Graphics Processing Unit를 이용한 섬기반 Real-Valued Genetic Algorithm의 체계적 평가

박현수O 김경중
세종대학교 컴퓨터공학과
rex8312@gmail.com, kimkj@sejong.ac.kr

Systematic Evaluation of Island based Real-Valued Genetic Algorithm with Graphics Processing Unit

Hyunsoo ParkO Kyungjoong Kim
Dept. of Computer Engineering, Sejong University

요 약

최적해를 구하는 효과적인 방법 중 하나인 GA (Genetic Algorithm)은 높은 품질의 해를 구하기 위해서 많은 연산시간이 필요하지만, GPU (Graphics Processing Unit)의 높은 데이터 병렬처리 능력과 우수한 부동소수 연산능력을 이용하면 빠르게 처리 가능하다. 이 논문에서는 GPU를 이용하여 가속한 섬 기반의 RVGA (Real-Valued Genetic Algorithm)와 GPU를 이용하지 않는 RVGA를 비교하여 평가하였으며, 또한 GPU를 이용하지만 RVGA가 아닌 Simple GA인 경우와도 비교하여 평가 하였다. 그 결과, GPU를 이용한 경우 속도 향상을 할 수 있었으며, Simple GA보다 RVGA가 더 속도가 향상되었다.

1. 서 론

GA (Genetic Algorithm)는 자연의 진화과정을 모방하여 확률적으로 해를 탐사하는 방법으로, 각종 최적화 문제를 효과적으로 해결 가능하다[1]. 다수의 개체를 이용하여 모의 진화를 수행하는데, 하나의 개체는 해공간에서 하나의 해를 나타낸다. 진화과정을 거치며 최적해에 가까운 개체들은 살아남지만, 최적해와 거리가 먼 개체들은 도태되어 사라진다. 그 결과, 마지막에는 최적해에 가까운 개체들이 살아남을 확률이 높다. 이런 개체들을 해로 변환하면 최적해에 가까운 해를 얻을 수 있다.

개체의 총 수를 집단의 크기(population)이라고 하는데, 만약 집단의 크기가 너무 작게 한다면 연산할 양이 줄어들어 속도는 빨라지지만 최적해를 찾지 못할 확률이 높아진다. 반면 집단의 크기를 크게 하면 더 좋은 해를 찾을 확률이 높아지지만 더 많은 시간이 소모된다. 하지만 집단의 커질수록 성능이 향상되는 것은 아니기 때문에 일반적으로는 30~200을 사용한다[2].

GPU를 이용해 병렬처리를 한다면 CPU만을 사용한 것에 비해 매우 빠른 속도로 해를 구할 수 있다. 집단의 크기가 크더라도 GA를 GPU로 가속하여 실험한 결과에 의하면 큰 속도 향상을 얻을 수 있었다[3][4]. 비록 모든 문제에서 집단의 크기를 크게 하는 것이 그 만큼 더 좋은 해를 구하는 것을 보장하지는 않지만, 집단이 너무 작을 경우에는 탐색능력이 떨어지게 된다.

2. 관련연구 및 배경

2.1 GPGPU (General-Purpose Computation on Graphics Processing Unit)

GPGPU란 주로 그래픽스 처리를 위해서만 사용되던 GPU를 일반적인 작업에서도 사용하는 것을 뜻한다. GPU는 그래픽스 처리를 위해 데이터 병렬화에 최적화되어 있으며, 강력한 부동소수 연산능력을 가지고 있고, 발전 속도가 빠르다[5].

2.2 CUDA (Compute Unified Device Architecture)

CUDA는 GPGPU를 위해 표준 C의 문법을 확장하여 nVidia에서 정의하였다. CUDA에서는 CPU와 GPU를 각각 host와 device라 정의하며, 주 메모리와 GPU에 있는 메모리를 각각 host 메모리와 device 메모리라 한다. Device 메모리는 global, shared, constant, texture 메모리로 나뉜다.

