A Novel Approach to Real-Time Processing: Implementing Complex Calculations on GPU

Seyed Ali Salehi Neyshabouri, Mohammad Reza Niknezhad, Ehsan Kamali

Iran University of Science and Technology
University St., Hengam St., Resalat Ave., Tehran, Iran

{nysalehi, mrniknejad, kamali.ehsan}@gmail.com

Abstract **- Since the very beginning days of developing intelligent robots, achieving higher speed in data processing has been one of the most problematic challenges for designers. Intelligent robots have always been coupled with heavy and complex calculations, which are vital for them to be computed in the least time, achieving the least delay. Regular processors known as CPUs leave designers with no option but to replace accurate and expensive algorithms with inaccurate and easy to compute ones to achieve the ultimate goal which is real-time processing. This paper suggests a method to boost data processing speed by executing some result-independent functions on the GPU. This method results a dramatic improvement in processing speed, promising emersion of some powerful but expensive, thus abandoned algorithms.**

I. Introduction

In computer science, real-time computing (RTC), or "reactive computing", is the study of hardware and software systems subjected to a "real-time constraint"—i.e., operational deadlines from event to system response. Such tasks were executed on CPU which forced programmers to either ignore maximum accuracy or utilize fast enough but expensive processors which don't solve the problems. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for a range of complex algorithms.

This paper aims to use highly parallel structure to speed up calculation of algorithms used in AI and image processing. Results of the application of the method on some algorithms like field evaluation, edge detection, and potential-field path planner on various CPUs and GPUs are illustrated.

II. Problem Description

Image-processing and artificial intelligence are some of complex challenges that robot designers in various fields encounter. Most of these challenges are satisfactorily answered, but only in static domains. Dynamic domains require real-time processing, which due to the complexity of calculations which such algorithms are coupled with, is hard to achieve. This problem is so serious that some advanced AI methods have not yet been implemented even years after development. To overcome this difficulty the designers should inevitably utilize very fast processors or replace the original algorithms with simplified but less accurate ones. The later is not ideal and in some cases it even makes the solution impractical. Using fast processors has proven not to be the satisfying procedure, even using fastest chips available in the market.

III. Calculations on GPU using CUDA

Driven by the insatiable market demand for real-time, high-definition 3D graphics, the programmable Graphic Processor Unit or GPU has evolved into a highly parallel, multithreaded, many-core processor with tremendous computational power and very high memory bandwidth.
In November 2006, nVidia® introduced CUDA™, a general purpose parallel computing architecture – with a new parallel programming model and instruction set architecture – that leverages the parallel compute engine in nVidia GPUs to solve many complex computational problems in a more efficient way than on a CPU [9]. In comparison with CPU, graphic cards have much more cores each having a separate memory unit which results in reducing the flow control for a single process.
CUDA comes with a software environment that allows developer to use C as a high-level programming language. It lets the programmer to get the desired results from some heavy and complex tasks much faster and allows some algorithms to be implemented in real-time which were almost abandoned due to complexity and the time required to get reasonable results.

The presented method is to parallelize AI algorithms, so they could be implemented on GPU using CUDA.

IV. Results

Robocup Small-Size league is an appropriate filed for testing this method. In this league 2 teams consisting of 5 robots capable of moving with the speed up to 4m/s and a ball moving at the speed up to 10m/s play in a very dynamic and adversative field. Real-time processing for a team playing in such field is vital.

The algorithms descriptions and the results from both CPU and GPU are followed.

A) Computer Vision

Among the various sensors that can be utilized, one of the most powerful and inexpensive is through machine vision, which has its own limits like complexity of some algorithms used in it such as edge detection. Edge detection uses relative contrast in nearby pixels to determine boundaries in an image. Although popular in traditional machine vision and robotics, it has been difficult to run this type of processing at real time rates without specially designed hardware. This is due to the usual method requiring neighborhood processing of pixels in order to generate a value [2]. The most popular methods generally employ linear shift-invariant filters like Sobel operator [4]. However, they usually result in poor output and just fulfill minimum requirements.

