International Journal of Grid and Distributed Computing
Vol. 9, No. 12 (2016), pp.143-158
http://dx.doi.org/10.14257/ijgdc.2016.9.12.13

An Effective FastSLAM Algorithm Based on CUDA

Heng Zhang[1], Yanli Liu[1], Mengyu Zhu[1], Naixue Xiong[2,3], Tai-hoon Kim[4]

[1]*School of Information Engineering, East China Jiaotong University, Nanchang, China*
[2]*School of Computer, Hubei University of Education, Wuhan, P.R. of China*
[3]*Dept. of Business and Computer Science, Southwestern Oklahoma State University, USA*
[4]*Department of Multimedia Engineering, Hannam University, Daejeon, Korea*
E-mail: hbzhangheng@126.com

Abstract

Compute Unified Device Architecture (CUDA) is a mature parallel computing architecture, which can significantly accelerate performance of the computation intensive algorithm. In this paper, FastSLAM algorithm based on the probability model is further studied and the resampling algorithm for the path estimation is improved. In the resampling phase, resampling rules are redesigned and the previous data limitations are broken for the purpose of parallelization. We propose the FastSLAM algorithm based on CUDA, which accelerates robot localization and mapping. The experiment results show that FastSLAM_CUDA can achieve a significant speedup over the FastSLAM with many particles.

Keywords*: compute unified device architecture (CUDA); simultaneous localization and mapping (SLAM); resampling algorithm; parallel computing*

1. Introduction

Autonomous localization and mapping in an unknown environment is an important problem for mobile robots. It is described as SLAM (Simultaneous Localization and Mapping). In an unknown environment, the mobile robots get the real-time data of environment by sensor, estimating posture of robot and state of environment simultaneously during map building in the motion model and observation model, finally realizes the function of localization and navigation. Clipp B et al.[1] proposed that using stereo cameras as sensors achieve real-time visual SLAM. Jia S et al.[2] proposed a parallel particle filter SLAM based on OpenMP. In this paper, a multi-thread particle filter technique is proposed to reduce the computation time and the execution time of SLAM. This method allows the PF-SLAM to run smoothly and ensures accuracy at the same time. Vincke B et al.[3] proposed using OMAP multi-core architecture and SIMD technology optimization in the embedded system, accelerated EKF-SLAM in order to achieve the goal of improving the localization quality. Jin S et al.[4] proposed the design of accelerated matrix multiplication using CUDA to achieve the acceleration of 3D EKF SLAM. Zhang H et al.[5] proposed the use of FPGA to accelerate localization and mapping, the weight calculation in particle filtering process and the memory copied from the host side to the device side. Montemerlo M and Thrun S et al.[6] proposed FastSLAM which is widely studied because of low complexity of the algorithm, low update dimension, easy implementation and so on. This paper combines with the technology of CUDA based on the study of FsatSLAM in depth. For the resampling phase during the process of path estimation, the paper redesigns sampling rules, breaks the original data dependencies, makes it satisfy the data parallel condition and proves the feasibility of new

ISSN: 2005-4262 IJGDC

sampling rules. The experimental results illustrate that FastSLAM based on CUDA can improve the computational efficiency greatly.

Based on the FastSLAM algorithm, this paper studies the resampling process and the landmark estimation process, and improves the algorithm in order to implement on CUDA. The improved algorithm takes full advantage of CPU and GPU resources to achieve the purpose of speeding up. This research effectively makes use of the computing power of CUDA and let CPU focus on more complex logic calculations, which improve the utilization of the machine and promote the overall performance of the robot.

2. CUDA Technology

CUDA is a general-purpose parallel computing architecture introduced by NVIDIA Corporation. It enables the GPU to solve the computation-intensive problem well [7, 8]. It could be more reasonable about allocation of computing resources between GPU and CPU under the platform of CUDA. Numerous graphical computing is processed by GPU instead of CPU which is good at data caching and stream processing.

One CUDA program is usually composed of a host program and a kernel function. The code is executed by the host program in the CPU, which includes preparations before the kernel starts, initialization of data and data exchange between kernels. Kernel is a piece of code executed in GPU side and a computational task that can be executed in parallel.

The process of CUDA program is a heterogeneous schema in co-processing of CPU/GPU. And it uses the CPU as the host, GPU as the device. One host can works together with multiple devices in the system. The tasks are divided explicitly between CPU and GPU. Logical tasks and the tasks in high serialization are processed by CPU, but the parallel tasks in high threads are processed by GPU. The programming model of the whole CUDA is dealt with in the form of serial and parallel interleaving with each other: when a parallel task occurs, the host takes the task to the device through calling a kernel function. In an optimal environment of parallel, the work of serial code in the host is only clearing the last completed kernel and calling the next kernel. The execution model of CUDA is illustrated in Figure 1.

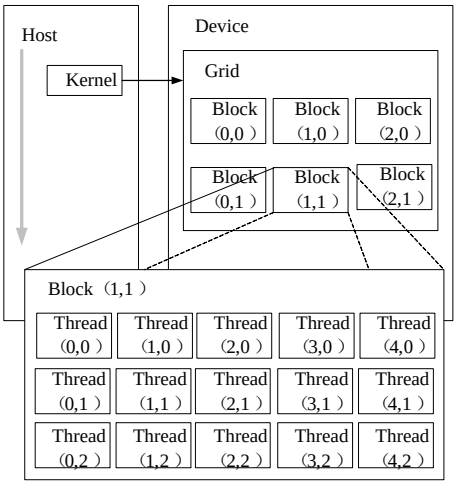


Figure 1. CPU/GPU Cooperative Computing Model

When a kernel function is mapped to the GPU, it is called a Grid. Every Grid is composed of one or more blocks, and every block has a blockIdx in order to recognize the only block. Every block is also composed of one or more threads, and every thread has a threadIdx to recognize the only thread. According to the blockIdx and threadIdx, every thread can access memory and video memory in a certain range including communication

with other threads. The partition of data in various domains (vector, matrix, space) is become visualized because of this thread form.