Thread는 CUDA에서 가장 기본적인 작업의 단위가 되며, 여러 thread가 모여서 block을 형성한다. 그리고 각 block마다 16kB의 shared 메모리를 가지고 있다.

병렬처리가 필요할 때, host 메모리에서 global 메모리로 데이터를 복사한 뒤, device에서 수행되는 함수인 kernel을 호출하여 필요한 만큼 thread를 생성하여 처리한다. 각 thread는 입력 받은 data를 처리하여 출력 data를 global 메모리에 복사한다. 이때 속도 향상을 위해 shared 메모리를 이용할 수 있다.

여기서, 병렬로 처리되어 속도가 향상되는 부분은 kernel이 수행되는 부분이다. 반면, Host와 device 메모리간에 데이터를 복사하는 과정은 매우 느리기 때문에 최소화 되어야 한다. 또한 global 메모리는 shared 메모리에 비해 매우 느리기 때문에 가능한 shared 메모리를 최대한 이용하도록 최적화 시켜야 한다[5].

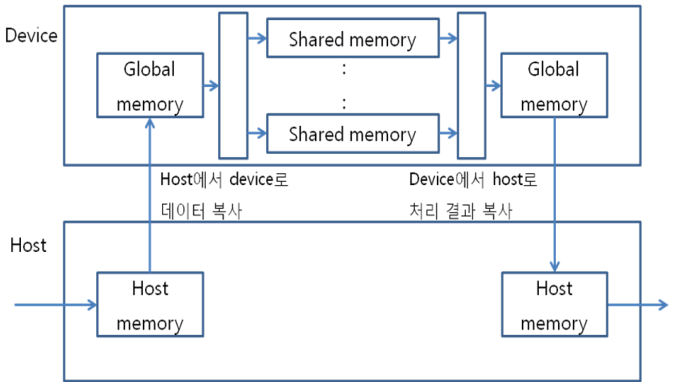


그림 1. 데이터의 흐름

2.3 Genetic Algorithm

Procedure Genetic Algorithm
Set k=0
Create an initial population
Evaluate the population
While *(the termination condition are not met)*
Set k = k + 1
Selection
Crossover
Mutation
Evaluate the population
End while
Output the solution

그림 2. GA algorithm

그림 2는 GA의 기본 알고리즘을 보여준다[5]. Simple GA에서 염색체는 0, 1로 구성된 비트열이며 RVGA (Real-Valued Genetic Algorithm)에서는 실수열이다. 개체의 염색체에서 실제 문제의 해를 구하기 위해서는 염색체를 복호해야 하는데, Simple GA에서는 비트열을 실수로 복호하여 해를 구한다. 만약 비트열이 너무 짧을 때에는 해의 정밀도가 떨어지고, 너무 길면 연산시간이 오래 걸린다. 반면, RVGA에서는 염색체가 이미 실수 형태이므로 복호 과정이 필요 없으며, 일반적으로 해의 정밀도가 더 우수하다[6].

2.4 Island 기반 GA

섬 기반의 GA에서는 전체 집단을 섬이라고 하는 작은 여러 개의 소집단으로 분리하고 각 집단에서 독립적으로 진화시킨다. 만약 어느 한 섬에서 적합도가 매우 높은 개체가 나타날 경우 그 영향은 하나의 섬에만 국한되고 나머지 섬에서는 독자적으로 진화를 이루어 종의 다양성을 유지할 수 있다[6]. 다양성이 유지되면 해공간을 폭넓게 탐색할 수 있어 지역해에 수렴할 가능성이 낮다. 일정 기간 고립이 지속된 후에는 각 섬의 일부 개체를 다른 섬으로 이주시켜 독립적으로 진화한 개체들을 서로 경쟁시킬 수 있다.

