Journal of Korea Multimedia Society Vol. 18, No. 5, May 2015(pp. 620-630)
http://dx.doi.org/10.9717/kmms.2015.18.5.620

다중 CPU 서버 환경에서 동시 사용자를 위한 부하조절 기반 볼륨 가시화 시스템

이웅규†, 계희원††

Load Balanced Volume Rendering System for Concurrent Users in Multi-CPU Server Environment

Woongkyu Lee†, Heewon Kye††

ABSTRACT

This research suggests a load balancing method for a volume rendering system which supports concurrent users. When concurrent users use a volume rendering server system, the computational resources are occupied by a particular user by turns because each process consumes the computational resources as much as possible. In this case, the previous method shows acceptable throughput but the latency is increased for each user. In this research, we suggest a method to improve the latency without performance degradation. Each process makes concessions for taking the resources according to the number of users connected to the system. And we propose a load balancing method in the dynamic situation in which the number of users can vary. Using our methods, we can improve the latency time for each user.

Key words: Volume Rendering, Medical Image Visualization, Multi Processor Server, Load Balancing

1. 서 론

볼륨 가시화는 3차원으로 구성된 볼륨 데이터를 가시화 하는 방법[1,2]으로, 물체를 미세한 정육면체나 미립자의 합으로 표현하는 기술이다. 볼륨 가시화는 주로 의학적, 공학적 가시화에 사용되며 의학용으로는 임상 진단이나 수술 모의실험 등에 사용할 수 있다. 볼륨 데이터는 일반적으로 복셀(voxel)이라고 부르는 스칼라(scalar)값의 삼차원 배열 형태로 구성되어 있다. 볼륨 데이터의 크기는 매우 크기 때문에 가시화 시간이 오래 걸리는 문제가 있다. 이러한 문제를 해결하기 위해 다중 프로세서를 이용한 가속화 방법들이 연구되었다.

최근 설치비용과 유지보수 같은 비용 편익 관점에서 클라이언트 서버 모델의 가시화 시스템이 주목받고 있다. 클라이언트 서버 모델은 서버에서 다수의 클라이언트 요청에 대해 연산이나 서비스를 제공하는 방법이다. 하나의 서버에 수 백 명 이상의 사용자가 동시에 접속하는 컴퓨터 게임과 다르게, 의료 정보 가시화 시스템에서는 하나의 가시화 서버를 보통 2명에서 6명 이하의 의사 또는 방사선사가 동시에 사용하게 된다. 각 사용자에게 별도의 완성 시스템을 제공하면, 하드웨어 비용 외에도 업그레이드와 시스템 유지보수에 비용이 발생하기 때문에, 한 대의 가시화 서버와 다수의 클라이언트를 운영하는 방법이 비용 측면에서 효율적이 되는 것이다.

※ Corresponding Author: Heewon Kye, Address: (136-792) 116 Samseongyoro-16Gil, Seongbuk-Gu, Seoul, Korea, TEL: +82-2-760-8014, FAX: +82-760-4347, E-mail: kuei@hansung.ac.kr
Receipt date: Dec. 9, 2014, Revision date: Mar. 30, 2015
Approval date: Apr. 14, 2015

† Dept. of Information Systems Eng., Hansung University (E-mail: teriusbin@naer.com)
†† Dept. of Information Systems Eng., Hansung University
※ This research was financially supported by Hansung University.

실시간 볼륨 가시화에 사용되는 서버 프로세서는 CPU(central processing unit)와 GPU(graphic processing unit)가 있다. GPU는 그래픽 작업에 특화된 연산 장치로 화소나 정점의 고속 연산이 가능하다. 그러나 GPU는 그래픽 메모리 용량이 시스템 메모리에 비해 부족하기 때문에, 다중 사용자 처리 시 볼륨 가시화에 제약이 있다. 이에 반해 다중 CPU를 이용한 볼륨 가시화의 경우 시스템 메모리 용량을 충분하게 확장할 수 있기 때문에 다중 사용자 접속 환경에서 유리하다. 또한 다중 CPU를 이용한 볼륨 가시화는 범용의 프로그래밍 언어로 작성 가능하므로 이식성이 높다는 장점이 있다. 그리고 다중 CPU의 성능이 점차 향상되고 있기 때문에 볼륨 가시화에서의 다중 CPU 활용은 중요한 사항이라 할 수 있다.

기존에 다중 CPU 기반의 실시간 볼륨 가시화 가속화 방안이 많이 연구되어 왔다. POSIX 시스템에서는 일반적인 표준 API인 Pthread(POSIX Threads)를 이용한다[3]. 이때 마스터 스레드는 다수의 종속 스레드를 동작시켜 볼륨 가시화 연산을 수행한다, 이후 마스터와 종속 스레드 간의 공유 메모리를 통하여 종속 스레드로부터 전달받은 결과 값을 마스터 스레드가 계산하여 최종 결과 영상을 가시화 한다. 그리고 공유 메모리 다중 처리 프로그래밍 API인 open MP를 사용하여 다수의 프로세서에 독립적으로 다중 스레드를 동작시켜 볼륨 가시화 알고리즘을 수행하는 방안[4]도 있다. 또한 병렬 볼륨 가시화 연산을 MPI(message passing interface)를 통하여 수행하는 방법[5]도 연구되었다. MPI란 메시지 전달 병렬 프로그래밍을 위해 표준화된 데이터 통신 라이브러리이다. 해당 연구는 마스터 노드 PC가 다수의 슬레이브 노드 PC에 분할된 데이터와 데이터 처리량을 메시지를 통해 분배하고 처리된 결과를 메시지를 통해 전달 받아 최종 결과 영상을 출력하는 방식이다. 최근에는 MPI방식과 openMP 및 Pthreads가 융합된 형태의 볼륨 가시화 가속화 방법[6]이 연구되고 있으며, 가장 작은 수준의 병렬 프로그래밍인 SIMD(single instruction single data)를 이용하여 단일 프로세서에 적합한 아키텍처를 제안하는 연구[7]와 SIMD와 비동기적이면서 독립적으로 동작하는 다수의 PC의 프로세서를 통해 병렬 연산을 수행하는 MIMD(multiple instruction multiple data)가 융합된 형태의 볼륨 가시화 가속화 방안[8]들이 제시되었다.

