http://dx.doi.org/10.3745/KIPSTB.2012.19B.4.243

GPU 병렬성을 이용한 문서 유사도 계산 성능 개선

박 일 남[†] · 배 병 걸[†] · 임 은 진[††] · 강 승 식[†††]

요 약

정보검색 분야에서 벡터 모델, 문서 클러스터링 등은 입력 문서 개수가 증가할수록 유사도 계산 속도가 시스템의 성능에 많은 영향을 미치고 있다. 본 논문에서는 문서 유사도 계산 성능을 향상시키기 위하여 유사도를 계산하는 연산을 CPU 대신에 GPU를 이용하는 CUDA 프레임워크에서 병렬처리 기법으로 구현하는 방법을 제안하였다. 이 방법은 보편적인 방식인 CPU 환경에서 구현했을 때와 비교할 때 최대 15배까지 성능이 향상되었다. 또한, 기존의 CUDA 라이브러리인 CUBLAS와 Thrust를 사용한 방법보다도 각각 5.2배, 3.4배의 성능 개선 효과가 있음을 확인하였다.

키워드 : GPU 병렬성, 문서 유사도, 문서 클러스터링, CUDA 라이브러리

Improving the Performance of Document Similarity by using GPU Parallelism

Il-Nam Park[†] · Byunggurl Bae[†] · Eun-Jin Im[††] · Seung-Shik Kang[†††]

ABSTRACT

In the information retrieval systems like vector model implementation and document clustering, document similarity calculation takes a great part on the overall performance of the system. In this paper, GPU parallelism has been explored to enhance the processing speed of document similarity calculation in a CUDA framework. The proposed method increased the similarity calculation speed almost 15 times better compared to the typical CPU-based framework. It is 5.2 and 3.4 times better than the methods by using CUBLAS and Thrust, respectively.

Keywords : GPU Parallelism, Document Similarity, Document Clustering, CUDA Library

1. 서 론

정보 검색에서 검색된 문서를 랭킹하기 위하여 유사도 계산에 의해 질의(query)와 유사도가 높은 순으로 정렬하는 벡터 모델을 사용한다[1]. 유사도 계산은 벡터 모델뿐만 아니라 문서 분류 및 문서 클러스터링 등 다양한 정보 검색 응용 분야에서 임의의 2개 문서의 유사도를 비교하는데 매우 중요한 역할을 한다. 특히, 문서 클러스터링 시스템은 문서 개수 n에 대해 문서 유사도 계산을 위해 O(n2) 연산이 필요하기 때문에 유사도 계산 속도가 성능 향상에 큰 영향을 미치게 된다. 임의의 문서 di와 dj의 코사인 유사도(cosine similarity)는 두 개의 문서 벡터에 대한 곱셈과 덧셈 연산으로 유사도를 계산하며 문서 개수가 많아질수록 연산량이 기하급수적으로 증가한다.

$$\mathrm{sim}(d_i, d_j) = \frac{\vec{d_i} \cdot \vec{d_j}}{|\vec{d_i}| \times |\vec{d_j}|} = \frac{\sum_{k=1}^{t} w_{ki} \times w_{kj}}{\sqrt{\sum_{k=1}^{t} w_{ki}^2} \times \sqrt{\sum_{k=1}^{t} w_{kj}^2}}$$

NVIDIA는 프로그래밍이 가능한 그래픽 카드를 출시하여 CUDA(Compute Unified Device Architecture)라는 프레임워크를 공개하였는데 이로부터 개인용 슈퍼 컴퓨터 시대가 시작되었다[2,3]. 이를 기반으로 대량의 연산을 병렬처리 기법으로 구현 가능한 분야에서 CUDA를 이용하는 프로그래밍 기법이 개발되고 있다[4,5,6,7,8].