This paper suggests the implementation of the edge detection algorithms for all pixels of the image simultaneously using CUDA. This algorithm can process a 780*580 image in about 0.34ms which is averagely 1442% faster comparing other implementations in Intel® OpenCV™.

In computer vision, edge detection and color segmentation are a good combination, since their processing advantages are complimentary. Color segmentation helps define object areas, while edge detection aids in defining more exact edges and does not rely on prior classification definitions as color thresholds do [3].

It is planned to work on determining the ball altitude in the small-size league as a novel approach by using edge detection.

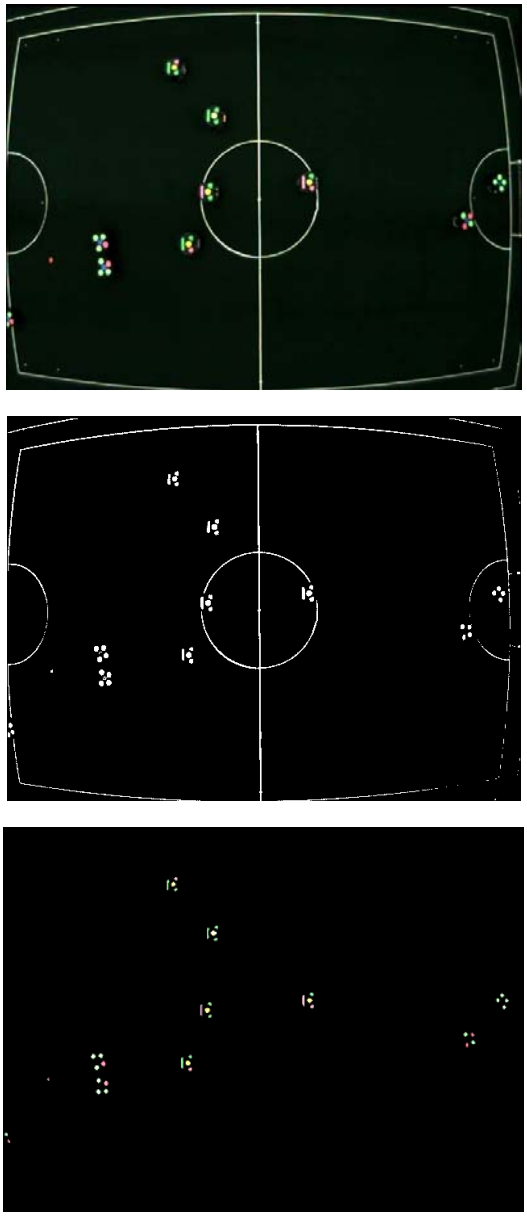

Figure 1: Source image (up) vs. image with edge detection and flood-fill applied (middle) and image combined with color segmentation (down).

B) Field Evaluation

One of the most important aspects of artificial intelligence in a small-size league robot is the field evaluation methods. In these methods each individual point of the field is evaluated for a specific purpose, for example to find the best point for receiving a pass from a teammate possessing the ball (Fig. 2). As explained in [1]

each point of the field is evaluated considering some parameters like the largest free angle toward the goal.

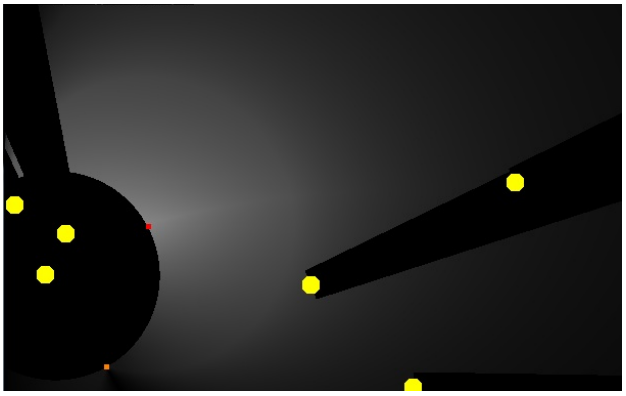

Figure 2: An example of evaluated field points for receiving pass from where the ball is located (the orange rectangle down left). Yellow circles are robots. Brighter points have more value. The red rectangle has maximum value and is the answer of the evaluation function.