이러한 다중 CPU 기반의 볼륨가시화 연구의 공통점은 단일 사용자 환경에서 고속의 실시간 볼륨 가시화를 제공한다는 점이다. 하지만 기존의 방법은 CPU의 모든 자원을 한명의 사용자가 독점하여 가속화하는 방법이며 다중 사용자 환경은 고려되지 않았다. 따라서 기존 방법을 다중 사용자 환경에 적용시킬 경우, 각 사용자는 운영체제의 스케줄링(scheduling)에 따라 순차적으로 시스템 자원을 독점하여 사용하게 된다[9].

여러 사용자가 작업을 수행하고 있는 경우, 시스템에 새로 접속한 신규 사용자는 시스템 자원을 할당받을 때까지 매우 오래 기다려야 하는 문제가 있다. 이러한 단점을 해결하려면, 다중 사용자를 위한 부하조절 연구가 필요하다. 하지만 볼륨 가시화 시스템에서의 자원 분배와 관련된 선행 연구는 찾기 어려운 실정이다. 서버 급 시스템 환경에서의 볼륨 가시화 방안[10]이 연구되었지만 다중 사용자 접속 시 자원의 부하 문제를 개선하기 위한 방법은 공개되어 있지 않다. 다중 GPU 기반의 서버 시스템[11]의 경우, 각 사용자의 요청은 하나의 GPU가 전담하여 처리한다. 여러 사용자가 하나의 GPU에 연결되며, 각 GPU는 사용자들의 요구를 순차적으로 처리[11]한다. GPU의 가시화 속도는 매우 빠르기 때문에 순차 처리가 큰 문제가 아니지만, 상대적으로 느린 CPU 기반의 시스템에서는 순차 처리로 인한 응답속도 지연이 문제가 된다.

만약 전체 시스템의 부하를 관리할 권한이 있다면, 각 각 프로그램의 작업 부하량에 따라 병렬적으로 CPU 자원을 분배하면 된다. 그러나 응용프로그램은 전체적인 자원 분배에 대한 권한을 가지고 있지 않다. 따라서 본 연구는 각 볼륨 가시화 응용프로그램이 자원을 양보하여 전체적인 성능을 향상시키는 방법을 연구하려 한다.

본 논문은 서버 급 시스템의 다중 CPU 환경에서 다수의 사용자 접속 시, 시스템에 접속한 사용자 수에 따라 CPU 자원을 효과적으로 분배하는 방법을 제안한다. 이를 통해 시스템의 응답 속도(latency)를 개선 할 수 있다. 이후 2장에서 다중 CPU 환경에서의 볼륨 가시화에 대해 설명한다. 그리고 3장에서 일반적인 가시화 알고리즘에 대해 설명하고 본 연구의 제안 알고리즘을 설명한다. 4장에서는 실험 결과를 보이고 5장에서 결론을 맺는다.

2. 관련 연구

이번 장에서는 제안 시스템이 병렬 가속화 하고자 하는 볼륨 가시화 알고리즘에 대하여 설명한다.

Fig. 1은 볼륨 가시화의 대표적인 기법인 광선 투사법(volume ray-casting)으로 볼륨 가시화 기법 중 높은 화질의 영상을 제공하는 기법[1,2,12]이다. 광선 투사법은 2차원 평면에 3차원 물체를 표현하는 기법으로 2차원 결과 영상의 각 화소 색상을 결정하기 위해 모든 화소별로 가상의 광선을 3차원 볼륨 데이터에 발사하여 이와 교차하는 모든 복셀의 밀도 값을 계산하고 색상을 결정해서 누적해야 한다. 광선 투사법은 높은 화질의 결과 영상을 생성 할 수 있지만 광선의 방향에 따라 임의의 순서로 복셀을 참조해야 하고 최종 색상을 결정하기 위해 누적연산을 반복적으로 수행해야 한다[13]. 이 반복 누적 연산은 볼륨의 크기만큼 수행되어야 하기 때문에 연산량이 많다.

하지만 다중 프로세서 기반의 병렬 볼륨 가시화는 다수의 프로세서를 이용하여 동시에 광선을 투사함으로써 고속의 볼륨 가시화가 가능하다. 각 프로세서는 서로 독립적으로 연산을 수행할 때 높은 효율을 유지할 수 있다[14]. 광선 투사법의 각 광선들은 서로의 진행방향과 색상결정이 독립적이므로 각 프로세서에 서로 다른 광선을 할당하여 병렬 연산을 수행하면 효과적이다.

3. 연구 방법

본 연구는 서버 급 다중 CPU 기반의 볼륨 가시화 환경에서 다중 사용자 접속 시 다중 CPU 자원을 효율적으로 분배하는 방법을 제안한다. 본론 3.1절에서 다중 사용자 접속 시 일반적인 볼륨 가시화 알고리즘을 소개하고 3.2절에서 동일 환경 내에서 CPU 부하 제어 제안 알고리즘을 소개한다. 그리고 3.3절에서 제안 알고리즘의 최적화 방법을 소개한다.

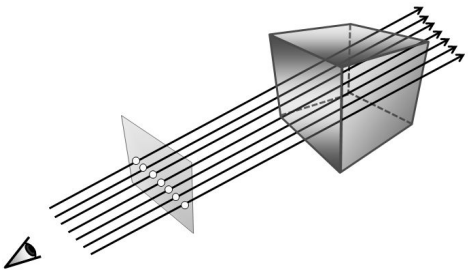


Fig. 1. Volume ray-casting.

3.1 기존 방법

Fig. 2는 서버 급 다중 CPU 기반의 일반적인 볼륨 가시화 알고리즘이다. 일반적인 볼륨 가시화 시스템의 경우 한명의 사용자 접속 후 프로그램이 시작되면 해당 프로그램이 다중 프로세서의 모든 자원을 점유하여 광선 투사법을 병렬적으로 수행하도록 설계되어있다[3,4,7]. 그렇기 때문에 다중 사용자 접속 시 운영체제는 프로세스 스케줄링을 통해 각 사용자가 순차적으로 자원을 점유 하도록 한다.