GPU의 ALU는 CPU의 복잡한 ALU에 비해 단순한 연산밖에 할 수 없으나 개수는 많기 때문에 덧셈, 곱셈과 같은 단순 연산이 반복되는 벡터 공간 모델의 유사도 계산에 적

※ 이 논문은 국민대학교 교내연구비를 지원 받아 수행된 연구임.
† 준 회 원 : 국민대학교 컴퓨터공학과 석사과정
†† 정 회 원 : 국민대학교 컴퓨터공학부 부교수
††† 정 회 원 : 국민대학교 컴퓨터공학부 교수
논문접수 : 2012년 2월 27일
수 정 일 : 1차 2012년 4월 6일
심사완료 : 2012년 4월 7일
* Corresponding Author : Seung-Shik Kang(sskang@kookmin.ac.kr)

합하다. GPU를 사용하기 위해 NVIDIA의 GPU 프로그램 개발 환경인 CUDA를 이용하여 병렬 연산을 수행하고자 한다. NVIDIA GPU는 다수의 연산 코어들이 SM(Streaming Multiprocessor)의 단위로 계층구조를 가지고 있으며, CUDA 프레임워크의 관점에서도 개별 연산을 담당하는 쓰레드들은 블록 단위로 계층구조를 가지고 있다[9,10,11].

본 논문은 유사도 연산 속도를 개선하기 위해 CPU (Central Processing Unit) 대신에 GPU(Graphic Processing Unit)를 사용하여 문서 유사도 계산을 병렬로 연산하는 방법을 제안하고자 한다. GPU 프로그래밍 도구는 NVIDIA의 CUDA 프로그래밍 환경을 사용하였다.

2. CUDA를 이용한 유사도 계산 프로그래밍

CUDA 프로그래밍에서 하나의 그리드는 실제 GPU 디바이스에서 수행될 커널을 의미한다. 하나의 그리드 안에는 여러 개의 쓰레드 블록이 존재하며 각 쓰레드 블록 안에는 쓰레드들이 존재한다. 하나의 쓰레드 블록 안에 존재할 수 있는 쓰레드 개수는 정해져 있으며 최대 512개의 쓰레드로 구성할 수 있다. 블록 안에 존재하는 쓰레드들 간에는 동기화가 보장되나 서로 다른 블록끼리의 쓰레드들은 동기화가 보장되지 못하므로 덧셈과 같은 이전의 결과에 영향을 미치는 연산에 대하여 병렬 프로그래밍으로 구현할 경우 그리드 안의 블록과 블록 안의 쓰레드들끼리의 관계를 생각하여 설계할 필요가 있다[9].

본 논문에서는 벡터 공간 모델의 유사도 계산을 NVIDIA GPU의 개발 환경인 CUDA를 활용하여 프로그래밍하여 입력 벡터에 대하여 하나의 블록당 최대 512개의 연산을 수행하도록 설계 및 구현하고자 한다. CUDA 프로그래밍에 의하여 작성된 쓰레드 블록들은 멀티프로세서로 수행하게 되며, 쓰레드들은 쓰레드 프로세서로 GPU 커널에서 작동하게 된다.

입력 데이터에 대하여 문서 개수를 나타내는 행을 n, 단어 개수를 나타내는 열을 m이라고 하자. n×m 크기의 문서 벡터가 입력으로 들어왔을 때, n번째 문서를 d, j는 1~n-1을 나타내며, 문서 d와 d_j간의 유사도 sim(d_j, d) 연산에 대하여 크기 m에 대해 병렬처리를 하여 시간을 줄이는 방법으로 CUDA 프로그래밍을 설계 및 구현하였다.

Fig. 1은 병렬처리를 위한 1차원 그리드와 블록에 대한 쓰레드별 아이디 주소를 나타낸 CUDA 주소 체계이다. 만약 m=1,000이라면 블록 개수는 ⌈1000/512⌉ = 2 가 된다. 2개의 블록에서 각 블록은 동시 수행 가능한 최대 쓰레드 개수인 512개 쓰레드로 구성된다. 그림 1과 같이 각 블록 안의 쓰레드들은 고유의 주소를 가지며, 1차원적인 주소는 blockIdx.x * blockDim.x + threadIdx 로 구할 수 있다. blockIdx.x는 블록의 주소, blockDim.x는 블록 안의 쓰레드 개수, threadIdx는 블록 안에서의 쓰레드 주소를 나타낸다.