3. FastSLAM Algorithm

3.1 Probabilistic SLAM

Let s_t is the current pose of robot at time t, when the robot moves in the plane, its motion model can be described as:

$$p(s_t \mid u_t, s_{t-1}) \tag{1}$$

where the distribution is the conditional probability. The role is to control u_t and the pose on previous moment named s_{t-1}.

The robot obtains the distance and the orientation of the environment and locates the environment by sensing landmarks. The landmark and its local coordinate system with respect to the environment are denoted by θ. The observation at time t is denoted by z_t, and the robot can observe many landmarks at time t. When several landmarks are observed in a moment, the serialization can be derived. The observation model can be described as:

$$p(z_t \mid s_t, \theta, n_t) \tag{2}$$

where $\theta = \{\theta_1, \cdots, \theta_k\}$ is a collection of landmarks in the environment, and $n_t \in \{1, \cdots, K\}$ is data association of landmark observation at time t.

In general, the problem of SLAM is considered the pose s_t under the location q, the observation $z^t = z^1, \cdots, z^t$ and the control inputs $u^t = u^1, \cdots, u^t$. It is expressed in form of the probability as $p(s^t, \theta \mid z^t, u^t)$. If the corresponding landmark has been known, the problem of SLAM can be simply described as:

$$p(s^t, \theta \mid z^t, u^t, n^t) \tag{3}$$

3.2 FastSLAM

The general process of FastSLAM is shown in Figure 2, and the assumptions need to be made as follows: 1) all landmarks can be estimated independently under the circumstance of path s^t for robot and the corresponding amount of landmarks n^t; 2) the distance and orientation of the K landmarks have been measured. So according to Rao-Blackwellized[9] theorem, the Eq. (3) can be factored as:

$$\begin{aligned} & p(s^t, \theta \mid z^t, u^t, n^t) \\ & = p(s^t \mid z^t, u^t, n^t) \prod_k p(\theta_k \mid s^t, z^t, u^t, n^t) \end{aligned} \tag{4}$$

In fact, this is equivalent to $K+1$ estimation problems including an estimation of robot path s^t and K estimation problems of landmarks in condition of path estimation.

In the algorithm of FastSLAM, it uses improved particle filtering for path estimation $p(s^t, \theta \mid z^t, u^t, n^t)$ and adopts EKF (Extended Kalman Filter) to complete location estimation of landmarks $p(\theta_k \mid s^t, z^t, u^t, n^t)$. Different landmarks use different filters. There are $K \times M$ EKF for M particles and K landmarks, and every particle has two dimensions.

1. The Path Estimation of Particle Filtering

At each time point, these two algorithms ensure that the set of particle represents the posterior probability $p(s^t, \theta | z^t, u^t, n^t)$. Every particle $s^{t,m} \in S_t$ represents an estimation of the robot path:

$$S_t = \{s^{t,[m]}\}_m = \{s_1^{[m]}, s_2^{[m]}, \cdots, s_t^{[m]}\}_m \tag{5}$$

Superscript m denotes the mth particle in the set. S_t is computed by the set S_{t-1} at time $t-1$, the control input u_t and the observation z_t. Firstly, every particle in S_{t-1} is used to estimate the pose at time t :

$$s_t^{[m]} \sim p(s_t | u_t, s_{t-1}^{[m]}) \tag{6}$$

It is obtained by the motion model of sampling probability, which is added to a temporary particle set and obtained the path $s^{t-1,[m]}$. Assuming the distribution of the particles in set S_{t-1} obeys $p(s^{t-1} | z^{t-1}, u^{t-1}, n^{t-1})$, the distribution of new particles obeys $p(s^t | z^{t-1}, u^t, n^{t-1})$. This distribution is commonly referred to as the proposed distribution of particle filter.

New set S_t is sampled from the temporary set of particles after generating M particles in this way. Every particle $s^{t,[m]}$ is represented as the weighting function $w_t^{[m]}$, it can be called acceptance function. The computing method is as follows:

$$w_t^{[m]} = \frac{\text{object distribution}}{\textit{proposal distribution}} = \frac{p(s^{t,[m]} | z^t, u^t, n^t)}{p(s^{t,[m]} | z^{t-1}, u^t, n^{t-1})} \tag{7}$$

The set of resampling S_t approximates posterior probability of target pose $p(s^t | z^t, u^t, n^t)$, and this approximation will be correct when the number of particles tends to infinity. We can observe that it only uses the nearest pose $s_{t-1}^{[m]}$ of robot to estimate when S_t is generated. So we can ignore the particle pose of other time series.

