Solving 6x6 Othello on Volunteer Computing System

Shi-Jim Yen[†1], Tsan-Cheng Su [†2], Jr-Chang Chen [†3], Shun-Chin Hsu [†4]

[†1] Department of Computer Science & Information Engineering, National Dong Hwa University, Hualien, Taiwan. sjyen@mail.ndhu.edu.tw
[†2] Department of Computer Science & Information Engineering, National Dong Hwa University, Hualien, Taiwan. d9821009@ems.ndhu.edu.tw
[†3] Department of Applied Mathematics, Chung Yuan Christian University, Taoyuan, Taiwan. jcchen@cycu.edu.tw
[†4] Department of Information Management, Chang Jung Christian University, Tainan, Taiwan. schsu@mail.cjcu.edu.tw

Abstract

A job-level system, named CGDG, is suitable to solve computer games in parallelization. In this paper, we propose a volunteer computing application in the desktop grids based system. A strongly solved game tree for 6x6 Othello is constructed and applied on CGDG. We use CGDG to parallelize the game tree by splitting it to sub-trees, and to accelerate the process of construction. This research cooperates with five universities in different locations to use all sparing computing power to solve the game of 6x6 Othello. The experimental result shows that CGDG system is efficient on generating a Othello game tree.

Keywords: Othello, Game Tree, Computer Game Program, Parallelization

I. Introduction

The development of computer game programs and solvers are popular research topics [2][3][4]. Program parallelism is essential for a program running on a system with enormous computing resources of many workers. A desktop grid system, named CGDG, is suitable to solve computer games in parallelization [5][6]. The CGDG framework was developed by modifying and improving the system in [1]. In this paper, we propose a volunteer computing application on CGDG for solving 6x6 Othello. CGDG is executed on a mass of computer connected by Taiwan Academic Network (TANet), which is a high-speed internet connection among all universities in Taiwan. Its network delays are about 5ms or less. We also propose a method for splitting a game tree into sub-

trees. Then, we can take advantage of massively parallel computing of CGDG, and thus a complete game tree can be searched by using remote fast computers simultaneously.

II. Architecture of CGDG

CGDG is divided into two parts: the user program and volunteer workers. The broker is a bridge between both parts, as shown in Fig. 1. There are six steps for a round of a job:

(1) The user program encodes a game state as a job and sends it to the broker.

(2) The broker dispatches jobs from the user program to a remote idle volunteer worker, as shown in Fig. 2.

(3) The volunteer worker receives, decodes and executes the job.

(4) After finishing the job, the volunteer worker encodes the result and sends information back to the broker.

(5) The broker passes the result from the volunteer worker to the user program.

(6) The user program receives, decodes and handles the result, as shown in Fig. 3.

III. The Split of a Game Tree

Volunteer workers on CGDG provided from many universities are connected by TANet. A target tree should be split into many sub-trees before sending jobs. The purpose is to generate an enough amount of jobs lest some volunteer workers become idle at the same level. Because the network delays in TANet are short, and the number of child nodes for a node in the game tree of Othello is limited, the use of multiple sub-trees that are generated together can effectively utilize all volunteer workers.

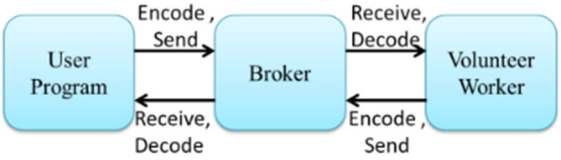

Fig. 1.The model for paralleling games.

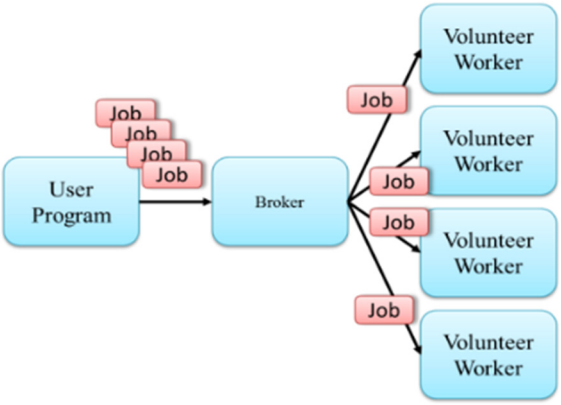

Fig. 2. Dispatch jobs in CGDG.

