유전 알고리즘의 GPU 구현 기법 및 비교 연구

GPU Implementation Techniques of Genetic Algorithm and Comparative Studies

현 병 용, 서 기 성*

(Byeong Yong Hyeon¹ and Kisung Seo¹)

¹Seokyeong University

Abstract: GPU (Graphics Processing Units) is consists of SIMD (Single Instruction Multiple Data) architecture and provides fast parallel processing. A GA (Genetic Algorithm), which requires large computations, is implemented in GPU using CUDA (Compute Unified Device Architecture). Three kinds of execution models are presented according to different combinations of processing modules in GPU. Comparison experiments between GPU models and CPU are tested for a couple of benchmark problems by variation of population sizes and complexity of problem sizes.

Keywords: GA (Genetic Algorithm), GPGPU, CUDA

I. 서론

GPU는 1999년 NVIDIA에 의해 처음 발표된 이래로 기존의 CPU보다 강력한 연산능력으로 많은 관심을 받아왔다. GPU는 CPU에 많은 부담을 줄 수 있는 특정 컴퓨터 그래픽스를 위한 연산에 사용되었으며, 대량의 데이터를 고속으로 처리할 수 있는 특징을 가지고 있다[1,2].

2006년도에 NVIDIA에 의해 GPU를 범용목적으로 쉽게 사용할 수 있도록 하는 프레임워크인 CUDA가 발표되면서, 개인용 컴퓨터에서 쉽게 고속의 병렬 연산 처리를 할 수 있게 되었다[1,2]. CUDA는 C/C++ 언어에 바탕을 두어 일반 사용자가 쉽게 구현할 수 있는 특징을 가지고 있다[2,3].

유전 알고리즘(GA: Genetic Algorithm) [4]은 자연계의 진화원리를 컴퓨터 알고리즘화 한 것으로, 강한 탐색 능력으로 인해 최적화 문제에 많이 적용되고 있다. 그러나 수행 시간이 오래 걸리는 단점이 있어서, 이를 개선하기 위한 노력이 다각도로 기울여져 왔다. 한편, 교배, 변이 등의 유전 연산과 목적함수의 연산 등이 GPU에서 처리하기 적합한 구조를 가지고 있기 때문에, GPU를 사용하여 이를 구현하고 수행하기 위한 시도가 이루어져왔다[5-9]. 대표적으로 두 가지 형태의 구현 방식으로 나눌 수 있는데, 적합도 평가만을 GPU를 통해 처리하는 방식[5,6]과 GA의 전반적인 프로세스를 GPU상에서 처리하는 방식[7-9]이 있다. 특히, 적합도 평가보다는 선택 연산자가 GPU를 이용하여 병렬화하기 어렵다. 이를 해결하기 위한 방법으로는 군집을 GPU의 메모리에 2차원 형태로 저장을 하고, 선택 연산자의 대상을 이웃하는 4방위의 개체로 제한을 두거나[9], 8방위의 개체로 제한을 두는 방법이 있다[8].

그러나 위의 대부분의 접근이 일반적인 유전 알고리즘의

GPU 구현보다는, GPU의 구조나 특성에 맞게 진화과정의 처리를 변형한 것이다.

본 논문은 선행연구[12,13]의 확장으로서, 일반적인 유전 알고리즘의 전 과정을 GPU에 구현하고, 유전 연산이나 적합도 평가와 같은 프로세스별 처리 시간에 대해 분석하여, 주요 프로세스 별로 GPU에 구현한다. 즉, GPU에서 수행하는 내용에 따라서 1) 적합도 평가만, 2) 유전 연산과 적합도 평가, 3) 유전 연산과 적합도 평가 및 난수 생성과정을 모두 수행 하는 3가지 종류 모델로 구분하여 구현한다. 그리고, 이를 대표적인 4가지 벤치마크 문제에 대해서 CPU 방식과의 다양한 지표의 비교 실험을 수행한다.

II. GPU와 CUDA

GPU가 발전하면서, 개인용 컴퓨터에서 고속의 대량 연산 처리가 가능해졌다. NVIDIA에 의하면 기존의 프로그램의 경우, 최소 20배에서 137배 가까이 빠른 것으로 알려졌다[1].

