유전 알고리즘과 게임 트리를 병합한 오목 인공지능 설계 및 GPU 기반 병렬 처리 기법

(Design of Omok AI using Genetic Algorithm and Game Trees and Their Parallel Processing on the GPU)

안 일 준 † 박 인 규 ††
(Il Jun Ahn) (In Kyu Park)

요 약 본 논문에서는 GPU(graphics processing unit)를 이용하여 오목의 인공지능 알고리즘 연산을 고속으로 수행하기 위한 효율적인 알고리즘 설계와 구현 방법을 제안한다. 본 논문에서 제안하는 게임 인공지능은 최소-최대 게임 트리(min-max game tree)와 유전 알고리즘(genetic algorithm)의 협업적 구조로 설계된다. 게임 트리와 유전 알고리즘의 평가함수(evaluation function) 부분은 많은 계산 량을 소모하지만 해 공간(solution space)의 수많은 후보 벡터에 대해 독립적으로 수행되기 때문에 본 논문에서는 이를 GPU 상에서의 대량 병렬처리를 통해 수행한다. NVIDIA CUDA(compute unified device architecture) 환경에서의 실제 구현을 통해 CPU에서의 처리에 비해 게임 트리는 400배 이상의 수행 속도의 향상을, 유전 알고리즘은 300배 이상의 수행 속도의 향상을 각각 보였다. 본 논문에서는 스레드(thread)의 넘침(overflow)을 피하고 보다 효과적인 해 공간 탐색을 위해, 게임 트리를 이용하여 근방의 몇 단계까지 전역 탐색(full search)을 수행한 후 이후 단계는 유전 알고리즘을 이용하여 선별 탐색을 수행하는 협업적 인공지능을 제안한다. 다양한 실험 결과를 통해 제안하는 알고리즘은 게임의 인공지능을 향상시키고 게임의 규칙으로부터 주어진 시간 내에 문제를 해결할 수 있음을 보인다.

키워드 : GPU, 게임 트리, 유전 알고리즘, 오목 인공지능, NVIDIA CUDA, 전역 탐색, 선별 탐색, 협업적 인공지능

Abstract This paper proposes an efficient method for design and implementation of the artificial intelligence (AI) of 'omok' game on the GPU. The proposed AI is designed on a cooperative structure using min-max game tree and genetic algorithm. Since the evaluation function needs intensive computation but is independently performed on a lot of candidates in the solution space, it is computed on the GPU in a massive parallel way. The implementation on NVIDIA CUDA and the experimental results show that it outperforms significantly over the CPU, in which parallel game tree and genetic algorithm on the GPU runs more than 400 times and 300 times faster than on the CPU. In the proposed cooperative AI, selective search using genetic algorithm is performed subsequently after the full search using game tree to search the solution space more efficiently as well as to avoid the thread overflow. Experimental results show that the proposed algorithm enhances the AI significantly and makes it run within the time limit given by the game's rule.

Key words : GPU, Game tree, Genetic algorithm, Omok AI, NVIDIA CUDA, Full search, Selective search, cooperative AI

· 이 논문은 인하대학교의 지원에 의하여 연구되었음. 이 논문은 인하대학교 IT공과대학 정보통신공학부의 "정보통신 프로젝트" 결과물로 작성되었음

† 학생회원 : 한국과학기술원 전기및전자공학과
ijahn@issserver.kaist.ac.kr
†† 종신회원 : 인하대학교 정보통신공학부 교수
pik@inha.ac.kr

논문접수 : 2009년 7월 29일
심사완료 : 2010년 2월 4일

정보과학회논문지: 시스템 및 이론 제37권 제2호(2010.4)

1. 서 론

최근 수년간 GPU(graphics processing unit)의 연산 속도는 급격히 발전해왔다. 또한, 이와 더불어 GPU에서 쉽게 사용할 수 있는 고수준의 언어는 기존의 3차원 그래픽스뿐만 아니라 GPGPU(general purpose GPU)라고 불리는 일반적인 용도로의 GPU 사용을 가능하게 하였다[1,2]. 최근 공개되어 활발히 이용되고 있는 NVIDIA사의 CUDA(compute unified device architecture)[3,4]와 같은 통합 컴퓨팅 프레임워크는 스레드(thread) 레벨에서 사용자가 제어할 수 있는 병렬처리 방법을 제공하였고 전역 메모리(global memory) 전체 공간에 대한 읽기와 쓰기를 자유롭게 사용 할 수 있게 하였다[4]. 따라서 기존의 OpenGL shading language(GLSL)[5] 또는 DirectX HLSL(high level shading language)과 같은 shading 언어 기반의 GPGPU에 비해 프로그래밍의 자유도가 획기적으로 높아져 많은 분야의 알고리즘을 CUDA 기반으로 병렬 구현할 수 있다[6].

GPGPU의 목적은 대용량 데이터 처리나 복잡한 알고리즘의 처리에서 CPU를 능가하여 눈부신 속도 향상을 얻는 것이다. 인공지능(artificial intelligence)을 위한 알고리즘은 광범위한 해 공간을 순차적으로 탐색하면서 각 탐색 대상에 대해 동일하지만 많은 양의 연산을 처리하는 방식, 즉 SIMD(single instruction multiple data) 방식의 접근을 필요로 한다. 이는 GPU의 대용량 병렬 처리 구조에 적합하다.

본 논문에서는, 다중 코어를 갖는 대용량 병렬 GPU에서 오목의 인공지능 알고리즘 연산을 고속으로 수행하기 위한 효율적인 구조와 방법을 제안한다. 본 논문에서 제안하는 오목 인공지능은 광범위한 해 공간에서 최적의 해(solution)를 탐색하기 위해, 게임 트리를 이용한 전역적 탐색과 유전 알고리즘을 이용한 선별적 탐색을 조합하여 설계되었다. 또한 각각의 알고리즘의 평가함수 부분을 GPU상에서 모두 병렬 화시켜 주어진 시간 내에 CPU에서는 수행할 수 없는 탐색 문제를 짧은 시간 내에 해결할 수 있도록 하였다.

본 논문의 구성은 다음과 같다. 제 2장에서는 본 논문과 관련된 기존의 연구에 관해 간략히 설명한다. 제 3장에서는 오목의 인공지능 구현 방법을 설명한다. 제 4장에서는 오목 인공지능의 GPU 병렬처리에 대한 구체적 방법을 제시한다. 제 5장에서는 실험 결과를 보이고, 제 6장에서는 결론을 제시한다.

