 + f_g} \quad (4.23)$$

4.2.4 Parallelization of new track initialization task

In terms of forming the new tracks, the measurements that are not associated to already existing tracks from all the sensors are used. If the probabilities of detections of the targets are high for all the sensors, then the measurements at one time step from all the sensors can be used to initialize new tracks using the logic of at least n measurements from m sensors. However, this approach will fail in low probability of detection cases. Hence, measurements over multiple time steps must be considered in track initialization, as shown in Figure 4.6.

The above approach will be computationally demanding even with few sensors. A suboptimal approach, in which new tracks are formed first for each sensor separately by considering multiple time steps and then the new tracks from all the sensors are fused together, can be used to reduce the computational load so that track initialization can be done in real-time.

Since this above process requires high computational power for forming the new tracks from each sensor and then for fusing together each formed tracked, this process is implemented in parallel, for gaining the power of CPU-GPU integrated computer environment. Here the PTO algorithm is applied while implementing in the integrated environment.

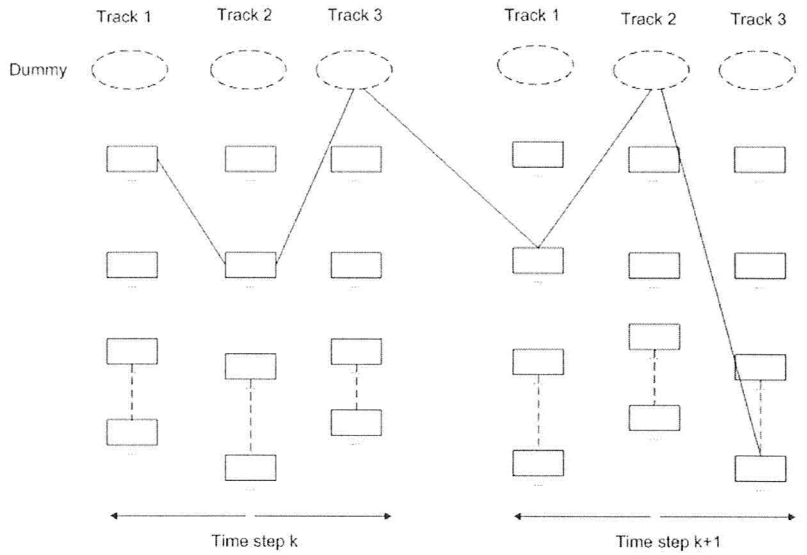

Figure 4.7: Assignment tree with multiple transmitter-receiver pairs and time steps

In this tracker, distributed tracking is used for track initialization and centralized tracking is used to update the already initialized tracks. The distributed tracking for initialization might result in slight performance degradation compared to the centralized tracking. However, the advantage of diversity of sensor field is not totally lost. It is reasonable to assume that target-sensor geometry will not change significantly in two or three measurement time steps. Hence, even if only one sensor has better detections of a target due to target-sensor geometry, a track will be initialized at least by that sensor. After initializing, the measurements from all the sensors will be used in the following time steps to confirm the track.

4.2.5 When to Use the GPU for Initialization?

This decision making is same as the one given for data association. However, for track initialization, only the measurements need to be transferred to the GPU. Then,

it is beneficial to perform the data association in the GPU if and only if

$$T^I_{CPU}(n_{m_k}) < T^I_{CPU}(n_{m_k})/f_g + T_c(n_{m_k}) \quad (4.24)$$

$$T^I_{CPU}(n_{m_k})\left(1+\frac{1}{f_g}\right) < T_c(n_{m_k}) \quad (4.25)$$

$$T^I_{CPU}(n_{m_k}) < \frac{T_c(n_{m_k})f_g}{1+f_g} \quad (4.26)$$

where $T^I_{CPU}(n_{m_k})$ is the track initialization computation time for n_{m_k} measurements in the CPU.

However, if the data association for the existing target is already performed in the GPU, then there is no need to transfer the measurement to the GPU. Hence, it is always beneficial to perform the track initialization in the GPU, if the data association is performed in the GPU.

The process of MFA tracker sequential computation on a stand-alone CPU is shown in Figure 4.8 and Figure 4.9. As can be seen in the process flow chart Figure 4.9, a nested iteration of computations is performed in the sequential MFA algorithm. Because of this nested iteration and the tracker's measurement data set capacity is also high in size, there is a computational overhead in this sequential process. In order to avoid this computational bottleneck, the process is partitioned into multiple threads, and each task is assigned to separate threads. The measurement association and the cost calculation are performed in parallel as shown in Figure 4.10.