GPU는 3D 작업에서 사용되는 실수 벡터 연산에 최적화되어있는 구조를 가진 프로세서이다[2]. 일반적인 CPU와는 달리 다수의 코어가 각각 독립적인 FPU와 ALU를 가지고 있으며, 이것이 병렬적으로 연산이 이루어지는 형태이다. CPU와는 다른 처리 구조로 인해 기존의 프로그래밍 방식으로는 접근하기 어려운 단점이 있다.

CUDA는 NVIDIA의 GPU를 범용목적으로 사용할 수 있도록 도와주는 프레임워크이다. C/C++ 언어를 기반으로 사용할 수 있도록 구현되어 일반 사용자가 쉽게 접근할 수 있고, 그로 인해 많은 연구자들이 다양한 분야에 폭넓게 적용하고 있다[1].

CUDA의 프로그래밍 모델에서 Device(GPU)는 Host(CPU)의 계산을 도와주는 역할을 하며, 계산량이 많은 함수를 Device에서 실행하도록 지정할 수 있다. 이러한 기능은 커널(kernel)을 이용해서 수행한다. 또한, CUDA에서 Device와

* 책임저자(Corresponding Author)

논문접수: 2010. 7. 15., 수정: 2011. 2. 8., 채택확정: 2011. 3. 13.

현병용, 서기성: 서경대학교 전자공학과

(mrhyunby@naver.com/ksseo@skuniv.ac.kr)

Host의 메모리는 서로 명확히 구분되고, Device와 Host사이의 메모리는 복사가 가능하다. 그리고 Device의 코드가 수행되는 동안에 Host의 코드가 수행되는 형태를 가지고 있다[2].

III. GPU를 통한 유전 알고리즘 구현 기법 및 실험

1. 유전 알고리즘의 GPU 구현 방법 및 실험 환경

유전 알고리즘의 GPU 구현 부분의 정도에 따라 3가지의 기법으로 나누고 이를 CPU 구현 방식과 비교한다. GPU 구현기법은 GPU에서 수행되는 내용에 따라서 적합도 평가수행, 유전 연산과 적합도 평가 수행, 그리고 유전 연산과 적합도 평가 및 난수 생성과정을 모두 수행하는 3가지로 나누어진다. 각 기법별 전체 수행시간 및 부분 프로세스별 처리 시간에 대해서 분석한다.

실험 대상 문제는 유전 알고리즘의 테스트에 많이 쓰이는 대표적인 벤치마크 목적 함수를 선택하였다. 각각 Sphere(식 1), Schwefel(식 2), Griewank(식 3), Weierstrass function(식 4)으로 불린다. 특히, Weierstrass 함수는 다른 목적 함수들에 비해 계산 복잡도가 높은 것으로 알려져 있고, 내부적으로 반복되는 부분을 따로 가지고 있어서, 차수 R의 설정에 따라 복잡도의 조정이 가능하다.

$$F_{Sphere} = \sum_{k=1}^D x_k^2 \quad (1)$$

$$F_{Schwefel} = \sum_{k=1}^D \left(\sum_{j=1}^k x_j \right)^2 \quad (2)$$

$$F_{Griewank} = 1 + \frac{1}{4000} \sum_{k=1}^D x_k^2 - \prod_{k=1}^D \cos\left(\frac{x_k}{\sqrt{k}}\right) \quad (3)$$

$$F_{Weierstrass} = \sum_{k=1}^D \left(\sum_{j=1}^R b^{-j \cdot h} \sin(b^j x_k) \right) \quad (R=5) \quad (4)$$

Intel Core 2 Duo 6420(2.13GHz), nVidia Gefore 9800GT (14 Multiprocessor, 512MB), 4 GB Ram의 PC에서 실험을 수행하였으며, 운영체제는 Ubuntu Linux 9.04, CUDA 프레임워크는 v2.3을 사용하였다. 실험에는 아래와 같은 GA 파라미터를 사용했으며, 각 실험은 20회 반복되었다.

Number of generations: 1000

Population sizes: 2500

Selection: Tournament Selection, size = 7

Crossover: Arithmetic Crossover, 0.9

Mutation: Random Mutation, 0.1

Dimension of objective function(Genome Lengths): 10

Objective function: Sphere, Schwefel, Griewank, Weierstrass

Iteration of Weierstrass (R): 5

2. CPU에서의 GA 수행

CPU에서의 유전 알고리즘은 그림 1과 같이 구현하였다. 토너먼트 선택 연산자와 산술 교배 연산자, 랜덤 돌연변이를 사용하였고, 매 세대마다 유전연산, 적합도 평가, 엘리티즘을 CPU에서 반복 수행한다. 본 논문에서는 이러한 수행방법을 Host로 명한다.