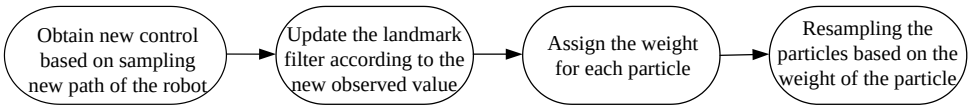


Figure 2. General Process of FastSLAM Algorithm

2. Location estimation of landmarks

All posterior probability of the path and location of landmarks can be shown as the sample set.

$$S_t = \{S^{t,[m]}, \mu_1^{[m]}, \Sigma_1^{[m]}, \cdots, \mu_K^{[m]}, \Sigma_K^{[m]}\}_m \tag{8}$$

Where $\mu_k^{[m]}$ and $\Sigma_k^{[m]}$ represent Gauss mean and covariance of the k th landmark that belong to the mth particle. In the location scene of planar robot, each mean $\mu_k^{[m]}$ is a bivector, and $\Sigma_k^{[m]}$ is a 2×2 matrix. The acquisition of the posterior pose θ_k of the k-th landmark depends on whether n_t equals k at time t .If $n_t = k$, we can get:

$$\begin{aligned} p(\theta_k | s^t, z^t, u^t, n^t) &\overset{\text{Bayes}}{\propto} p(z^t | \theta_k, s^t, z^{t-1}, u^t, n^t)\, p(\theta_k | s^t, z^{t-1}, u^t, n^t) \\ &\overset{\text{Markov}}{=} p(z_t | \theta_k, s_t, n_t)\, p(\theta_k | s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \end{aligned} \tag{9}$$

If $n_t \neq k$, it means the probability that no landmark is observed at time t. Since there are no new landmark observations and landmarks location at time t and time $t-1$ have no change, so the function can be shown as:

$$p(\theta_{n_t \neq k} \mid s^t, z^t, u^t, n^t) \overset{\text{Markov}}{=} P(\theta_k \mid s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \tag{10}$$

The FastSLAM algorithm updates Eq. (9) using EKF (Extended Kalman Filter). In the observation model of linear Gaussian, the nonlinear motion model is still Gaussian distribution.

3. Calculate the Weight

The weights in the particle filter are calculated as follows:

$$\begin{aligned} w_t^{[m]} &\propto \frac{p(s^{t,[m]} \mid z^t, u^t, n^t)}{p(s^{t,[m]} \mid z^{t-1}, u^t, n^{t-1})} \\ &\overset{\text{Bayes}}{=} \frac{p(z_t, n_t \mid s^{t,[m]}, z^{t-1}, u^t, n^{t-1})}{p(z_t, n_t \mid z^{t-1}, u^t, n^{t-1})} \\ &\overset{\text{Markov}}{=} \int p(z_t, n_t \mid \theta, s_t^{[m]}) p(\theta \mid s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1}) d\theta \\ &= \int p(z_t \mid \theta, s_t^{[m]}, n_t) p(n_t \mid \theta, s_t^{[m]}) p(\theta \mid s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1}) d\theta \\ &\propto \int p(z_t \mid \theta, s_t^{[m]}, n_t) p(\theta \mid s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1}) d\theta \\ &\overset{\text{EKF}}{\approx} \int p(z_t \mid \theta_{n_t}^{[m]}, s_t^{[m]}, n_t) p(\theta_{n_t}^{[m]}) d\theta \end{aligned} \tag{11}$$

In our hypothesis, the distribution $p(n_t \mid \theta, s_t^{[m]})$ is a uniform distribution. EKF approximately observes the model $p(z_t \mid \theta_{n_t}^{[m]}, s_t^{[m]})$ using the linearized model and estimates $p(\theta_{n_t}^{[m]})$ using Gauss posterior.

4 FastSLAM Algorithm Based on CUDA

4.1 Algorithm Description of Resampling

Since the particles of robot path are not sampled on the real path, each particle is weighted by importance to show the difference. The new particle set S_t is resampled from the weight sequence of particle set. So, Resampling is useful to reflect the true path of the particle. The computation of resampling process is complex. Resampling is only a tiny part of landmark estimation in the process of FastSLAM, and takes much less time in the whole process of SLAM. But it can be seen from Eq. (4) that the estimation of landmarks must be completed on the premise of path estimation. The estimation of path has $K \times M$ EKF processes. So it is significant to study the acceleration of resampling for improve overall pace. Algorithm 1 is the resampling algorithm of general system.

Algorithm 1 resampling algorithm of system

Input: A list of samples $\{s_i[k], k=1,2\cdots N\}$ and corresponding weigh $w_i[k]$.

Output: New sample sequence $s_o[k]$ and corresponding weigh $w_o[k]$.

1 Draw $u \sim U(0, \frac{1}{N}]$ // u is a uniform random number.
2 $c[1] = w_i[1]$ // $c[k]$ is cumulative sum of weigh $w_i[k]$.
3 for $k=2$ to N do
4 $c[k] = c[k-1] + w_i[k]$ // cumulative sum.
5 end for
6 $j=1$ // Starting with the first element.
7 for $k=1$ to N do
8 $u_k = u + \frac{k-1}{N}$
9 while $u_k > c[j]$ do
10 $j = j+1$
11 end while
12 $s_{o[k]=s_i[j]}$
13 $w_o[k] = \frac{1}{N}$
14 end for

4.2 Analysis and Improvement of Parallelization

The main reason that resampling of system doesn't run in parallel is existing data dependency from the 11th to the 13th step in the algorithm 1(SR algorithm). In order to realize the parallelization, we redesign the regulation of resampling in the new algorithm, break original limits of data dependency, and make it possible to accelerate. We improve this algorithm. The specific design and analysis are as follows.

1) The uniform random number is generated by CurandGenerateUniformDouble function in CURAND library.

2) Parallel sweeping algorithm.