Fig. 3은 다중 CPU 환경에서 다중 사용자 접속 시 CPU자원의 점유 과정이다. N개의 프로세서를 가진 환경에서 K명의 사용자가 접속한다면 각각의 사용자는 돌아가면서 자원을 점유한다. 이 때 나머지 사용자들은 대기 상태가 된다. 예를 들어 프로세서 개수가 12개인 환경에서 3명의 사용자가 접속하였다면 1번 사용자는 12개의 자원을 모두 점유하여 작업을 수행하고 나머지 사용자인 2번과 3번은 대기 상태에 들어간다. 이는 전체적인 작업속도 효율은 좋으나 각 사용자마다의 응답 속도가 느리다는 단점이 있다.

3.2 제안 방법

본 연구는 볼륨 가시화 알고리즘 수행 중 다중 CPU의 부하 조절 및 응답 속도 향상을 위해 각 사용

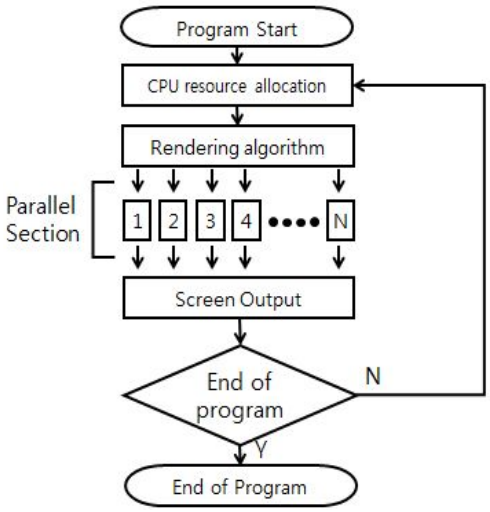


Fig. 2. Traditional algorithm.

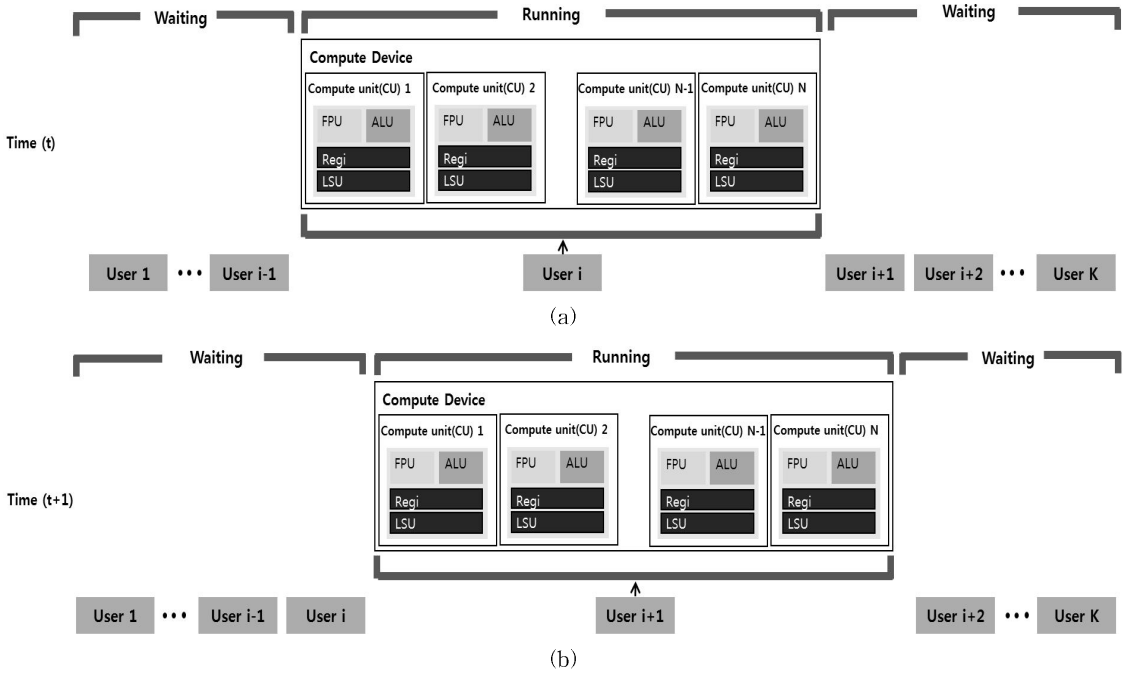


Fig. 3. Multi processor allocation (a) at time t, (b) at time t+1.

자에게 프로세서 자원을 동적으로 분배한다. 자원 분배의 기준은 시스템 내부에 접속한 사용자수로 판단한다. 볼륨 가시화는 각각의 사용자의 작업 부하량이 유동적으로 변하기 때문에 사용자들의 작업량은 동일하다고 가정하고 자원을 동등하게 분배한다.

Fig. 4는 본 논문에서 제안하는 CPU 부하 제어

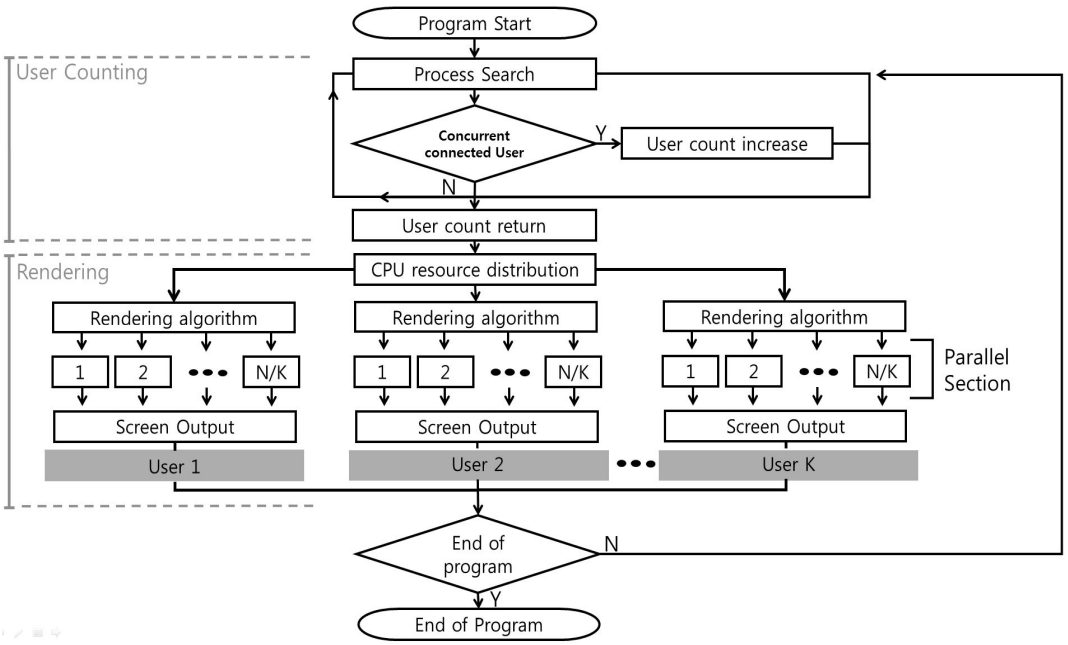


Fig. 4. Proposed algorithm.