The process of MFA tracker is parallel algorithm is shown in Figure 4.10 Firstly, initialize all parameters, and transport the parameters to GPU for computing the cost of associating multiple targets. Then, initialize GPU and start multiple threads to compute cost calculation with many different data sets for the multiple targets.

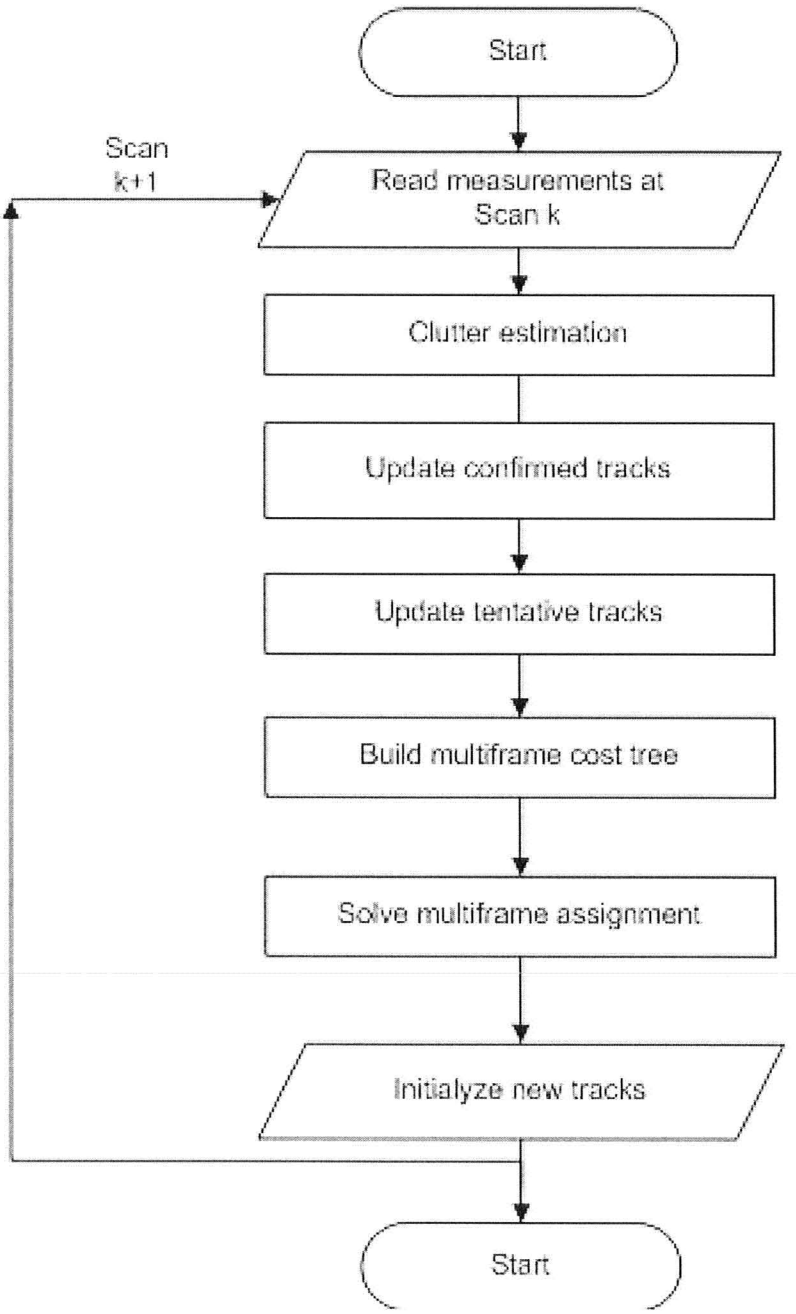

Figure 4.8: Flow chart of MFA tracker

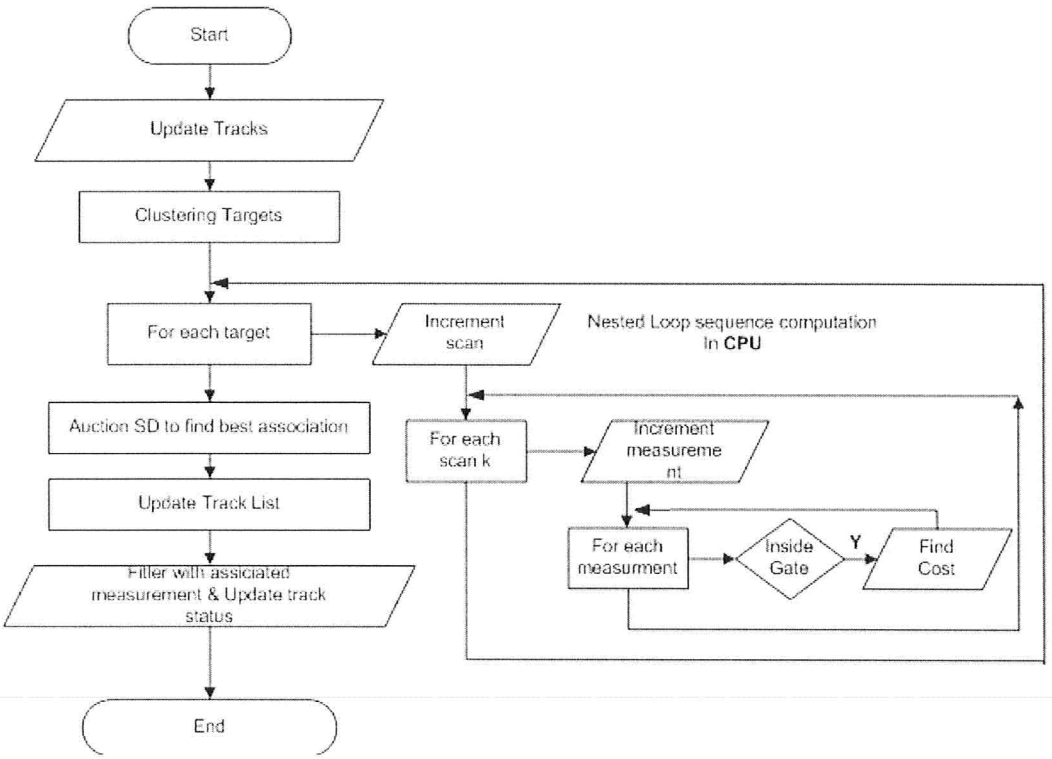

Figure 4.9: Flow chart for sequential MFA tracker computation in stand-alone CPU

According to the results, retain the most likelihood path and transport the path parameters back to host. The CPU is mainly responsible for GPU initialization, memory management, data preparation, and receiving the results from GPU. The GPU is mainly responsible for parallel computing and transporting the results to host. Due to the CPU accessing memory device only by PCI-E interface, the transmission is slow as shown in the previous section became of the bottleneck in the CPU-GPU data communication. Frequent data exchanging between host and device should be avoided in the data parallel algorithm.

