시간 분석을 통한 그리드 응용 프로그램의 성능 향상을 위한 연구

김형준[0] 우영제 권용원 류소현 정창성
고려대학교 대학원 전자컴퓨터학과
박형우* 한국과학기술정보연구원 슈퍼컴퓨팅센터
{hjkim[0],gossamor,messias,luco}@snoopy.korea.ac.kr, csjeong@charlie.korea.ac.kr

Developing Performance of Grid Application through Analysis of Time

Hyung-Jun Kim[0] ,Young-Je Woo, So-Hyun Ryu, Yong-Won Kwon, Chang-Sung Jeong
Dept. of Electronic and Computer, Korea University

요 약

최근 단순한 데이터 공유만이 아니라 모든 자원(중앙 처리 장치, 메모리, 저장 장치, 네트워크)을 공유하는 그리드 컴퓨팅이 주목을 받고 있다. 현재 그리드 응용 시스템들은 대부분 Globus toolkit을 통해 프로세스를 생성 관리하고 있으며 MPICH-G2를 가지고 그 프로세스간의 통신을 하고 있다. 본 논문은 이러한 요소를 가지고 볼륨 렌더링을 분산 처리하는 프로그램을 구현 분석함으로써 그리드 응용 프로그램 작성 시 성능향상을 위해 고려해야할 요소를 지적하고 대안을 이야기하고자 한다.

1. 서론

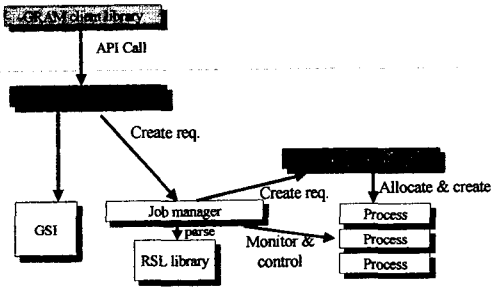
월드 와이드 웹 (WWW)의 개발은 정보공유에 대한 혁명을 가져 왔다고 말할 수 있다. 그러나 이제 이런 단순히 데이터를 공유하는 차원에서 머무는 것이 아니라 상대의 리소스 (CPU, MEMORY, HDD, NETWORK) 까지도 사용할 수 있는 새로운 정보 혁명을 향해 진행되고 있다. 이것이 그리드 컴퓨팅이다. 이런 그리드는 단순히 연구실 안에서 몇 대의 저렴한 컴퓨터를 모아서 하는 단순 병렬 처리를 넘어서 가상 조직(VO. Virtual Organization)을 형성한다. 그리고 그 VO는 나라와 나라, 대륙과 대륙을 묶어 강력한 컴퓨팅 파워를 가진 가상 수퍼 컴퓨터가 되는 것이다.[1] 예를 들어 기존의 과학자들이 많은 데이터 량과 계산 속도를 요하는 실험을 하기 위해 고가의 수퍼 컴퓨터를 이용하곤 했다. 그러나 이것은 경제적, 물리적 제약이 따를 수 밖에 없었다. 그러나 그리드는 이런 문제를 해결하게 된다. 위의 과학자는 이 실험을 위해 고가의 수퍼 컴퓨터를 구입하거나 그 쪽으로 이동해서 연구를 수행하지 않고 자신의 VO에 접속해서 자신의 컴퓨터와 그룹안의 가용 컴퓨터 자원을 묶어서 이런 실험을 할 수 있게 된다. 그리드는 활용 형태에 따라 크게 계산 그리드, 데이터 그리드, 액세스 그리드 로 분류할 수 있다. 본 논문에서는 이런 그리드 중 계산 그리드에 우선 초점을 맞추기로 한다. 우선 이를 위해 가장 많이 사용하는 그리드 미들웨어인 글로버스 툴킷과 MPICH-G2에 대해 2절에서 먼저 설명하고 3,4절에서 이를 바탕으로 분산 처리되어진 그리드 응용 프로그램인 볼륨렌더링의 분석을 통해 성능향상을 위한 고려할 점을 짚어 보고자 한다.

2. 관련 연구

2.1 글로버스 툴킷 (Globus Toolkit 2.0)

글로버스 툴킷은 미국 ANL(Argonne National Lab.)에 의해 개발된 그리드 미들웨어로서 크게 세가지의 서비스를 하게 된다.[2] MDS (Metacomputing Directory Service)로 이루어진 그리드 정보 서비스,GRAM (Globus Resource Allocation Manager)로 이루어진 자원 관리 서비스 그리고 Grid-Ftp로 이루어진 데이터 관리 서비스(Data Management)이다.