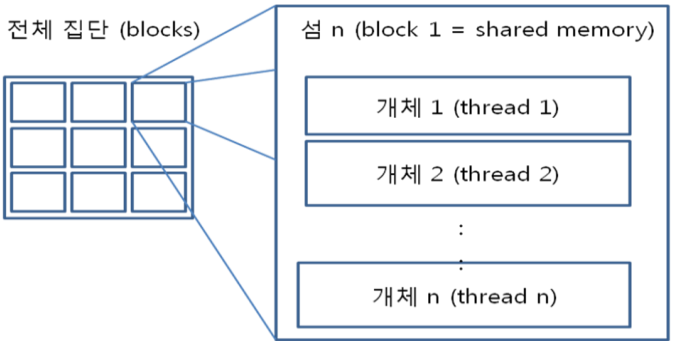


그림 3. 하나의 섬의 구성

본 논문에서 하나의 섬은 CUDA에서 하나의 block에 대응한다. 그리고 하나의 thread에 하나의 개체가 대응한다. 이주를 하지 않는 동안에는 빠른 shared 메모리에서 연산을 수행하고 이주를 수행할 때만 global 메모리를 이용한다[3].

2.5 기존의 GPGPU에서 구현된 GA와 차이점

기존에 CUDA로 구현된 GA는 일반적으로 Simple GA였다[2][3]. 하지만, GPU의 강력한 부동 소수 처리 능력을 고려하면 실수열을 직접 사용하는 RVGA가 더 적합한 선택이다. 복호과정을 제외하여 속도를 높일 수가 있고, 일반적으로 해의 정밀도가 더 높다. 실험에서는 단정도 부동 소수형을 기본 데이터 형으로 사용한다.

3. GPU 기반 RVGA

3.1 개요

초기화 후에 이주 조건이 만족될 때까지 GA과정을 수행한 뒤, 일정 세대가 진화하여 이주조건이 만족되면 이주를 수행하고 다시 GA과정을 반복한다. 종료조건이 만족할 때까지 이 과정을 반복하며, 종료 조건이 만족되면 결과를 출력한다. 따라서, *이주 회수* × *부분 세대 수*가 전체 세대 수가된다. 그림 4에 전체 과정이

나와 있으며, 점선으로 묶인 부분이 GPU에서 처리되는 부분이다.

3.2 초기화

초기화는 host에서 수행한다. 우선, 해 영역 안에서 난수를 생성하여 초기 집단을 생성한다. 총 생성해야 하는 난수의 개수는 *섬 개수* × *지역집단 크기* × *염색체 길이* 이다. 생성된 값들은 GPU에서 사용하기 위해 device 메모리로 복사한다. 전체집단은 여러 개의 섬으로 구성되어 있으며, 각 섬은 여러 개체가 모여있다.

GPU 상에서 표준 C의 rand() 함수는 사용이 불가능하므로, 난수 발생기가 별도로 필요하다. 본 논문에서는 [8]을 참조하여 난수발생기를 구현하였다. 초기 seed값으로 사용된 난수들은 host에서 생성한다.

3.3 선택

토너먼트 선택을 사용한다. 각 thread는 담당하고 있는 개체와 무작위로 선택된 개체 *n*개를 차례대로 비교한다. [0,1) 사이의 수를 선택하여 Selection threshold를 넘지 않으면 두 개체 중 적합도가 높은 개체가 살아남고 아니면 적합도가 낮은 개체가 살아남는다. 최종적으로 살아남은 개체는 현재 위치에 복사된다.

3.4 유전연산

교배과정에서 thread는 [0,1) 사이의 난수를 골라서 crossover ratio 이하의 수가 나오면 무작위로 자신의 개체와 교배할 개체를 선택한 뒤에 1점 교배를 한다. 교배하고 난 자식 개체 둘 중에 하나를 무작위로 골라서 자신의 자리에 복사한다.

RVGA에서 돌연변이 연산은 각 염색체의 숫자 하나마다 [0,1) 사이의 난수를 골라서 Mutation ratio이하의 숫자가 나오면 다시 난수를 골라서 나온 수를 염색체의 원소에 더한다. 만약 돌연변이가 문제의 해 영역을 벗어난다면 돌연변이 과정을 취소한다.