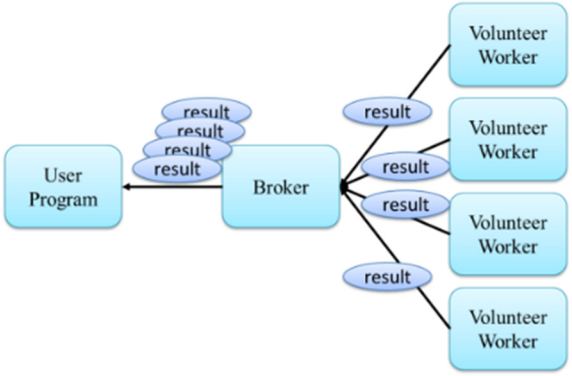

Fig. 3. Integrate results in CGDG.

A game tree is split into three sub groups as shown in Fig. 4. Each leaf node

is encapsulated as a job, and then all the jobs are sent to the broker. In Fig. 5, both of group A and group C have three jobs at the simultaneous time. The six jobs are sent to the broker, and then the broker assigns six volunteer workers to compute each of these jobs simultaneously.

The above method generates a game tree for Othello efficiently. Also, it is suitable for computing a game tree via machines located far away geographically and connected by the network with about 5ms latency. The latency is 50 times slower than that of the cluster computer architecture where all nodes in a tree are computed locally. Furthermore, our method to distribute tree nodes fits the CGDG architecture, and thus we maximize the use of all available resources with a variety of computers with different numbers of cores and clock speed.

The above method generates a game tree for Othello efficiently. Also, it is suitable for computing a game tree via machines located far away geographically and connected by the network with about 5ms latency. The latency is 50 times slower than that of the cluster computer architecture where all nodes in a tree are computed locally. Furthermore, our method to distribute tree nodes fits the CGDG architecture, and thus we maximize the use of all available resources with a variety of computers with different numbers of cores and clock speed.

IV. Experimental Results and Discussions

The experiments compare CGDG with a single server system. The single server system is Intel 12-cores i7-3930k CPU @ 4.2GHz with main memory 32GB; and CGDG have volunteer workers comprised of Intel 8-core Xeon CPU @ 2.2GHz with main memory 16GB and AMD 8-cores FX-8350 CPU @ 4.0GHz with main memory 16GB. Note that the hardware specifications for all volunteer workers are varied.

Table 1 is the experimental result to solve the game of 6x6 Othello. A "node" means a different state in the game. Each node is computed by one core. Numerous nodes are generated by searching from the current state to the end, and thus enormous computing resources are required.

The CGDG systems with 4 and 8 volunteer workers are used, and each volunteer worker owns 8 cores. Thus, there are 32 and 64 cores in total respectively. For the generation of a complete “strongly solved” game tree for 6x6 Othello, the result shows that CGDG with 8 volunteer workers performs about 4 times faster than the single server system. If there are extra idle computers joining the system, the system scales linearly in Fig. 6.

Table 1. Comparison of a single server and CGDG.

System / Nodes	**Single Server System (12 cores)**	**CGDG (4 workers) (1 worker is a 8-core machine)**	**CGDG (8 workers) (1 worker is a 8-core machine)**
10000	1217.95 sec	615.66 sec	305.26 sec
30000	3696.37 sec	1747.12 sec	898.49 sec
50000	6328.45 sec	3011.47 sec	1476.63 sec

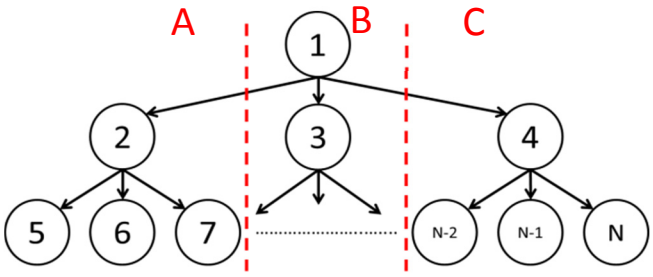

Fig. 4. Split to sub game trees.

