한국항행학회 논문지 제15권 제2호 2011년 4월

GPU가 장착된 PC를 위한 혼합 정렬 알고리즘 설계

Designing Hybrid Sorting Algorithm for PC with GPU

권오영*

Oh-Young Kwon*

요 약

데이터 정렬은 현대 사회에 존재하는 수많은 디지털 데이터에 대한 중요한 가공 작업 중의 하나이지만, 데이터가 방대할수록 정렬 과정 자체도 많은 연산시간을 소비한다. 본 논문에서 데이터 배열을 분할하여 PC에 있는 CPU와 GPU에서 각각 동시에 정렬을 수행하는 혼합 정렬 알고리즘을 제안하였다. 각 장치의 처리 성능을 바탕으로 가장 효율적인 배열의 분할 범위를 결정하고 각각 분할된 영역을 CPU와 GPU에서 동시에 정렬함으로써 전체 정렬 시간을 단축시켰다. 실험결과에서 알 수 있듯 혼합 정렬이 GPU만 활용한 정렬보다 8%이상 정렬 수행 속도를 향상시켰다.

Abstract

Data sorting is one of important pre-process to utilize huge data in modern society, but sorting spends a lot of time by sorting itself. In this paper, we presented hybrid sorting algorithm that splits array to sort concurrently in CPU and GPU. To do this, we decided most effective range of array based on hardware performance, then accomplished reducing whole sorting time by concurrent sorting on CPU and GPU. As shown in results of experiment, hybrid sorting improved about eight percent of sorting time in comparison with the sorting time using only GPU.

Key words : Hybrid sorting, GPU, Sorting, many-core, multi-core

I. 서 론

현대 사회에는 수많은 디지털 데이터가 존재한다. 이 데이터들을 사용하기 위해 대부분 목적에 맞는 가공 작업이 선행되어야 하는데, 대표적으로 데이터의 정렬(sorting)을 들 수 있다. 데이터가 방대할수록 정렬 과정 자체도 많은 연산 시간을 소모한다. 정보화된 사회에서 데이터의 양은 항상 증가하는 추세에 있다. 예를 들어 지리적으로 구역을 분할하여 만든 구역별 지형 데이터의 정밀도를 향상시킨다면, 요구되는 데이터는 컴퓨터 연산 속도의 향상보다 폭발적으로 증가하게 된다. 따라서 방대한 데이터의 정렬 시간을 단축시키기 위한 다양한 연구들이 수행되고 있다[1]-[4]. 이러한 연구들은 GPU나 multi-core CPU를 대상으로 정렬 속도를 높이는 데 주력하고 있다. 최근 GPU는 단순 그래픽 처리를 넘어서 고속 병렬 연산 장치로써 주목을 받고 있다[5]-[7]. 대부분의 컴퓨터에 고속 연산이 가능한 그래픽 카드가 탑재되고 있으므로, CPU에서 수행되던 정렬 연산을 GPU와 분담하여 전체 정렬 시간을 단축할 수 있다. GPU만을 사

* 한국기술교육대학교 컴퓨터공학부(School of Computer Science & Engineering, Korea University of Technology and Education)
· 제1저자 (First Author) : 권오영
· 투고일자 : 2011년 3월 2일
· 심사(수정)일자 : 2011년 3월 3일 (수정일자 : 2011년 4월 15일)
· 게재일자 : 2011년 4월 30일

용할 경우 CPU는 GPU의 연산이 종료될 때까지 기다려야 하는 단점이 있다. 따라서 본 논문에서 CPU와 GPU가 동시에 정렬을 수행하는 혼합 정렬 알고리즘을 제안하였다.

혼합 정렬을 수행하기 위해 먼저 CPU와 GPU의 하드웨어 성능을 기준으로 데이터를 분할한다. 분할된 두 개의 데이터 배열에 대해 논리적인 정렬 종료 추정 시간을 산정한 후, CPU와 GPU에서 각각 수행되는 정렬 연산이 최대한 같은 시간 안에 종료될 수 있도록 분할된 배열의 범위를 재조정한다. 그런 후 분할된 배열을 각각 CPU와 GPU에서 동시에 정렬함으로써 전체 정렬에 소요되는 시간을 단축한다. 실험 결과에 따르면 혼합 정렬이 GPU만 활용한 정렬보다 8%이상 정렬 수행 속도를 향상시켰다.