There is a dataset $X=[x_0, x_1, \cdots x_{n-1}]$. We introduce the concept of prefix sum[10] in order to compute prefix sum. The reduced from of $X=[x_0, x_1, \cdots x_{n-1}]$ is defined as $x_0 \oplus x_1 \oplus x_2 \oplus \cdots\cdots x_{n-1}$, where $\oplus$ is a binary associative operator. The operator maybe identical equation, addition or multiplication. For the sake of simplicity, the algorithm uses an addition operation, so it can be obtained the result array $[I, x_0, x_0+x_1, \cdots, x_0+x_1+x_2+\cdots x_{n-2}]$. The algorithm is divided into up-sweep (lines 2-6 in algorithm 2) and down-sweep (lines 7-14 in algorithm 2) [11, 12]. The algorithm uses $2\times\log_2(n)$ step to scan x. Reducing x on the up-sweep phase, the active thread only has half of the first $i-1$ iteration during the first $i(i=0,1,2,\cdots,\log_2(n)-1)$ iteration, and the distance of additive elements is 2^i. The down-sweep phase is the inverse process of up-sweep phase. There are 2^i active threads, and the distance of additive elements is $n/(2^{i+1})$ (n is the total number of elements). The movement of the element is also done

by the active thread. The algorithm of up and down sweep is explained in Fig. 3 In Fig. 3(a), there are four active elements in the 0 iteration (the summed elements are represented by Σ In the figure 3), two active elements in the first iteration and only one active element in the second iteration; Fig. 3 (b) is down-sweep phase, there are one active element in the 0 iteration, two active elements in the first iteration and four active elements in the second iteration.

3) In this algorithm, we designs the left and right boundaries, the specific definition is as follows:

$$l = \lfloor (c[k] - u) \times N \rfloor + 2 \tag{12}$$

$$r = \lfloor (c[k] + w_i[k] - u) \times N \rfloor + 1 \tag{13}$$

when $l \le r$, resampling, otherwise do not carry out any operation.

We can briefly demonstrate the parallelism of the new resampling rule. And it's not difficult to find that is to prove the validity of Eq. (14) and Eq. (15).

$$l = l_1, r_N = N \tag{14}$$

$$r_k < l_{k+1}; k = 1, 2, \cdots, N-1 \tag{15}$$

The proving process of Eq. (14) and Eq. (15) are as follows:

a) The proof of Eq. (14).

According to the calculation rules of the prefix sum, $c[k]$ is the prefix sum of $w_i[k]$. In the algorithm:

$$c[1] = 0 \tag{16}$$

$$c[k] = c[k-1] + w_i[k-1] \tag{17}$$

$$c[N] + w_i[N] = 1 \tag{18}$$

Because of $u \sim U(0, 1/N]$, u is a uniform random number. Eq. (16) and Eq. (17) are integrated to derive the following formula:

$$l_1 = \lfloor (c[1] - u) \times N \rfloor + 2 = 1$$

$$r_N = \lfloor (c[N] + w_i[N] - u) \times N \rfloor + 1 = N$$

b) The proof of Eq. (15).

$$\because \quad r_k = \lfloor (c[k] + w_i[k] - u) \times N \rfloor + 1$$

$$= \lfloor (c[k+1] - u) \times N \rfloor + 1$$

$$l_{k+1} = \lfloor (c[k+1] - u) \times N \rfloor + 2$$

where $k = 1, 2, \cdots, N-1$.

$$\therefore r_k < l_{k+1}; k = 1, 2, \cdots N-1.$$

4) The detailed flow chart of parallel resampling algorithm refers to Algorithm 2:

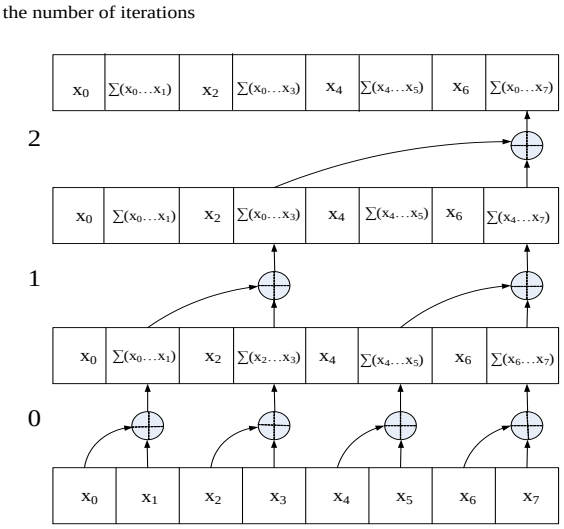
Algorithm 2 parallel resampling algorithm of system

Input: A list of samples $\{s_i[k], k = 1, 2 \cdots N\}$ and corresponding weigh $w_i[k]$.