2. 기존의 연구

2.1 GPU를 이용한 병렬 컴퓨팅 기술

NVIDIA는 2007년 프로그램이 가능한 그래픽 프로세서 G80 및 병렬처리 SDK인 CUDA를 출시하였다. CUDA는 NVIDIA의 그래픽 하드웨어를 제어하는 명령어들을 C언어 기반으로 제공하여 사용자가 스레드 레벨에서의 대용량 병렬 처리를 GPU에서 수행할 수 있게 한다. GPU는 CPU 대비 10배 가까운 메모리 인터페이스 속도와 128개(G80)의 프로세서에서 대용량 데이터를 병렬처리함으로써 계산 속도를 높일 수 있으며 또한 시스템을 최적화 시킬 수 있다[3,4]. CUDA는 영상처리, 의료 데이터 처리, 컴퓨터 비젼, 그래픽스의 비주얼 컴퓨팅 분야뿐만 아니라, 연산이 많은 대형 행렬연산, 편미분 방정식 풀이, 천체 시뮬레이션, 정렬 알고리즘, 탐색알고리즘(바이러스, 유전자, 단백질), DB 분석계산 등 다양한 분야에서 응용되고 있다[2,7]. 그러나 방대한 탐색 공간에서 최적의 해를 찾는 모델로 표현되는 게임 인공지능 알고리즘에의 응용은 아직 미미한 상황이다.

2.2 딥 블루 (Deep Blue)[8]

딥 블루는 IBM의 8년에 걸쳐 개발한 슈퍼컴퓨터로 1997년 세계 체스 챔피언인 Garry Kasparov와의 대결로 관심을 얻었다. 딥 블루는 가능한 모든 경우를 조사하여 다음 수를 결정하기 때문에 방대한 병렬처리 능력이 필요하다. 딥 블루는 30개의 노드로 구성된 RS/6000 SP기반 컴퓨터이고 480개의 전용 VLSI를 장착하고 있다. 체스 인공지능은 C언어로 구현되었으며 AIX 운영체제에서 실행했다. 1초당 2억개의 위치를 계산할 수 있으며 11.38 GFLOPS의 최대 성능을 가진다. 이에 따라 딥 블루 시스템은 규정 시간 내에 12수 앞을 내다볼 수 있는 예측 능력을 가지고 있다.

딥 블루의 예측 함수는 많은 예측 가능한 경우를 포함하는 일반화된 형태로 짜여 있다. 이 함수의 값들은 수 천 개 이상의 대국을 분석한 컴퓨터 자신에 의해서 결정된다. 예측 함수는 8,000개의 부분으로 나뉘어 있고 이들의 대부분은 특정한 위치에 최적화되어 있다. 초기 시스템에는 70만개 이상의 대국 정보와 4000개 정도의 위치가 기록되어 있다.

2.3 강화 학습 알고리즘

최근 강화 학습 알고리즘을 보드게임의 인공지능에 적용하는 연구가 활발히 진행되고 있다. 강화 학습 알고리즘은 환경과 에이전트와의 상호 정보교환을 통하여 에이전트의 행동을 개선해 나가는 학습방법이다[9-11]. 환경에는 목적 달성에 필요하거나, 필요하지 않은 많은 상태들이 존재한다. 에이전트는 각각의 상태를 경험하여 목적달성의 경우 환경으로부터 보상을 받게 된다. 목적 달성을 할 수 없는 상태인 경우 보상을 받을 수 없다. 그러므로 많은 시행착오를 겪을 수 있으며, 모든 상태를 경험해 보아야 한다. 강화학습 알고리즘으로서 일반적으로 Q-learning을 많이 사용한다.

모든 가능한 상태공간에서의 특정 상태에서의 행위 중 가장 좋은 보상 값을 저장하고 산출한다. 처음에는 모든 상태에 대하여 보상을 얻기 위해 무작위로 행위를 하여 경험을 쌓게 된다. 거의 모든 상태를 경험함으로써 학습이 완료되어 지능적으로 동작하게 된다[9-11].

강화 학습 알고리즘을 오목 인공지능에 적용할 경우 하나의 상태 공간은 15×15 이므로 225개의 격자 셀로 구성된다. 사용자의 말들의 위치, 컴퓨터의 말들의 위치, 사용자의 말들의 공격 또는 방어를 위해 이동 가능한 공간 등의 요소를 고려할 때 전체 상태공간의 크기는 225×225×225×225×8 로 20,503,125,000 이 된다. 평균 학습 횟수가 1,000~4,000 임을 고려할 때 고려해야 할 상태 공간의 크기는 기하급수적으로 증가한다. 이를 CPU로 구현한다면 제한된 시간 내에 최적의 해를 탐색할 수 없을 뿐만 스택 오버플로가 일어날 가능성도 배제할 수 없다.

강화학습 알고리즘은 각 상황에서 선택할 수 있는 점수 중 가장 유리한 것을 선택하게 된다. 그러나 학습의 초기에는 저장된 데이터가 많지 않아 좋은 선택을 못하게 된다. 학습을 시키는데 많은 시간이 필요하다는 점을 감안한다면 주어진 시간 안에 최적의 해를 찾을 확률은 낮아지게 된다.

3. 오목 인공지능 설계

일반적으로 무한히 빠른 컴퓨터와 무한한 양의 메모리가 존재할 수 없으므로 게임의 인공지능을 설계할 때 모든 경우의 수를 탐색하여 최적의 해를 도출하는 것은 불가능하다. 따라서 제한된 컴퓨팅 환경과 주어진 시간 내에 효율적으로 전역 최소값 또는 이에 근사한 값을 찾는 것이 게임 인공지능 구현의 핵심요소가 된다. 장기, 바둑, 오목과 같은 보드 게임의 경우 현재로부터 몇 수 앞까지의 모든 경우의 수를 최소-최대 게임 트리의 구조를 이용하여 탐색한 후 가장 최선의 선택을 하는 알고리즘이 일반적이나 많은 수를 예측하기 위해서는 슈퍼컴퓨터와 같은 고성능의 컴퓨터가 필요하다.

본 논문에서는 광범위한 해 공간에서 최적의 해를 탐색하기 위해 게임 트리를 이용한 전역적 탐색과 유전 알고리즘을 이용한 선별적 탐색을 조합한다. 각각의 평가함수 부분은 GPU상에서 모두 병렬 처리된 고속 연산으로 수행된다.