본 논문의 구성은 다음과 같다. 2장에서는 CPU와 GPU를 이용한 혼합 정렬의 전체 과정을 설명하였다. 3장에서는 배열의 분할 방법을, 4장에서는 분할된 배열의 정렬 수행 방법을 상세히 설명하였다. 논문에서 제안한 방법에 대한 실험방법과 결과를 5장에 제시하였다.

Ⅱ. 혼합 정렬 과정

CPU와 GPU의 협업 연산을 통한 정렬은 인덱스 추출, 데이터 배열 분할, 정렬 수행, 정렬 결과 재구성의 단계를 거친다.

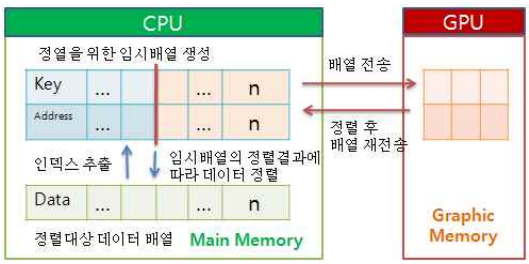

그림 1. CPU와 GPU간 데이터 흐름
Fig. 1. Data flow between CPU and GPU

정렬할 데이터가 입력되면 메모리 자원 소모를 최소화하기 위해 초기 데이터 배열에서 실제 정렬에 사용될 키와 각 데이터 원소의 주소를 추출한 임시 배열을 생성한다. 이후 CPU와 GPU의 연산 능력을 비교하여 비교 결과에 따라 정렬 대상이 되는 임시 배열을 CPU에서 처리할 영역과 GPU에서 처리할 영역으로 분할한다. 분할한 각 배열의 영역은 CPU와 GPU에서 담당하여 정렬을 수행한 후 결과를 다시 조합하여 최종 정렬 결과를 얻게 된다. 혼합 정렬 과정에서 CPU와 GPU 사이의 데이터 흐름은 그림 1에 나타나 있다.

초기에 주어진 정렬 대상 데이터 배열은 시스템의 메인 메모리에 위치한다. 정렬 대상이 되는 데이터 배열의 원소들은 정렬에 필요한 키 이외에 다른 데이터를 포함하고 있다. 이 데이터들을 전체 데이터 배열이 위치한 메인 메모리에서 GPU의 메모리로 전송한다면 메모리 공간 뿐 아니라 전송 시간에도 많은 비용이 소모된다. 각 원소들의 실제 정렬에 사용될 키와 원소 데이터의 메모리 주소를 각각 32비트 플로팅 포인트(32bit floating point) 자료형에 저장한 임시 배열을 생성한다. 배열을 32비트 플로팅 포인트 자료형으로 정의하는 이유는 키 값과 데이터의 다양한 자료형을 포괄할 수 있을 뿐만 아니라 GPU에서 가장 효율적으로 처리하는 데이터 형식이기 때문이다.

임시 배열을 만든 후에 CPU와 GPU의 연산능력 비교 결과에 따라 이 배열을 CPU영역과 GPU영역으로 분할하여 GPU영역은 GPU 메모리로 전송하여 정렬하고, CPU 영역은 CPU에서 처리한다. GPU의 정렬 연산이 끝나면 결과는 다시 시스템의 메인 메모리로 전송된다.

Ⅲ. 배열의 분할

추출된 키와 데이터 주소로 구성된 인덱스 배열의 분할은 CPU와 GPU의 연산 능력과 메모리 전송 시간을 고려하여 결정한다. 연산 능력은 flops (floating-point operations per second)를 사용하여 표시한다. CPU와 GPU에서 사용할 정렬 알고리즘의 복잡도를 이용하여 정렬에 사용되는 연산수를 추정한 후 각 장치의 flops로 나누어 수행 시간을 추정한다.

정렬 알고리즘의 복잡도는 흔히 빅 오(Big O)로 표현되는 복잡도를 사용하지만 추가적으로 고려해야 할 사항이 있다. 빅 오 복잡도는 단순히 정렬 대상의

원소 개수 N에 대해 정렬 알고리즘이 몇 번의 비교를 수행해야 하는지를 나타낸다. 하지만 정렬이 컴퓨터에서 수행될 때는 각종 분기문이나 메모리 접근 등의 이유로 비교 연산이외에 다수의 연산이 요구되므로, 빅 오로 산출한 복잡도에 실제 정렬 루틴의 연산수를 곱해주어야 한다. 즉, 정렬에 필요한 연산의 수는 수식 (1)에 의해 산출된다. (1)에서 C는 정렬 루틴의 연산수를 의미하며, $O(Sorting)$ 은 정렬에 사용된 알고리즘의 복잡도이다.