3.5 엘리트 전략

진화 과정에서 가장 높은 적합도를 가진 개체가 도태되어 사라지는 것을 방지하기 위하여 엘리트 전략을 사용할 수 있다. 각 섬마다 적합도가 높은 개체를 선택하여 각 집단이 저장된 배열의 제일 앞부분에 가져다 놓고, 엘리트가 위치한 자리의 thread는 수행하지 않아 엘리트가 사라지는 것을 방지한다. 하지만 다른 위치의 thread가 엘리트를 교배/토너먼트 상대로 지정할 수 있다.

3.6 이주

이주는 지역집단끼리 개체를 교환하는 과정이다. 각 집단이 저장된 배열의 앞부분부터 일정비율을 바로 다음 위치에 있는 섬에 보낸다. 마지막 위치의 섬에서는 처음에 위치한 섬으로 이주한다.

엘리트 전략을 사용할 경우 엘리트가 이주 대상에 반드시 포함되므로 이주가 자주 일어나면 적합도가 높은 개체들이 모든 섬에 빠르게 증식하여 전체 집단의 다양성을 해칠 수 있다. 이것을 막기 위해 이주 횟수에 비해 섬의 개수를 충분히 많게 한다.

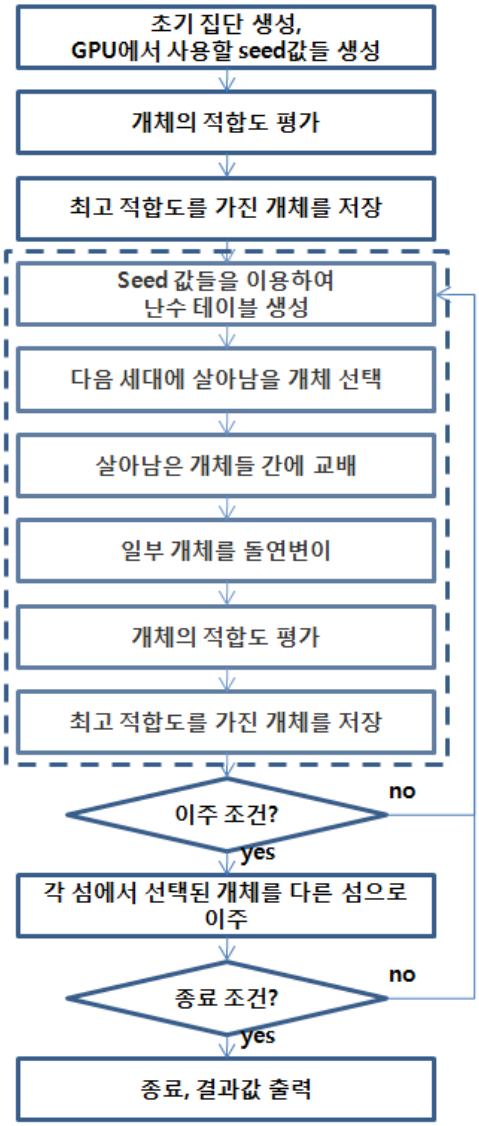


그림 4. 구현한 RVGA

4. 실험 및 결과

4.1 평가함수

평가에는 De Jong의 테스트 함수 5개[9]를 사용하였다. 모두 최소값을 찾는 문제이며 평가함수는

[0,1]을 반환하도록 설계되었다. 최소값을 찾았을 경우 적합도 1.0을 반환한다.