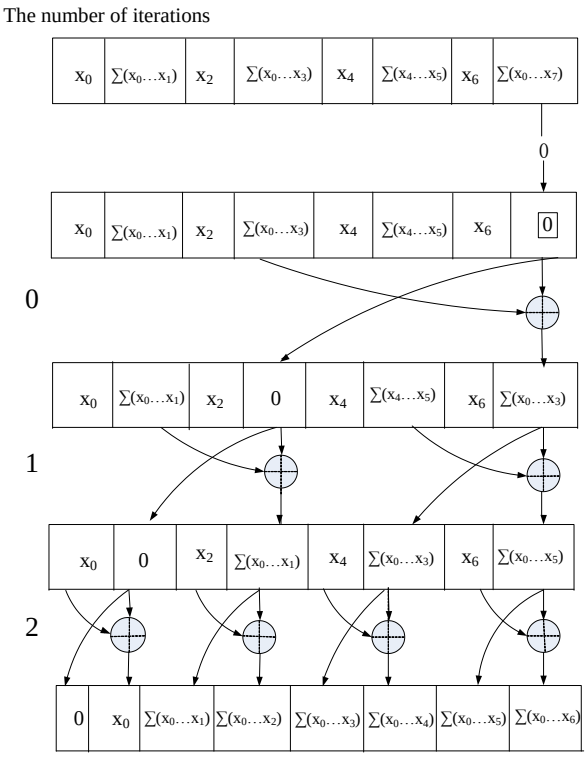
Output: New sample sequence $s_o[k]$ and corresponding weigh $w_o[k]$.

1 Draw $u \sim U(0, \frac{1}{N}]$ in parallel // u is a uniform random number.
2 for $d = 0$ to $\log_2(N) - 1$ do // up-sweep algorithm
3 for k from 0 to $N - 1$ by 2^{d+1} in parallel do
4 $x[k + 2^{d+1} - 1] = x[k + 2^d - 1] + x[k + 2^{d+1} - 1]$
5 end for
6 end for
7 $x[N - 1] = 0$ // down-sweep algorithm
8 for $d = \log_2(N) - 1$ to 0 do
9 for k from 0 to $N - 1$ by 2^{d+1} in parallel do
10 $t = x[k + 2^d - 1]$
11 $x[k + 2^d - 1] = x[k + 2^{d+1} - 1]$
12 $x[k + 2^{d+1} - 1] = t + x[k + 2^{d+1} - 1]$
13 end for
14 end for
15 for $k = 1$ to N by 1 in parallel do
16 $l = \lfloor (c[k] + w_i[k] - u) \times N \rfloor + 1$ // left boundary
17 $r_k = \lfloor (c[k] + w_i[k] - u) \times N \rfloor + 1$ // right boundary
18 for $j = l$ to r do // $l \le r$ resampling
19 $s_o[j] = s_i[k]$
20 $w_o = \frac{1}{N}$
21 end for
22 end for

Algorithm 1 (SR algorithm) can't be realized by the method of parallel processing because of data dependency of lines 9 to 11. It's easy to realize the parallel processing for algorithm 2 because the data dependency has been eliminated after the prefix sum computed. The computation of the prefix sum is realized through parallel scanning algorithm. It uses two boundaries (boundary of left and right) to obtain the storage location of viable particle and their number of copy. The two boundaries are computed by prefix sum of particle weight and uniform random number. These two variables eliminate the dependency between data and keep the parallelism of algorithm 2. Transmission cost is reduced because all steps of particle filter are parallelized.



(a) up-sweep phase



(b) down-sweep phase

Figure 3. Parallel Exclusive Scan Algorithm

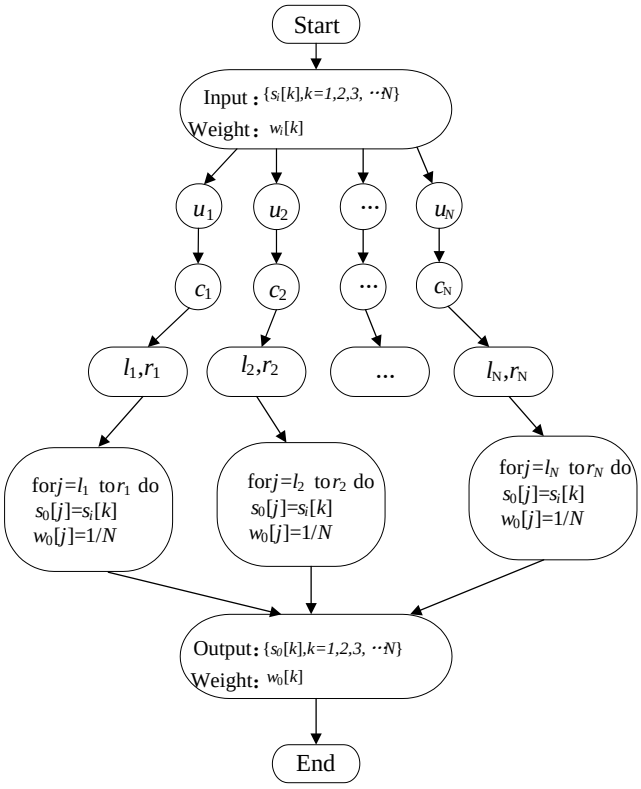


Figure 4. Parallel Scan Algorithm

5 Simulation Experiment and Analysis

5.1 Experimental Object and Experimental Environment

In the experiment, we adopt the way that described in algorithm 2 to resample. FastSLAM 2.0 controls the number of particles. This experiment compares the operation speed between pure CPU and the GPU under acceleration, and analyzes corresponding efficiency simultaneously.

The hardware environment of this experiment is: GPU is NVIDIA GeForce GTX 960, video memory is 2GB, computing core number of CUDA is 1024, CPU is Intel Core i5 3470, memory is 8GB, CPU clock speed is 3.9GHz. The soft environment is installing CUDA 7.5 under Ubuntu 15.04 X86_64 system. Compile environment is gcc 4.9.2, CMake3.0.2. Dependency of math library C++ standard library Boost 1.55.0, Linear algebra library Eigen 3.1.4. Gperftools 2.1(X86_64 system need to install libunwind-1.1) is used for efficiency analysis of CPU. The tool of efficiency analysis is CUDA with the tool nvprof. The comparisons between GPU and CPU are as shown in Table 1.