* 본 연구는 2002년 한국과학기술정보연구원의 그리드 미들웨어 연구인 어플리케이션을 위한 PSE 기술 과제에서 지원을 받았습니다



[그림 1]

[그림 1]은 각각 노드에 있는 GRAM에게 작업요청을 하는 것을 나타낸다. Client의 작업 요청을 각 노드에 있는 Gatekeeper 데몬이 받고 GSI 인증을 거친 후 job manger를 통해 RSL[9]을 파싱한 후 로컬 자원 관리자에게 프로세스 생성 요청을 하게 된다.[3]

2.2 MPICH-G2

현재 글로버스 툴킷을 사용한 그리드 응용 프로그램에서는 통신 미들웨어로 가장 보편적으로 사용되는 것이 MPICH-G2[6] 이다. 이것은 ANL에서 개발한 MPI v1.1[4][5]을 바탕으로 만든 grid-enabled implementation이다. MPI는 불연속적인 데이터 전송하는 데 편리 점이 있고 이기종간의 통신을 지원한다. 또한 메시지 구별 능력 (Separation),네이밍 프로세스 (Naming Processes)과 커뮤니케이터 (Communicator), 집합적 통신(Collective Communication)등이 있다. MPI는 많은 API들이 있지만 가장 적게는 6개의 기본적 API (MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Send, MPI_Recv, MPi_Finalize)를 가지고 일반적인 통신을 구현할 수 있다.

3. 그리드 응용 프로그램 구현

3.1 분산 볼륨 렌더링 (Distributed Volume Rendering)

모든 process들이 Raycasting을 사용하여 skeleton, engine, brain 세 가지의 볼륨 데이터를 계산 영역을 나누어 계산하고 결과를 master process가 디스플레이 하는 구조로 이루어져 있다.[7][8]

[표 1]

	skeleton	engine	brain
size (MB)	14.0	6.87	10.4

본 실험에서 분산 볼륨 렌더링은 작업 할당 방식에 의해 두 가지 방법으로 구현되었다.

3.1.1. 정적 작업 할당(Static Job Assignment)

```
/*slave, master*/
process start;
MPI_Init();
read data from file into buffer;
rendering
{
 preprocess buffer[rank];
 assignment job area;
 calculation of raycasting;
}
MPI_Gather(..);     //comunication for gathering
                    //for caculated data
End;
```

볼륨 데이타를 읽어 들인 다음, 렌더링을 가속화하기 위한 선행 작업을 한다.[8] 그 후 전체 영역을 프로세스의 갯수로 나누어 고르게 각각의 프로세스들마다 작업할 영역을 할당해 준다. 그리고 각각의 프로세스들은 자신이 작업한 결과를 MPICH-G2의 Collective Operation 중 하나인 MPI_Gather를 사용하여 동시에 마스터 프로세스로 전송한다. MPI_Gather는 Barrier 알고리즘에 의해 모든 프로세스가 MPI_Gather에 도달하기까지 기다리고 있고, 전송을 받는 마스터 프로세스는 모든 전송이 완료되기까지 기다리게 된다.

3.1.2 동적 작업 할당(Dynamic job assignment)

```
/*master*/
loop()
{
  MPI_Recv();
//get the data
  display;
  MPI_Send();
//reassign next job
}
```

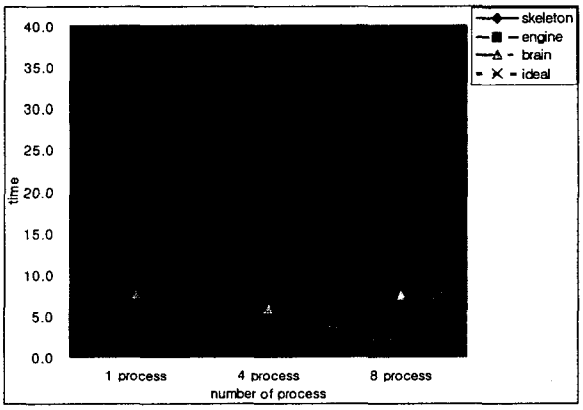
```
/*slave1*/
read data from file;
preprocessing;
rendering;
MPI_Send(); //send result
loop()
{
 MPI_Recv();
        //get the index of
        //a assigned job
 preprocessing;
 rendering;
 MPI_Send();
  //send the caculated data
}
```

```
/*slave 2*/
```

위의 pseudo 코드와 같이 프로그램이 초기화 될 때 일차적인 일이 slave들에게 할당된다. 여기서 작업의 영역 개수는 slave의 개수보다는 많아야 한다. 그리고 master는 루프 상태에서 slave중 하나가 연산을 마치고 send해 올때 까지 MPI_Recv()에서 block된다. send가 오면 그 데이터를 받아 디스플레이하고 데이터를 넘겨준 그 slave에게 다음 일을 다시 할당 시킨다. 그리고 다시 receive상태로 다른 통신 요청을 기다리게 된다. 이렇게 각각의 slave와 일대일로 통신함으로써 통신 시간과 계산 시간이 중복 되어 master가 가장 늦은 slave를 기다리며 block 당하는 것을 줄일 수 있고 slave들은 일이 끝난 상태에서 바로 다시 일을 할당 받음으로 load balancing을 향상시킨다.