3.1 평가 함수

본 연구에서 설계한 평가 함수는 오목에서 일어나는 21가지의 특정한 패턴을 포함하는 일반화된 형태로 이루어져 있으며 각 경우에 대해서 유리한 순서로 평가치를 부여한다. 표 1은 평가 함수에 저장된 21가지의 특정 패턴과 그에 대한 평가치를 보여준다. 특정 좌표에 놓인 돌에 대해, 평가 함수가 포함하는 경우와 논리적으로 일치하면 그 좌표에 평가치를 부여한다.

표 1 특정 패턴에 대한 평가치

	상태	평가치
1	오목	100
2	사사	특이 조건
3	상대편의 방어가 없는 사삼	특이 조건
4	상대편의 방어가 있는 사삼	70
5	상대편의 방어가 없는 사목	특이 조건
6	상대편의 방어가 없고 중간에 하나의 빈칸이 있는 사목	33
7	한쪽에 상대편의 방어가 있고 중간에 하나의 빈칸이 있는 사목	32
8	양쪽에 상대편의 방어가 있고 중간에 하나의 빈칸이 있는 사목	31
9	상대편의 방어가 없는 삼삼	특이 조건
10	상대편의 방어가 없는 이삼	21
11	상대편의 방어가 없는 이이이	20
12	상대편의 방어가 없는 삼목	10
13	한쪽에 상대편의 방어가 있는 사목	9
14	상대편의 방어가 있는 삼삼	8
15	상대편의 방어가 없고 중간에 하나의 빈칸이 있는 삼목	7
16	한쪽에 상대편의 방어가 있고 중간에 하나의 빈칸이 있는 삼목	6
17	상대편의 방어가 없는 이이	5
18	상대편의 방어가 없는 삼목	4
19	한쪽에 상대편의 방어가 있는 삼목	3
20	상대편의 방어가 없는 이목	2
21	한쪽에 상대편의 방어가 있는 이목	1

표 1에 나오는 21가지의 특정 패턴은 모두 독립적으로 구현하는 것이 아니고 1번, 5번, 18번, 20의 기본 패턴을 이용하여 나머지는 이것에 대한 약간의 변형 또는 조합으로 이루어진다. 예를 들어 3번 패턴은 5번 패턴과 12번 패턴의 결합으로 이루어지며, 4번 패턴은 5번 패턴과 19번 패턴의 결합, 13번 패턴과 12번 패턴의 결합, 또는 13번 패턴과 19번 패턴의 결합으로 이루어진다. 평가치에서 명시된 특이 조건은 다음 절에서 정의한다.

3.2 특이 조건

알고리즘을 설명하기 위하여 그림 1과 같은 특이 조건을 정의한다. 특이 조건 발생 시 무조건 게임에 승리한다. 연속해서 4개의 말이 놓인 경우, 3개의 말과 4개의 말이 조합되어 놓인 경우, 4개의 말과 4개의 말이 조합되어 놓인 경우, 3개의 말과 3개의 말이 조합되어 놓인 경우가 그것이다. 가로, 세로, 대각선, 가운데 빈칸이 바뀌면서 발생하는 다른 조건이라도 위와 논리적으로 같다면 특이 조건으로 간주한다.

3.3 최소-최대 게임 트리와 유전 알고리즘의 협업 알고리즘

본 논문에서는 게임 트리와 유전 알고리즘을 이용하여 오목 인공을 구현한다. 최대-최소 게임 트리는 깊이 우선 탐색을 적용하며, 한 수를 더 내다볼 때 마다 그 단계에서의 모든 경우의 수를 게임 트리로 조직하고, 최종 단계의 노드들을 평가한 뒤, 각 단계의 레벨에 맞게 최대값과 최소값을 탐색하여, 결과를 도출한다[12]. 게임에서 n개의 수가 지속적으로 존재 한다면, p레벨의 게임 트리를 조직하고 평가할 때, 그림 2와 같이 최종 레벨에서 n^p개의 노드가 생성되고 모든 노드에 대해 평가함수를 구한 후 각 레벨별로 나에게 유리한 최대값과 상대편에게 유리한 최소값을 취합하여 최상위 단계에서 의사결정을 수행한다. 즉 기본적으로 전역 탐색의 개념이다. 하지만 트리의 깊이가 증가할수록 탐색해야 하는 노드의 수는 기하급수적으로 증가하므로 주어진 시간 내에 탐색할 수 있는 게임 트리의 최대 깊이가 제한된다. 따라서 프로세서에서 처리할 수 있는 컴퓨팅 능력에 따라 최대 레벨의 크기를 제한해야 하는 단점이 있다.

이 단점을 극복하기 위해 본 논문에서는 GPU의 병렬

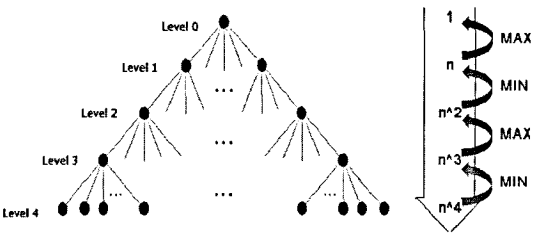

그림 2 최대 최소 알고리즘이 적용된 게임 트리[12]