$$I_{Sorting} = C \cdot O(Sorting) \quad (1)$$

CPU와 GPU의 연산능력인 flops 수치로 (1)을 나누면 정렬 추정 시간 T를 구할 수 있다. (2)와 (3)은 각각 CPU와 GPU에서 정렬이 수행될 때 필요한 시간을 의미한다.

$$T_{CPU} = \frac{I_{Sorting}^{CPU}}{flops_{CPU}} \quad (2)$$

$$T_{GPU} = \frac{I_{Sorting}^{GPU}}{flops_{GPU}} \quad (3)$$

정렬 추정 시간이란 실제 시간을 의미하는 것이 아니라 CPU와 GPU의 상대적인 수행 시간 비율을 나타내기 위한 것이다. 가장 효율적인 정렬을 위해서는 CPU와 GPU의 정렬이 동시에 끝나야 한다. 따라서 수식 (4)에서 CPU의 정렬 분량(M_{CPU})은 GPU 추정 시간을 CPU 추정시간과 GPU 추정시간의 합으로 나누어 구한다. 즉, 정렬 추정 시간이 작을수록 그 비율만큼 더 많은 분량의 배열을 할당받게 된다. 배열의 개수는 항상 정수이므로 얻어진 분할 범위 결과는 반올림한다. 수식 (4), (5)에서 M_{Array}는 인덱스 배열의 크기를 의미한다.

$$M_{CPU} = M_{Array} \cdot \frac{T_{GPU}}{T_{CPU} + T_{GPU}} \quad (4)$$

$$M_{GPU} = M_{Array} - M_{CPU} \quad (5)$$

배열의 분할 범위가 결정되면, 다음으로 시스템의 메인 메모리에서 GPU의 메모리로 정렬할 데이터를 전송하는 시간을 고려해야 한다. 이때 전송될 배열의 크기를 PC의 그래픽 버스(bus) 속도로 나누어 전송 추정 시간($T_{Transmit}$)을 구한다. 그래픽 버스 속도는 시스템의 메인 메모리에서 그래픽 카드의 메모리로 데이터를 전송하는 버스의 속도이며 하드웨어에 따라 정해진 표준 속도 수치를 적용한다. 정렬을 위한 인덱스 배열은 메인 메모리에 위치하는데, 위의 분할 결과에 따라 GPU의 할당량을 GPU의 메모리로 전송한 후 정렬이 끝나면 다시 메인 메모리로 전송받게 되므로 정렬 전과 정렬 후 두 번 전송된다. 전송 시간은 GPU의 정렬을 위해 소모되는 시간이므로 수식 (3)의 GPU 정렬 소요시간에 더한다. 변경된 GPU 소모 시간 (7)을 수식 (4)에 다시 적용하고, (4)의 결과를 (5)에 다시 적용하여 CPU와 GPU에서 수행될 배열의 분할 범위를 재설정한다.

$$T_{Transmit} = \frac{2M_{GPU}}{BUS_{speed}} \quad (6)$$

$$T_{GPU} = T_{GPU} + T_{Transmit} \quad (7)$$

본 논문에서 정렬을 위하여 생성한 임시 배열의 크기는 2^n개로 구성된다고 가정하였다. 일반적으로 GPU에서 정렬에 사용하는 병렬 정렬 알고리즘은 2^n개의 배열에 최적의 효율을 보인다[3]. 따라서 최종적으로 얻어진 GPU 배열 크기가 2^n으로 구성되는지 검사한 후 만약 아니라면 2^n이 되도록 CPU 배열과 조절하여 최종 분할 범위를 결정한다. 두 개로 분할된 행렬의 한 부분은 GPU의 메모리로 전송되고 남은 행렬은 CPU에서 정렬을 수행한다.

Ⅳ. 혼합 정렬

4-1 정렬 수행

분할된 배열을 CPU와 GPU에 할당한 후 각 장치에 적합한 정렬 알고리즘으로 정렬을 수행한다. 본 논문은 CPU에서는 쾌속 정렬(Quick sort)을, GPU에서는 GPU의 구조에 따른 효율적인 병렬연산을 위해 바이토닉 정렬(Bitonic sort)을 사용한다. 본 논문은

GPU에 적합한 정렬 알고리즘을 개발하는 것이 아니기 때문에 GPU와 CPU의 컴퓨팅 능력을 최대한 사용할 수 있도록 배열을 분할하여 각각 최적의 알고리즘을 이용한다.