If the SSL field having the dimensions of 6.05*4.05m is broken down to 6050*4050 points, implementing passing evaluation method on the CPU takes about 1500ms as shown in figure 3. Such processing time is not acceptable in a field as dynamic as the field of small-size robot.

The first solution that emerges into mind is to breakdown the field into bigger parts. It is clear that decreasing the number of points in each dimension by factor of n will result in processing time to become $1/n^2$ of the original time. Thus to make the task executable in the desired time (1ms) the field should be divided into 67*100 point. Such allocation has been tested in previous robotic competitions by our team Immortals and the results were not satisfactory.

Since evaluation functions for each point of the field are calculated independently of the other points, they could be implemented in parallel using CUDA. Executing passing evaluation method on the GPU with 6050*4050 allocations takes only 43ms (Fig. 3) which is a real-time evaluation of a high definition allocation (35 times faster than CPU).

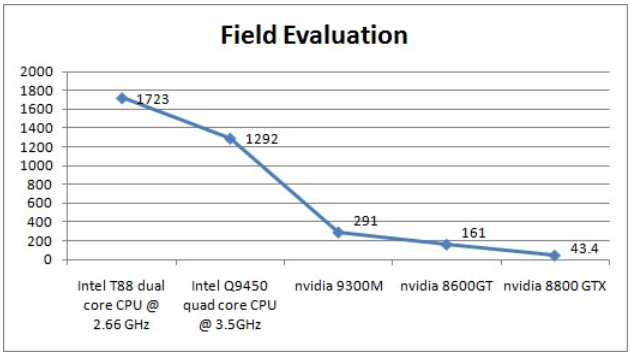

Figure 3: Field evaluation execution time on multiple CPUs and GPUs.

C) Path Planning

Path planning has always been a challenge for designers of mobile robots. There are many path planning methods like random planners, potential fields and A*; none of which has both satisfying accuracy and high speed of calculation. The main reason for that is the huge number of calculation steps which CPU does the computations serially, based on its architecture, while a GPU, having numerous cores could compute them simultaneously and more rapidly. Using CUDA, these algorithms could be implemented on GPU. Implementation of two of these methods is explained in the following sections.

1) Parallel-expanding Rapidly-Exploring Random Trees (RRT):

Like every other path planner, RRTs suffer from long execution time. Some developments have been made to improve performance of this algorithm, like Execution-Extended RRT [6], but they decrease the efficiency and accuracy of the path planned by the algorithm in highly dynamic domains. Although these new algorithms have slightly reduced the time required for calculation, they cannot be named real-time path planners yet. For example ERRT developed by CMDragons in Small-Size league competitions takes 2 ms, so it takes 10 ms to calculate the path for all 5 robots [6].
In the present work it is suggested to parallelize the expansions of this method, with inserting many nodes in the tree simultaneously. This method is not efficient enough in initial steps; but it reveals its capabilities as size of the tree increases.

The results of the tests show some improvements in calculation time. Executing an RRT plan in a regular small size game using CUDA takes at most 0.61ms.

2) Potential Field Planning

In [7] an approach is proposed for robot path planning that consists of incrementally building a graph connecting the local minima of a potential field defined in the robot's configuration space and concurrently searching this graph until a goal configuration is attained.

For every expansion step of the graph, all neighboring points should be evaluated. This paper suggests to evaluate all points of the field simultaneously on GPU using CUDA, and to determine local minima of every step by CPU. The experiments show a huge improvement in results as shown in figure 4, especially in multi-agent cases, because same evaluation results would be reused for all agents.