이 쓰레드 아이디 주소를 가지고 m개의 데이터가 한번에 병렬로 곱셈을 수행할 수 있도록 해야 한다. 하나의 쓰레드는 0에서 (m-1)값 중 각자 하나씩 쓰레드 아이디 주소를 가지고 있으며 이 쓰레드 주소를 이용하여 vector[n][m-1]의 데이터 주소를 계산하여 병렬 곱셈 연산을 하도록 CUDA 프로그래밍을 하였다.

문서 d_i와 d_j간의 코사인 유사도 계산은 벡터의 내적과 벡터 크기를 구하는 제곱 연산으로 이루어진다. Fig. 1에서 제시한 idx(쓰레드 아이디 주소)를 사용하여 1차원 idx 표현을 가지고 2차원적인 입력 벡터에 대한 내적은 Fig. 2와 같이 기술된다. j는 문서 d_i와 현재 계산될 문서 d_j에 대한 행을 나타내므로 j×m은 현재 문서의 첫 번째 시작 주소를 나타낸다. 이 주소에 idx(0~m-1)를 더하여 각 m개의 데이터에 대하여 한번에 내적 연산을 수행한다.

벡터의 크기 연산도 벡터 내적과 마찬가지로 동일 쓰레드 주소 체계로 제곱 연산을 수행한다. 벡터의 내적과 벡터 크기 연산이 끝난 뒤 __syncthreads() 함수를 호출하였는데, 이는 벡터 내적 연산이 종료된 후에 벡터 크기 연산을 시작하기 위한 것이다. 마찬가지로 벡터 크기의 연산이 끝난 뒤 덧셈 연산을 하기 위하여 __syncthreads() 함수를 호출하였다. 이 함수는 블록 안의 쓰레드들끼리 동기화를 시켜준다.

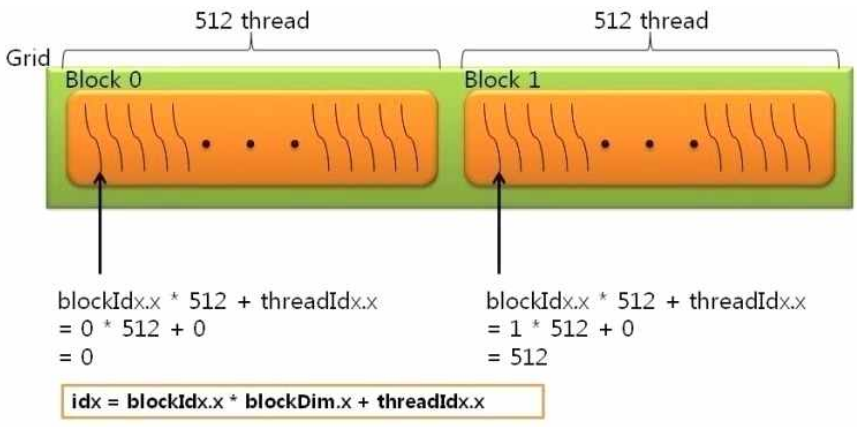

그림 1. 병렬처리를 위한 그리드와 블록 레이아웃
Fig. 1. Grid block layout for parallel processing

```
for (int j=0; j<n-1; j++) {  // 벡터 내적
  if (idx < m)
  d_arr_ip[j*m+idx] = d_arr[j*m+idx] *
  d_arr[(n-1)*m+idx];
}
__syncthreads();

for (int j=0; j<n; j++) {  // 벡터 크기
   if (idx < m)
   d_arr[j*m+idx] *= d_arr[j*m+idx];
}
__syncthreads();
```

그림 2. GPU를 이용한 벡터 유사도의 곱셈 연산
Fig. 2. Multiplication operation for vector similarity in GPU