4-2 정렬 결과 재구성

GPU 정렬이 완료되면 정렬 결과는 임시 배열의 GPU가 담당하도록 분할된 영역으로 다시 저장된다. CPU의 정렬 결과는 추출 배열을 대상으로 논리적 분할 범위만 지정하여 내부 정렬이 수행되어 있다. CPU와 GPU의 결과는 분할된 범위로 따로 정렬되어 있으므로 마지막으로 조합하는 과정이 필요하다. 이 과정에는 머지소트(Merge sort)의 한 단계가 사용된다. 두 논리적 분할 정렬을 각 왼쪽부터 비교하여 하나의 행렬로 조합하며 정렬과정에서 최초의 정렬대상 행렬을 각 원소의 주소를 참조하여 치환한다. 정렬 결과를 최종적으로 재구성하는 과정은 그림 2와 같다.

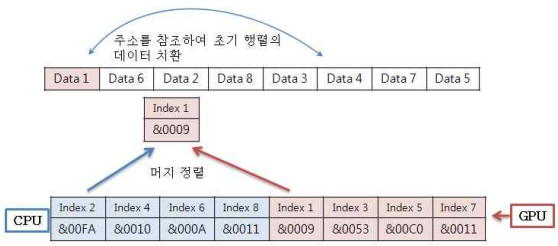

그림 2. 정렬 결과의 최종 재구성
Fig. 2. Final Restructuring of Sorting Result

V. 실 험

본 논문에서 제안한 혼합 정렬 방법을 실제 배열에 적용하여 정렬 수행 시간을 측정하였다. 실험에 사용된 배열은 1048576(2^{20})개의 원소로 구성되었으며 각 원소의 키 값은 랜덤하게 생성되었다. 정렬 알고리즘으로 CPU는 퀵 정렬을, GPU는 바이토닉 정렬을 사용하였다. 각 알고리즘의 구현은 퀵 정렬의 경우 C 라이브러리 함수인 qsort를 이용하였고, 바이토닉 정렬은 Nvidia에서 제공하는 CUDA SDK[5]를 사용하였다.

장치의 연산 능력은 장치 제작사가 공식적으로 명시한 수치를 기준으로 하였다. 실험에 사용된 시스템의 그래픽 카드는 Nvidia GeForce 9300 MGS 로 34 GFlops 의 연산능력을 가지며 CPU 는 Intel P9700 모델로 22.4 GFlops 의 연산능력을 가진다고 보고되었다. GPU 는 PCI-Express 버스를 통해 시스템과 연결되므로 초당 메모리 대역폭은 8 GByte로 추정하였다.

CPU에 사용된 퀵 정렬의 복잡도는 $n \log n$ 이고, 퀵 정렬의 루틴은 하나의 원소에 대한 정렬에 75번의 인스트럭션을 수행하였다. 인스트럭션 수는 퀵 정렬의 루틴을 어셈블 코드로 변경하여 한 번의 비교에 소요되는 명령어 수로 결정하였다.. 실험에 사용된 듀얼 코어(Dual-Core) CPU의 연산 능력은 22.4 GFlops 이지만, 실험에 사용된 퀵 정렬은 하나의 코어로 수행하게 되므로 11.2 GFlops 로 가정하였다.

GPU에 사용된 바이토닉 정렬의 복잡도는 배열의 모든 원소가 병렬적으로 처리된다면 $O(\log n^2)$ 이 된다. 하지만 실제로 모든 원소를 병렬적으로 처리할 수는 없으므로 GPU에서 정렬을 수행하는 실제 쓰레드의 수를 고려해야 하였다. 실험에서 사용한 Nvidia SDK 의 바이토닉 정렬은 쓰레드 블록 당 512개의 쓰레드가 사용되었으므로, 복잡도는 $O(n/512)(\log n^2)$ 이 되었다. 실험에서 사용한 바이토닉 정렬 루틴은 53번의 인스트럭션을 수행하였다. 퀵 정렬과 바이토닉 정렬의 인스트럭션 수는 실험에 사용된 코드를 직접 측정한 것을 사용하였다. 이 인스트럭션 수는 다른 연구에서 측정된 수치와 유사하였다[8]. 여기서 결정된 변수들을 바탕으로 3장에 제시된 방법을 이용하여 배열을 분할한 결과가 표 1에 기술하였다.