표 1. Benchmark functions

F1	$f(x)=\sum_{i=1}^{3}x_i^2$ $\min value: 0 = f(0,0,0)$
F2	$f(x,y)=100(y-x^2)^2+(1-x)^2$ $\min value: 0 = f(1,1)$
F3	$f(x)=\sum_{i=1}^{5}[x_i]$ $\min value: -30 = f(-5.12,-5.12,-5.12,-5.12,-5.12)$
F4	$f(x)=\sum_{i=1}^{30}ix_i^4+N(0,1)$ $\min value: 0 = f(0,0,0,0) \quad when \quad N(0,1)=0$
F5	$f(x_1,x_2)=[0.002+\sum_{j=1}^{25}[j+(x_1-a_{1j})^6+(x_2-a_{2j})^6]^{-1}]^{-1}$ $a_{ij}=\begin{pmatrix}-32 & -16 & 0 & 16 & 32 & -32 & \dots & 0 & 16 & 32\\ -32 & -32 & -32 & -32 & -32 & -16 & \dots & 32 & 32 & 32\end{pmatrix}$ $\min value: 1 \simeq f(-32,-32)$

- F3 의 $[x]$ 는 x 보다 작은 정수
- F4 의 $N(0,1)$ 은 평균이 0, 분산이 1인 정규분포 잡음

4.2 실험 조건

표 2. 실험조건

CPU	Intel Core2 6600 2.4GHz
RAM	2 GB
GPU	nVidia GeForce GTS 250
OS	Microsoft Windows 7
Tool	Microsoft VS2008, nVidia CUDA 2.3
토너먼트 상대 수	3
Selection threshold	0.7
Crossover ratio	0.8
Mutation ratio	0.05
Elite 수	1
이주 비율	지역 집단의 10%

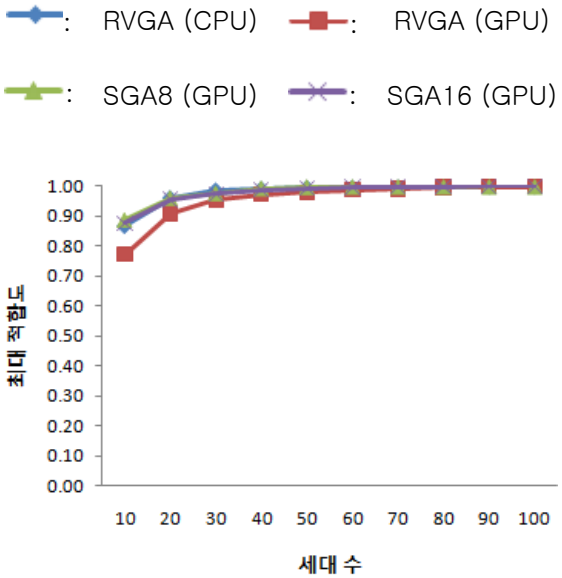
RVGA를 CUDA와 표준 C로 구현하고, 변수마다 8 비트, 16 비트를 사용하는 Simple GA를 CUDA로 구현하였다. 모든 GA는 섬 기반으로 동일하게 동작하며 표준 C로 구현된 RVGA는 병렬처리 대신 반복문을 사용한다

Simple GA에서 하나의 비트는 실제 비트를 사용하는 것이 아닌 float을 사용한다. 비록 메모리 낭비가 심하지만 비트연산에 따른 시간낭비가 없다.

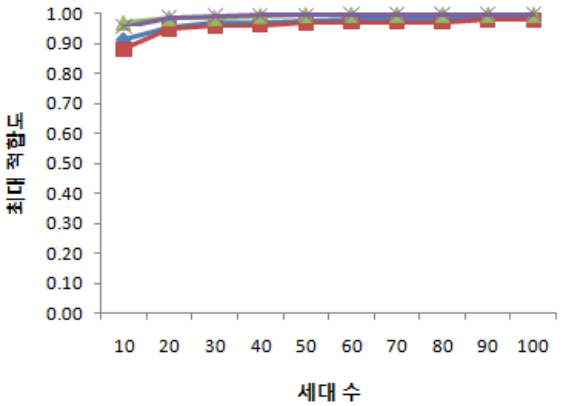
Simple GA가 RVGA와 다른 점은 개체의 적합도를 평가하는 과정에서 염색체의 복호과정이 추가된다는 점과 돌연변이 과정에서 염색체의 각 비트마다 난수를 골라서 mutation ratio이하의 숫자가 나오면 비트를 다른 값으로 변화 시킨다는 점 뿐이다.