알고리즘이다. 먼저 가시화 알고리즘이 수행되기 전에 시스템에 접속한 사용자들의 수를 탐색한다. 탐색 범위는 시스템에서 실행되는 전체 프로세스 이며, 접속한 사용자 수를 파악하여 사용자 전체에게 CPU 자원을 균등하게 분배한다. 각 사용자는 분배된 자원을 이용하여 가시화 알고리즘을 수행한다.

위 Fig. 5는 다중 CPU 환경에서 자원의 균등 분배 과정이다. N개의 프로세서를 가진 환경에서 K명의 사용자가 접속한다면 각각의 사용자는 N/K개의 프로세서만을 사용한다. 예를 들어 프로세서 개수가 12개인 환경에서 3명의 사용자가 접속한다면 각 사용자가 사용하는 프로세의 개수는 4개가 된다. 그리고 1명의 사용자가 추가적으로 더 접속 한다면 사용자가 4명이 되어 각 사용자에게 프로세서는 3개씩 분배된다. 이처럼 시스템 내에 접속한 사용자의 수에 따라 시스템은 동적으로 CPU 자원을 양보하여 사용한다.

3.3 유휴 자원 재분배

Fig. 6은 각 사용자에게 분배되는 CPU자원의 개수가 균등하게 나누어 떨어 지지 않는 경우를 나타낸다. 예를 들어 프로세서가 12개인 환경에서 5명의 사용자가 접속했을 경우 3.2절의 균등 분배 알고리즘에 따라 각 사용자는 프로세서를 2개씩 점유하게 되고 남은 2개의 자원은 유휴상태로 남는다. 이때 나머지 유휴자원을 분배하는 방법이 필요하다.

본 연구는 프로세서가 균등하게 분배 될 경우 3.1절의 균등분배 가시화 알고리즘을 수행한다. 하지만 균등하게 분배되지 않을 경우 남은 유휴 자원을 특정 사용자에게 차등 분배한다. Fig. 7은 본 연구에서 제안하는 차등 분배 과정이다. N개의 프로세서를 가진

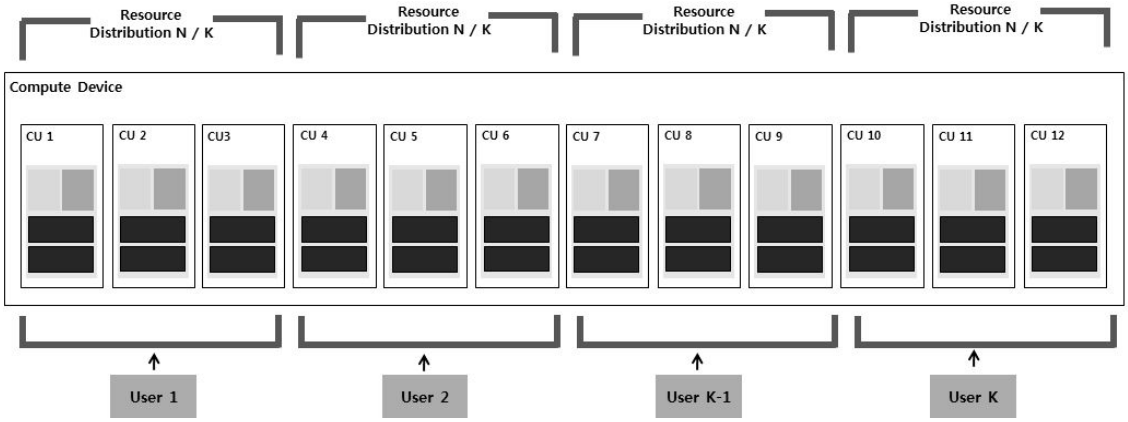


Fig. 5. Even resource distribution for multi processor (N=12, K=4).

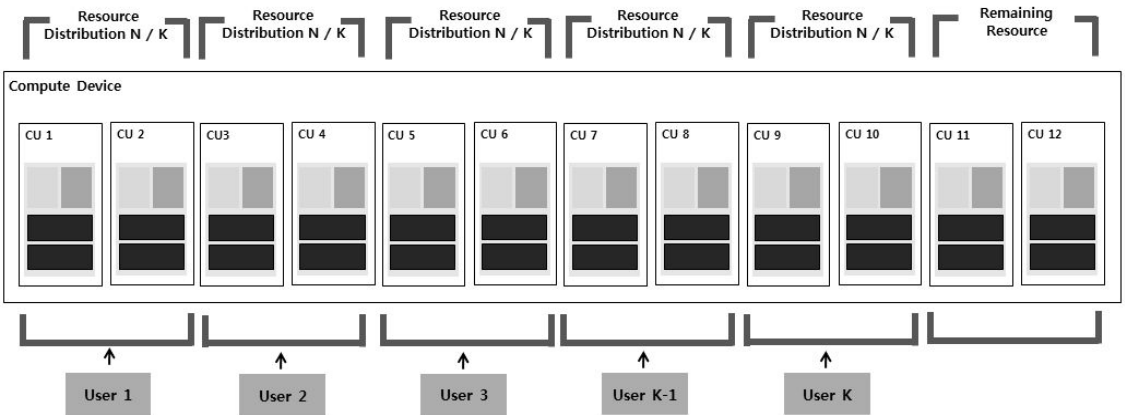


Fig. 6. Remaining computing resources (N=12, K=5).