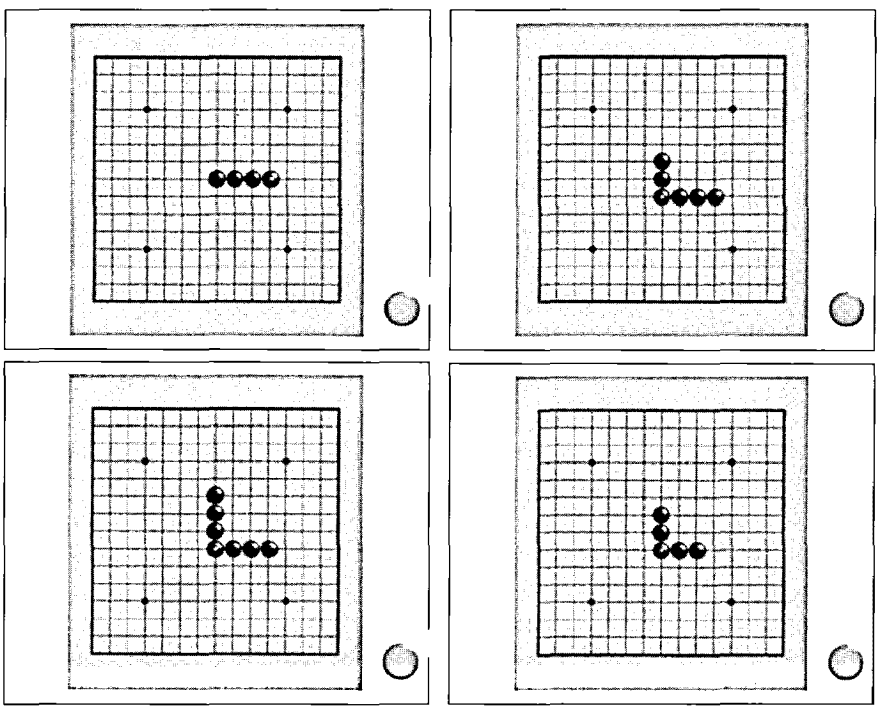

그림 1 특이 조건의 사례

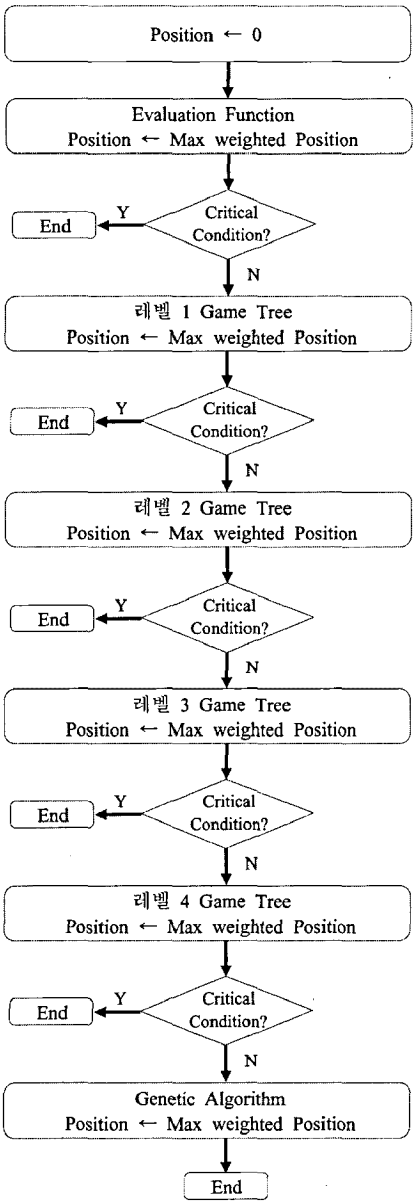

그림 3 오목 인공지능의 순서도

처리 능력 이상의 레벨이 되면 최소-최대 게임 트리를 유전 알고리즘으로 전환한다. 유전 알고리즘은 적자생존과 유전의 메커니즘을 바탕으로 하는 탐색 알고리즘이다. 다시 말해 주어진 환경에 잘 적응하는 유전자만을 선택하고 교배하며 때에 따라서는 돌연변이를 하여 다음 세대에 우수한 유전 형질을 전달하게 된다. 따라서 진화가 거듭될수록 주어진 환경에 더 적합한 유전자들만이 남아있게 될 것이다. 유전 알고리즘에서 사용하는 선택, 교차, 변이, 대치와 같은 연산자들은 일정 범위 안에서 선별적으로 이루어지며 이는 선별적 탐색의 개념이다. 모든 범위를 탐색하는 대신 가능성이 높은 곳만을 탐색한다.

그림 3은 본 논문에서 제안하는 오목 인공지능의 기본 구조를 나타낸다. 레벨 4까지는 게임 트리로 전역적 탐색을 한 후에 유전 알고리즘으로 선별적 탐색을 수행한다. 각 단계에서는 평가함수가 반환하는 최대 평가치 좌표를 저장한다. 그리고 특이 조건을 만족하는 단계에서 탐색을 종료한다. 유전 알고리즘 이 후에도 특이 조건을 찾지 못한다면 각 단계에서 축적된 최대 평가치 좌표를 반환한다.

3.4 유전 알고리즘 설계

본 절에서는 오목 인공지능을 유전 알고리즘으로 설계하기 위해 사용된 연산자들의 구체적 구현에 대해 기술한다.

3.4.1 유전자 초기화

유전자 집합은 (염색체 길이×염색체 개수)의 1차원 동적 배열(dynamic array)로 선언된다. 즉 상황에 따라 길이와 개수를 자유롭게 변경할 수 있다. 각각의 유전자는 실수로 표현되며 0~224까지의 좌표(오목은 15×15의 바둑판에서 수행된다고 가정)중 염색체 길이만큼의 좌표가 임의로 선택되어 하나의 염색체를 이룬다.

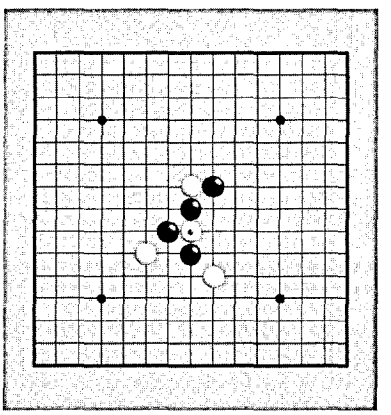

그림 4 염색체 초기화 사례

그림 4의 경우를 표 2에 맵핑시킨다면 한 개의 염색체는 112, 97, 98, 127, 126, 140, 142, 158로 초기화된다. 게임 초기에는 불필요한 좌표들이 포함될 가능성이 크므로 놓여 있는 수들로부터 일정 범위 이내에 있는 좌표들만으로 초기화시킨다. 실수 표현의 가장 큰 장점은 선택되어질 좌표와 유전자가 일대일 대응되는 것이다. 만약 오목 인공지능을 위해 유전자를 이진수로 표현하였다면 각 선택되어질 좌표의 위치 관계와 상관없는 불필요한 교차나 변이가 발생하는 단점이 발생한다.