Table 1. Comparison between GPU and CPU

GPU	
GPU model	GeForce GTX 960
Max clock rate	1.28 GHz
Memory clock rate	3505 MHz
computing core number of CUDA	1024
video memory	2 G

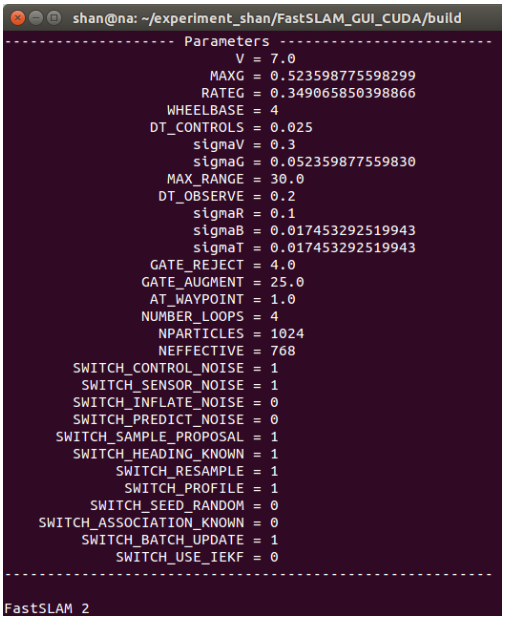
Global memory	4 G
Shared memory	49152 bytes
CPU	
CPU model	Intel Core i5 3470
Clock rate	3.19GB
Core number	2
Memory	8GB

5.2 Experimental Results and Analysis

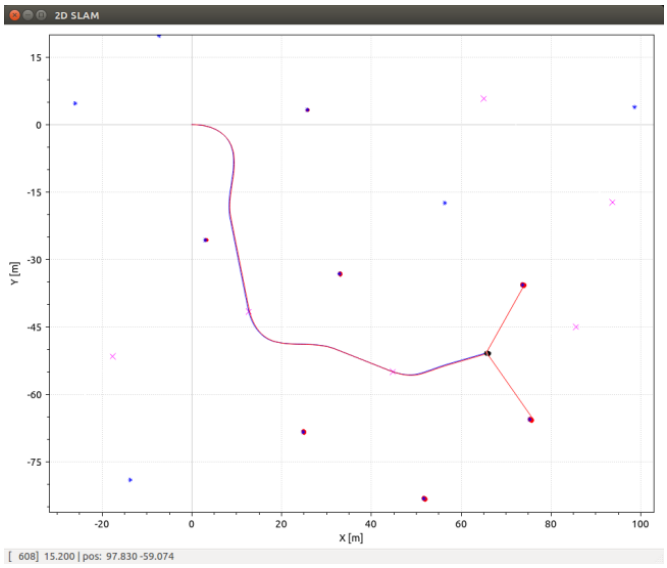
5.2.1 Experimental Results

Fig. 5 shows the effect of FastSLAM 2.0 when the number of particles is 1024. (a) The configuration of FastSLAM operating parameters at the beginning of the algorithm; (b) the distribution of landmarks at the beginning. The landmarks are blue in the figure. After the program runs for a period of time, there are red points. The red points are the estimation points of landmarks, and the red laser lines are denoted to realize the data association. The blue line is the true path and the red line is the estimated path in the figure; (c) the red point in the graph increases gradually, at the same time the data realize association, and determine whether the new observed landmarks belong to the existing sequence landmarks in the environment; (d) The algorithm achieves the first closed-loop, and the red points in the figure increase more. The position uncertainty of the observed landmarks is gradually decreased after the robot completes loop closing movement; (e) the algorithm ends after the fourth loop closing.

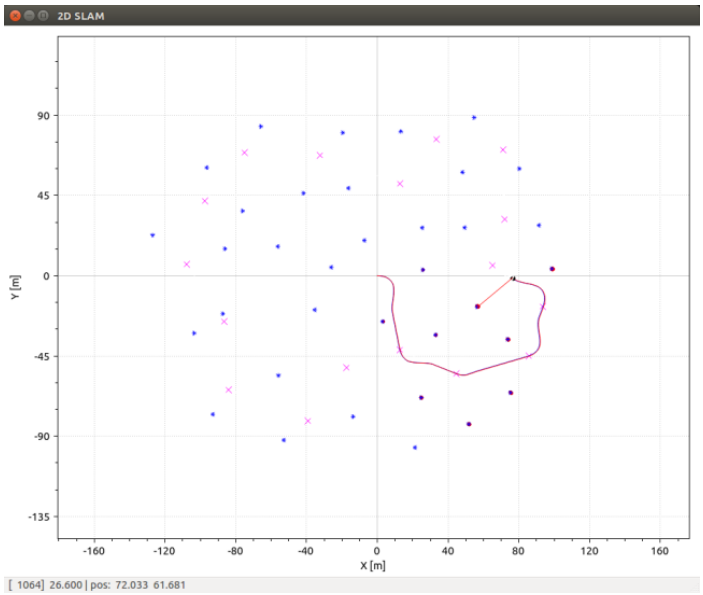
Figure 6 shows that the execution time of FastSLAM 2.0 is proportional to the number of particles in the phase of path estimation. Where the x-axis represents the number of particles and the y-axis represents the execution time. When the number of particles is larger, the difference of execution time between FastSLAM and FastSLAM_CUDA is bigger and bigger, and FastSLAM_CUDA is significantly more efficient than FastSLAM.