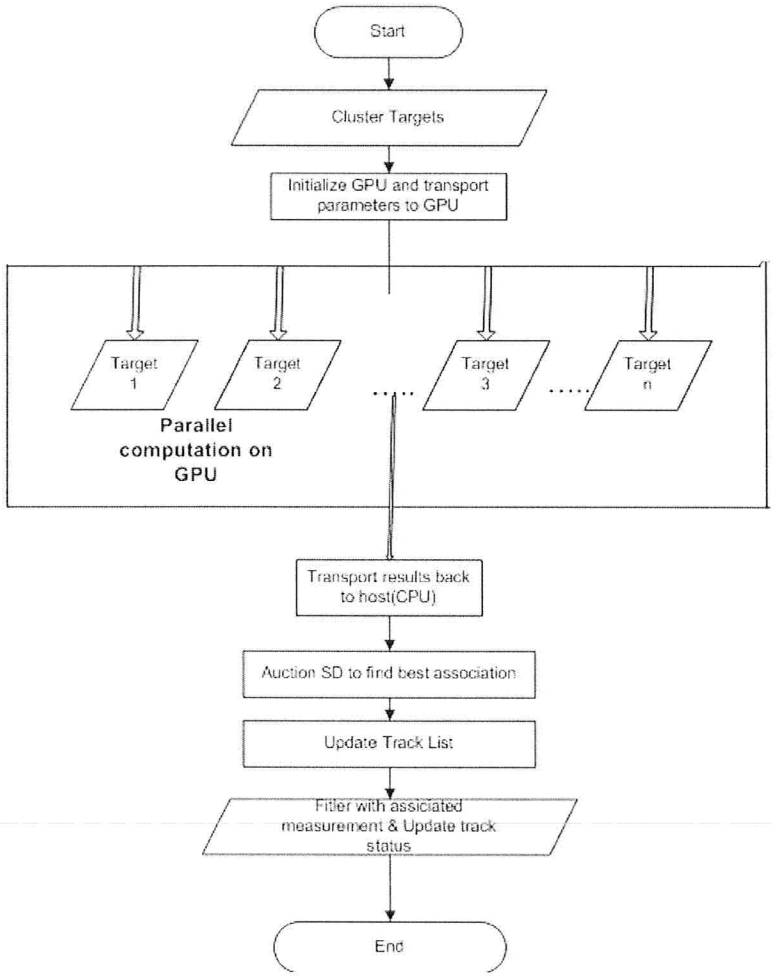

Figure 4.10: Flow chart for parallel MFA tracker computation on CPU-GPU integrated environment

Chapter 5

RESULTS AND DISCUSSIONS

5.1 Performance Measures

The performance measures that show the effectiveness of an MFA parallel algorithm are the speedup factor and parallel efficiency. These measures depend on how a given set of tasks is assigned and executed onto the GPGPU architecture. The speedup of a parallel algorithm is given by

$$\lambda = \frac{Ex_(c)}{Ex_(cg)} \tag{5.1}$$

Where is $Ex_(c)$ Execution time on CPU and $Ex_(cg)$ is execution time on CPU-GPU integrated environment.

5.2 Simulations

This section presents a two dimensional tracking example to illustrate the new parallel task scheduling technique on CPU-GPU integrated environment and to compare with the direct implementation of the sequential algorithm on CPU only computing environment. The single target Markov transition model that characterizes the j^{th} target dynamic at time k is given by

$$x_k^j = A_k^j x_{(k-1)}^j + w_k^j \tag{5.2}$$

where $x_k^j = [x_k^j, y_k^j, \dot{x}_k^j, \dot{y}_k^j]$ is the state of the the j^{th} target, which consists of target position (x_k^j, y_k^j) and target velocity $(\dot{x}_k^j, \dot{y}_k^j)$ at time step k, and w_k^j is an i.i.d. sequence of zero-mean Gaussian noise vectors with covariance Σ_k^j. The matrix A_k^j is given by

$$A_k^j = \begin{pmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

The matrix Σ_k^j is given by

$$\Sigma_k^j = \begin{pmatrix} \frac{T^3}{3} & \frac{T^2}{2} & 0 & 0 \\ \frac{T^2}{2} & T & 0 & 0 \\ 0 & 0 & \frac{T^3}{3} & \frac{T^2}{2} \\ 0 & 0 & \frac{T^2}{2} & T \end{pmatrix} l \tag{5.4}$$

where $l = 1 \times 10^{-4} \mathrm{m}^2\mathrm{s}^{-3}$.