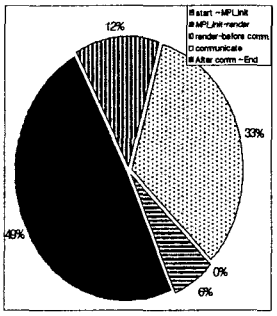
4. 결과 및 분석

4.1 정적 할당 결과 및 분석

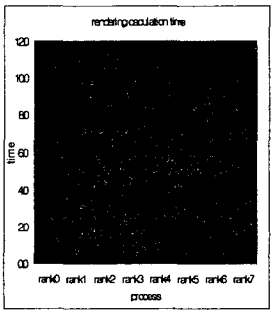


[그래프 1]

원래 이상적으로는 사용되어지는 프로세스의 개수에 대해 선형적으로 시간이 줄어 들어야한다. [그래프 1]을 참조하여 보면 이상적인 그래프의 기울기와 상당한 차이를 보임을 알 수 있다. 그리고 세 가지 data set안에서도 서로 다른 기울기를 나타낸다. brain 볼륨 데이터의 경우는 오히려 8개의 프로세스를 사용할 때 더 시간이 많이 걸리는 현상까지 볼 수 있다. 이러한 현상들은 아래 그래프를 통해 분석할 수 있다.



[그래프 2]



[그래프 3]

[표2]

rank	0	1	2	3	4	5	6	7
통신	5.082	0.090	0.087	0.090	0.086	0.100	0.099	0.099

[그래프 2]는 8개의 프로세스를 사용할 때 skeleton의 master 프로세스의 시간을 분석한 그래프이다. 가장 많은 시간을 차지한 것이 MPI_Init부터 Raycasting전 까지 볼륨 데이타를 오픈하는 과정이다. 이 시간은 단일 프로세스로 연산할 때도 동일하게 소모 되는 시간이다. 그런데 실제 볼륨 렌더링의 핵심인 Raycasting을 계산하게 되는 영역은 불과 전체 시간의 12%밖에 차지 하지 않는다. 나머지 영역 39%는 분산 처리로 인해 생긴 over load가 된다.

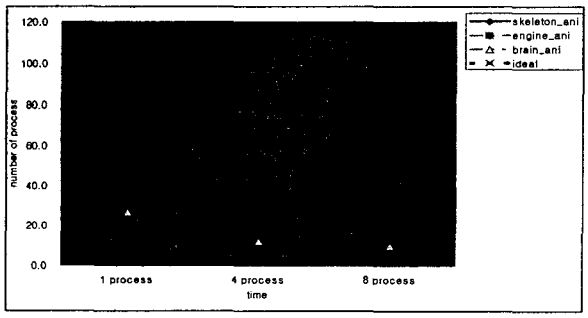
MPICH-G2는 MPI를 초기화하는 MPI_Init 단계에서 글로

버스안의 DUROC을 통해 프로세스들의 초기 동기를 맞추게 된다. 즉 모든 프로세스가 MPI_Init까지 도달하기 전까지는 이미 도달한 프로세스는 wait 상태가 된다. 모두 여기에 도달한 후에 다음을 진행하게 된다. [그래프 2]에서 6%의 시간을 차지하고 있는 요소가 바로 이런 과정에서 생긴 over load가 된다.

위 도표에는 나오지 않았지만 이런 종류의 오버 로드는 또 있다. 처음 실행을 요청하기 위해 글로버스의 클라이언트에서 globusrun을 통해 RSL (Resource Specification Language)[9]를 각 gatekeeper로 보내게 되는데 여기에 들어가는 시간도 본 실험에서는 평균적으로 1.377sec정도 걸리는 것으로 나타났다. 이것은 [그림 1]에서 gram client와 gatekeeper와의 Mutual Authentication 과정에서 소용되는 시간이다.

overload의 나머지 한 요소인 통신은 33%이다. 이렇게 많은 시간이 들어가는 이유는 [그래프 3]과 [표2]을 통해 알 수 있다. 우선[표1]을 보면 통신 시간이 나와 있다. 그런데 마스터인 master 프로세스를 제외하고 나머지 프로세스의 통신 시간은 불과 0.1sec 미만 에 불과하다. 그런데 마스터만 이들의 50배가 넘는 5.02sec가 된다. [그래프 3]은 8개의 프로세스들의 실제 raycasting 연산하는 시간이다. 동일한 사이즈의 영역을 할당했음에도 불구하고 계산 양이 동일하지 않다는 것을 알 수 있다. 이것은 볼륨 렌더링을 가속화하기 위한 방법으로, leaping [8] 이라는 전 처리 과정을 통해 볼륨 데이터 중 불필요한 영역을 제외시키기 때문이다. 그래프에서 [그래프3]에서 가장 계산이 많이 걸린 프로세스와 적게 걸린 프로세스의 시간 차이가 master process의 통신 시간 대부분을 차지하게 된 것이다. 결국 3.1.1. 절에서 언급한 것처럼 정적 할당의 경우 master는 모든 slave가 계산을 완료하기를 기다려야 함으로, 가장 계산 시간이 많은 slave가 끝나는 시각까지 master process의 통신 시간에 포함된 것이다. 그러나 이런 문제를 동적 작업 할당 방식을 통해 상당히 해결할 수 있었다.

