Leveraging Parallel Architectures in AI Search Algorithms for Games

Nicolas A. Barriga
University of Alberta
Edmonton, AB, Canada
barriga@ualberta.ca

Abstract

This document contains a summary of research performed by the author on the topic of search algorithms for games. An outline of the problems being addressed is provided, along with the progress already made, and planned future work. The specific subjects studied are: parallelizing UCT search on GPUs, the development of a hierarchical search framework for Real-Time Strategy (RTS) games and the building placement problem in RTS games. We propose to take advantage of different parallel architectures to help solve these problems.

Introduction

Since the early beginnings of Artificial Intelligence research, game playing has been an important source of inspiration. From classical board games like chess or checkers, to video games like Pac-Man or Unreal Tournament, games have provided a seemingly endless supply of interesting and challenging problems. Some of these games have been solved (checkers), for some, techniques have been developed to play at or above expert human level (chess), and others are getting close, but have yet to defeat a professional human player (go). However, on some game types, such as Real-Time Strategy (RTS) games, AI techniques have not even reached amateur level playing strength.

On a wide variety of search algorithms (game tree search algorithms, local search), more time to search, or faster computation when time is limited, usually leads to better solution quality. For the last few decades computer scientists and software engineers have been able to rely on steady processor clock speed increases to tackle bigger and bigger problems. This trend is slowly coming to a halt as clock speed increments stagnate and hardware advances focus instead on parallelism.

This parallelism comes in many flavours:

Instruction Level Parallelism (ILP): refers to several complementary techniques designed to execute multiple instructions from a single stream in parallel. Some of these techniques include superscalar processors, instruction pipelining, out-of-order execution, register renaming, speculative execution and branch prediction.

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Vector processing instructions: such as SIMD extensions in x86 processors (MMX, SSE, 3DNow!). These instructions perform the same computation on all the elements of a register vector. Vector lengths in modern desktop computers are commonly two to sixteen elements.

Single Instruction, Multiple Thread (SIMT): the basis of modern Graphics Processing Units (GPUs), in which several hundred threads can be executing the same instruction on independent data (as opposed to data explicitly packed on vector registers as in SIMD).

Symmetric Multiprocessing (SMP): two or more processors connected by a system bus, sharing main memory. They can run multiple processes or threads in parallel.

Multi-Core: found in most modern CPUs and GPUs, each processor chip contains many processing cores, with independent level 1 (and sometimes level 2) cache memory. As the programming techniques for the last two items are basically the same, these two types of systems are usually treated interchangeably.

Distributed Computing: multiple computing nodes (usually each of them an SMP system) each with its own independent main memory, interconnected by a network. While communication speeds between nodes are much slower and with higher latency than the system bus connecting SMP processors, these systems scale much more, both in terms of memory and raw processing power.

Since the hardware takes care of Instruction Level Parallelism, and optimizing compilers are beginning to automatically vectorize code to take advantage of SIMD extensions, we will focus on the latter techniques.

Application Domain and Research Goals

Real-Time Strategy (RTS) is a genre of video games in which players gather resources, build structures from which different types of troops can be trained or upgraded, recruit armies, and command them in battle against opponent armies. RTS games are an interesting domain for Artificial Intelligence (AI) research because they represent well-defined complex adversarial decision problems and can be divided into many interesting and computationally hard subproblems (Buro 2004).

The best AI systems for RTS games still perform poorly against good human players (Buro and Churchill 2012). Hence, the research community is focusing on developing RTS agents to compete against other RTS agents to improve the state-of-the-art. For the purpose of experimentation, the RTS game StarCraft: Brood War (Blizzard Entertainment 1998) has become popular because it is considered well balanced, has a large online community of players, and has an open-source interface — BWAPI, (Heinermann 2014) — which allows researchers to write programs to play the full game. Several StarCraft AI competitions are organized every year (Buro and Churchill 2012). Such contests have sparked increased interest in RTS game AI research and many promising agent frameworks and algorithms have already emerged.

While developing these agents, many sub-problems have been tackled, such as build order optimization (Churchill and Buro 2011), combat (Churchill and Buro 2013) and strategy selection (Synnaeve 2012). There are still problems that have received little attention, like building placement, or high level search (searching over strategic macro-actions, as opposed to the actions of individual units).