표 1. 배열 분할 결과
Table 1. Array Partition Result

	CPU	GPU
논리적 추정시간(sec)	0.04227	0.00116
백분율(%)	2.68	97.32
원소개수	28102	1020474

표 2는 전체 정렬 수행에 소요된 시간을 구간별로 나타내었다. 시간 단위는 밀리 초(milisecond)이며 전송 시간에서 Host 는 CPU, Device 는 GPU를 의미한

다. 정렬 수행 시간과 전송 시간은 연산을 수행할 때마다 복합적인 조건에 의한 편차가 발생하기 때문에 열 번의 연산을 수행한 평균값으로 나타내었다.

표 2. 혼합 정렬의 최종 정렬 시간
Table 2. Final Sorting Time of Mixed Sorting.

	CPU	GPU
분할 원소 개수	28102	1020474
총 수행 시간 (ms)	18.121	17.593
퀵 정렬 (ms)	18.029	
바이토닉정렬 (ms)		0.021
머지 정렬 (ms)	0.092	
전송 시간 (ms) (Host to Device)		5.033
전송 시간 (ms) (Device to Host)		12.447

CPU와 GPU를 이용한 혼합 정렬이 최고의 효율을 얻기 위해서는 분할된 두 정렬의 수행이 동시에 끝나야 한다. 표 2 의 결과에서 보듯 CPU 와 GPU의 정렬 시간에는 약간의 차이가 존재하지만 유사한 시간 범위 내에 종료되는 것을 알 수 있었다. GPU의 정렬이 조금 일찍 종료되었지만 두 배열은 하나로 통합되어야 하므로 최종 정렬 시간은 CPU를 기준으로 한 18.121 ms 이 되었다. 1048576 개 원소를 가진 전체 배열을 CPU 혹은 GPU에서만 수행한 결과, CPU 의 퀵 정렬 소요 시간은 249.544 ms 이고 GPU 의 바이토닉 정렬은 19.681 ms 였다. 따라서 혼합 정렬의 수행 시간은 CPU에서만 퀵 정렬을 수행하였을 경우 비해 1377 %, GPU에서 바이토닉 정렬만 수행하였을 경우에 비해 8.6 % 향상되었음을 알 수 있었다.

데이터 크기에 따른 성능 변화를 측정하기 위해 배열 원소 5242880개, 10485760 개에 대해 동일한 방법으로 실험을 수행하였다. 이것은 최초 실험에 사용한 배열 원소의 개수가 각 5배, 10배 증가한 것이며 결과는 아래의 그림 3과 같다. 비교에 사용된 정렬 방법은 바이토닉 정렬과 본 논문에서 제안한 혼합 정렬이었다. CPU만을 사용한 정렬은 수행시간이 GPU에 비하여 상당히 많은 시간이 소요되어서 비교에서 제외하였다. 바이토닉 정렬은 GPU만 사용하여 정렬을 수행하였고, 혼합 정렬은 CPU와 GPU 모두를 사용하여 정렬을 수행하였다. 그림 3에서 알 수 있듯이 혼합 정렬의 경우 모든 실험 구간에서 가장 좋은 성능을 보여주었다.

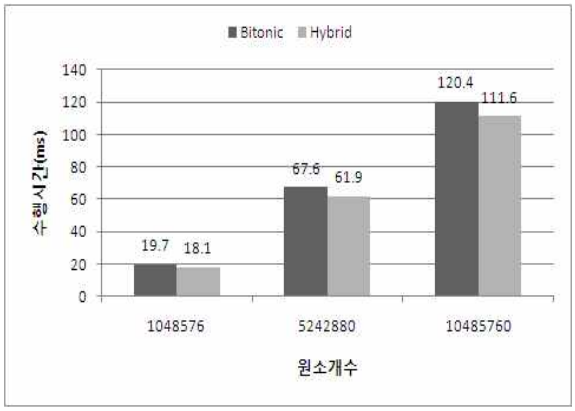

그림 3. 수행 시간 비교
Fig. 3. Comparison of execution time

VI. 결 론

정보화된 사회에 존재하는 수많은 디지털 데이터들을 목적에 맞게 사용하는 과정에서 데이터의 정렬에 대한 필요성은 빈번하게 발생한다. 하지만 시간이 지날수록 방대해지는 디지털 데이터의 정렬 과정 자체도 많은 연산 시간을 소모하고, 알고리즘 개선만을 통한 방법으로는 정렬 효율의 극적인 향상은 이루어지기 힘들다.

