GPU의 병렬 처리 기능을 이용한 PSO(Particle Swarm Optimization) 알고리듬 구현

김은수, 김조환, 김종욱
동아대학교 전자공학과

Implementation of PSO(Particle Swarm Optimization) Algorithm using Parallel Processing of GPU

Kim Eun-Su, Kim Jo-Hwan, Kim Jong-Wook
Department of Electronic, Dong-A University.

Abstract - 본 논문에서는 연산 최적화 알고리듬 중 PSO(Particle Swarm Optimization) 알고리듬을 NVIDIA社에서 제공한 CUDA(Compute Unified Device Architecture)를 이용하여 새롭게 구현하였다. CUDA는 CPU가 아닌 GPU(Graphic Processing Unit)의 다양한 병렬 처리 능력을 사용해 복잡한 컴퓨팅 문제를 해결하는 소프트웨어 개발을 가능케 하는 기술이다. 이 기술을 연산 최적화 알고리듬 중 PSO에 적용함으로써 알고리듬의 수행 속도를 개선하였다. CUDA를 적용한 PSO 알고리듬의 검증을 위해 C 언어 기반으로 프로그래밍하고 다양한 Test Function을 통해 시뮬레이션 하였다. 그리고 기존의 PSO 알고리듬과 비교 분석하였다. 또한 알고리듬의 성능 향상으로 여러 가지 최적화 분야에 적용 할 수 있음을 보인다.

1. 서 론

GPU(Graphic Processing Unit)란 대용량 부동소수점 데이터를 병렬 고속 처리를 통해 매우 빠르게 계산할 수 있는 컴퓨터 부품의 핵심으로 3D게임과 자동차 시뮬레이션, H.264 등의 방송 콘텐츠와 U-헬스케어 분야에서 주로 활용되고 있다. 반면 CPU는 고정, 부동소수점을 모두 처리할 수 있지만 부동소수점연산에서 GPU 성능을 따라잡기엔 한계가 있다.

최근 GPU의 성능이 획기적으로 높아지면서 CPU가 처리해야 할 데이터를 대신 처리하게 되었다. 특히 NVIDIA에서 CUDA(Compute Unified Device Architecture), AMD에서 OpenCL(Open Computing Language), Intel에서 Larrabee를 선보이면서 'GPU컴퓨팅'이라는 말이 대두되었다. GPU컴퓨팅은 GPGPU(General Purpose computation on GPUs)로 표현할 수 있는데, GPGPU는 GPU를 비 그래픽 애플리케이션에 응용하려는 것을 통칭해 부르는 것을 말한다.

GPGPU 프로그래밍은 각종 셰이딩 언어 등을 알고 있어야 하는 고급 프로그래밍 기술에 속하기 때문에 일반 개발자가 프로그래밍 하기 어렵다. 그러나 CUDA는 C언어 기반의 개발 환경이기 때문에 그래픽 API를 모르더라도 손쉽게 GPU를 활용해 성능 향상을 시킬 수가 있다.

현재 NVIDIA에서 CUDA를 이용해 동영상 인코딩 고속 구현, 금융 파생상품 측정 소프트웨어 가속화 등 다양한 어플리케이션의 고속화를 이루었다. 그리고 CUDA를 이용한 고속화 및 구현에 관한 논문들이 발표되는 등 다수의 관련 연구 보고가 이루어지고 있다. CUDA 라이브러리를 사용한 블록 암호의 고속 구현화도 발표되었고[1], CUDA와 OpenMP를 이용한 신경망 구현에 대한 연구가 보고되고 있다[2].