그림 1. 전체 유전 알고리즘의 Host 수행.

Fig. 1. Processing of entire genetic algorithm on Host.

표 1. 전체 유전 알고리즘의 Host 수행 결과(초, 백분율).

Table 1. The results for processing of entire genetic algorithm on Host(second, percentage).

목적 함수	Sphere	Schwefel	Griewank	Weierstrass
합계	1.498 (100 %)	1.842 (100 %)	3.984 (100 %)	27.241 (100 %)
유전 연산	1.451 (96.86 %)	1.456 (79.05 %)	1.432 (35.94 %)	1.427 (5.24 %)
적합도 평가	0.036 (2.40 %)	0.376 (20.41 %)	2.539 (63.73 %)	25.770 (94.60 %)
기타	0.011 (0.74 %)	0.01 (0.54 %)	0.013 (0.33 %)	0.044 (0.16 %)

일반적으로 진화연산중 적합도 평가 수행시간의 비중이 매우 높은 것으로 알려져 있으나, 사전 실험 결과 벤치마크 함수의 복잡도에 따라서 상당한 차이가 있음을 알 수 있다(표 1). Sphere 함수와 Schwefel 함수의 경우는 유전 연산이 높은 비중을 차지하였고, 계산 복잡도가 높은 Griewank 함수와 Weierstrass 함수의 경우에는 적합도 평가가 높은 비중을 차지하였다.

3. 적합도 평가의 GPU 병렬 처리

먼저, 적합도 평가만을 GPU(Device)를 통해 병렬 처리하도록 구현하였으며, D-eval이라고 명한다. 그림 2와 같이 GPU에서 적합도 평가를 수행하기 이전에 GPU의 전역 메모리에 군집을 전달하고, 적합도 평가를 수행한 이후에 적합도를 HOST로 전달한다.

GPU를 통해 적합도 평가를 병렬 처리한 알고리즘을 동일한 GA 파라미터를 사용하여 실험하였다. 표 2에서 볼 수 있듯이 적합도 평가의 병렬 처리는 계산 복잡도가 작은 함수에서는 Host 방식보다 수행 시간이 오래 걸리거나 약간의 향상만이 있었다. 계산 복잡도가 큰 목적 함수일수록 시간 단축량이 큰 것을 볼 수 있었다. Sphere 함수의 경우에는 적합도 평가 프로세스의 수행 시간의 향상이 있었지만, 군집과 메모리 전송의 시간과의 합까지 고려할 경우에는 Host 방식보다 약간 지연되는 현상이 나타남을 알 수 있다.

그림 2. 적합도평가(evaluation)의 Device 수행 방식(D-eval).
Fig. 2. Processing of evaluation on Device.

표 2. 적합도 평가의 디바이스 수행 결과(초, 백분율).

Table 2. The result for Processing of evaluation on Device(second, percentage).

목적 함수	Sphere	Schwefel	Griewank	Weierstrass
합 계	1.621 (100 %)	1.685 (100 %)	1.627 (100 %)	1.704 (100 %)
유전 연산	1.457 (89.88 %)	1.448 (85.93 %)	1.461 (89.8 %)	1.436 (84.27 %)
군집 전송 Host→Device	0.069 (4.26 %)	0.069 (4.1 %)	0.069 (4.24 %)	0.069 (4.05 %)
적합도 평가	0.029 (1.79 %)	0.102 (6.05 %)	0.031 (1.9 %)	0.133 (7.81 %)
적합도 전송 Device→Host	0.02 (1.23 %)	0.019 (1.13 %)	0.02 (1.23 %)	0.02 (1.17 %)
기타	0.046 (2.84 %)	0.047 (2.79 %)	0.046 (2.83 %)	0.046 (2.7 %)

4. 적합도 평가와 유전 연산의 GPU 병렬 처리

CPU와 GPU간의 메모리 전송에 소비되는 시간을 줄이기 위해서는 군집을 GPU의 메모리에 보관하여야 한다. 따라서 유전 연산 역시, GPU에서 수행되어야 한다. 유전 연산을 위해서는 선택, 교배, 돌연변이 연산자에 필요한 난수를 제공하여야 한다. 산술 교배 연산자에 필요한 가중치는 돌연변이 연산자에 해당하는 난수를 공유한다. 한 세대에서 필요한 난수의 개수는 아래와 같다.