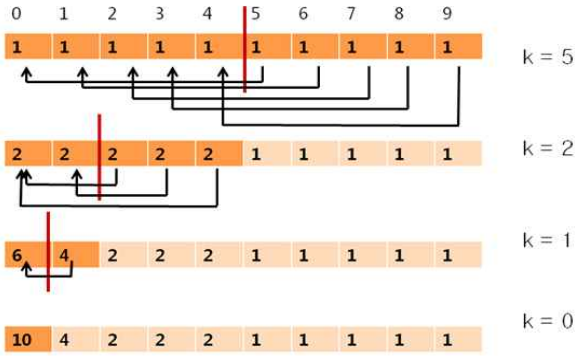

그림 3. 덧셈 병렬 처리 방법
Fig. 3. Add operation in GPU programming

덧셈 연산은 Fig. 3과 같이 $\log_2 k$개의 연산으로 설계하여 단어 개수를 2로 나눈 값을 초기값으로 한다. 단어 개수가 10이라면 k는 5이다. 0~4까지 주소에 각각 k를 더한 만큼의 주소값들끼리 더하여 0~4에 값을 저장을 한다. 계속해서 다시 k를 2로 나눈 값을 가지고 위 작업을 반복하여 k값이 1이 될 때까지 수행한다. 최종적으로 배열 0번지에 크기 10인 배열의 덧셈 값을 얻을 수 있다.

여기서 한가지 주의할 점은 k가 2일 때 배열주소4의 값이 덧셈을 못하는 에러를 범하게 된다는 것이다. (k=2, k보다 작은 주소의 덧셈은 각각 0+2 번지와 1+3번지가 수행된다.) 이 에러는 이전의 k값이 홀수일 때(k=5) 덧셈 결과에서 k-1 주소값을 미리 더해줌으로 해결할 수 있다. 이러한 덧셈 알고리즘을 CUDA에 적용하기 위하여 Fig. 4와 같이 설계하였다.

CUDA는 블록당 최대 512개의 쓰레드를 처리할 수 있으므로 초기의 k값은 512로 한다. 여기서 블록 단위로 처리하는 이유는 __syncthreads() 함수를 사용하여 동기화할 수 있는 범위가 동일 블록 안에 있는 쓰레드로 제한되기 때문이다. 블록당 512개의 데이터에 대하여 병렬 처리에 의해 $\log_2 256 = 8$회에 512개 덧셈 연산을 수행할 수 있다. 블록당

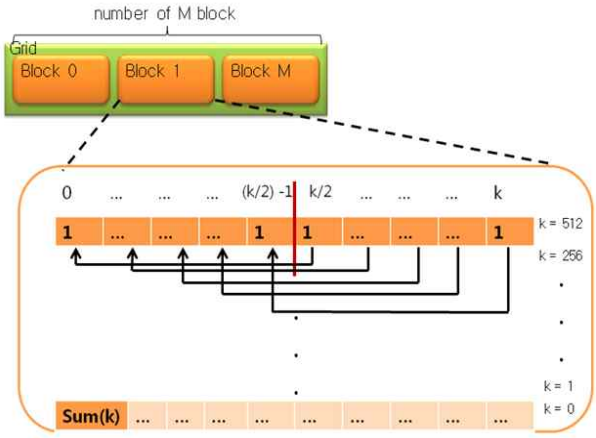

그림 4. 블록당 512개 쓰레드를 사용한 병렬 덧셈
Fig. 4. Add operation by using 512 threads/block

512개를 처리하므로 입력 벡터의 m이 1,000이면 2개의 블록이 생성되고, 여러 블록이라 하더라도 병렬로 처리되므로 1,000개의 데이터에 대하여 8회만에 덧셈이 수행된다.