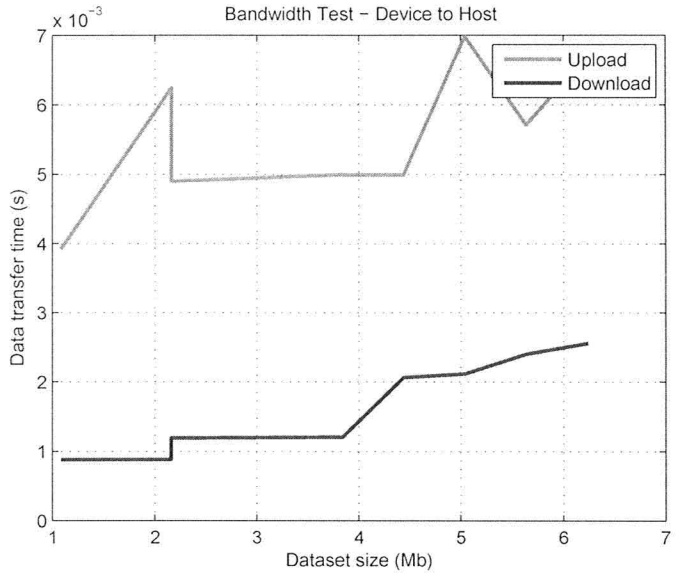

Figure 5.1: GPU bandwidth grpah

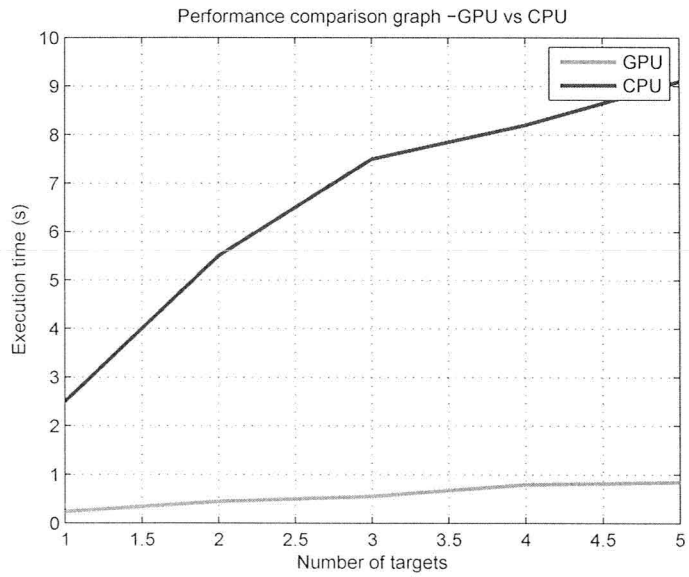

Figure 5.2: Performance comparison graph of GPU vs CPU

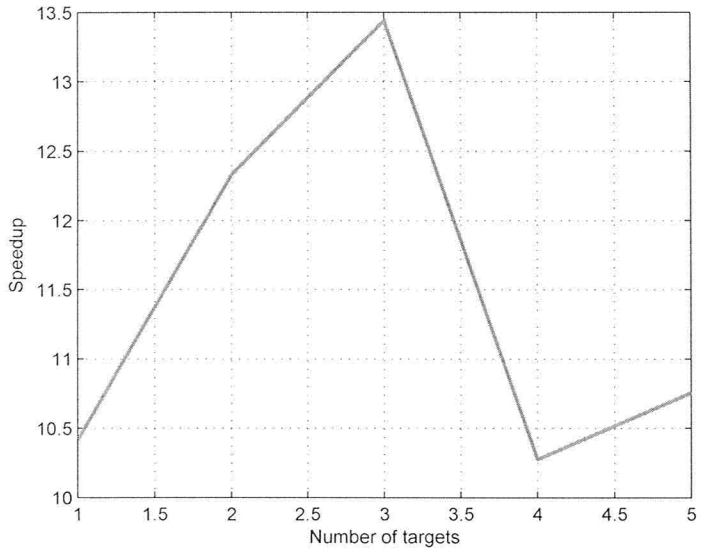

Figure 5.3: Efficiency of parallel MFA algorithms

5.3 Results

This simulation considered a varying number of targets in the scenario to illustrate the MFA parallelization efficiency. Simulation results show the improved speedup of the CPU-GPU integrated environment when compared to the traditional CPU only computational environment.

The GPU parallel platform that is used to analyze the performance of the MFA algorithm consists of a NVIDIA GeForce 285GTX as graphics card (GPU Device). This card has a 240 graphics processor clocked at 1476MHz, 1GB GDDR3 video memory clocked at 1242MHz and the data transfers between CPU and GPU device occur through the PCI-Express 16x interface, which is the connecting bridge between the host (CPU) and device (GPU).

Figure 5.1 shows the communication characteristics of the parallel architecture on

which simulations are performed. The communication speedup is calculated from the communication characteristics graph.

As the dimension of the state vector of the target increases, the data transmission between the host (CPU) and device (GPU) increases linearly. However, the computational requirement for each target increases exponentially due to the data association in each track.

The simulation results presented in Figures 5.2 shows that the computational time of parallel MFA implementation on CPU-GPU integrated environment is always below the direct implementation on CPU only environment. This time reduction is achieved by reducing the huge amount of data transmission at each time step using PTO algorithm.

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This thesis considered the parallelization of a MFA tracker for multitarget tracking problems. The high computational load of the standard MFA tracker, which typically consists of vectors, is made tractable for real-time applications through parallelization in a primary-secondary architecture using optimization techniques.

This work focused on how to exploit computation performance in CPU-GPU integrated environment. The contributions of the work presented in this paper are two-fold. First, this work presents CPU-GPU integrated computing model and algorithm to optimize traditional implementation of MFA on GPU computing model. Second, this work analyzes the performance of the parallel task optimization methods to prove the model and algorithm. The results of the experiments indicate that our parallel task algorithm can better adapt to the collaboration of heterogeneous

computing processors.