선택 연산자: 군집의 크기 $\times$ (2 $\times$ 토너먼트 크기)
 돌연변이 연산자: 군집의 크기 $\times$ 유전자의 길이
 유전 연산 방법: 유전 연산 방법 결정에 필요한 확률(1)
 전체 필요한 난수의 개수
 = 군집의 크기 $\times$ (2 $\times$ 토너먼트 크기 + 유전자의 길이 + 1)

그림 3과 같이 매 세대마다 CPU에서 생성한 난수는 유전 연산 수행 이전에 GPU에 전송되고, GPU에서는 전송받은 난수를 바탕으로 진화 연산을 수행한다. 새로 생성된 군

그림 3. 적합도평가(evaluation)와 유전연산(evolution)의 Device 수행 방식(D-evol+eval).

Fig. 3. Processing of evolution and evaluation on Device.

표 3. 적합도 평가 및 유전 연산의 디바이스 수행 결과(초, 백분율).

Table 3. The result for Processing of evaluation and evolution on Device(second, percentage).

목적 함수	Sphere	Schwefel	Griewank	Weierstrass
합 계	1.915 (100 %)	1.961 (100 %)	1.964 (100 %)	1.985 (100 %)
난수 생성	1.531 (79.95 %)	1.503 (76.64 %)	1.58 (80.45 %)	1.49 (75.06 %)
난수 전송 Host→Device	0.157 (8.2 %)	0.156 (7.96 %)	0.156 (7.94 %)	0.157 (7.91 %)
유전 연산	0.14 (7.31 %)	0.14 (7.14 %)	0.14 (7.13 %)	0.141 (7.1 %)
군집 교체 Device→Device	0.015 (0.79 %)	0.015 (0.77 %)	0.015 (0.76 %)	0.015 (0.76 %)
적합도 평가	0.029 (1.5 %)	0.102 (5.2 %)	0.03 (1.53 %)	0.134 (6.75 %)
기타	0.043 (2.25 %)	0.045 (2.29 %)	0.043 (2.19 %)	0.048 (2.42 %)

집은 GPU에서 GPU로의 메모리 전송을 통하여 복사된다. 본 논문에서는 이러한 수행 방법을 D-evol + eval이라고 명한다.

표 3에 GPU를 통해 유전 연산과 적합도 평가, 엘리티즘을 수행한 결과가 나와 있다. 전체적인 수행 시간이 적합도 평가만을 수행하는 방법보다는(표 2) 10-15%정도 약간 늘어나는 것을 알 수 있다. CPU에서 유전 연산을 수행할 경우에는 각 유전 연산자에 필요한 만큼의 난수를 생성하는 반면에, GPU에서 유전 연산을 수행할 경우에는 필요한 만큼의 충분한 난수를 CPU에서 생성하여 GPU에 제공하여야 한다. 이러한 이유로 난수 생성 및 전송에서의 수행 시간이 기존의 방법보다 오래 걸리기 때문이다.

5. 적합도 평가, 유전 연산, 난수 생성의 병렬 처리

CPU에서 난수를 생성하고 전송하는 과정에 소모되는 시

그림 4. 적합도평가(evaluation)와 유전연산(evolution), Mersenne Twister의 Device 수행 방식(Device).

Fig. 4. Processing of evolution, evaluation, and Mersenne Twister on Device.

표 4. 적합도 평가, 유전 연산 및 Mersenne Twister의 디바이스 수행 결과(초, 백분율).

Table 4. The result for Processing of evaluation, evolution, and Mersenne Twister on Device(second, percentage).

목적 함수	Sphere	Schwefel	Griewank	Weierstrass
합 계	0.375 (100 %)	0.449 (100 %)	0.379 (100 %)	0.483 (100 %)
시드 생성	0.042 (11.2 %)	0.042 (9.35 %)	0.042 (11.08 %)	0.042 (8.7 %)
Mersenne Twister	0.099 (26.4 %)	0.099 (22.05 %)	0.1 (26.39 %)	0.1 (20.7 %)
유전 연산	0.146 (38.93 %)	0.145 (32.3 %)	0.145 (38.26 %)	0.145 (30.02 %)
군집 교체 Device→Device	0.015 (4 %)	0.015 (3.34 %)	0.015 (3.96 %)	0.015 (3.10 %)
적합도 평가	0.03 (8 %)	0.103 (22.94 %)	0.032 (8.44 %)	0.136 (28.16 %)
기타	0.043 (11.47 %)	0.045 (10.02 %)	0.045 (11.87 %)	0.045 (9.32 %)