표 2 염색체 초기화 사례

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134
135	136	137	138	139	140	141	142	143	144	145	146	147	148	149
150	151	152	153	154	155	156	157	158	159	160	161	162	163	164
165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189	190	191	192	193	194
195	196	197	198	199	200	201	202	203	204	205	206	207	208	209
210	211	212	213	214	215	216	217	218	219	220	221	222	223	224

3.4.2 선택

선택 연산을 위해 본 논문에서는 품질 비례/룰렛 휠에 의한 선택을 적용한다. 각 해의 평가치를 평가한 다음 가장 좋은 해의 평가치가 가장 나쁜 해의 적합도의 k배가 되도록 조절한다. 해 집단 내의 i번째 해의 평가치 f_i는 다음과 같이 구할 수 있다.

$$f_i = (C_i - C_w) + \frac{(C_b - C_w)}{k-1}, \quad k > 1 \qquad (1)$$

여기서 C_b와 C_w는 해 집단 내에서 가장 좋은 해의 평가치 및 가장 나쁜 해의 평가치를 의미하며, C_i는 해 i의 평가치를 나타낸다. 일반적으로 사용하는 k값은 3~4이다. 본 논문에서는 k값을 3으로 하여 가장 좋은 해의 평가치가 가장 나쁜 해의 평가치의 3배가 되도록 조절하였다. 평가치를 조절하지 않는다면 해 집단에서 가장 좋은 해의 선택압(selection pressure)이 너무 커져 해의 다양성을 급속히 떨어뜨리므로 바람직하지 않다.

앞과 같이 계산한 각 염색체의 평가치를 모두 합한 값만큼의 크기를 가진 룰렛 휠을 가정한다. 각 염색체는 이 룰렛 휠 상에 자신의 적합도 만큼의 공간을 배정받는다. 여기에 활을 쏘면 각 염색체의 선택 확률은 배정된 공간의 크기에 비례하게 된다. 룰렛 휠 선택은 그림 5와 같이 간단히 구현할 수 있다.

그림 5에서 f_i는 염색체 i의 적합도이고, SumOfFitnesses는 모든 염색체들의 적합도 값을 더한 수치다. random(0, SumOfFitnesses)은 [0, SumOfFitnesses) 구간에서의 난수를 의미한다[13].

3.4.3 교차

각 염색체의 순서까지 섞기 위해 순서 교차(order crossover)를 적용한다. 오목의 경우 수를 두는 순서가 매우 중요한다. 하지만 보편적으로 사용하는 일점 교차(one-point crossover)는 부모 각각의 일정 부분을 순

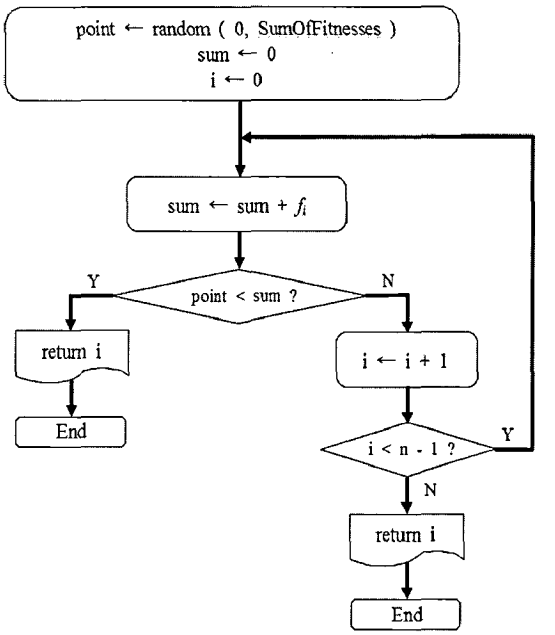

그림 5 품질 비례 룰렛 휠 선택의 순서도

서는 변형시키지 않고 서로 교환한다. 따라서 본 논문에서는 인자의 순서도 교차시킬 수 있는 순서 교차를 사용한다. 그림 6은 순서 교차의 방법을 나타낸다. 우선 선택된 두 부모 s_1, s_2에 임의로 두 개의 자름 선을 정한 다음 자름선 사이에 있는 부분을 s_1로부터 복사한다. 나머지 위치는 s_2로부터 복사하고, 두 번째 자름선 바로 다음 위치부터 시작해 사용된 위치는 제외하고 순서대로 복사한다[13]. 이렇게 되면 s_1의 뒷부분이 자식해의 앞부분으로 오게 된다. 만약 s_1의 뒷부분의 패턴이 후반보다 초반에 더 효과적인 패턴이라면 자식해는 부모해보다 높은 평가치를 받을 것이다.

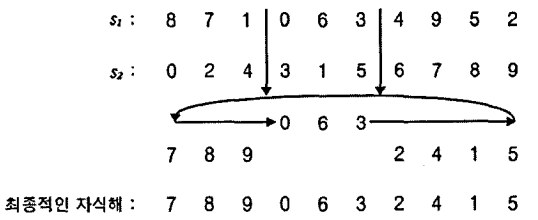

그림 6 순서 교차[13]

3.4.4 변이

유전 알고리즘의 초기에는 해들의 평가치 좋지 않은 것이 보통이고 시간이 지남에 따라 점차적으로 품질이 개선된다. 따라서 초반에는 변이의 정도가 다소 강하여도 평가치 향상이 일어날 가능성이 있지만, 해의 개선이 상당한 수준에 이른 후반에는 변이가 강하면 평가치 향상이 일어나기 어렵다. 이에 착안하여 비균 등 변이(non-uniform mutation)를 적용한다.

비균등 변이는 이진 난수 r을 발생시킨 다음 아래와 같이 이루어진다.

$$f=\begin{cases} f+\Delta(t, U-f), & \text{if } r=0 \\ f-\Delta(t, f-L), & \text{if } r=1 \end{cases} \quad (2)$$

여기서 f는 염색체의 평가치고, U와 L은 염색체가 가질 수 있는 평가치의 상한 값과 하한 값이다. 즉, 본 논문에서는 U는 224이고, L은 0이다. $\Delta(t, y)$는 0과 y 사이의 값을 갖는데 시간 t가 증가함에 따라 점점 0으로 근접하는 특성을 갖는다. $\Delta(t, y)$을 위해 난수 r_2을 발생시켜 아래와 같은 식을 사용할 수 있다[13]. 본 논문에서는 최대 세대수를 1,000으로 제안하였다.

$$\Delta(t, y)=(1-r_2^{(1-\frac{t}{T})^2})y \quad (3)$$

여기서

r_2 : [0, 1]범위의 난수

T : 최대 세대수

3.4.5 대치

유전 알고리즘에서 해 집단의 다양성을 유지하는 것은 매우 중요하다. 유전 알고리즘 초반에 해 집단이 설익은 수렴(premature convergence)을 한다면 최적화된 해를 구하기 어렵다. 본 논문에서는 다양성을 알고리즘 후반까지 유지하기 위해 교차 연산 후 자식 해를 두 부모 해 중 품질이 나쁜 해와 대치한다. 해 집단에서 자신과 닮았을 가능성이 가장 높은 해가 부모 해이므로 다른 해보다 부모 해 중의 하나를 제거하는 것은 해 집단의 다양성을 오래 유지시키는 데 도움이 된다. 만일 자식 해를 해 집단 중 평가치가 가장 낮은 해와 대치한다면 다양성이 급격히 줄어 설익은 수렴을 할 가능성이 커진다.