However, the ultimate goal is not just to win a competition (though that is interesting by itself), but to develop algorithms that can have a wider impact:

- In commercial games, smarter robot enemies and allies can greatly enhance the gaming experience (Buro 2004).
- Specific algorithms have wider applications, such as pathfinding in routing and transportation problems.
- RTS research can also be of interest in military applications.

My proposal is to use parallel algorithms to overcome one of the main difficulties posed by RTS games: the short time to compute a solution. As an example, in StarCraft, actions can be issued at every game frame, which is every 42 ms. Taking advantage of all the resources at our disposal, such as multiple cores, one or more GPUs, and even possibly other networked machines, can give us the computing power we need to reach expert play in this types of games.

Research Progress

The following subsections give a succinct introduction to the problems being tackled and are for the most part taken verbatim from introductions to (Barriga, Stanescu, and Buro 2014), (Stanescu, Barriga, and Buro 2014) and a paper currently under review, all of which are authored or co-authored by the author of this document. These sections also contain details on the progress made so far by the author during the first two years in the PhD program, and how parallel computing can help solve these problems.

Hierarchical Adversarial Search

Despite recent advances in RTS game playing programs, no unified search approach has yet been developed for a full RTS game such as StarCraft, although the research community is starting to tackle the problem of global search in smaller scale RTS games (Chung, Buro, and Schaeffer 2005; Sailer, Buro, and Lanctot 2007; Ontañón 2013). Existing StarCraft agents rely on a combination of search and machine learning for specific sub-problems (build order (Churchill and Buro 2011), combat (Churchill and Buro 2013), strategy selection (Synnaeve 2012)) and hard-coded expert behaviour.

Even though the structure of most RTS AI systems is complex and comprised of many modules for unit control and strategy selection (Wintermute, Joseph Xu, and Laird 2007; Churchill and Buro 2012; Synnaeve and Bessiere 2012), none comes close to human abstraction, planning, and reasoning abilities. These independent modules implement different AI mechanisms which often interact with each other in a limited fashion.

In (Stanescu, Barriga, and Buro 2014) we presented a hierarchical adversarial search framework in which each level implements a different abstraction — from deciding how to win the game at the top of the hierarchy to individual unit orders at the bottom. We applied a 3-layer version of our model to SparCraft — a StarCraft combat simulator — and show that it outperforms state of the art algorithms such as Alpha-Beta, UCT, and Portfolio Search in large combat scenarios featuring multiple bases and up to 72 mobile units per player under real-time constraints of 40 ms per search episode.

Because of these time constraints, during the plan selection in the middle layer, a heuristic is used to select which plans are evaluated in the minimax search, while the rest are discarded. Vast improvements could be made, if more nodes could be expanded at this level, which could be achieved either by parallelizing the current alpha-beta search, or moving to a parallel Monte Carlo Tree Search. These algorithms have been successfully parallelized for other domains, both on SMP and distributed arquitectures (Weill 1996; Chaslot, Winands, and van Den Herik 2008).

Building Placement

A sub-problem of RTS game AI that has seen little research is building placement, which is concerned with constructing buildings at strategic locations with the goal of slowing down potential enemy attacks as much as possible while still allowing friendly units to move around freely.

Human expert players use optimized base layouts, whereas current programs do not and therefore become prone to devastating base attacks.

Finding good building locations is difficult. It involves both spatial and temporal reasoning, and ranges from blocking melee units completely (Certicky 2013) to creating bottlenecks or even maze-like configurations that maximize the time invading units are exposed to own static and mobile defenses.

Important factors for particular placements are terrain features (such as ramps and the distance to expansion locations), the cost of constructing static defenses, and the type of enemy units.

Human expert players are able to optimize building locations by applying general principles such as creating chokepoints, and then refining placement in the course of playing the same maps over and over and analyzing how to counter

experienced opponent attacks. Methods used in state-of-the-art RTS bots are far less sophisticated (Ontanón et al. 2013). Some programs utilize the terrain analysis library BWTA (Perkins 2010) to identify chokepoints and regions to decide where to place defenses. Others simply execute pre-defined building placement scripts program authors have devised for specific maps. Still others use simple-minded spiral search around main structures to find suitable building locations.