본 논문에서는 연산 최적화 알고리듬 중 PSO(Particle Swarm Optimization) 알고리듬을 CUDA를 이용하여 새롭게 구현하였다. CUDA는 CPU가 아닌 GPU의 다양한 병렬 처리 능력을 사용해 복잡한 컴퓨팅 문제를 해결하는 소프트웨어 개발을 가능케 하는 기술이다. 이 기술을 연산 최적화 알고리듬 중 PSO에 적용함으로써 알고리듬의 수행 속도를 개선하였다. CUDA를 적용한 PSO 알고리듬의 검증을 위해 C 언어 기반으로 프로그래밍하고 다양한 Test Function을 통해 시뮬레이션 하였다. 그리고 기존의 PSO 알고리듬과 비교 분석하였다. 또한 알고리듬의 성능 향상으로 여러 가지 최적화 분야에 적용 할 수 있음을 보인다.

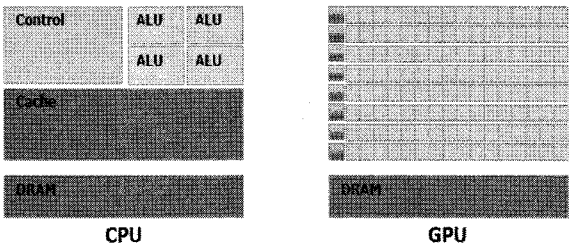
2. 본 론

2.1 CUDA(Compute Unified Device Architecture)

본 절에서는 먼저 CUDA 기술에 대한 이해를 돕기 위해 GPU의 구조에 대해 간단히 설명한다. 이어서 PSO 알고리듬 구현에 사용된 CUDA 및 CUDA 프로그래밍 기법에 대해 설명한다.

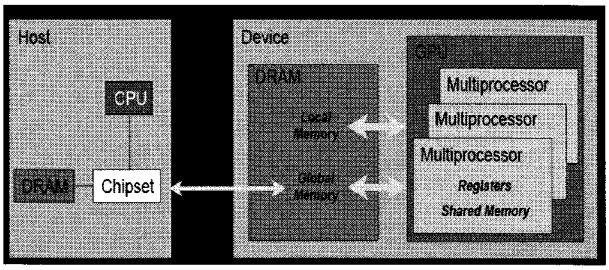
2.1.1 GPU 구조

그래픽 카드는 그래픽 프로세서(GPU)와 메모리를 가지고 있다. GPU는 CPU의 부족한 그래픽 작업을 해결하기 위해 고안된 특수 목적 처리 장치로 그래픽에 관련된 부동 소수점 연산들을 주로 담당한다. 호스트 컴퓨터로부터 받은 데이터를 버퍼에 넣기 위한 병렬 구조를 가지고 있어서 32비트 부동소수점 연산을 빠르게 수행할 수 있다. 그림 1은 CPU와 GPU의 비교 그림으로써, CPU는 대부분을 제어부분과 캐시 메모리가 차지하는 반면 GPU는 제어 부분이 거의 없고 대부분이 연산기(ALU)가 차지하고 있어서 연산이 많은 프로그램 수행 시 상당한 성능을 보여준다[3].



<그림 1> CPU와 GPU 비교

2.1.2 CUDA 프로그래밍 기법



<그림 2> 호스트 PC와 GPU의 프로그램 연산 과정

CUDA는 NVIDIA에서 2007년에 발표한 범용 GPU 프로그래밍 라이브러리로써 C언어를 사용하여 GPU를 통한 범용 컴퓨팅을 할 수 있게 해주는 GPGPU 기술이다. NVIDIA에서는 CUDA 드라이버 및 개발 툴킷을 배포하고 있으며 Matlab, 포토샵 플러그인 또한 제공하고 있다. CUDA는 Geforce 80시리즈 이상부터 동작을 한다.

CUDA는 GPU의 메모리를 register, shared메모리 등의 온칩(on-chip)과 global, local 메모리 등의 오프칩(off-chip)으로 나누어서 관리한다. 그리고 메모리는 그리드(grid), 블록(black), 쓰레드(thread)의 구성에 의해 영향을 미친다. 메모리의 계층 구조를 활용하여 그리드와 쓰레드로 프로그램을 구성한다.