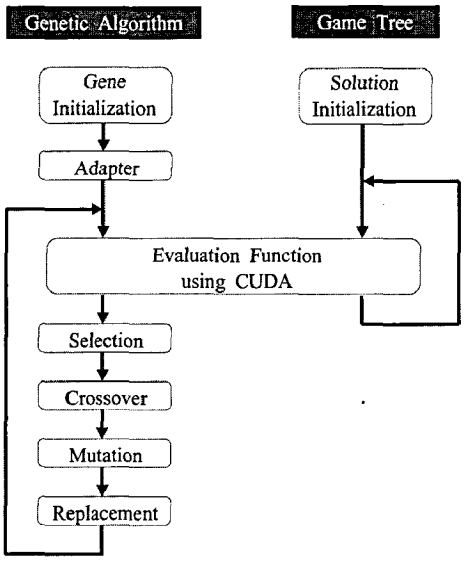

그림 7 유전 알고리즘과 게임 트리의 순서도

4. 오목 인공지능의 GPU 기반 구현

4.1 최소-최대 게임 트리의 GPU 기반 구현

그림 7에 제시한 바와 같이 유전 알고리즘과 게임 트리는 동일한 평가함수를 사용한다. 보드(board)의 각 좌표(225개)에 평가함수를 적용하면 상황별 21가지의 평가치 중 하나를 반환한다. 평가함수를 CUDA로 구현하기 위해 좌표 1개를 하나의 스레드에 할당하고, 보드 1개를 하나의 블록(block)에 할당한다. 즉 하나의 블록은 225개의 스레드로 구성된다. 게임 트리의 경우 한 개의 노드(node)에 한 개의 블록이 할당되어 적합도(fitness)를 평가한다.

커널 (kernel) 함수는 각각의 상황을 판단할 수 있는 21개의 __device__함수로 구성된다. 각 좌표에 할당된 스레드는 커널 함수를 통해 적합도를 계산한다. 좀 더 정교한 평가함수를 구현하고자 한다면 더욱 구체적인 상황을 판단할 수 있는 __device__함수를 구현한 후 추가하면 되므로 확장성이 용이하다.

각 노드는 보드의 상태를 나타내는 1×225의 1차원 배열을 포함하는 구조체로 구현된다. 먼저 플레이어가 어떤 행동을 취한 경우, 인공지능은 거기에 어떤 행동으로 대응할 것인가를 결정해야 한다. 이때 다음에 나타날 수 있는 모든 보드의 상황을 평가함수로 분석하여 다음에 취할 가장 유리한 행동을 탐색한다. 게임 초반에는 다음에 둘 수 있는 가능한 수가 많고, 유효한 수가 적기 때문에 일정 turn까지는 heuristic 기법을 사용하여 이미 놓여 있는 수들로부터 일정 범위 이내에 있는 좌표들만 탐색한다.

표 3 레벨 별 노드, 블록, 스레드의 개수

	노드	블록	스레드
레벨 2	225 (= $_{225}C_1$)	225	50,625
레벨 3	25,200 (= $_{225}C_2$)	25,200	5,670,000
레벨 4	1,873,200 (= $_{225}C_3$)	1,873,200	421,470,000

인공지능은 평가함수의 반환 값이 특이조건을 만족시킬 때 까지 게임 트리를 레벨 2에서 레벨 4까지 순차적으로 적용한다. 표 3은 heuristic 기법을 사용하지 않았을 때 각 레벨에서의 게임 트리의 노드의 개수, CUDA에서의 블록과 스레드의 개수를 나타낸다. 레벨 1의 경우 노드의 개수는 225($_{225}C_1$) 개이다. 이 노드들을 평가함수에 적용할 시 CUDA는 블록은 225개, 스레드는 50,400(224×225)개를 발생시킨다. 이 때, 게임 초반에는 heuristic 기법을 사용하기 때문에 각 레벨의 노드의 개수는 40%이상 줄어든다. 또한 게임 후반에는 보드에 놓일 수 있는 수의 위치가 줄어들기 때문에 노드의 개수는 매 turn 마다 일정하게 줄어든다.

GPU에서 커널 함수를 수행하기 전에 레벨 4까지의 노드 정보와 평가 함수가 반환하는 평가치를 저장할 수 있는 메모리를 device의 전역 메모리(global memory)에 할당한다. 그리고 레벨 4까지의 모든 보드 정보를 시스템 메모리에서 전역 메모리로 복사한다. 이후, GPU에서 수행되는 모든 연산들은 전역 메모리에 저장된 정보를 액세스하여 수행한다.

커널 함수가 수행되면 각 스레드는 전역 메모리에서 해당 구조체를 읽어, 내부적으로 전역 메모리에 평가함수가 반환하는 평가치를 저장하는 연산을 수행한다. 각 레벨 별로 평가치의 최소값과 최대값을 가지는 노드는 별도로 저장한다. 그 후, CPU에서는 전역 메모리에 저장된 정보를 시스템 메모리로 받아 각 레벨의 최소-최대 노드를 선택하고 최종적으로 주어진 상황에서의 의사결정을 수행한다.

4.2 유전 알고리즘의 GPU 기반 구현

유전 알고리즘과 게임 트리에 동일한 평가함수를 적용하기 위해 유전 알고리즘에서 사용하는 해의 형태와 게임 트리에서 사용하는 해의 형태를 맞춰줘야 한다. 이를 위해 평가함수의 입력 변수는 1×225의 1차원 배열로 하고 유전 알고리즘에서 평가함수 적용 전에 해의 형태를 변형시킨다. 염색체 초기화 후 유전 알고리즘에서 사용하는 해의 형식을 게임 트리의 해의 형식으로 변형시키는 adapter 함수를 별도로 구현한 후 적용한다. adapter 함수는 각각의 염색체를 보드에 매핑(mapping) 시켜 염색체 개수만큼의 1×225의 1차원 배열로 변형 후 평가함수에 적용한다.

유전 알고리즘은 평가함수의 반환 값이 특이조건을 만족시키거나, 특정 좌표의 평가치가 정해진 값을 넘어설 때까지 반복된다. 표 4는 평균 세대수를 500이라고 가정했을 때 유전 알고리즘 수행 중 발생하는 염색체 개수에 따른 CUDA에서의 블록과 스레드의 개수를 나타낸다.