In a paper currently under review at the *Second Workshop on Artificial Intelligence in Adversarial Real-Time Games at AIIDE-14* we propose using a Genetic Algorithm to optimize the placement of buildings in Real-Time Strategy games. Candidate solutions are evaluated by running fast combat simulation for gauging building placement quality with data gathered from human and bot replays for attack force estimation and stochastic hill-climbing for improving placements. The end result is a system that requires little domain knowledge and is quite flexible because the optimization is driven by an easily adjustable objective function and simulations rather than depending on hard-coded domain rules — described for instance in (Certicky 2013).

The current implementation works well for offline optimization of the building placement, but using this technique for online (i.e. during a game) optimization could prove very valuable, as the solution could be adapted to the enemy's army composition as soon as new scouting information is available. Solution quality under real-time game constraints would be greatly improved by parallelization, and thus faster convergence. Parallel Genetic Algorithms have been widely studied and successfully applied to a wide range of problems, on SMP systems, distributed memory systems and GPUs (Pospíchal, Jaros, and Schwarz 2010; Luque and Alba 2011).

Parallel UCT Search on GPUs

Monte-Carlo Tree Search (MCTS) has been very successful in two player games for which it is difficult to create a good heuristic evaluation function. It has allowed Go programs to reach master level for the first time (Coulom 2007; Gelly 2008). Recent results include a 4-stone handicap win by Crazy Stone against a professional player (Wedd 2013).

MCTS consists of several phases, which for our purposes can be classified as in-tree and off-tree. During the in-tree phases, the algorithm needs to select a node, expand it, and later update it and its ancestors. The off-tree phase consists of possibly randomized playouts starting from the selected node, playing the game until a terminal node is reached, and returning the game value which is then used for updating node information. For our research we choose UCT (Upper Confidence bounds applied to Trees, (Kocsis and Szepesvári 2006)) as a policy for node selection and updating because it has been proven quite successful in computer game playing (Teytaud and Teytaud 2010).

As with most search algorithms, the more time MCTS spends on selecting a move, the greater the playing strength. Naturally, after searching the whole game tree, there are no further gains. However, games such as Go which are characterized by a very large branching factor have large game trees that cannot be fully searched in a feasible amount of time. Parallelizing MCTS has led to stronger game play in computer programs (Chaslot, Winands, and van Den Herik 2008), and state of the art UCT implementations use distributed algorithms which scale well on thousands of cores (Yoshizoe et al. 2011). Unfortunately, the prohibitive cost of highly parallel machines has limited the full exploration of the potential of these algorithms.

However, a new type of massively parallel processors capable of running thousands of threads in Single Instruction Multiple Thread (SIMT) fashion, with performance in the Teraflops range, has become mainstream. These processors, called Graphics Processing Units (GPUs), are widely available on standard computer systems, ranging from smartphones to supercomputers. So far, there have been only a few attempts of harnessing this computing power for heuristic search.

In (Barriga, Stanescu, and Buro 2014) we proposed a parallel UCT search algorithm that takes advantage of modern GPU hardware. Experiments using the game of Ataxx are conducted, and the algorithm's speed and playing strength is compared to sequential UCT running on the CPU and *Block Parallel* (Rocki and Suda 2011) UCT that runs its simulations on a GPU. Empirical results show that the proposed *Multiblock Parallel* algorithm outperforms other approaches and can take advantage of the GPU hardware without the added complexity of searching multiple trees.

Our plan is to explore in depth the characteristics of the *Block Parallel* and *Multiblock Parallel* algorithms. For this purpose, we believe that experimenting on artificial game trees will give us the flexibility to achieve a deeper understanding of the relationships between the game particulars, the algorithms' parameters and the specific hardware on which it is executed.

Research Timeline

Table 1: Timeline

Topic	Details	Pub./Est. Date
Parallel UCT on GPUs	Preliminary research using the game of Ataxx	CIG '14
Hierarchical Adv. Search	Initial implementation and experiments	AIIDE '14
Building Placement	Initial implementation and experiments	Under Review
Building Placement	Parallel Online GA	Q4 2014
PhD Candidacy	Examination and thesis proposal	Jan 2015
Parallel UCT on GPUs	Artificial game trees	Q2 2015
Hierarchical Adv. Search	Parallel UCT/$\alpha\beta$	Q4 2015
Thesis Defense	Final oral examination	Aug 2016

Table 1 shows the research already published in grey, ongoing in light grey and planned in white background.