블록 내에서 덧셈 연산 수행이 끝나면 각 블록 안에서 0번째 주소에는 512개의 덧셈 결과가 구해질 것이다. 최종적으로 2개의 블록, 블록0의 0번째와 블록1의 0번째를 더하면 1,000개의 데이터에 대한 덧셈 결과를 얻을 수 있다. 블록 2개에 대하여 덧셈은 추가적으로 1번만 하면 되므로, 덧셈 연산 수행은 각 블록에 대하여 8번, ('생성된 블록 개수' -

```
for(int j=0; j<n; j++) //덧셈 연산
{ // 각 블럭당 log 덧셈 계산
  int k = 256;
  for(; j>1;) {
    if(idx % B_SIZE < k && idx+k < m) {
      d_arr_ip[(j*m)+idx] += d_arr_ip[(j*m)+idx+k];
      d_arr[(j*m)+idx] += d_arr[(j*m)+idx+k];
    }
    if(k%2!=0 && (idx % B_SIZE == 0)) { //홀수일때
      d_arr_ip[(j*m)+idx] += d_arr_ip[(j*m)+idx + k-1];
      d_arr[(j*m)+idx] += d_arr[(j*m)+idx + k-1];
    }
    k /= 2;
    __syncthreads();
  }
  if(idx % B_SIZE == 0) {
    d_arr_ip[(j*m)+idx] += d_arr_ip[(j*m)+idx+1];
    d_arr[(j*m)+idx] += d_arr[(j*m)+idx+1];
  }
  __syncthreads();
}
```

그림 5. GPU를 이용한 벡터 유사도의 덧셈 연산
Fig. 5. Add operation for vector similarity in GPU

1)번 만큼이 길이 m의 덧셈 연산을 하는데 필요한 횟수이다. 이를 일반화하면 m개의 데이터에 대하여 덧셈 연산을 처리하는 시간은 ⌈m/512⌉ + 8 이다.

Fig. 5는 (⌈m/512⌉ + 8) 만큼의 덧셈 연산을 수행하도록 각 블록 덧셈을 CUDA로 작성한 것이며, 이 연산이 끝난 후 CPU에서 각 블록 0번 주소끼리 더하는 연산을 하도록 하였다. d_arr_ip는 벡터 내적에 대한 결과 벡터이고, d_arr은 문서 d와 d_j벡터 크기이다.

Fig. 2와 Fig. 5의 GPU 연산에 의해 벡터 유사도 연산에서 GPU를 이용한 병렬 처리 시간은 n×⌈m/512⌉+8으로 CPU로 수행하는 n×m 보다 향상되었음을 알 수 있다.

3. 실험 및 비교 평가

벡터 모델에 대하여 CPU로 연산할 때 수행 시간은 n×m이고, GPU 방법으로 연산하면 n×⌈m/512⌉+8 이다. 구현한 알고리즘에 대해 실험을 통해 CPU를 사용한 경우와 기존의 CUDA 라이브러리를 사용하여 구현한 방법과 비교하고자 한다.

3.1 실험 환경 및 실험 방법

표 1. 실험 환경
Table 1. Experiment environment

CPU	인텔 코어i3-530 : 듀얼코어, 2.93GHz
GPU	Device : Tesla C2050 Number of multiprocessors : 14 Number of cores : 448

표 2. 실험 데이터
Table 2. Input data specification

문서 개수 × 벡터 크기	10×1000, 20×2000, 30×3000, 40×4000, 50×5000, 60×6000, 70×7000, 80×8000, 90×9000, 100×10000, 200×20000, 300×30000 , 400×40000, 500×50000, 1000×100000

Table 2의 실험 데이터에 대해 아래 방법들에 대한 유사도 계산을 수행한다. 각 방법에 대한 연산 수행 시간을 측정하여 제안한 방법의 성능을 평가하였다.

(1) CPU: 기존 CPU 연산 시간을 측정하는 방법
(2) GPU_CS(Cublas Sdot): CUDA 프로그래밍 라이브러리 CUBLAS[10]를 사용하여 GPU 연산 시간 측정. CUBLAS는 벡터의 dot 연산 제공
(3) GPU_TR(Thrust Reduce): CUDA 프로그래밍 라이브러리 Thrust[11]를 사용하여 GPU 연산 시간 측정. 벡터 연산은 없으나 주어진 배열을 하나의 값으로 더해주는 Reduce 연산 제공
(4) GPU_PR(Proposed Reduce): 본 논문에서 제시하는 방법으로 GPU 연산 시간을 측정하고자 그림 2의 곱셈 연산과 그림 5의 덧셈 연산을 이용하여 그림 6과 같이 구현

```
Kernel_function_call_Figure2_and_5()
//Fig. 2, 5의 GPU 곱셈, 덧셈 커널 호출
for(j=0; j<n-1; j++) {
for(b=0; b<생성된블럭개수; b++) {
   _and_d_inner += h_arr_ip[(j*m) + (b*512)];
   _sum_of_squares += h_arr[(j*m) + (b*512)];
   d_ sum_of_squares += h_arr[((j-1)*m) + (b*512)];
}
 sim[j] = _and_d_inner / (sqrt(_sum_of_squares) *
sqrt(d_ sum_of_squares));
}
```