표 4 염색체 개수 별 블록, 스레드의 개수

	블록	스레드의 개수
1	500	112,500
10	5,000	1,125,000
100	50,000	11,250,000
200	100,000	22,500,000
300	150,000	33,750,000
400	200,000	45,000,000
500	250,000	56,250,000
1,000	500,000	112,500,000

5. 실험 결과

오목의 인공지능을 위한 평가함수는 우선 CPU 코드로 구현되었고 CUDA를 이용하여 모두 병렬화 하였다. 모든 실험은 Intel CPU(21.28 GFLOPS의 E6750)와 128개의 코어를 갖는 NVIDIA G92b(705 GFLOPS의 GeForce 9800 GTX+ 및 70.4 GB/s의 메모리 대역폭) 그래픽카드를 이용하여 수행되었다. 실험은 게임 트리와 유전 알고리즘에서 평가함수를 CPU와 GPU에서 실행시켰을 때 수행속도를 비교하였다.

5.1 게임 트리의 실험 결과

표 5에 게임 트리의 각 레벨에 따른 실험 결과를 제시하였다. 이 때 GPU의 수행 시간은 커널의 수행 시간뿐만 아니라 메모리의 이동까지 모두 포함하고 있다. CPU상에서의 실행은 최적화된 CPU 코드를 사용하지 않았으나, GPU상에서의 구현에 의한 전반적인 수행 속도 향상을 비교하는 데 큰 문제는 없을 것으로 생각된다.

스레드의 개수가 225개인 레벨 1에서는 5배 정도의 성능향상을 보였으나 스레드가 많아질수록 속도향상도 커지는 것을 확인할 수 있다. 특히 레벨 3인 경우 CPU 수행시간 대비 0.3%이하로 대폭 감소하였음을 알 수 있다. 레벨 4는 CPU의 속도가 실시간으로 게임을 진행할 수 있는 시간을 벗어나기 때문에 실험에서 제외시켰다.

표 5 한 수를 두기 위한 CPU 대비 GPU 기반 게임 트리의 속도 향상

레벨	수행 시간		CPU 대비 속도 향상
	CPU (E6750)	GPU (9800GTX+)	CPU/GPU
1	0.0175	0.00339	5.14x
2	0.3910	0.00753	51.94x
3	45.0490	0.10200	442.11x

5.2 유전 알고리즘의 실험 결과

표 6에 유전 알고리즘의 염색체 개수에 따른 실험 결과를 제시하였다. 염색체를 보드에 맵핑시킨 후 게임 트리와 동일한 평가함수를 적용시킨 후 수행시간을 측정하였다. 제시된 바와 같이 염색체 개수가 증가할수록 수행시간이 단축되었음을 알 수 있다. 염색체 개수가 1,000개인 경우 CPU 수행시간 대비 0.3%이하로 대폭 감소하였음을 알 수 있다.

표 6 한 수를 두기 위한 CPU 대비 GPU 기반 유전 알고리즘의 속도 향상

염색체 개수	수행 시간		CPU 대비 속도 향상
	CPU (E6750)	GPU (9800GTX+)	9800GTX+ / E6750
1	0.00369	0.00719	0.51x
10	0.05790	0.00779	7.39x
100	0.41600	0.01050	39.53x
200	0.80500	0.00896	89.78x
300	1.23200	0.09140	134.85x
400	1.58600	0.01110	143.01x
500	1.66100	0.01361	122.47x
1,000	4.18500	0.01110	375.58x

5.3 제안된 인공지능의 성능 평가

본 논문에서는 GPU가 동시에 실행할 수 있는 전역적 탐색 이후, 최적의 해를 찾지 못할 경우 범위를 넓혀 선별적 탐색을 수행할 것을 제안했다. 그러므로 지능 평가를 위해, 제안된 인공지능과 레벨 4의 게임 트리로 만든 인공지능을 서로 대전시켜 성능을 비교하였다.

그림 8의 (a), (b)는 게임 트리의 레벨 4를 이용한 인공지능이 흑으로 본 논문에서 제안된 인공지능이 백으로 대전한 결과이다. (a), (b) 모두 제안된 인공지능이 승리하였다. (c), (d)는 흑과 백의 순서를 바꾸어 대전한 결과이다. 이 또한 제안된 인공지능이 모두 승리하였다. 결과적으로 게임 트리 레벨4 이후 유전 알고리즘의 사용이 훨씬 효과적임을 나타낸다.

5.4 사용자 평가

제안하는 인공지능 알고리즘과 GPU 구현의 효용을 주관적으로 검증하기 위하여 사용자 평가를 실시하였다. 평가 대상은 제안된 인공지능과 동일한 평가함수를 사용하는 게임트리 레벨2로 구현한 인공지능에 승리한 초등학생 3명, 게임트리 레벨3에 승리한 중학생 3명, 그리