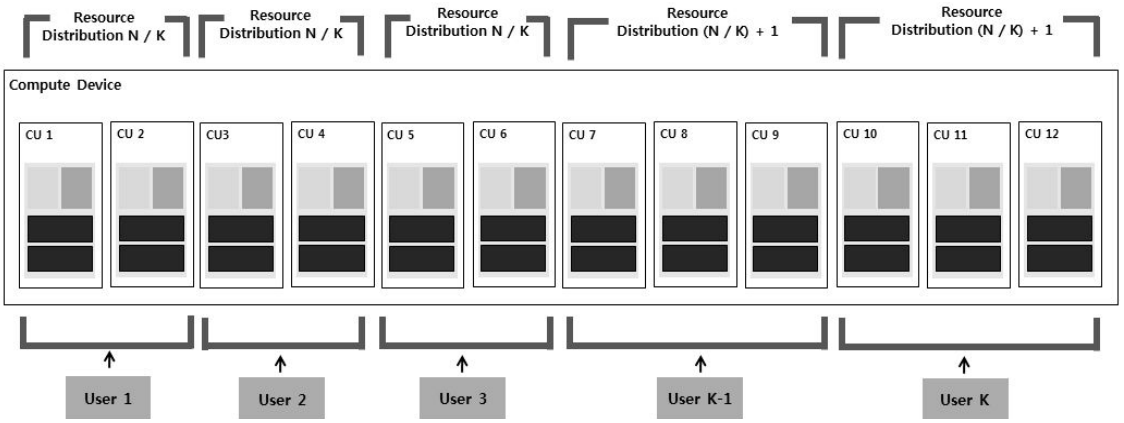


Fig. 7. Differential resource distribution for multi processor (N=12, K=5).

환경에서 K명의 사용자가 접속했다 가정하자. 이때 프로세서가 균등하게 분배 되지 않는다면 남은 유휴 프로세서 (N mod K)개를 각 사용자에게 1개씩 추가적으로 재분배 한다. 예를 들어 프로세서 개수가 12개인 환경에서 5명의 사용자가 접속한다면 각 사용자에게 균등하게 분배되는 프로세서의 개수는 (12 / 5) = 2개가 된다. 그리고 남은 유휴 프로세서 (12 mod 5) = 2개를 2명의 사용자에게 1개씩 차등 분배한다. 즉 사용자 1번, 2번, 3번은 2개의 프로세서를 점유하게 되고 사용자 4번과 5번은 (12 mod 5) + 1 = 3개를 점유하게 된다. 이때 추가적으로 자원을 할당 받은 사용자가 지속적으로 자원을 소유하면서 발생하는 불균형을 고려해야 한다.

본 연구는 차등 분배과정 이후 불균형을 개선하기 위해 추가 자원 소유자를 주기적으로 변경한다. Fig. 8은 본 연구에서 제안하는 차등 분배 순환 과정 이다. 예를 들어 프로세서 개수가 12개인 환경에서 5명의 사용자가 접속하였다면 그림 8(a)의 과정에 따라 사용자 1번, 2번, 3번은 2개의 프로세서를 점유하게 되고 사용자 4번과 5번은 3개를 점유하게 된다. 이 후 시스템 내에서 정의한 일정 주기의 시간이 지나면 추가 자원 소유자를 순차적으로 변경한다. 즉 기존 추가 자원 소유자인 사용자 4번과 5번은 2개의 자원을 소유하게 되고 사용자 1번과 2번이 새롭게 추가 자원을 분배 받는다. 해당 과정은 균등 분배 유무 판별에서 자원이 균등하게 분배되지 않는 동안 각각의 모든 사용자에게 반복적으로 적용된다. 차등 분배 순환 과정을 통해 추가자원을 특정 사용자가 독점하면서 생길 수 있는 불균형을 방지 할 수 있다.

4. 실험 결과

본 연구의 실험은 세 종류의 컴퓨터를 사용하였다. 첫 번째는 보급형 데스크탑 컴퓨터(PC 1)로 CPU는 i5-2520M 2.5GHz로 4개의 프로세서가 장착되어 있다. 두 번째는 보급형 데스크탑 컴퓨터(PC 2)로 CPU는 intel의 i7-3770 3.4GHz로 동일하게 4개의 프로세서가 장착 되어있다. 세 번째는 서버 급 워크스테이션(PC 3)으로 두 개의 intel xeon CPU E5645 2.40GHz가 장착되어 있으며, 각 CPU는 6개의 프로세서가 포함되어 있어, 총 12개의 프로세서가 장착되어 있다. 소프트웨어 개발 환경은 Visual studio 2008, C++를 사용하였으며 화면 출력은 DirectX[15]를 사용하였다. 볼륨 가시화 알고리즘 병렬 가속화를 위해 OpenMP[16,17]를 사용하였으며 출력 영상 픽셀의 y축 방향으로 다중 스레드(thread)를 생성하여 가시화 알고리즘을 구현하였다. 또한 분배 알고리즘을 수행하기 위해 현재 가용 가능한 스레드의 개수 및 분배 스레드의 개수를 가변적으로 지정하였다.

또한, 병렬처리와 빈 공간 도약 기법(empty space leaping), 조기 광선 종료 기법(early ray termination)을 통한 가속화를 적용했다[18-21]. 빈 공간 도약 기법은 광선 투사법 기반의 볼륨 가시화의 대표적인 가속화 기법으로써 출력 영상에 반영되지 않을 투명한 영역을 미리 추출하여 걸러내는 방법이다. 볼륨 데이터를 블록으로 나누어 각 영역의 최댓값, 최솟값 등을 검색하여 별도의 메모리에 저장한다. 이를 가시화 수행 중에 참조하여 광선이 진행 도중 마주친 블록이 투명한지를 판단하고 투명한 블록이라면 건