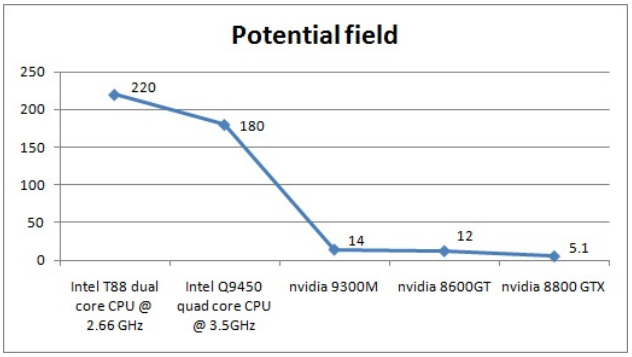

Figure 4: Potential field execution time on multiple CPUs and GPUs.

D) Physic-Based Behavioral Control

Separately implementing higher level tactics and lower level navigation control can lead to tactics which do not fully utilize the robot's dynamic actuation abilities. It can furthermore create the problem of the navigational code breaking the constraints of the higher level tactical goals when avoiding obstacles [1]. For example when a robot is executing "shoot on goal" skill [8], it shoots the ball to the center of the largest free angle toward the goal from its position. Doing so may not be necessarily the best solution. In some cases dribbling a defender might put the robot in a much better situation, or shooting the ball toward a defender might cause the ball to deflect toward the goal. In the method presented by Zickler [1] this problem is almost solved. This method finds the best solution by simulating different ways to run a probabilistically modeled play or skill [8] by a rigid-body-simulator like nVidia PhysX™.

The main difficulty of this approach is its low speed which has not allowed it to be implemented in real-time even 2 years after its development. As mentioned in [1] it takes 1 to 30 seconds to calculate the path, depending on the size of the tree. It should be considered that the tree which takes 1 second to be computed consists of 1000 nodes, and its average success rate in an example run by the authors is less than that of the linear tactic. In fact a tree with reliable success rate should have at least 10,000 nodes taking about 4 seconds to be calculated.

One of the main reasons for its high time consumption is that the computations of physics engine requires a lot of time. This problem could be easily resolved by using PPUs (Physics Processing Unit). It would resolve half of the algorithm's time-consumption problem [1] and the other half could be solved by using CUDA with parallel-Expanding RRT mentioned in the previous section.

This method is still under development and furthermore, the question how to re-use the previously computed result during re-planning needs to be answered.

V. Conclusion

A method is presented to implement parallel calculations of AI, image processing, and path planning on GPU using CUDA software introduced by nVidia. This method dramatically + speed, and is highly capable of calculating such computations much faster in comparison with CPU.

Apparently this approach enables designers to experience a real-time image processing with sufficient accuracy, evaluate points of the field with a reasonable allocation and implement expensive but accurate path-planning algorithms. It generally seems to solve some of the old problems of real-time processing

It is planned to work on the implementation and optimization of more image processing and AI tasks on GPU to form a general library to face the needs of robot designers to achieve a real-time processing ability.

References

[1] Zickler, S., Veloso, M. Playing Creative Soccer: Randomized Behavioral Kinodynamic Planning of Robot Tactics, Pennsylvania: Carnegie Mellon University, 2008

[2] Bruce, J. Real-time Machine Vision Perception and Prediction, Pennsylvania: Carnegie Mellon University, May 2000

[3] Marr. D. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. W. H. Reeman and Company, San Francisco, 1982

[4] Jain R., Kasturi R., and Schunck, B.G. Machine Vision. McGraw-Hill, 1995

[5] Zickler, S. Bruce, J. Biswas, J. Licitra, M. and Veloso, M. CMDragons 2009 Extended Team Description, Pennsylvania: Carnegie Mellon University, 2009

[6] Bruce, J. and Veloso, M. Real-Time Randomized Path Planning For Robot Navigation, Pennsylvania: Carnegie Mellon University, 2008

[7] Barraquand, J. Langlois, B. and Latombe, J. Numerical Potential Field Techniques for Robot Path Planning, Pennsylvania: Carnegie Mellon University, 1992

[8] Browning, B. Bruce, J. Bowling, M. and Veloso, M. STP: Skills tactics and plans for multi-robot control in adversarial environments. In: Journal of System and Control Engineering, 2005

[9] nVidia web page: http://www.nvidia.com