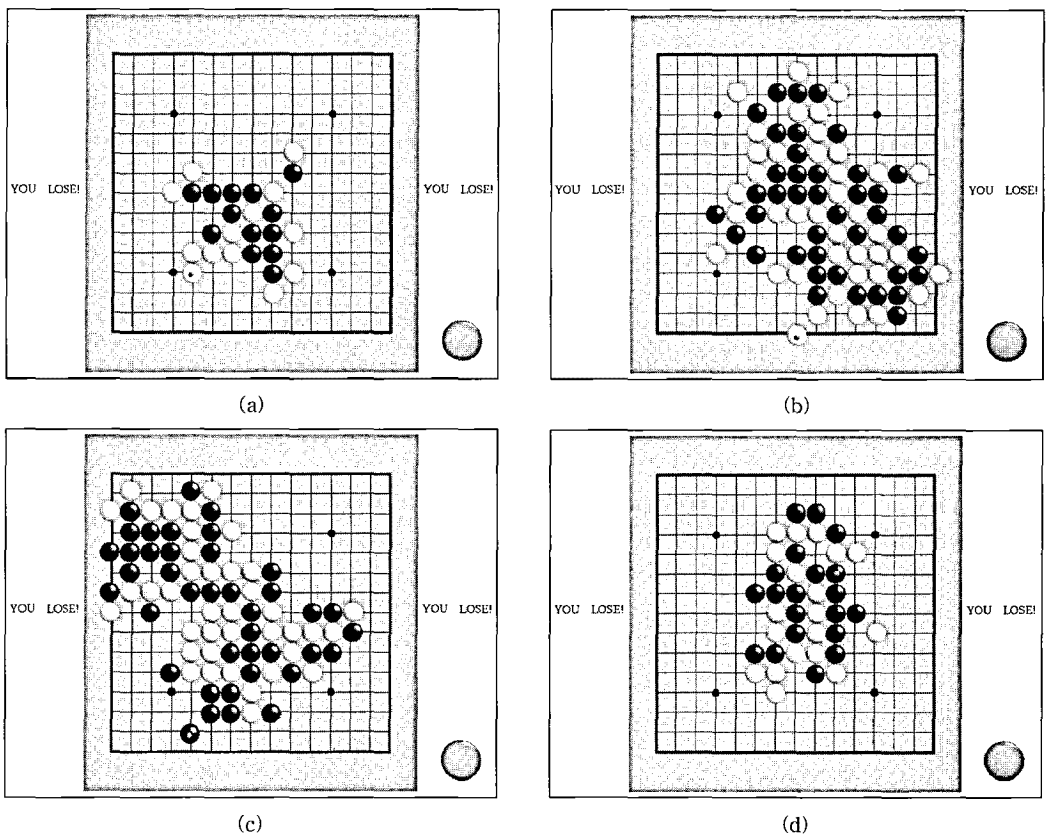

그림 8 (a),(b)흑(게임 트리 레벨4) VS 백(제안된 인공지능), (c),(d)흑(제안된 인공지능) VS 백(게임 트리 레벨4)

표 7 사용자 평가(User Study) 결과

User	승		제안된 인공지능 승률
	User	제안된 인공지능	
초등학생 3명	0	3	100%
중학생 3명	0	3	100%
고등학생 5명	1	4	75%
대학생 20명	6	14	70%

고 게임트리 레벨 4로 구현한 인공지능에 승리한 고등학생 5명, 대학생 20명의 총 31명을 선택하였다. 오목은 흑으로 두었을 때 항상 유리하므로 모든 실험은 제안된 인공지능을 백으로 하였다. 사용자 1명 당 3전 2선승제로 하였으며, 평가 결과는 표 7과 같다.

결과를 통해 유추하면, 초등학생과 중학생의 대전을 통해서도 게임트리 레벨 2나 레벨 3로 구현된 인공지능보다 본 논문에서 제안한 인공지능이 훨씬 효과적임을 알 수 있다. 고등학생과 대학생의 대전을 통해서는 게임트리 레벨 4보다 실전에서 평균 73%의 성능향상이 있음을 알 수 있다.

6. 결 론

본 논문에서는 오목의 인공지능을 게임 트리와 유전 알고리즘을 이용하여 구현하고 NVIDIA의 차세대 GPU 구조인 CUDA를 이용하여 고속으로 수행하였다. 수행속도 비교를 위해 모든 알고리즘은 우선 CPU 코드로 구현되었고 CUDA를 이용하여 모두 병렬화 하였다. 실험결과 CPU에서의 처리에 비해 CUDA에 맵핑한 경우 최대 442배의 수행 속도 향상을 보였다. GPU는 지속적으로 고성능의 신제품이 출시되므로 본 논문에서 제시된 결과는 상위 기종의 GPU를 사용함에 따라 그 성능에 비례하여 수행 속도가 빨라질 것이다. 또한 제한된 시간 안에 최적의 해를 찾기 위해 게임 트리를 이용하여 일정 레벨까지는 전역적 탐색을 하고, 이 후 유전 알고리즘으로 선별적 탐색을 할 것을 제안하였다. 제안된 방법을 이용하면 주어진 시간 안에 답을 찾는 의사결정 문제를 더 효율적으로 접근할 수 있다.

향후 인공지능의 평가함수의 성능을 개선하여 오목 인공지능의 승률을 높일 계획이다. 또한 CPU코드를 최적화 시켜 전반적인 성능을 더 개선시킬 것이다. 또한, 인공지능의 평가함수 부분에 좀 더 구체적인 상황에 대한 평가치를 추가한다면 승률은 더욱 높아질 것이라 기대한다.

참 고 문 헌

[1] General Purpose GPU (GPGPU) Homepage, http://www.gpgpu.org.

[2] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol.96, no.5, pp.879-899, May 2008.

[3] NVIDIA Corporation, *Compute Unified Device Architecture* (CUDA), http://developer.nvidia.com/object/cuda.html

[4] NVIDIA Corporation, *CUDA™ Programming Guide*, June 2008.

[5] R. Rost, OpenGL Shading Language Second Edition, Addison-Wesley, 2006.

[6] I. K. Park, N. Singhal, M. H. Lee, and S. Cho, "Efficient design and implementation of visual computing algorithms on the GPU," *Proc. IEEE International Conference on Image Processing (ICIP 2009)*, pp.2321-2324, November 2009.

[7] J. Lee and H. Ryu, "Current status and future prospect of personal supercomputer using GPU parallel computing," *The Magazine of the IEEK*, vol.36, no.5, pp.18-27, May 2009.

[8] http://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

[9] R. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[10] N. J. van Eck and M. van Wezel, "Application of reinforcement learning to the game of Othello," *Computers and Operations Research*, vol.35, no.6, pp.1999-2017, June 2008.

[11] I. Ghory, "Reinforcement learning in board games," CSTR-04-004, Dept. of Computer Science, University of Bristol, May 2004.

[12] M. Deloura, *Game Programming Gems*, Charles River Media, 2001.

[13] B.-R. Moon, *Easy-to-learn genetic algorithm : Evolutionary approach*, Hanbit Media, 2008.

안 일 준

2010년 2월 인하대학교 정보통신공학부 공학사. 2010년 3월~현재 한국과학기술원 전기및전자공학과 석사과정. 관심분야는 GPGPU, 게임 인공지능, 의료영상처리

박 인 규

1995년 2월 서울대학교 제어계측공학과 공학사. 1997년 2월 서울대학교 제어계측공학과 공학석사. 2001년 8월 서울대학교 전기컴퓨터공학부 공학박사. 2001년 9월~2004년 3월 삼성종합기술원 전문연구원. 2007년 1월~2008년 1월 미국 Mitsubishi Electric Research Laboratories (MERL) 방문연구원. 2004년 3월~현재 인하대학교 정보통신공학부 조교수. 관심분야는 컴퓨터그래픽스 및 비젼, 영상처리, 멀티미디어응용분야