간 문제를 해결하기 위해서, CUDA SDK에서 제공하는 Mersenne Twister 알고리즘을 사용하여 GPU 상에서 난수 생성을 구현하였다[10]. Mersenne Twister는 긴 주기와 효율적인 메모리 사용, 좋은 분포 특성, 높은 성능으로 알려져 있다[11]. 그림 4와 같이 매 세대마다 Mersenne Twister에 사용되는 Seed 값을 Host에서 생성한 난수 값을 사용함으로써 알고리즘의 신뢰성을 향상시켰다. 이후의 유전 연산 및 적합도 평가, 엘리티즘은 3절에서 사용한 방법과 동일하게 수행되었다. 본 논문에서는 이러한 수행 방법을 Device 라고 명한다.

이번 실험에서는 CPU를 통해 Mersenne Twister에 필요한 Seed를 생성하고, GPU를 통해 난수 생성, 유전 연산, 적합도 평가, 엘리티즘을 수행하였다. 표 4에서 볼 수 있듯이

GPU를 이용한 앞의 두 방법에 비하여 많은 시간 단축이 있었다.

표 1-4를 종합적으로 분석해 볼때, 2500 개체수에서의 실험시 Sphere 함수와 Schwefel 함수의 경우에는 GPU에서의 전체 구현시의 수행시간 결과가(표 4) CPU 방식보다는(표 1) 약 1/4로 감소되었으나, 적합도 평가 또는 유전연산의 일부 구현의(표 2와 3) 경우는 오히려 CPU 에서의 구현보다 별 차이가 안 나거나 약간 높은 경우도 나타났다.

이는 Sphere 함수와 Schwefel 함수의 구성이 수행되는 연산에 비해서, GPU의 전역 메모리로부터 유전자를 불러오는 횟수, 즉 메모리 접근량이 많기 때문인 것으로 보인다.

Griewank, Weierstrass 함수에 대한 결과에서는 부분적인 GPU 구현을 포함한 모든 GPU 구현 방식이 CPU 구현 방식보다 모두 상당한 폭으로 감소함을 보였고, 문제의 복잡도가 큰 Weierstrass 함수의 경우에는 약 1/56 로 대폭 감소하였다.

IV. GPU 수행 방법 및 개체수 변화의 실험 및 결과

III 장에서 구현하고 실험한 내용을 개체수의 크기에 따른 확장하여 실험한다. 유전 알고리즘의 파라미터는 동일하며, 개체집단의 크기를 1000부터 5000까지 500씩 증가시켜 가며 실험을 수행한다.

Population sizes : [1000, 5000], $\Delta 500$

실험 결과는 각 요소별로 20번의 반복 실험에 대한 평균 값을 산출하였다. 그림 5-8의 각 인덱스는 III 장에서 소개한 4가지 구현 방법을 나타낸다.

Host: Host를 통해 전체 GA 수행

D-eval: Device를 통해 적합도 평가만 수행

Host를 통해 유전 연산과정 수행

D-evol+eval: Device를 통해 유전 연산과 적합도 평가,

Host를 통해 유전 연산에 필요한 난수 생성

Device: Device를 통해 전체 GA 수행

Host를 통해 난수 생성에 필요한 Seed 생성

그림 5는 Sphere 함수에 대한 4가지 구현방식의 수행 시간을 보여준다. Sphere 함수의 경우 1000~5000의 군집 크기에서 Host 방식에 비하여 D-eval과 D-evol+eval 방식의 수행 시간이 더 큰 것을 볼 수 있다. 반면에 Device 방식은 군집 크기가 증가할수록 Host 방식과의 시간차가 커지는 것을 볼 수 있다. 또한 군집 크기가 증가함에 따라서 두 방식의 속도향상(Host/Device)이 약 2.3배~5.6배로 증가하는 것을 볼 수 있다.

이를 통해, 계산 복잡도가 낮은 Sphere 함수에서는 군집 크기가 커지더라도 D-eval 방식과 같이 매 세대마다 Host와 Device간의 메모리 전송이 전체적인 수행시간에 큰 영향을 주는 것을 알 수 있다.

