GPU와 지역성을 이용한 행렬 곱셈 가속

권오영, 이창묵

한국기술교육대학교

Matrix Multiplication Acceleration with GPU and Locality

Oh-Young Kwon, Chang-Mug Lee

Korea University of Technology and Education

E-mail : oykwon@kut.ac.kr, atlantis13@kut.ac.kr

요 약

행렬 곱셈은 과학 및 공학분야에 다양하게 응용되고 있다. 행렬 곱셈의 경우 지역성을 활용하면 수행 성능을 크게 개선할 수 있다. GPU가 장착된 PC에서 CPU의 컴퓨팅 능력과 GPU의 컴퓨팅 능력을 같이 활용하여 행렬 곱셈을 가속하는 방법을 제시하였다. 제안된 방법이 GPU만을 사용하는 것보다 약 15%~30%의 성능을 향상시켰다.

ABSTRACT

Matrix multiplication is widely used in scientific and engineering field. Locality can improve the execution performance of matrix multiplication. A method for accelerating matrix multiplication is presented. This method uses both CPU and GPU computing power in PC. The presented method improved execution time about %15~30% than the method which uses only GPU.

키워드

지역성(Locality), 행렬곱셈(Matrix Multiplcation), GPU, 병렬프로그래밍(Parallel Programming)

Ⅰ. 서 론

행렬 곱셈의 성능을 향상하기 위한 다양한 방법들이 연구되었다. 단일 CPU의 경우는 지역성을 증진시켜서 수행성능을 개선하였고[1], 병렬 시스템의 경우는 병렬 시스템에 적합한 병렬 행렬 곱셈 알고리즘을 개발하였다[2]. 그래픽 처리를 위한 GPU가 병렬처리를 지원하고, GPU 연산처리 능력이 CPU 보다 우월해지면서 최근에 GPU를 활용한 고성능 컴퓨팅이 활발히 연구되고 있다[3]. 최근에 출시되는 대부분의 PC는 GPU를 탑재한 그래픽 카드를 가지고 있다. GPU를 위한 프로그래밍 환경도 제공되고 있으며, 이 프로그래밍 환경을 이용하면 GPU를 이용한 병렬 프로그래밍을 쉽게 할 수 있다.

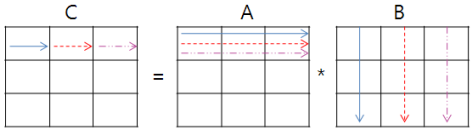
본 논문은 GPU활용이 가능한 PC에서 CPU와 GPU를 동시에 활용하여 행렬 곱셈을 가속화 하는 방안을 제시하였다. 제안된 방법은 GPU나 CPU를 단독으로 사용한 방법보다 행렬 곱셈의 수행시간을 단축하였다.

Ⅱ. CPGPU: 행렬 곱셈 가속 방법

일반적인 행렬 곱셈은 그림 1과 같다.

```
L1 : for i=1 to n do
L2 :   for j=1 to n do
L3 :     for k=1 to n do
           C[i,j]=C[i,j]+A[i,k]*B[k,j];
```

(a) 행렬곱셈코드



(b) 메모리 참조 패턴

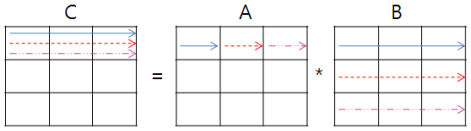
그림 1. 전형적인 행렬 곱셈 방법

그림 1의 메모리 참조 패턴을 보면 행렬 B는 컬럼위주로 참조가 된다. 이러한 경우 B[k,j]에 존재하는 공간 지역성(spatial locality)을 활용할 수

없다. 공간 지역성을 높이는 방법을 적용하여 행렬을 재구성하면 그림 2와 같은 행렬 곱셈 코드와 메모리 참조 패턴을 얻을 수 있다[1].

```
L1 : for i=1 to n do
L3 :   for k=1 to n do
L2 :     for j=1 to n do
           C[i,j]=C[i,j]+A[i,k]*B[k,j];
```