호스트 PC에서 GPU로 호출되는 서브프로그램을 커널(kernel)이라고 한다. 커널은 그리드로 구성되는데, 그리드는 블록들로 구성되고 블록은 다시 여러 개의 쓰레드로 구성된다. 블록 내의 쓰레드들은 서로 영향을 주지만 다른 블록 내부의 쓰레드들에게는 영향을 주지 않는다.

GPU로 데이터를 전달하는 방법은 대량의 데이터를 메인 메모리에서 그래픽 카드의 메모리로 데이터를 복사한 후 그 포인터를 전달한다. 그 후 GPU에서 처리를 한 후 실행 결과를 다시 그래픽 카드의 메모리에 넣은 후, 메인 메모리에서 다시 포인터로 읽어온다.

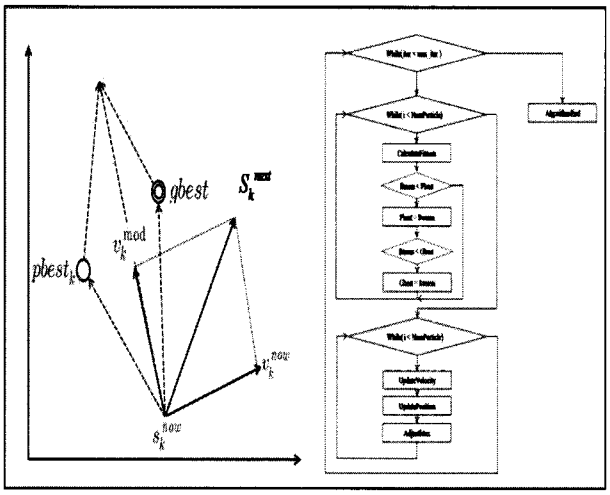
프로그램 빌드 과정은 CUDA로 작성된 코드(*.CU)는 CUDA 컴파일러를 통해 기존 C 컴파일러에 전달된다. cudaMalloc(), cudaMemcpy(), cudaFree() 함수를 통해 GPU 메모리를 할당하고 호스트의 데이터를 복사한다. 그리고 할당된 메모리를 해제한다.

그림 2는 호스트 PC와 GPU 사이의 프로그램 연산 과정을 나타낸 것이다. 호스트 PC의 DRAM에 저장된 데이터를 그래픽 카드의 global 메모리를 통해 복사한다. 그리고 global 메모리와 local 메모리를 사용해서 GPU의 register와 shared 메모리를 통해 연산을 하게 된다. 그 후 다시 역과정을 통해 호스트 PC의 DRAM으로 데이터를 복사한다.

2.2 PSO(Particle Swarm Optimization) 알고리듬

본 절에서는 PSO 알고리듬에 대해 간단히 설명하고 CUDA를 사용한 PSO 알고리듬과 기존의 PSO 알고리듬 사이에 정밀도 및 수행속도를 비교한다.

2.2.1 PSO 알고리듬



<그림 3> PSO 연산과정과 순서도

입자 군집 최적화(PSO) 알고리듬은 1995년 Kennedy와 Eberhart에 의해 제안된 알고리듬이다. 이 알고리듬은 조류나 어류 등의 생물의 무리가 각각의 개체가 가지고 있는 정보와 무리 전체가 가지고 있는 정보를 공유하면서 먹이를 찾아가는 과정을 모의한 방법이다.