Schwefel 함수에 대한 수행 시간의 비교가 그림 6에 나와있다. D-eval 방식은 Host 방식에 비하여 약간의 수행 시간 이득(약 0.03초~0.32초)이 있었고, D-evol+eval은 Host 방

그림 5. Sphere 함수의 평균 수행 시간.

Fig. 5. Average execution time of Sphere function.

그림 6. Schwefel 함수의 평균 수행 시간.

Fig. 6. Average execution time of Schwefel function.

그림 7. Griewank 함수의 평균 수행 시간.

Fig. 7. Average execution time of Griewank function.

식에 비하여 약간의 수행 시간 손실(약 0.07초~0.44초)이 있었다. D-eval 방식은 군집 크기가 커질수록 Host 방식과의 수행시간 차가 커졌다. 반면에 Device 방식은 Sphere 함수 일 때와 마찬가지로 Host 방식과의 수행시간 차가 커졌지만, 속도향상(host/device) 약 2.56배~5.26배로 Sphere 함수와 비슷하였다.

그림 8. Weierstrass 함수(R=5)의 평균 수행 시간(log scale).

Fig. 8. Average execution time of Weierstrass function(R=5)(log scale).

그림 7에는 Griewank 함수에 대한 수행 결과가 나와있다. Griewank 함수는 앞의 두 함수에 비하여 적합도 평가의 수행시간이 전체 알고리즘의 수행 시간에 큰 비중을 차지한다. 이런 이유로 Host 방식과 GPU를 통해 일부의 GA를 구현한 D-eval, D-eval+eval 방식, GPU를 통해 전체 GA를 구현한 Device 방식의 수행시간 차이가 분명히 나타난다. III 장 4절에서 명한 것처럼 D-eval 방식보다 D-eval+eval 방식의 수행 시간이 더 걸린다.

그림 8에는 Weierstrass 함수에 대한 수행 결과가 나와 있으며, 앞의 3가지 경우와는 달리 계산 복잡도가 높기 때문에 수행 시간을 로그스케일로 나타내었다. Weierstrass 함수는 다른 함수들보다, CPU만 사용한 Host 방식과 GPU를 사용한 3가지 방식간의 수행시간 차이가 더 크게 나타난다. Host 방식에 비하여 D-eval 방식은 약 13.5배~16.4배, D-eval+eval 방식은 약 11.2배~14.2배, 그리고 Device 방식은 28.9배~78.3배의 속도향상을 보였다.

본 장에서는 계산 복잡도가 다른 4개의 목적함수를 대상으로 군집 크기에 따른 4가지 구현 방법에 대한 수행 시간 비교 실험을 하였다. 계산 복잡도가 낮은 Sphere 함수와 Schwefel 함수에서는 GPU를 이용하여 GA의 일부분만을 수행하는 D-eval, D-eval+eval 방식이 CPU 수행방식에 비해서 비슷하거나 오히려 약간 효율이 떨어짐을 보였으나, Griewank과 Weierstrass 함수에서는 상당한 수행시간의 감소를 나타내었다. 더욱이, GPU를 이용하여 전체 GA를 수행하는 Device 방식은 목적함수의 계산 복잡도가 낮은 함수를 포함한 모든 문제에서 충분히 높은 성능 향상을 보였다.

V. 문제의 크기 변화 실험 및 결과

IV 장의 실험결과 GPU(Device)를 통해 전체 GA를 수행하는 방식의 성능이 가장 우수함을 알 수 있었다. 본 장에서는 이 방식과(device) CPU 방식만의 비교를 문제의 크기 변화, 즉 문제를 표현하는 유전체의 크기 변화를 통해, 평균 수행 시간과 CPU/GPU의 속도 향상비율을 실험한다. 유전 알고리즘 파라미터는 앞과 유사하며, 다음 파라미터만 추가된다.

Problem Size(Genome length) : 10, 20, 40, 60, 80, 100

그림 9, 11, 13, 15에서는 각 목적 함수의 크기가 20, 60, 100일 경우에 대한 CPU와 GPU 방식의 평균 수행 시간의 실험 결과이고, 그림 10, 12, 14, 16은 군집의 크기와 문제의 크기에 따른 4가지 목적함수에서의 CPU와 GPU 방식의 수행 시간 비율이다.

먼저, 그림 9는 Sphere 문제의 수행 시간의 차이를 보여 주는데, GPU의 수행시간이 CPU에 비해 다소 짧음을 알 수 있다. 참고로 y 축의 수행시간은 로그-스케일이다.