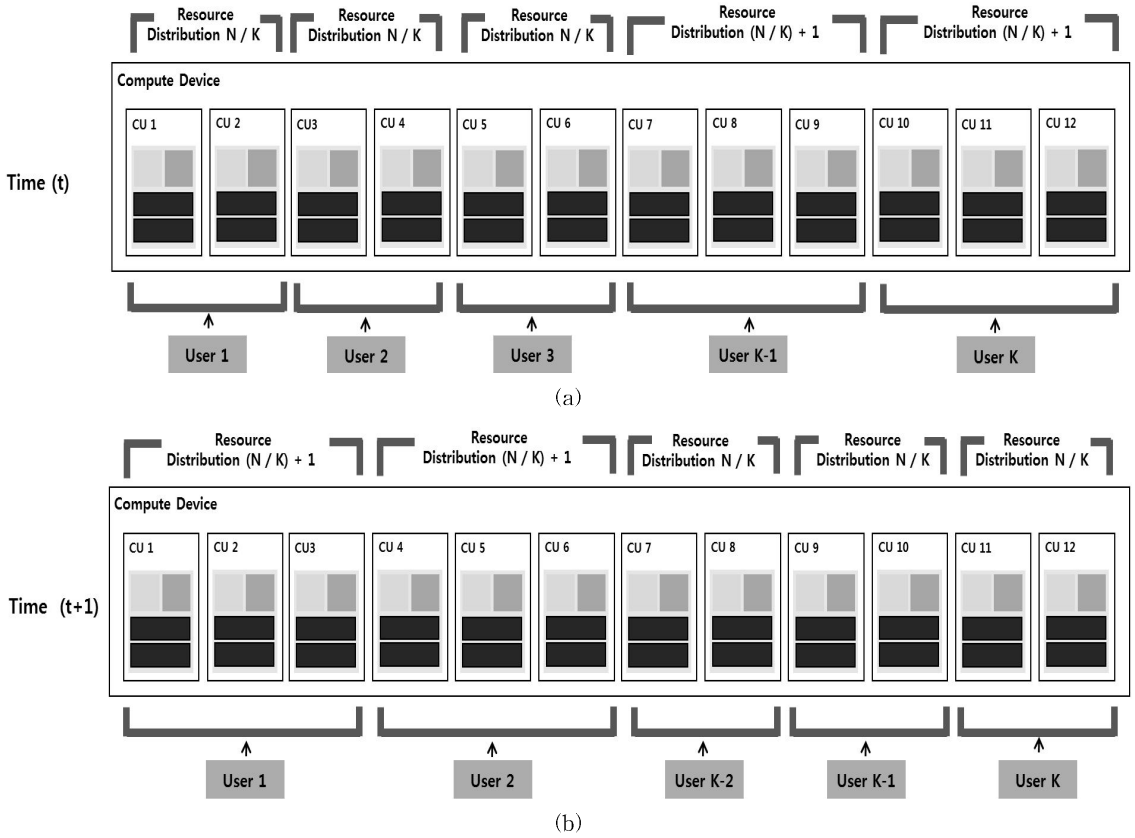
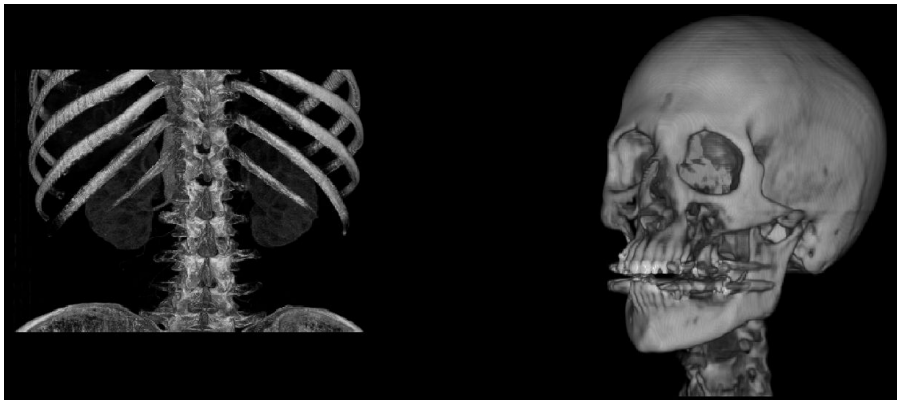


Fig. 8. Differential resource distribution for multi processor (N=12, K=5). (a) at time t, (b) at time t+1.

너뛰어 가속화 한다[20,21]. 조기 광선 종료 기법은 광선 투사법 에서 각 광선이 누적해 나가는 색상의 불투명한 정도가 매우 커지면 더 이상 광선의 진행이 불필요하므로 종료하는 방법이다.

Fig. 9는 본 연구에서 실험에 사용한 데이터로 좌측 그림은 512 × 512 × 300의 복부 데이터(abdomen data)이고, 우측 그림은 256 × 256 × 225의 머리 데이터(head data)이다. 실험 방법은 기존 방법(한 명의



(a) (b)

Fig. 9. The experimental data. (a) Abdomen Data, (b) Head Data.

사용자가 CPU의 자원을 독점하는 방법)과 제안 방법(사용자간에 CPU 자원을 공유하는 방법)을 비교했다. 실험은 두 가지로 기획하였다. 첫 번째 실험은 총 가시화 시간을 알기 위해 100번의 장면을 가시화한 후 평균 가시화 시간을 측정하였다. 그리고 각 실험 마다 10번의 실험을 통해 얻은 수치의 평균을 계산하였다.

Table 1과 Table 2는 기존 방법 대비 제안 방법의 총 가시화 시간 변화를 접속 사용자 수에 따라 비교한 것이다. 머리 데이터의 경우 보급형 데스크탑인 PC 1과 PC 2환경에서 제안 방법이 기존 방법에 비해 약 1% 정도의 시간이 더 소요 되었고, 서버 급 워크스테이션인 PC 3의 경우 약 5-7%정도 더 소요 되었다. 복부 데이터는 보급형 데스크탑인 PC 1과 PC 2환경에서 약 2~3% 정도의 시간이 더 소요 되었고, 서버 급 워크스테이션인 PC 3은 약 7~10%정도 더 소요 되었다. 제안 방법은 기존 방법에 비해 약간의 시간이 더 소요되지만 비슷한 수준이다.