각각의 에이전트들은 자신들이 발견한 최적의 목적함수 F(pbest)와 그 해의 위치 벡터인 pbest를 기억하고 있다. 그리고 집단에서 발견한 해중에 최적의 목적함수 F(gbest)와 그 해의 위치 벡터 gbest의 정보를 공유하고 있다. 각 에이전트는 현재의 위치 벡터와 속도벡터, 그리고 pbest와 gbest의 정보를 이용해서 식 1에 의해 이동을 하게 된다. 그리고 각 에이전트의 위치 벡터의 수정은 현재의 위치와 수정된 속도를 이용해서 식 2와 같이 행해진다[4]. 그림 3은 PSO 알고리듬의 연산 과정과 그 순서도를 나타낸 것이다.

$$V_k^{next} = a_1 V_k^{now} + a_2 rand(pbest_k - S_k^{now}) + a_3 rand(gbest_k - S_k^{now}) \quad (k=1,2,...,N) \quad (1)$$

$$S_k^{next} = S_k^{now} + V_k^{next} \quad (k=1,2,...,N) \quad (2)$$

여기서, V_k^{now}, V_k^{next} : 현재와 다음의 속도 벡터
S_k^{now}, S_k^{next} : 현재와 다음의 위치 벡터
$pbest_k, gbest_k$: 각 에이전트의 최량해의 위치벡터와 전체 에이전트의 최량해의 위치벡터
a_1, a_2, a_3 : 가중치 계수

PSO 알고리듬은 다음의 연산 과정을 거친다[5].

1) 전체 에이전트에 대해 초기 위치 벡터 s와 속도 벡터 v를 난수를 이용해 설정한다. 이 때 초기의 위치벡터 x를 이제까지의 최량해의 위치 pbest로 한다.

2) 1)에서 가장 우수한 pbest를 전체 에이전트에 관한 최량의 벡터 gbest로 한다.

3) 에이전트의 속도 벡터 v와 위치 벡터 s를 갱신한다.

4) 에이전트에 관해 현재의 위치에서 목적함수값 F(s)가 F(pbest)보다 우수한 목적함수 값을 가지면 s를 pbest로 할당한다.

5) 전체 에이전트에 대해서 F(pbest)가 F(gbest)보다 우수한 목적함수 값을 가지면 pbest를 gbest로 할당한다.

6) 충분히 좋은 적합 도를 가진 해를 얻거나 충분히 많은 세대를 거치게 될 때까지 3단계부터 루프를 반복 수행한다.

2.2.2 기존 PSO 알고리듬과 CUDA를 이용한 PSO 알고리듬 비교

PSO 알고리듬은 Microsoft Visual Studio 2005를 사용하여 구현하였다. 식 1에서 a_1 가중치 계수는 0.8, a_2, a_3 가중치는 계수는 동등 가중치인 1로 설정하였다. 그리고 에이전트들은 30으로 설정하였다. 실험에 사용된 Test Function은 Six-hump camel-back function(CA), Branin function(BR), Goldstein-Price(GP), Rastrigin function(RA)이다[6].

Test Function의 기존에 알려진 최적 값과 그 때의 포인트를 알고 있기 때문에 최적 값을 찾을 때까지의 연산 과정을 시뮬레이션 하였다. 알려진 최적값과 구현된 PSO 알고리듬 최적값 사이의 error를 구해 error 수치가 10^{-6}이하가 되면 루프가 종료 되게 구현하였다. 구현된 두 개의 PSO 알고리듬과 비교하여 정밀도 및 수행속도를 측정하였다

표 1은 총 100번의 시뮬레이션 결과 값의 평균을 나타낸 것이다. 먼저 Test Function들의 기존에 알려진 최적 값과 그 때의 전역해를 나타내었다. 그리고 비용함수 계산횟수는 error 수치가 10^{-6}이하가 돼서 루프가 종료될 때 까지 몇 번의 루프가 수행 되었는지 나타낸 수치이다. 또한 그 때의 전역해와 루프가 종료될 때까지의 수행 시간을 나타내었다.

구현된 두 개의 PSO 알고리듬 모두 루프를 종료 할 때 나온 최적 값과 그 때의 전역해가 기존에 알려진 최적 값과 전역해와 거의 동일한 정밀도를 보여주었다. 그리고 비용함수 계산횟수 또한 거의 동일한 수치를 보여주었다.