(a) 재구성된 행렬곱셈코드



(b) 메모리 참조 패턴

그림 2. 공간 지역성이 증진된 행렬 곱셈 방법

모든 행렬은 NxN으로 구성되었고, N = g + c라 하자. 그러면 행렬 곱셈 $C=AB$는 $C_g = A_gB$와 $C_c = A_cB$로 분할하여 계산할 수 있다. 여기서 C_g는 GPU에서 C_c는 CPU에서 동시에 계산을 하면 행렬 곱셈을 가속할 수 있다. 즉, 행렬을 분할해서 GPU로 보내는 적절한 g값을 결정하면 CPU와 GPU를 동시에 활용하여 행렬 곱셈을 수행할 수 있다. 컴퓨팅 수행 성능을 알 수 있으면, 연산에 소요되는 수행시간은 연산수를 성능으로 나누어 얻을 수 있다. GPU의 성능을 P_g, CPU의 성능을 P_c라 하자. GPU의 수행시간은 $T_g = gN^2/P_g$, CPU의 수행시간은 $T_c = cN^2/P_c$로 추정할 수 있다. CPU는 메모리 전송을 담당해야하기 때문에 메모리 전송에 소요되는 시간도 고려하여야 한다. 메모리 전송시간은 $T_m = (2gN + N^2)/r$이고 여기서 r은 메모리 전송율이다. 계산의 편의를 위하여 모든 행렬을 전송한다고 가정하여 전송시간을 $T_m = 3N^2/r$로 재 정의한다. 메모리 전송은 GPU에서 연산을 수행하기 위한 부담이므로 GPU의 수행시간 감소를 의미한다. 결국 $T_g - T_m \cong T_c$ 를 충족하는 g를 결정하여 계산을 CPU와 GPU에서 동시에 수행하면 행렬 곱셈을 가속화할 수 있다. g에 대하여 정리하면 다음과 같다.

$$g \cong \frac{P_g}{P_c + P_g}N + \frac{P_cP_gT_m}{N^2(P_c + P_g)}$$

위 식에서 N이 충분히 크다고 가정하면 위의 식은 $g \approx \frac{P_g}{P_c + P_g}N$로 간략화 할 수 있다. g를 구하기 위한 성능은 직접 측정하거나 매뉴얼들을 통해서 확인할 수 있다.

III. 실험결과

실험에 사용된 컴퓨터는 Intel Core2Duo P9700 CPU와 Nvidia Geforce P9300 GPU를 탑재한 노트북을 사용하였다[4]. 실험에 사용된 코드는 Nvidia가 제공하는 행렬 곱셈 샘플 코드에 활용하였다[3]. 샘플코드를 활용하기 위하여 N은 256, 512, 1024, 2048로 설정하였다. 매뉴얼에 의하면 CPU와 GPU의 성능은 각각 18 Gflops와 34 Gflops였다. 그러므로 g = 0.65N이 되었다. 샘플코드가 N을 16의 배수로 설정하여 0.65N를 바로 적용할 경우 행렬을 적절히 분할할 수 없다. 그리하여, 16의 배수를 쉽게 나눌 수 있고, 메모리 전송 부담도 반영할 수 있도록 g를 0.75N(= 12N/16)로 설정하였다. 0.65에 11/16이 더 근접하나 GPU의 블록 구조에 적합한 분할은 11:5보다 12:4가 더 효율적이어서 0.75를 선정하였다. 표 1은 각 행렬 곱셈의 수행 시간을 보여주고 있다. 여기서 CPU1은 그림 1의 코드이고, CPU2는 그림 2의 코드이다. GPU는 GPU만을 활용한 방법이고, CPGPU는 본 논문에서 제시한 방법이다.

표 1. 행렬 곱셈 수행 시간(ms)

	CPU1	CPU2	GPU	CPGPU
256	31.19	19.52	6.92	5.37
512	244.12	153.19	51.66	39.44
1024	4029.8	1212.85	397.55	302.75
2048	110581.68	10993.68	3112.47	2717.97

제안한 행렬 곱셈 가속 방법이 GPU단독으로 계산하는 것 보다 효과적임을 보이고 있다.

IV. 결 론

행렬 곱셈을 GPU와 CPU로 분할하여 계산하는 행렬 곱셈 가속 방법을 제시하였다. 각 프로세서의 성능만으로 데이터 분할양을 결정하고, 이를 기준으로 CPU와 GPU가 연산할 부분을 구분하여 동시에 연산을 수행함으로써 행렬 곱셈을 가속할 수 있었다.

참고문헌

[1] Oh-Young Kwon, Gi-Ho Park, Tack-Don Han, Shin-Dug Kim, and Sung-Bong Yang, "CtrlTile: A Loop Tiling to Reduce Loop Control Overhead," 15th IASTED Conference for Applied Infomatica '97, Innsbruk, Austria, Feb. 1997, pp. 13-16.

[2] G. C. Fox, S. W. Otto, and A. J. G. Hey. Matrix Algorithms on a Hypercube I: Matrix Multiplication. Parallel Computing, 4:17 31, 1987.

[3] http://www.nvidia.com

[4] http://www.sony.com