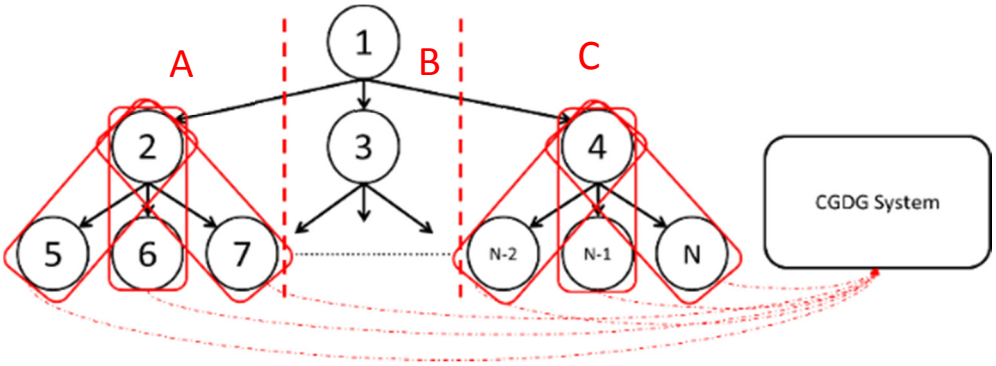

Fig. 5. Sample jobs to CGDG

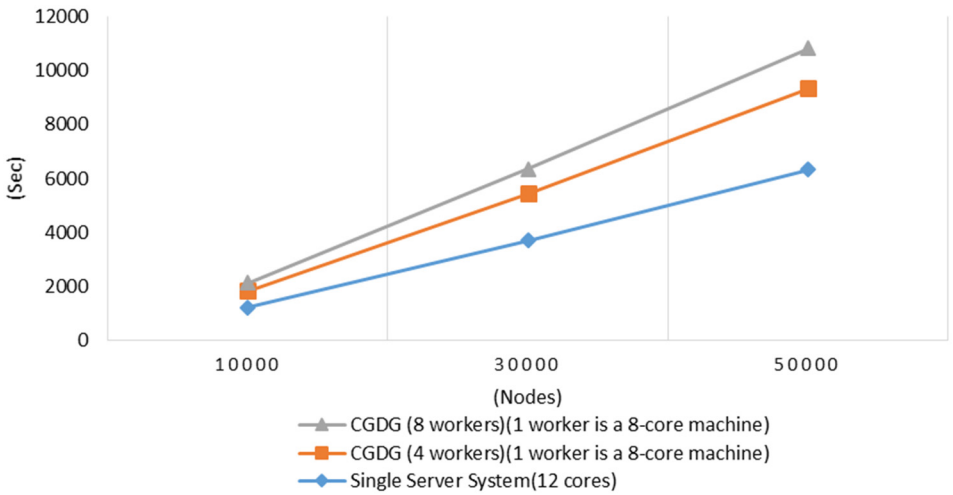

Fig. 6. Comparison of a single server and CGDG

References

[1] BOINC, available at http://boinc.berkeley.edu/.

[2] Chen, C.H., Lin, S.S. and Huang, M.H., “Volunteer Computing System Applied to Computer Games” *TCGA workshop 2012* (*TCGA2012*), June 2012, Hualien, Taiwan, pp.25-28.

[3] Chen, K.Y., “The Development of the Multi-Broker Desktop Grid for Computer Games”, National Chiao-Tung University, master thesis, 2012.

[4] Liou, H.Y., Wu, I.C., Kang H.H., Guo, J.H. and Liao, T.F., “General Framework Development on Board Game” *TCGA workshop 2012* (*TCGA2012*), June 2012, Hualien, Taiwan, pp.19-24.

[5] Liu, H.Y., Wu, I.C., Liau, T.F., Kang, H.H. and Chen, L.P., “Software Framework for Generic Game Development in CGDG,” *International Computer Symposium* (*ICS 2012*), Hualien, Taiwan, 2012.

[6] Wu, I.C. and Jou, C.Y., “The Study and Design of the Generic Application Framework and Resource Allocation Management for the Desktop Grid CGDG”, Institute of Computer Science and Engineering College of Computer Science NCTU, 2010. .