4.3 탐색능력 비교

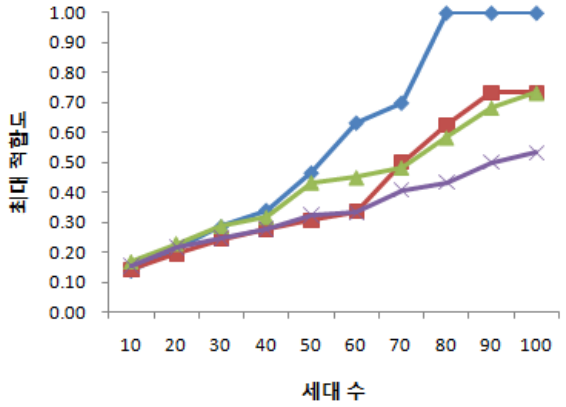
하나의 섬에 개체 수 10, 섬 개수 10개, 10세대마다 이주하며 총 100세대까지 진화 시킨다. 실험을 10번 하여, 그 평균값을 표시했다. Simple GA는 변수 마다 8비트를 사용한 것을 SGA8, 16비트를 사용한 것을 SGA16으로 한다.



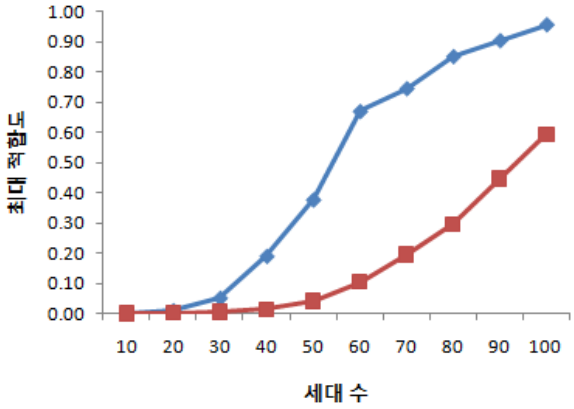
(a) De Jong F1 결과



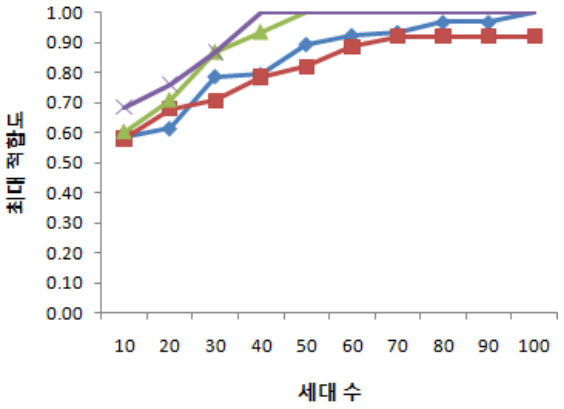
(b) De Jong F2 결과



(c) De Jong F3 결과



(d) De Jong F4 결과



(e) De Jong F5 결과

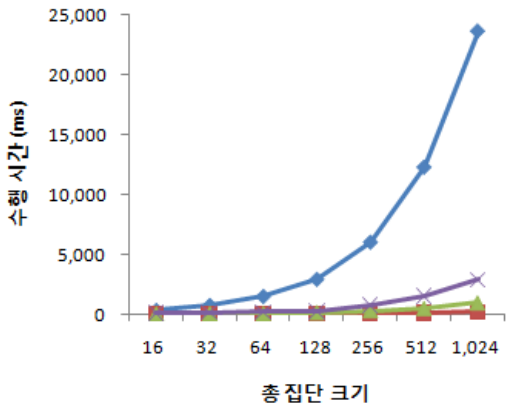
그림 5. 최대 적합도 비교

표준 C로 구현된 GA와 CUDA로 구현된 GA는 대체로 비슷한 탐색능력을 보여주지만, F3와 F4에서는 차이가 발생했다. 두 문제 모두 CUDA로 구현된 GA가 탐색성능이 떨어지는 것으로 나타났는데 표준 C로 구현된 GA와 차이 점은 다른 난수 발생기를 사용한다는 점이므로 난수 발생기의 성능이 탐색성능에 영향을 미친 것으로 보인다.