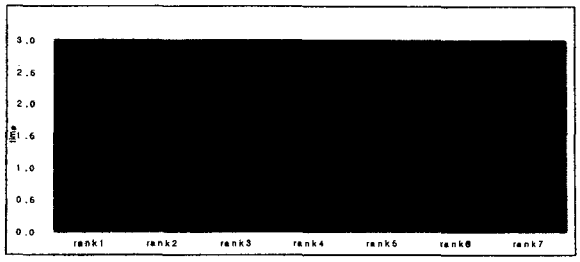
4.2 동적 할당 결과 및 분석



[그래프 4]

[표 3]

rank	1	2	3	4	5	6	7	avg
skeleton	0.0183	0.0164	0.0178	0.0144	0.0181	0.0159	0.0338	0.0192
engine	0.0184	0.0164	0.0178	0.0145	0.0169	0.0234	0.0222	0.0185
brain	0.0177	0.0165	0.0177	0.0165	0.0193	0.0238	0.0249	0.0195



[그래프 5]

[그래프 4]는 [그래프1]과 비교해 볼 때 각 구간의 기울기가 ideal과 상당히 비슷해졌음을 발견할 수 있다. [표3]는 MPI_Send와 MPI_Recv의 일회 통신 시간이다. 여기서 볼륨데이타의 종류의 크기와 상관없이 작은 일정한 통신 시간이 소요됨을 알 수 있다. 또한 이런 통신이 전체 작업동안 36회 반복되게 된다. 따라서 전체 소요 시간 24.804sec 중 통신 시간 0.6912sec로 2.8%로 정적인 부분과 비교해서 절대적으로 감소했다. 또한 [그래프 5]를 통해 load balancing도 개선됨을 알 수 있다.

5. 결론

글로버스와 MPICH-G2를 사용한 분산 응용프로그램 작성에 있어서 이상적인 수치를 내지 못하게 하는 요소로 (1) Mutual Authentication으로 인한 부하,(2) MPI_Init에 의한 프로세스 시작 시 동기 설정에 의한 부하,(3)Collective operation 사용함으로써 load balancing감소로 인한 부하 등이 나타남을 알 수 있었다. 이를 해결하기 위해서는 (3)의 경우는 앞에서 언급한 것과 같이 동적 일 할당을 통해 해결할 수 있다. (1)과 (2)는 사실 완전히 해결 할 수는 없다. 그러나 [그래프 1]과 [그래프 4]에서 동일한 렌더링 방식에서 그 기울기 값이 data set에 따라 달라짐을 볼 수 있다. 즉 처리해야할 data가 큼에 따라 ideal의 기울기에 가까움을 알 수 있다. 이것은 (1),(2)의 요소는 처리할 데이터의 양과 관계없이 일정하게 생기는 부하임으로 데이터의 양이 많아지면 자연적 그 상대적인 영향력이 줄어들게 된다.

따라서 우리는 대용량의 데이터를 동적 할당방식을 통해 그리드 응용 프로그램을 구현함으로써 그 성능을 개선할 수 있다.

참고문헌

[1] Ian Foster "The Anatomy of the Grid",International J. Supercomputer Applications, 15(3), 2001.

[2] Globus Project, http://www.globus.org

[3] Karl Czajkowski, Ian Foster "A Resource Mnagement Architecture for Metacomputing Systems", IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, po. 62-82, 1998

[4] William Gropp "Using MPI -Portable Parallel Programming with the Message-Passing Interface" 2nd Ed. MIT press

[5] William Gropp "MPI-The Complete Reference(The MPI Core) " 2nd Ed. vol -1 MIT Press

[6] MPICH-G2, http://www.hpclab.niu.edu/mpi/g2_body.html

[7] Knitte G ".High-speed volume rendering using redundant block compression Visualization", 1995. Visualization '95. Proceedings., IEEE

[8] Sung-Up. Jo and C.S. Joeng, "A Parallel Volume Visualization Using Extended Space Leaping Method" Lecture Notes in Computer Science, August 2000, pp. 76-83

[9] RSL, http://www-fp.globus.org/gram/rsl.html