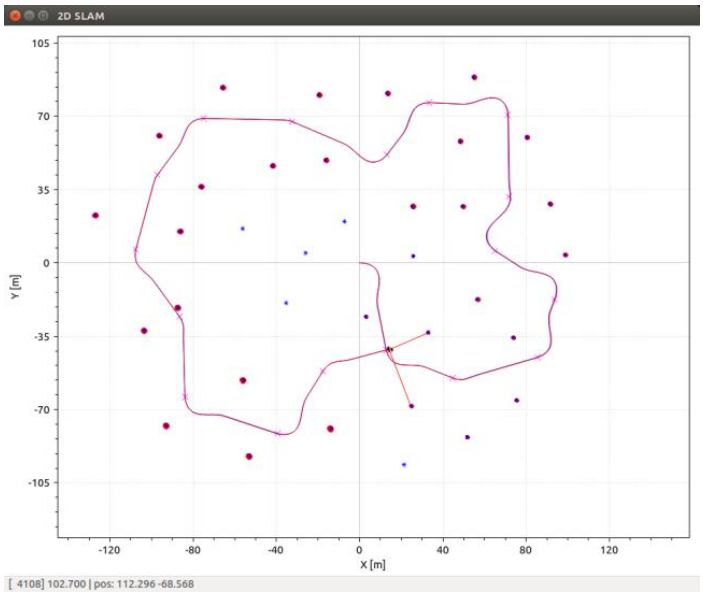
(a) Configuration parameters



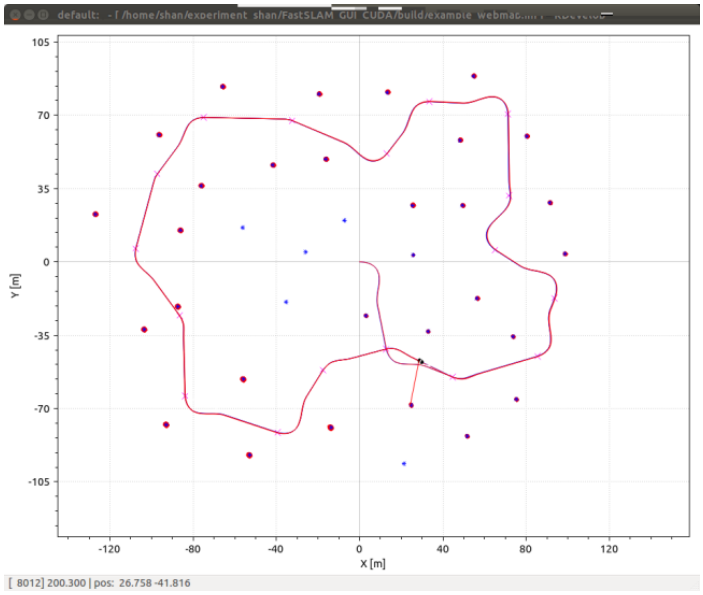
(b) At the beginning of algorithm



(c) After running for some time



(d) The first closed-loop



(e) The fourth closed-loop and algorithm ends

Figure 5. Operation Figures of FastSLAM 2.0

In Figure 5, the blue points represent the landmarks, the red points represent the estimated landmarks, the pink crosses represent the path points, the blue line is the true path, the red line is the estimated path, and the red laser line marks data association.

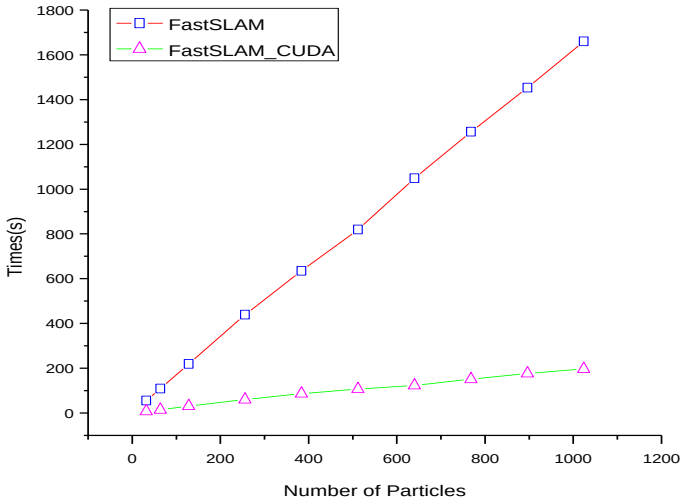


Figure 6. Comparison between FastSLAM_CUDA and FastSLAM

Figure 7 shows the acceleration ratio between FastSLAM and FastSLAM_CUDA. As the number of particles increases, it starts to rebound when it tends to 8 times, but it is always more than 8 times.

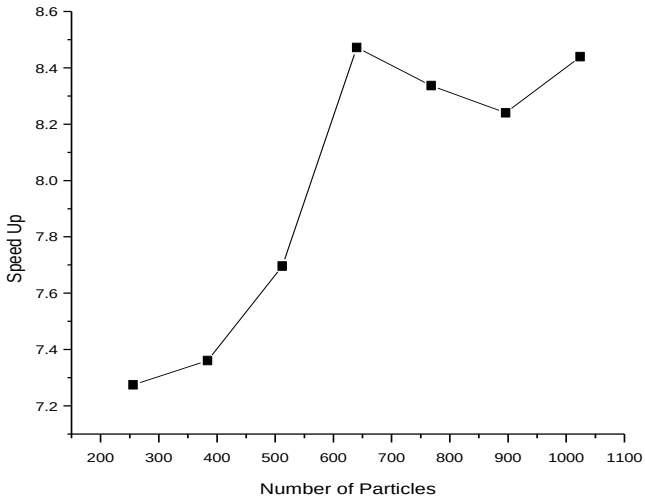


Figure 7. Speed-up Ratio between FastSLAM_CUDA and FastSLAM

5.2.2 Discussion

We process the parallel design on FastSLAM by CUDA. This algorithm makes full use of the multi-core computing power of CUDA and achieves the purpose of accelerating applications.