본 논문에서는 CPU 뿐만 아니라 단순 연산의 고속 병렬 처리에 최적화 된 구조를 가진 GPU를 데이터 정렬에 이용함으로써 정렬 시간의 단축을 달성하였다. CPU와 GPU를 동시에 사용하기 위해 프로세서의 성능과 사용된 정렬 알고리즘의 복잡도를 기준으로 데이터 배열을 분할하고, 분할된 두 개의 배열에 대해 데이터 전송 시간을 포함한 논리적 정렬 종료 추정 시간을 산정하였다. 추정 시간을 기반으로 두 개 행렬의 연산이 최대한 같은 시간 안에 종료될 수 있도록 행렬 범위를 재조정한 후 각각의 프로세서에서 동시에 정렬을 수행함으로써 정렬 수행시간을 단축할 수 있었다. 실험을 통해 본 논문에서 제안한 데이터 분할 알고리즘에 기반한 혼합 정렬 방법이 CPU에서의 단독 정렬은 물론 GPU 에서의 단독 정렬 보다 효율적임을 확인하였다. 혼합 정렬이 1048576개의 데이터에 대하여 GPU만 사용한 방법보다 8.6%정도

의 수행시간 개선이 있었으며, 데이터의 크기가 커질수록 더 우수한 성능을 보였다.

본 논문에서는 알고리즘의 복잡도, 정렬 루틴의 인스트럭션 수, 그리고, flops로 표시된 하드웨어 성능을 기반으로 데이터를 분할하였다. [9]의 Roofline 모델은 flops는 물론 메모리 대역폭당 인스트럭션 처리율까지 고려한 성능 분석모델이다. 이 모델을 활용하면 본 논문에서 제시한 수행시간 추정법보다 정확한 수행시간 추정은 물론 최적화의 방향까지도 예측할 수 있다. 따라서 이 모델을 활용한 혼합 정렬 알고리즘의 최적화 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] N. Satish, M. Harris, and M. Garland, “Designing efficient sorting algorithms for manycore GPUs”. *Proc. 23rd IEEE Int’l Parallel & Distributed Processing Symposium,* May 2009.

[2] N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha, “GPUTera Sort: High performance graphics coprocessor sorting for large database management”, *in Proc. 2006 ACM SIGMOD Int''l Conference on Management of Data,* 2006, pp. 325-336.

[3] J. Chhugani, W. Macy, A. Baransi, A. D. Nguyen, M. Hagog, S. Kumar, V. W. Lee, Y.-K. Chen, and P. Dubey, “Efficient implementation of sorting on multi-core SIMD CPU architecture”, *in Proc. 34th Int'l Conference on Very Large Data Bases*, Aug. 2008, pp. 1313-1324.

[4] F. Gavril, “Merging with parallel processors”, *Commun. ACM*, vol. 18, no. 10, pp. 588-591, 1975.

[5] NVIDIACorporation, “NVIDIA CUDA SDK”, http://www.nvidia.com/cuda, 2009.

[6] NVIDIA CUDA Programming Guide, *NVIDIA Corporation*, Jun. 2008, version 2.0.

[7] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA”, *Queue*, vol. 6, no. 2, pp. 40-53, Mar/Apr 2008.

[8] Tim Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan: "Photon Mapping on Programmable Graphics Hardware". *Graphics Hardware* 2003.

[9] Samuel Willams, Andrew Waterman, and David Patterson, "Roofline: an Insightful Visual Performance model for multicore architectures," *Comm. of ACM,* Vol. 52, Issue 4, Apr. 2009, pp. 65-76.

권 오 영 (權五暎)

1990년 2월 : 연세대학교 전산과학과 (이학사)
1992년 2월 : 연세대학교 컴퓨터과학과 (이학석사)
1997년 2월 : 연세대학교 컴퓨터과학과 (공학박사)
1997년 4월 ~ 2000년 2월 : 한국전자통신연구원 컴퓨터 · 소프트웨어연구소 선임연구원
2000년 3월 ~ 현재 : 한국기술교육대학교 컴퓨터공학부 부교수
관심분야 : 고성능 컴퓨팅, 임베디드 시스템, 미들웨어