그림 10은 CPU 대비 GPU 수행의 속도 향상을 나타낸다. 유전자의 길이가 10일 때, 군집의 크기별로 수행 시간 향상은 최소 2.2배에서 최대 5.5배의 향상이 있었다. 그러나 유전자의 길이가 커짐에 따라 수행 시간 향상이 1.6배 정도로 떨어지는 현상을 보였다. 이러한 현상은 목적 함수의 크기가 커짐에 따른 유전자에 대한 연산 횟수의 변동에 의한 수행시간 이득 보다 메모리로부터 유전자를 불러오는 횟수의 변동에 의한 수행시간 손실이 더 크기 때문으로 사료된다.

그림 11과 12는 Schwefel 함수의 수행 시간과 속도향상을 보여주고 있다. Sphere 함수에 비하여 CPU와 GPU의 수행 시간 영역이 분리되는 것을 볼 수 있다. 이러한 현상은

그림 9. Sphere함수의 평균 수행 시간(log scale).
Fig. 9. Average execution time of Sphere function(log scale).

그림 10. Sphere 함수의 속도 향상(CPU/GPU).
Fig. 10. Speedup on Sphere function (CPU/GPU).

Sphere 함수에 비하여 목적 함수의 계산 복잡도가 높기 때문인 것으로 사료된다.

유전자의 길이가 10일 때, 군집의 크기별로 수행 시간 향상은 최소 2.6배에서 최대 5.3배의 향상이 있었다. 그러나 Sphere 함수와 마찬가지로 유전자의 길이가 커짐에 따라 수행 시간 향상이 2배 정도로 떨어지는 현상을 보였다. 이러한 현상은 위에서 분석한 원인과 동일한 것으로 사료된다.

그림 13은 Griewank 함수의 평균 수행 시간, 그림 14는 속도 향상을 나타낸다. Griewank 함수는 앞의 두 목적 함수

그림 11. Schwefel함수의 평균 수행 시간(log scale).
Fig. 11. Average execution time of Schwefel function(log scale).

그림 12. Schwefel 함수의 속도 향상(CPU/GPU).
Fig. 12. Speedup on Schwefel function (CPU/GPU).

그림 13. Griewank함수의 평균 수행 시간(log scale).
Fig. 13. Average execution time of Griewank function(log scale).

에 비하여 더 복잡한 연산을 수행하는 특징을 가지고 있어서, 모든 경우의 문제크기가 커짐에 따라 CPU 대비 GPU의 수행이 더욱 우수한 성능을 보임을 알 수 있다. 또한, 전체 수행 시간 향상은 최소 6배에서 최대 18배로 나타났다.

그림 15와 16은 Weierstrass 함수($R=5$)에 대한 실험 결과이다. Weierstrass 함수의 경우도 Griewank 함수와 같이 문제 크기 증가에 따라 더욱 CPU에 대한 GPU의 수행 속도 향상이 월등함을 보여 주었으며, 최소 28배에서 최대 106배

그림 14. Griewank 함수의 속도 향상(CPU/GPU).

Fig. 14. Speedup on Griewank function (CPU/GPU).

그림 15. Weierstrass 함수($R=5$)의 평균 수행 시간(log scale).

Fig. 15. Average execution time of Weierstrass function($R=5$) (log scale).

그림 16. Weierstrass 함수($R=5$)의 속도 향상(CPU/GPU).

Fig. 16. Speedup on Weierstrass function($R=5$)(CPU/GPU).

에 이르는 성능 향상을 보여주고 있다.

다만, 3500 개체에서 전후의 개체 수에 비해서 성능 향상비율이 낮게 나오는 특이한 현상이 존재한다.

VI. 결론

논문에서는 GPU를 통해 유전 알고리즘의 연산을 병렬로 처리함으로써 CPU에서의 처리보다 단축된 수행시간을 얻고자 하였다. 적합도 평가, 유전 연산과 적합도 평가, 유전 연산과 적합도 평가 및 난수 생성과정을 GPU에서 수행하는 3가지 모델로 분리 구현하고, 이를 벤치마크 목적 함수에 대해서 비교 실험하였다.

실험을 통해, 유전 알고리즘의 전 과정을 GPU에서 수행하는 방법이 가장 높은 성능을 발휘함을 확인할 수 있었다. 또한, 목적 함수의 계산 복잡도가 높아질수록 GPU가 속도 향상에 있어서 큰 영향을 미침을 알 수 있었다.