그림 6. GPU+PR를 이용한 벡터 유사도 계산
Fig. 6. Vector similarity by GPU+PR

3.2 실험 결과

Table 3은 입력 문서에 대하여 CPU, GPU_CS, GPU_TR, GPU_PR 방법으로 유사도 계산에 대한 연산 시간을 측정한 결과이다. GPU_CS와 GPU_TR은 병렬 처리 데이터의 크기가 일정 규모 이상인 경우에만 CPU를 이용하는 방법보다 처리 효율이 좋아진다. 이에 비해, GPU_PR 방법은 모든 입력 데이터에 대하여 CPU 방법보다 효율이 향상되었다.

표 3. 유사도 계산 성능 측정 결과
Table 3. Performance evaluation

Document Vector size	CPU (msec)	GPU_CS (msec)	GPU_TR (msec)	GPU_PR (msec)
101000	0.176	5.725	2.160	0.173
505000	4.309	29.231	16.557	0.844
10010000	17.606	59.769	29.573	1.802
50050000	442.171	303.000	174.137	32.910
1000100000	1766.511	622.167	401.992	118.664

Fig. 7은 입력 벡터 크기에 따른 벡터 공간 모델 연산 시간을 나눈 것으로 각 방법 별 1msec 당 연산 처리율을 나타내는 그래프이다. 문서 벡터 크기가 증가할수록 CPU 방법은 항상 동일한 처리율을 유지하고 있는 반면, GPU_CS, GPU_TR, GPU_PR 방법들은 1msec 당 처리율이 증가하고 있다. 기존의 라이브러리를 사용한 GPU_CS와 GPU_TR은 연산량이 일정 규모 이상인 경우에만 CPU보다 효율이 좋아지지만 GPU_PR은 처음부터 CPU 연산 처리율과 비교하였을 때 효율이 좋아졌다.

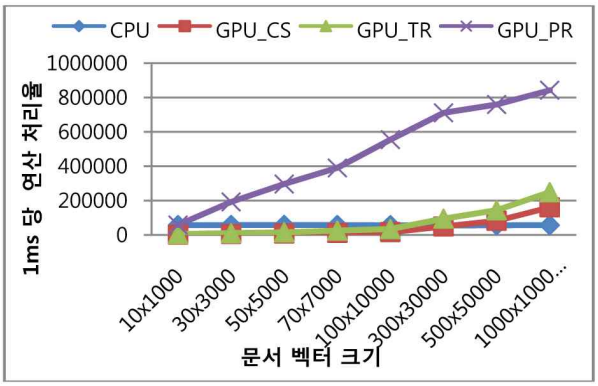

그림 7. 벡터 크기에 따른 1ms당 연산 처리율
Fig. 7. Processing speed change by vector size

실험 상의 최대 입력 벡터 1000100000에 대하여 CPU 처리율을 기준으로 하였을 때, GPU_CS는 2.8배, GPU_TR은 4.3배, GPU_PR은 15배로 CUDA를 이용하는 방법이 기존의 라이브러리를 사용한 경우보다 전반적으로 효율 개선 효과가 높게 나타났다.