（1）The algorithm of FastSLAM_CUDA redesign the resampling process of the FastSLAM algorithm to meet the CUDA parallelism. The original resampling method is improved, and then system resampling is selected to replace the original hierarchical sampling. In the system resampling, the sampling rules redesigned, such as

$l = \sum_{c_k < u} N + 2$ and $r = \sum_{c_k < u + w_i} N + 1$. In this design, the data association between the system samples is broken to satisfy the data non-relevancy in the CUDA parallel design, and then CUDA C/C++ is adopted to redesign this part of the program. Extern C is used as the interface and C/C++ code and CUDA code are combined to implement the mixed programming.

（2）The landmark estimation process of FastSLAM largely use EKF to estimate the path. EKF process is a continuation ofKalman Filter, which has a large number of matrix calculations. These calculations are time-consuming. In this algorithm, CUDA is adopted to design the matrix calculation, which greatly improve the efficiency of the program and further accelerate the FastSLAM real-time process.

6 Conclusion

In this paper, we present a FastSLAM algorithm based on CUDA and mainly discuss the improved resampling method. This method breaks the original data dependency and realizes the parallel computation. The simulation experiment is realized by co-processing between CPU and GPU under the platform of CUDA. Experimental results show that FastSLAM_CUDA is significantly more efficient than FastSLAM, and the new algorithm can improve the speed at least 8 times than the original algorithm.

In the following research work, we use Particle Swarm Optimization (PSO) instead of particle filtering because PSO has a better feature in parallel computing. The research in the future will focus on how to implement PSO-FastSLAM[13-15]. At the same time, we will transplant from single-GPU to multi-GPU, which will further accelerate the algorithm.

Acknowledgments

This work is supported by the National Nature Science Foundation of China (Grant No. 61563014, 61663010), the Nature Science Foundation of Jiangxi Province, China (No. 20161BAB202068).

References

[1] B. Clipp, J. Lim and J. M. Frahm, “Parallel, real-time visual SLAM”, Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference, IEEE, **(2010)**.
[2] S. Jia, X. Yin and X. Li, “Mobile robot parallel PF-SLAM based on OpenMP”, Robotics and Biomimetics (ROBIO), IEEE International Conference, IEEE, **(2012)**.
[3] B. Vincke, A. Elouardi and A. Lambert, “SIMD and OpenMP optimization of EKF-SLAM”, Multimedia Computing and Systems (ICMCS), International Conference, IEEE, **(2014)**.
[4] S. Jin, J. Cho and X. D. Pham, “FPGA design and implementation of a real-time stereo vision system”, Circuits and Systems for Video Technology, IEEE Transactions, vol. 20, no. 1, **(2010)**, pp. 15-26.
[5] H. Zhang and F. Martin, “CUDA accelerated robot localization and mapping”, Technologies for Practical Robot Applications (TePRA), IEEE International Conference, IEEE, **(2013)**.
[6] M. Montemerlo, S. Thrun and D. Koller, “FastSLAM: A factored solution to the simultaneous localization and mapping problem”, AAAI/IAAI, **(2002)**, pp. 593-598.
[7] Z. Yonglong, M. Kuizhi and J. Xiang, "Parallelization and optimization of SIFT on GPU using CUDA", High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), IEEE 10th International Conference, IEEE, **(2013)**.
[8] M. A. Chao, C. Y. Chu and C. H. Chao, “Efficient parallelized particle filter design on CUDA”, Signal Processing Systems (SPS), IEEE Workshop, IEEE, **(2010)**.
[9] B. D. Gouveia, D. Portugal and L. Marques, “Speeding up rao-blackwellized particle filter SLAM with a multithreaded architecture”, Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on. IEEE, **(2014)**.

[10] M. Harris, S. Sengupta and J. D. Owens, “Parallel prefix sum (scan) with CUDA”, GPU gems, vol. 3, no. 39, **(2007)**, pp. 851-876.
[11] M. Harris, “Optimizing parallel reduction in CUDA”, NVIDIA Developer Technology, vol. 2, no. 4, **(2007)**.
[12] B. E. Peng Gong, “Parallelizing Particle Filtering and Enabling WCET Analysis on Multi-Core Architectures for Hard Real-Time Systems”, The Ohio State University, **(2012)**.
[13] H. C. Lee, S. K. Park and J. S. Choi, “PSO-FastSLAM: an improved FastSLAM framework using particle swarm optimization”, Systems, Man and Cybernetics, SMC, IEEE International Conference, IEEE, **(2009)**.
[14] A. Ouyang, Z. Tang and X. Zhou, " Parallel hybrid PSO with CUDA for ID heat conduction equation”, Computers & Fluids, vol. 110, **(2015)**, pp. 198-210.
[15] T. Lan, M. Guo and J. Qu, "CUDA-based hierarchical multi-block particle swarm optimization algorithm”, Control and Decision Conference (CDC), 27th Chinese, IEEE, **(2015)**.