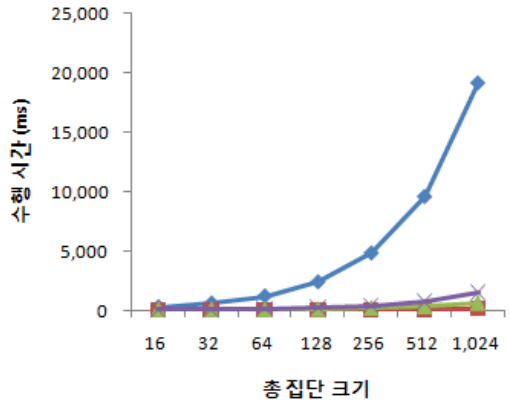
SGA8과 SGA16은 F4실험에서 공유메모리 부족 문제를 발생시켰다.

4.4 수행속도 비교

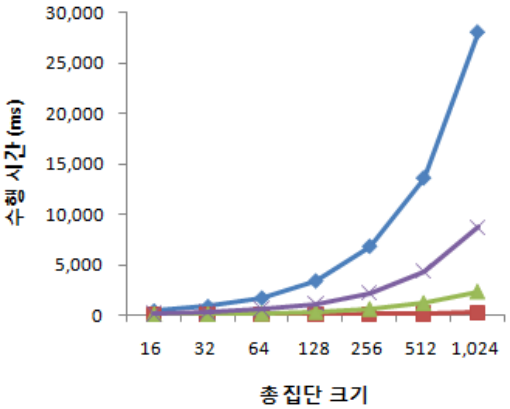
전체 집단 크기를 조정하면서 수행시간을 비교한다. 공유 memory가 16kb로 제한 되어 있어, 지역 집단의 크기를 자유롭게 정하는데 어려움이 있기 때문에, 지역 집단의 크기는 32로 고정하고 전체 집단의 크기를 늘리기 위해 섬의 개수를 늘리는 방법을 사용했다. 16, 32, 64, 128, 256, 512, 1024로 집단의 크기를 2배씩 늘려갔으며 그에 따른 수행시간을 각각 10번씩 측정하여 평균 수행시간을 측정한다. 이주는 100세대마다 수행하고 1000세대까지 진화시켰다. 시간의 단위는 ms이다.