4. 결 론

문서 유사도의 계산 속도를 향상시키기 위하여 NVIDIA사의 GPU 프로그램 개발 환경인 CUDA를 이용하는 방법을 제안하였다. 실험 결과로, 통상적인 CPU 환경의 연산 처리 방식보다 CUDA 개발 환경에서 구현했을 때 최대 15배의 성능 개선 효과가 있었다. 또한, 기존의 라이브러리 CUBLAS와 Thrust를 사용한 방법과 비교했을 때에도 CUDA를 이용하는 방법이 최대 각각 5.2배, 3.4배 향상됨을 확인하였다. 또한, CUBLAS와 Thrust 라이브러리를 사용할 때는 입력 벡터 크기가 일정 수준으로 커야 CPU 연산보다 빠르게 처리하는 반면에 CUDA는 입력 크기와 무관하게 항상 CPU 연산보다 효율이 더 좋게 나타났다. 정보검색 분야는 대용량 데이터를 다루는 경우가 많으므로 CPU 연산 대신에 GPU를 이용하는 병렬처리 방법으로 전체 시스템의 성능을 향상시킬 수 있음을 알 수 있었으며, 상위 기종의 서버를 구입하는 대신에 GPU의 병렬성을 이용함으로써 저비용으로 최대의 효과를 얻을 수 있는 새로운 방법을 발견하였다.

참 고 문 헌

[1] G. Salton, A. Wong and C. S. Yang, “A vector space model for automatic indexing,” Communications of the ACM, Vol.18, No.11, pp.613-620, 1975.
[2] J. S. Lee and H. K. Ryu, “Personalized super computer current and future by GPU parallel computing technology”, Korean Electronics, Vol.36, No.5, pp.562-571, 2009.
[3] J. Nickolls, I. Buck, M. Garland, K. Skadron, “Scalable parallel programming with CUDA,” queue - GPU computing, Vol.6, No.2, pp.40-53, 2008.
[4] Jason Sanders, ‘CUDA by Example: An introduction to general-purpose GPU programming’, Addison-Wesley, 2010.
[5] D. Luebke, “CUDA: Scalable parallel programming for high-performance scientific computing”, ISBI, pp.836-838, 2008.
[6] M Garland et al., “Parallel computing experiences with CUDA,” IEEE Micro, Vol.28, No.4, pp.13-27, 2008.
[7] T. Park, J. Woo, and C. Kin, “CUDA-based parallel bi-conjugate gradient matrix solver for BioFET simulation”, Korean Electronics Journal, Vol.48, No.1, pp.80-100, 2011.
[8] M. J. Kim, “An image processing speed enhancement in a multi-frame super resolution algorithm by CUDA”, Korean Journal of Military Science Technique, Vol.14, No.4, pp.663-668, 2011.
[9] NVIDIA CUDA, “NVIDIA CUDA C Programming guide version3.2”, http://developer.nvidia.com.
[10] NVIDIA CUDA, “NVIDIA CUDA CUBLAS library, PG-05326-032_V02”, http://developer.nvidia.com.
[11] “Thrust library”, http://code.google.com/p/thrust/.

박 일 남
e-mail : pin0156@naver.com
2011년 국민대학교 컴퓨터공학부(학사)
2011년~현 재 국민대학교 컴퓨터공학과 석사과정
관심분야 : 자연어처리, 한국어 정보처리, 정보검색, 텍스트마이닝 등

배 병 걸
e-mail : bazel1984@naver.com
2009년 국민대학교 컴퓨터공학부(학사)
2011년~현 재 국민대학교 컴퓨터공학과 석사과정
관심분야 : 자연어처리, 한국어 정보처리, 정보검색, 문서 유사 비교 등

임 은 진
e-mail : ejim@kookmin.ac.kr
1991년 서울대학교 컴퓨터공학과(학사)
1993년 서울대학교 컴퓨터공학과(공학석사)
2000년 U.C.Berkeley 컴퓨터공학과 (공학박사)
2001년~현 재 국민대학교 컴퓨터공학부 부교수
관심분야 : 고성능컴퓨팅, 병렬처리, 텍스트마이닝 등

강 승 식

e-mail : sskang@kookmin.ac.kr

1986년 서울대학교 컴퓨터공학과(학사)

1988년 서울대학교 컴퓨터공학과(공학석사)

1993년 서울대학교 컴퓨터공학과(공학박사)

1994년~2001년 한성대학교 정보전산학부 부교수

2001년~현 재 국민대학교 컴퓨터공학부 교수

관심분야 : 자연어처리, 한국어 정보처리, 정보검색, 텍스트마이닝 등