Furthermore, the proposed Parallel Optimization Algorithm (PTO) design method is shown to be more efficient in terms of resource utilization. In the parallel MFA algorithm using PTO design technique, the data transfer between the host computer (CPU) and the parallel computation hardware device (GPU) is reduced significantly without any apparent degradation in tracking performance. However, the proposed PTO algorithm makes the overall algorithm efficient and real-time feasible.

6.2 Future Work

In the future, the performance of multiple CPUs and GPUs in an integrated environment could be evaluated, and the optimization of load balancing to achieve higher performance cab be analyzed. Testing of the parallel MFA algorithm on different CPU-GPU integrated environments and exploring the new performance opportunities offered by newer generations of CPUs and GPUs would be more beneficial. It would also be interesting to test parallel MFA algorithm on large number of targets to get more experimental results. Finally, another interesting direction is exploring this parallel MFA algorithm on multiple GPGPUs.

Bibliography

C. Thompson, S. Hahn, and M. Oskin (2002). A framework and analysis of modern graphics architectures for general-purpose computing. *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pages 215–226.

C.H. Pin and H.L. Sheng (2001). Taxonomy of Task Scheduling in Parallel and Distributed Computing. *Computer Science.*

D. Luebke, M. Harris, J. Kruger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, and A. Lefohn (2004). Estimation with Applications to Tracking and Navigation. *Gpgpu: general purpose computation on graphics hardware.*

F. Wu, M. Cabral and J.Brazelton (2010). High Performance Matrix Multiplication on General Purpose Graphics Processing Unit. *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference.*

I. Buck (2005). Taking the plunge into GPU computing. *GPU Gems 2.*

J. Owens, D. Luebke, N. Govindaraju, M. Harris, J.Kruger, A. Lefohn, and T. Purcel (2005). A survey of general-purpose computation on graphics hardware. *Eurographics 2005, State of the Art Reports.*

J.D. Owens, D. Luebke, and N. Govindaraju (2007). A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, **41**, 80–113.

K. N. Levitt and W. T. Kautz (1972). Cellular arrays for the solution of graph problems. *Proceedings of the 35th Annual International Symposium on Microarchitecture*, **50**.

L. Wang, Y.Z Huang, X. Chen, and C.Y. Zhang (2008). Task Scheduling of Parallel Processing in CPU-GPU Collaborative Environment. *Computer Science and Information Technology,International Conference*, pages 228 – 232.

M. Ekman, F. Wang, and J. Nilsson (2005). An in-depth look at computer performance growth. *SIGARCH Comput. Archit.News*, **33**, 144–147.

M. S. Arulampalam, S. Maskell, N. Gordon and T. Clapp (2002). A tutorial on particle filters for online nonlinear or non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing.*

N. Govindaraju, S. Larsen,J. Gray and D. Manocha (2007). A Memory Model for Scientific Algorithms on Graphics Processors. *SC 2006 Conference, Proceedings of the ACM/IEEE*, page 6.

NVIDIA Corporation (2010). Taking the plunge into GPU computing. *NVIDIA Programming Guide 3.0. Internet: www.nvidia.com.*

Oxford University (2010). GPU Stochastic Simulation for statistical data analysis. *Tutorial. Internet: http://www.oxford-man.ox.ac.uk/gpuss.*

R. Tharmarasa, S. Sutharsan and T. Kirubarajan (2009). Multiframe Assignment Tracker for MSTWG Data. *Seattle, WA, USA.*

R. Tharmarasa, T. Kirubarajan, M.L. Hernandez and A. Sinha (2007). PCRLB-Based Multisensor Array Management for Multitarget Tracking. *IEEE Transactions on Aerospace and Electronic Systems*, **43**, 539–551.

R.L. Popp, K.R. Pattipati, Y. Bar-Shalom, and R.R. Gassner (1998). An adaptive m-best SD assignment algorithm and parallelization for multitarget tracking. *Aerospace Conference, 1998. Proceedings., IEEE*, **5**, 71 – 84.

Y. Bar-Shalom and X. R. Li (1995). Multitarget-Multisensor Tracking: Principles and Techniques. *YBS Publishing*.

Y. Bar-Shalom, T. Kirubarajan and C.Gokberk (2005). Tracking with classification-aided multiframe data association. *Aerospace and Electronic Systems, IEEE Transactions*, **41**, 868.

Y. Bar-Shalom, X. Li and T. Kirubarajan (2001). Estimation with Applications to Tracking and Navigation. *John Wiley and Sons*.