복부 데이터의 평균 가시화 시간이 머리 데이터에 비해 큰 이유는 데이터의 크기가 커짐에 따라 캐시에 대한 경합이 발생하게 되고 이에 따라 추가 손실이 발생된다고 판단된다. 기존 방법은 모든 프로세서가 하나의 데이터에 집중하여 연산하는데 비해, 본 연구는 여러 데이터를 동시에 다루게 되므로 시스템이 처리하는 작업 공간(working set)이 확대된다. 그 결과로 제안 방법이 기존 방법보다 평균 가시화 시간이 약간 더 소요되는 것으로 판단된다.

두 번째 응답 속도 실험은 사용자로부터 새로운 가시화 요청이 있을 때, 시스템이 응답하는 시간을 확인하기 위해 설계하였다. 이를 위해 1개 이상의 기존 프로그램이 계속 실행되고 있는 환경에서 새로운 프로그램이 실행되었을 경우, 신규 프로그램의 첫 번째 가시화 화면을 출력 하는데 소요되는 시간을 측정하였다. 접속한 사용자 별로 평균을 측정하였으며 마찬가지로 각 실험 마다 10번의 실험을 통해 얻은 수치의 평균을 계산하였다.

Table 3과 Table 4는 기존 방법 대비 제안 방법의 프로그램 응답 시간 증감률을 사용자 접속 수에 따라

Table 1. Head data rendering time

User (People)	PC 1			PC 2			PC 3		
	Existing Method (sec)	Proposed Method (sec)	Increased Time (%)	Existing Method (sec)	Proposed Method (sec)	Increased Time (%)	Existing Method (sec)	Proposed Method (sec)	Increased Time (%)
1	0.353	0.356	0.8	0.216	0.217	0.5	0.167	0.168	0.9
2	0.675	0.679	0.6	0.390	0.396	1.1	0.268	0.286	6.7
3	1.065	1.077	1.1	0.548	0.553	1.5	0.360	0.380	5.7
4	1.416	1.429	0.9	0.735	0.749	1.8	0.467	0.503	7.6
5							0.577	0.609	5.5
6							0.679	0.719	5.9

Table 2. Abdomen data rendering time

User (People)	PC 1			PC 2			PC 3		
	Existing Method (sec)	Proposed Method (sec)	Increased Time (%)	Existing Method (sec)	Proposed Method (sec)	Increased Time (%)	Existing Method (sec)	Proposed Method (sec)	Increased Time (%)
1	1.114	1.126	1.0	0.523	0.527	0.7	0.424	0.425	0.2
2	1.679	1.709	1.7	0.992	1.024	3.2	0.598	0.649	8.5
3	2.323	2.375	2.2	1.235	1.281	3.7	0.867	0.929	7.1
4	2.955	3.039	2.8	1.567	1.608	2.6	1.055	1.166	10.5
5							1.250	1.340	7.2
6							1.501	1.624	8.2

Table 3. Head data latency time

User (People)	PC 1			PC 2			PC 3		
	Existing Method (sec)	Proposed Method (sec)	Increased Speed (%)	Existing Method (sec)	Proposed Method (sec)	Increased Speed (%)	Existing Method (sec)	Proposed Method (sec)	Increased Speed (%)
1	0.472	0.477	-1.1	0.398	0.401	-0.8	0.325	0.325	0
2	0.971	0.781	24.3	1.082	0.945	14.4	0.443	0.413	7.2
3	1.780	1.456	22.2	1.543	1.158	33.2	0.604	0.520	16.1
4	2.902	1.389	108.2	2.041	1.035	97.1	0.817	0.654	24.9
5							1.324	0.881	50.2
6							1.968	0.912	115.7

Table 4. Abdomen data latency time

User (People)	PC 1			PC 2			PC 3		
	Existing Method (sec)	Proposed Method (sec)	Increased Speed (%)	Existing Method (sec)	Proposed Method (sec)	Increased Speed (%)	Existing Method (sec)	Proposed Method (sec)	Increased Speed (%)
1	1.383	1.387	-0.3	0.777	0.778	-0.2	0.750	0.766	-2.1
2	2.195	1.949	12.6	1.179	1.168	0.9	0.928	0.903	2.7
3	3.693	3.091	19.4	1.992	1.454	37.0	1.409	1.285	9.6
4	4.622	3.121	48.0	2.767	1.737	59.2	2.200	1.671	31.6
5							2.481	1.637	51.5
6							3.208	1.893	69.4

비교한 표이다. 제안 방법을 통해 응답 시간이 대폭 향상 되었으며 사용자의 접속 수가 증가할수록 반응성이 증가하는 것을 볼 수 있다. Table 3의 머리 데이터의 경우 PC 3에서 6명의 사용자 접속 시 응답 속도가 최대 115.7%까지 향상되었고, Table 4의 복부 데이터의 경우 PC 3에서 6명의 사용자 접속 시 응답 속도가 최대 69.4%까지 향상되었다.

반응성이 향상된 이유는 기존 방법의 경우 새로운 사용자가 접속했을 시 기존 사용자가 프로세서 자원을 독점하고 있기 때문에 신규 사용자가 자원을 사용하지 못하고 대기하지만, 제안 방법은 기존 사용자가 신규 사용자에게 프로세서 자원을 양보하기 때문에 반응 속도가 향상된다. 다만, 접속 사용자가 1명일 경우 속도가 약 0~2% 감소하는 이유는 본 연구에서 제안한 사용자 수를 확인 하는데 필요한 부하라고 판단된다.

본 연구는 하나의 다중 프로세서 시스템에 6명 이하의 소수 사용자가 매우 밀도 있는 작업을 수행하는 것을 가정하고 있다. 이러한 작업 방식은 기존의 작업 부하 조절 연구에서 찾아보기 어려운 것이 사실이다. 따라서 본 연구의 결과와 기존 연구의 결과를 직접 비교하기는 어렵다. 다만, 다수의 서버를 이용하여 초당 60회 이상의 요청을 처리하는 작업 부하 조절 연구[22]와 간접적인 비교를 수행 할 수 있다. 기존 연구의 RSIF 알고리즘[22]은 작업 부하를 조절하며 순차처리 방법에 비해, 처리속도(throughput)는 10%정도 저하되지만 응답시간(setup time)을 약 25%정도 향상시킨다. 본 논문의 제안 방법은 순차처리 방법에 비해 처리속도는 5-10% 정도 저하되지만 응답시간(latency time)을 69-115% 향상시키는 점과 간접적으로 비교가 가능하다.