Conclusion

Fully exploiting parallel processors is difficult, but if the current hardware trends continue, it may become imperative to do so. If we want to keep solving bigger problems, parallel seems to be the way to go. This research proposal is an attempt at expanding our knowledge of the area and to apply it on novel and promising new hardware and application domains.

References

Barriga, N. A.; Stanescu, M.; and Buro, M. 2014. Parallel UCT search on GPUs. In *Accepted for presentation at the IEEE Conference on Computational Intelligence and Games (CIG)*.

Blizzard Entertainment. 1998. StarCraft: Brood War. http://us.blizzard.com/en-us/games/sc/.

Buro, M., and Churchill, D. 2012. Real-time strategy game competitions. *AI Magazine* 33(3):106–108.

Buro, M. 2004. Call for AI research in RTS games. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 139–142.

Certicky, M. 2013. Implementing a wall-in building placement in StarCraft with declarative programming. *arXiv preprint arXiv:1306.4460*.

Chaslot, G.; Winands, M.; and van Den Herik, H. 2008. Parallel Monte-Carlo tree search. *Computers and Games* 60–71.

Chung, M.; Buro, M.; and Schaeffer, J. 2005. Monte Carlo planning in RTS games. In *IEEE Symposium on Computational Intelligence and Games (CIG)*.

Churchill, D., and Buro, M. 2011. Build order optimization in StarCraft. In *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 14–19.

Churchill, D., and Buro, M. 2012. Incorporating search algorithms into RTS game agents. In *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*.

Churchill, D., and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *IEEE Conference on Computational Intelligence in Games (CIG)*, 1–8. IEEE.

Coulom, R. 2007. Efficient selectivity and backup operators in Monte-Carlo tree search. *Computers and Games* 72–83.

Gelly, S. 2008. *A contribution to reinforcement learning; application to computer-Go*. Ph.D. Dissertation, Universite Paris-Sud.

Heinermann, A. 2014. Broodwar API. http://code.google.com/p/bwapi/.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Luque, G., and Alba, E. 2011. *Parallel Genetic Algorithms: Theory and Real World Applications*, volume 367. Springer.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game AI research and competition in StarCraft. *TCIAIG* 5(4):293–311.

Ontañón, S. 2013. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *AIIDE*.

Perkins, L. 2010. Terrain analysis in real-time strategy games: An integrated approach to choke point detection and region decomposition. *Artificial Intelligence* 168–173.

Pospíchal, P.; Jaros, J.; and Schwarz, J. 2010. Parallel Genetic Algorithm on the CUDA architecture. In *Applications of Evolutionary Computation*. Springer. 442–451.

Rocki, K., and Suda, R. 2011. Parallel Monte Carlo tree search on GPU. In *Eleventh Scandinavian Conference on Artificial Intelligence: Scai 2011*, volume 227, 80. IOS Press, Incorporated.

Sailer, F.; Buro, M.; and Lanctot, M. 2007. Adversarial planning through strategy simulation. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, 80–87. IEEE.

Stanescu, M.; Barriga, N. A.; and Buro, M. 2014. Hierarchical adversarial search applied to real-time strategy games. In *Accepted for presentation at the Tenth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Synnaeve, G., and Bessiere, P. 2012. Special tactics: a Bayesian approach to tactical decision-making. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, 409–416.

Synnaeve, G. 2012. *Bayesian programming and learning for multi-player video games*. Ph.D. Dissertation, Université de Grenoble.

Teytaud, F., and Teytaud, O. 2010. Creating an upper-confidence-tree program for Havannah. In *Advances in Computer Games*. Springer. 65–74.

Wedd, N. 2013. Human-computer go challenges. http://www.computer-go.info/h-c/index.html#2013.

Weill, J.-C. 1996. The ABDADA distributed minimax search algorithm. In *Proceedings of the 1996 ACM 24th annual conference on Computer science*, 131–138. ACM.

Wintermute, S.; Joseph Xu, J. Z.; and Laird, J. E. 2007. SORTS: A human-level approach to real-time strategy AI. In *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 55–60.

Yoshizoe, K.; Kishimoto, A.; Kaneko, T.; Yoshimoto, H.; and Ishikawa, Y. 2011. Scalable distributed monte-carlo tree search. In *Fourth Annual Symposium on Combinatorial Search*.