4개의 목적 함수에 대해서 군집의 크기, 유전자의 길이, 목적 함수의 계산 복잡도에 따른 변화를 실험한 결과, GPU에서의 수행시간이 CPU보다 적었으며, 수배에서 수십 배의 속도향상이 나타났다. 특히, 내부적으로 반복을 통해 계산 복잡도를 증가시킬 수 있는 Weierstrass 문제가 GPU의 수행 시간 단축 효과가 더 뛰어남을 알 수 있었다.

이를 통해서 문제의 규모나 특성에 따른 GPU 구현방법의 선택에 대한 정보를 제공할 수 있으면, 유전 알고리즘의 GPU 구현시의 예상효과 등을 사전에 파악하는데 도움을 줄 수 있다.

향후 GPU 상의 유전 알고리즘 구현을 최적화하고, 다중 군집의 병렬처리 구현 등의 연구가 필요하다고 생각된다.

참고문헌

- [1] 이주석, 류현곤, "GPU 병렬 컴퓨팅 기술을 이용한 개인용 수퍼 컴퓨터 현황과 전망," 전자공학회지, 제36권 제5호, 2009.
- [2] NVIDIA, CUDA Programming Guide, NVIDIA Corporation, 2009.
- [3] NVIDIA, CUDA C Programming Best Practices Guide, NVIDIA Corporation, 2009.
- [4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [5] O. Maitre, L. A. Baumes, N. Lachiche, A. Corma, and P. Collet, "Coarse grain parallelization of evolutionary algorithms on GPU cards with EASEA," *Proc. of the Genetic and Evolutionary Computation Conference, GECCO-2009*, Montreal Quebec, Canada, pp. 1403-1410, Jul. 2009.
- [6] O. Maitre, N. Lachiche, P. Clauss, L. Baumes, A. Corma, and P. Collet, "Efficient parallel implementation of evolutionary algorithms on GPU cards," *Proc. of the 15th International Euro-Par Conference*, Delf, Netherlands, 5704, pp. 974-985, Aug. 2009.
- [7] Q. Yu, C. Chen, and Z. Pan, "Parallel genetic

algorithms on programmable graphics hardware,” *Proc. of the First International Conference, ICNC 2005*, Changsha, China, vol. 3612, pp. 1051-1059, Aug. 2005.

- [8] J. M. Li, X. J. Wang, R. S. He, and Z. X. Chi, “An efficient fine-grained parallel genetic algorithm based on GPU-accelerated,” *Proc. of the IFIP International Conference, Network and Parallel Computing Workshops-2007*, pp. 855-862, Sep. 2007.
- [9] A. Munawar, M. Wahib, M. Munetomo, and K. Akama, “Hybrid of genetic algorithm and local search to solve MAX-SAT problem using nVidia CUDA framework,” *Genetic Programming and Evolvable Machines*, vol. 10, no. 4, Dec. 2009.
- [10] V. Podlozhnyuk, *Parallel Mersenne Twister, CUDA SDK Documentation*, 2007.
- [11] M. Matsumoto and T. Nishimura, “Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3-30, Jan. 1998.
- [12] 현병용, 권오성, 현수환, 서기성, “유전 알고리즘의 GPU 구현,” *한국지능시스템학회 학술발표 논문집*, 마산, 한국, vol. 20, no. 1, pp. 3-6, Apr. 2010.
- [13] 현병용, 김영균, 서기성, “전체 유전알고리즘의 GPGPU 구현 및 문제크기 변화에 따른 수행시간 비

교,” *ICROS 학술대회*, 춘천, 한국, pp. 436-438, May 2010.

현 병 용

2005년~현재 서경대학교 전자공학과 학사과정. 관심분야는 GA, GPGPU.

서 기 성

1986년 연세대학교 전기공학과 공학사. 1988년 연세대학교 전기공학과 공학석사. 1993년 연세대학교 전기공학과 공학박사. 1993년~1998년 서경대학교 산업공학과, 전자공학과 조교수. 1999년~2003년 Michigan State University, Genetic Algorithms Research and Applications Group, Research Associate. 2002년~2003년 Michigan State University, Electrical & Computer Engineering, Visiting Assistant Professor. 2004년~현재 서경대학교 전자공학과 부교수. 관심분야는 지능로봇, GA, GP, 진화신경망, 진화 디자인.