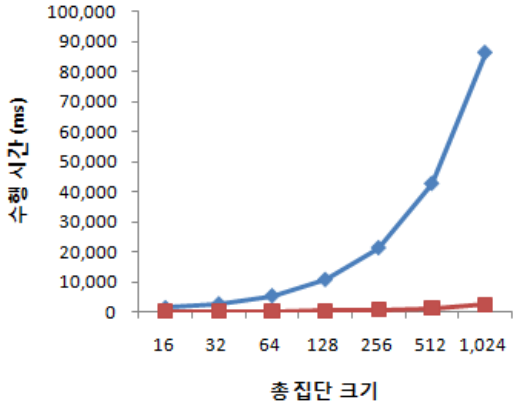
(a) De Jong F1 결과



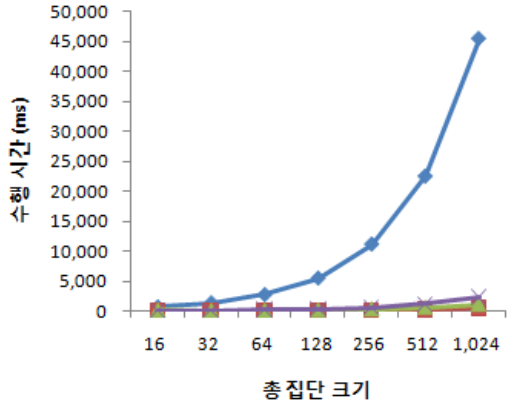
(b) De Jong F2 결과



(c) De Jong F3 결과



(d) De Jong F4 결과



(e) De Jong F5 결과

그림 6. 수행 시간 비교

전체 집단의 크기가 커질수록 모두 수행시간은 길어진다. CPU로 연산할 때는 집단의 크기가 2배 커질 때마다 전체 연산 시간은 약 2배 정도로 늘어나는 반면, GPU로 연산을 수행할 때는 2배 이하로 늘어난다. GPU의 경우에는 전체 프로세서가 모두 사용될 만큼 thread가 많지 않다면 집단의 크기가 커져도 수행시간은 거의 차이를 보이지 않는다.

GPU를 사용하는 RVGA가 복호과정이 필요 없어 가장 빠르며, simple GA는 비트를 더 많이 사용할수록 복호과정에서 시간이 더 오래 걸렸다.

이전 실험과 마찬가지로 F4에서 SGA8과 SGA16은 공유 메모리 부족 문제가 발생하였다.

5. 결론

GPU를 이용하여 GA를 병렬 처리하는 경우 약 4~64배의 향상이 있었다. 만약, 하나의 CPU만을 이용하여 동일하게 성능을 향상시키려 한다면 많은 비용을 들여야 하거나 심지어 전체 시스템을 교체하지 않는 이상 불가능할 수도 있다.

하지만 GPU의 한계도 있다. 첫째, 개발이 상대적으로 복잡하여, 복잡한 메모리 모델이나 GPU 하드웨어에 대해서 자세하게 알지 못하면 최적화가 힘들다.

둘째, 응용에 따라서는 적용하기 힘들 수도 있다. 실제 응용에서는 다양한 평가함수를 사용하게 될 것이다. 완벽하게 병렬처리 하기 위해서는 평가함수 또한 kernel로 작성해야 한다. 간단한 문제들의 경우에는 문제가 안되지만, 문제가 kernel로 작성하기 어렵거나 GPU의 개별 프로세서들이 처리하는 것이 CPU가 처리하는 것에 비해 매우 비효율적인 알고리즘의 경우에는 병렬화로 인한 이점을 많이 잃어버릴 것이다.

그 외에는 공유 메모리가 16kb로 제한되어 있다는 점과, double 형을 사용할 수 없어 해의 정밀도가 떨어진다는 점이 있다. 비록 최근에 출시한 고가의 GPU에서는 개선되었지만, 대부분의 GPU는 동일한 문제를 가지고 있다.

6. 감사의 글

이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2010-0012876)

7. 참고문헌

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Michigan, 1975.

[2] D. E. Goldberg, *Genetic algorithm in search, optimization and machine learning*, Addison-Wesley Publishing Co. Inc., N.Y., 1989.

[3] S. Debattisti, N. Marlat, L. Mussi, and S. Cagononi, “Implementation of simple genetic algorithm within the CUDA architecture.” *GPUs for Genetic and Evolutionary Computation Workshop*, 2009.

[4] P. Pospichal and J. Jaros, “GPU-based acceleration of the genetic algorithm.” *GPUs for Genetic and Evolutionary Computation Workshop*, 2009.

[5] nVidia Corporation, CUDA programming guide 2.3.

[6] 진강규, “유전알고리즘과 그 응용 제2판.” 교우사, 2004.

[7] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag Berlin Heidelberg New York, 1999.

[8] W. B. Langdon, “A fast high quality pseudo random number generator for nVidia CUDA.” *GPUs for Genetic and Evolutionary Computation Workshop*, 2009.

[9] K. A. De Jong, “An analysis of the behavior of a class of genetic adaptive systems”, *Doctoral dissertation*, The University of Michigan, Ann Arbor, Michigan, 1975.