하지만 수행시간에서 두 PSO 알고리듬 사이에 차이를 보였다. 기존 PSO 알고리듬보다 CUDA를 이용한 PSO 알고리듬이 약 30~40% 정도 더 빠른 수행속도를 보였다.

그리고 루프를 지정된 반복횟수를 만족할 때 까지 연산을 수행하도록 하는 시뮬레이션에서도 CUDA를 이용하여 구현했을 때 30%이상 수행속도가 빨라졌다. 또한 에이전트의 수를 늘리는 등 데이터의 양을 늘릴수록 수행속도의 차이가 커졌다.

<표 1> 최적 값을 찾을 때까지의 연산 시뮬레이션

Test Function		CA	BR	GP	RA
	알려진 최적값	-1.031628	0.397888	3.000000	-2.000000
	전역해	0.089880	3.142000	0.000000	0.000000
		-0.712600	2.275000	-1.000000	0.000000
기존 PSO 알고리듬	전역해	0.089918	3.141462	0.000000	0.000000
		-0.712531	2.274439	-1.000000	0.000000
	비용함수 계산횟수	39	45	182	125
	수행속도 (sec)	0.001612	0.002389	0.006458	0.006276
CUDA를 이용한 PSO 알고리듬	전역해	0.089946	3.141888	0.000000	0.000000
		-0.712885	2.274419	-1.000000	0.000000
	비용함수 계산횟수	35	42	178	118
	수행속도 (sec)	0.000750	0.001161	0.004128	0.004266

3. 결 론

CUDA를 사용하면 GPU에 대한 자세한 내용을 알지 못해도 손쉽게 C프로그래밍 할 수 있다. 그리고 기존의 알고리듬이나 어플리케이션과 동일한 정밀도를 보여주면서 수행속도를 크게 줄일 수 있다. 또한 알고리듬의 연산 과정이 복잡하거나 처리할 데이터의 양이 클수록 CUDA의 효율성은 더욱 증가하게 된다. C로 구현한 프로그램이나 Matlab으로 구현된 프로그램 등 다양한 분야에서 CUDA를 이용하여 프로그래밍 할 경우 시뮬레이션이나 어플리케이션의 수행속도를 비약적으로 줄일 수 있다.

향후 현재 구현된 PSO 알고리듬의 수행속도를 더 개선할 예정이고, 본 연구실에서 연구 중인 휴머노이드 로봇 시뮬레이션에 사용된 GA(Genetic Algorithm)와 DEAS(Dynamic Encoding Algorithm for Searches) 알고리듬에도 CUDA를 이용하여 구현해 수행속도를 높여 어플리케이션에 적용할 예정이다.

[참 고 문 헌]

[1] 염용진 외, "GPU용 연산 라이브러리 CUDA를 이용한 블록암호 고속 구현", 정보보호학회논문지 ,제18권 제3호, pp. 23 ~ 32, 2008. 6

[2] 장홍훈 외, "CUDA와 OpenMP를 이용한 신경망 구현", 한국정보과학회 2008 종합학술대회 논문집, 제35권 제1호, pp. 289 ~ 290, 2008. 6

[3] NVDIA CUDA Programming Guide V2.0, 8 Jun. 2008 <http://kr.nvidia.com/object/cuda_develop_kr.html>

[4] 유명련, "Particle Swarm Optimization 탐색과정의 가시화를 위한 툴 설계", 멀티미디어학회논문지 , Vol.6 No.2, pp. 332 ~ 339. 2003

[5] 이중상 외, "Particle Swarm Optimization을 이용한 블랙 숄츠 옵션가격 결정모형", 한국경영과학회 2005년 춘계학술대회논문집, pp. 745 ~ 747, 2005. 5

[6] Jong-Wook KIM and Sang Woo KIM, "A Fast Computational Optimization Method: Univariate Dynamic Encoding Algorithm for Searches (uDEAS)", IEICE Transactions on Fundamentals of Electronics, Vol. E90-A, pp. 1679 - 1689, August 2007