5. 결 론

본 연구는 다중프로세서 CPU 환경에서 다중 사용자 접속 시 효율적인 동적 부하 제어 기법을 제안하였다. 기존 볼륨 가시화 방법은 서버 급 PC 환경에서 다중 사용자 접속 시 하나의 사용자가 CPU의 모

든 자원을 독점 하면서 비효율이 생긴다. 본 연구는 다중 사용자 접속 환경에서 가시화 시간에는 큰 차이를 보이지 않으면서 반응속도를 대폭 향상 시켰다.

이를 위하여 볼륨 가시화 전에 접속 사용자수를 파악하여 프로세서를 동적으로 균등 분배 하는 방법을 제안하였고, 프로세서가 균등하게 나누어 떨어 지지 않는 경우에 대한 최적화 방법을 제안하였다. 제안 방법을 기존 방법과 여러 PC환경에서 비교한 결과, 본 연구는 기존 방법에 비해 가시화 시간은 10% 이하의 추가시간이 소요되면서 반응 속도 측면에서 최대 116% 까지 향상되었다. 제안 방법은 다중 사용자 기반 클라이언트 서버 볼륨 가시화 시스템에 적용 가능하며, 제안 방법을 통하여 사용자의 가시화 요청에 의해 발생하는 응답 시간을 향상시킬 수 있다. 향후 연구로서 추가적인 대용량 메모리를 사용하는 고화질 볼륨 가시화[23]에 본 알고리즘을 적용하여 메모리 측면에서의 부하 조절을 수행할 예정이다.

REFERENCE

[1] M. Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, Vol. 8, pp. 29-37. 1988.

[2] P. SabelIa, "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Proceedings of ACM SIGGRAPH 1988 the 15th Annual Conference on Computer Graphics and Interactive Techniques,* pp. 51-58. 1988.

[3] E.W. Bethel and M. Howison, "Multi-core and Many-core Shared-memory Parallel Raycasting Volume Rendering Optimization and Tuning," *International Journal of High Performance Computing Applications,* Vol. 26, No. 4, pp. 399-412, 2012.

[4] E. Okuyan and U. Güdükbay, "Direct Volume Rendering of Unstructured Tetrahedral Meshes using CUDA and OpenMP," *The Journal of Supercomputing*, Vol. 67, No. 2, pp. 324-344, 2014.

[5] L. Castanie, C. Mion, X. Cavin, and B. Levy. "Distributed Shared Memory for Roaming Large Volumes," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, pp. 1299-1306, 2006.

[6] M. Howison, E.W. Bethel, and H. Chikls, "MPI-hybrid Parallelism for Volume Rendering on Large, Multi-core Systems," *Proceedings of the 10th Eurographics Conference on Parallel Graphics and Visualization*, pp. 1-10, 2010.

[7] M. Meissner, S. Grimm, W. Strasser, J. Packer, and D. Latimer "Parallel Volume Rendering on a Single-chip SIMD Architecture," *Proceedings of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics,* pp. 107-113, 2001.

[8] C.S. Leo and H. Schroder, "Fast Processing of Medical Images using a New Parallel Architecture the Hybrid System," *Proceeding of Fifth IEEE Southwest Symposium on IEEE, in Image Analysis and Interpretation,* pp. 148-152, 2002.

[9] A. Silberschatz, P. Baer, and G. Gagne, *Operating System Concepts Eight Edition,* Wiley, New York, 2008.

[10] L. Parsonson, L. Bai, S. Grimm, A. Bajwa, and L. Bourn, "Medical Imaging in a Cloud Computing Environment," *Proceeding of International Conference on Cloud Computing and Services Science,* pp. 327-332, 2011.

[11] T. Sharp, D. Robertson, and A. Criminisi, *Volume Rendering on Server GPUs for Enterprise-Scale Medical Applications*, Microsoft Tech Report MSR-TR-2010-72, 2010.

[12] K. Engel, M. Hadwiger, J.M. Kniss, C.R. Salama, and D. Weiskopf, *Real-Time Volume Graphics,* Wellesley, Massachusetts, 2006.

[13] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 2, pp. 99-108, 1995.

[14] Y. Jung, *CUDA Parallel Programing*, Freelec, Seoul, 2011.

[15] DirectX SDK, http://msdn.microsoft.com/en-us/directx (accessed Nov, 18, 2014).

[16] Y. Jung, *OpenMP Parallel Programing*, Freelec, Seoul, 2011.

[17] OpenMP, http://openmp.org/wp/ (accessed Nov, 24, 2014).

[18] F. Hernell, P. Ljung, and A. Tnnerman, "Local Ambient Occlusion in Direct Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 16, No. 4, pp. 548-559, 2010.

[19] L. Marsalek, A. Hauber, and P. Slusalled, "High-Speed Volume Ray Casting With CUDA," *Proceeding of IEEE Symposium Interactive Ray Tracing*, pp. 185, 2008.

[20] M. Levoy, "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, Vol. 9, No. 3, pp. 245-261, 1990.

[21] W. Li, K. Mueller, and A. Kaufman, "Empty Space Skipping and Occlusion Clipping for Texture-Based Volume Rendering," *Proceedings of IEEE Visualization Conference*, pp. 317-324, 2003.

[22] R. Logeswaran and L.C. Chen, "A Novel Strategy for Load Balancing of Distributed Medical Applications," *Journal of Medical Systems*, Vol. 36, No. 2, pp. 483-490, 2012.

[23] J. Nam and H. Kye, "Fast Ambient Occlusion Volume Rendering using Local Statistics," *Journal of Korea Multimedia Society*, Vol. 18, No. 2, pp. 158-167, 2015.

이 웅 규

2014년 한성대학교 정보 시스템 공학과 학사

2014년~현재 한성대학교 정보 시스템 공학과 석사 과정

관심분야: Volume rendering, Medical image visualization, CPU Multi processing, GPU Multi processing.

계 희 원

1999년 서울대학교 전산과학과 학사

2001년 서울대학교 전기컴퓨터 공학부 석사

2005년 서울대학교 전기컴퓨터 공학부 박사

2007년~현재 한성대학교 정보시스템공학과 부교수

관심분야: 볼륨 가시화, 실시간 렌더링, 대용량 영상처리