ISBN 978-89-6211-609-0

ParaView 병렬 가시화 시스템을 활용한 거대 과학 데이터의 가시화

(Visualization of Huge Scientific Dataset using ParaView Parallel Visualization System)

이 중 연 (jylee@kisti.re.kr)

한 국 과 학 기 술 정 보 연 구 원
Korea Institute of Science & Technology Information

목차

표 차례

그림 차례

수식 차례

1. 서론

계산과학이란 이론상으로 믿을 만한 예측을 하기에는 너무 복잡한 현상, 또는 실험실에서 수행하기에는 너무 위험하거나 비용이 많이 드는 현상 등을 슈퍼컴퓨터와 같은 고성능 컴퓨터를 이용해서 모의 실험하여 실제 실험을 대체하는 것을 말한다. 이러한 모의실험 후에는 매우 방대한 결과가 생성되는데, 이런 결과들은 일반인은 물론 전문가들조차 이해하기 어려운 숫자의 나열인 경우가 대부분이다. 이런 숫자의 나열을 보고 이것이 의미하는 바를 바로 알아볼 수 있는 사람은 거의 없을 것이고, 전문가라 할지라도 이해하는데 많은 시간이 걸릴 것이다. 그러나 이 수치 데이터를 그림으로 표현하는 경우에는 이 수치의 의미를 이해하는데 걸리는 시간은 획기적으로 단축될 수 있다. 이런 데이터는 2차원, 혹은 3차원으로 구성되어 있고 때로는 4차원 이상의 다차원으로 이루어져 있을 때도 있다. 이렇게 복잡한 데이터를 2차원, 또는 3차원 영상으로 나타내어 쉽게 데이터의 특성 및 의미를 이해할 수 있게 해주는 것을 과학적 가시화라고 한다.

컴퓨터를 이용한 과학적 가시화는 물리, 화학, 천문, 대기, 기계, 항공, 생명 등 과학 기술 분야 전반에 걸쳐서 널리 활용되어지고 있다. 과학자들은 컴퓨터를 이용, 자신들의 연구 결과를 2차원이나 3차원 영상으로 표현하여 계산 결과를 보다 빠르게, 그리고 보다 포괄적으로 이해할 수 있게 되었다. 또, 가시화 결과를 다른 사람들과 공유함으로써 자신의 연구 결과를 타인이 쉽게 이해할 수 있도록 하여 다른 과학자들과의 공동 연구에 활용하기도 한다.

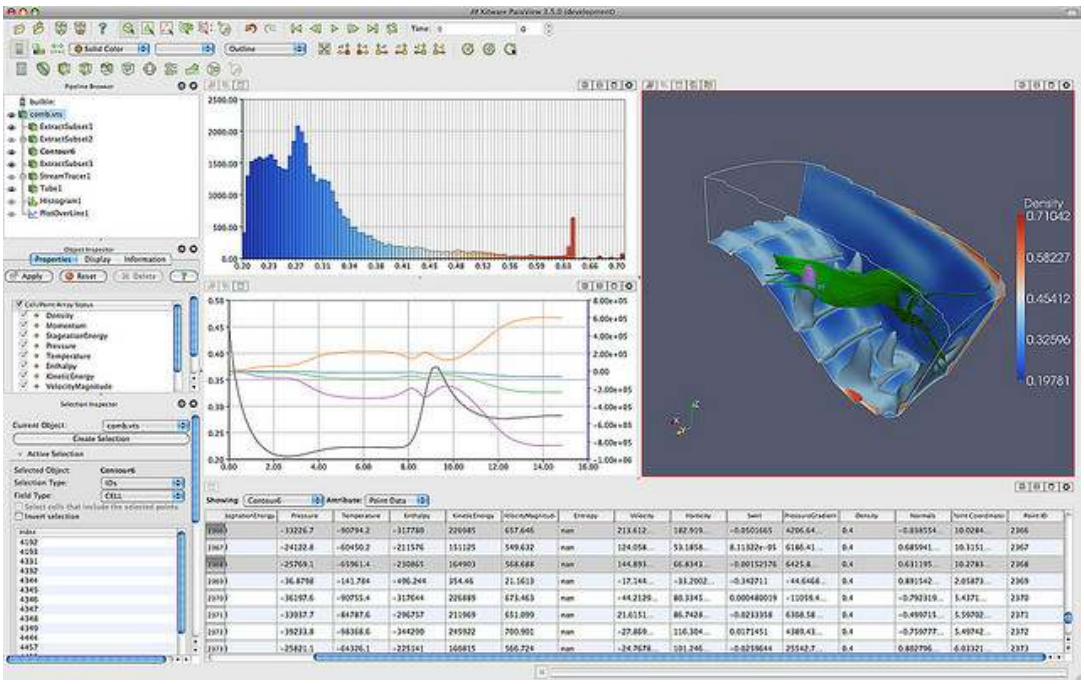
2000년대 들어 슈퍼컴퓨터를 비롯한 계산 자원의 성능이 비약적으로 발전하여 이를 통한 과학 데이터의 크기 또한 급격히 커지고 있다. 이로 인해 일반 PC를 이용한 과학 데이터의 가시화가 매우 어려워지고 있다. 이러한 문제를 극복하고자 여러 해결 방안이 제시되고 있는데, ParaView는 대용량 데이터를 여러 노드의 클러스터 컴퓨터를 이용, 빠르게 가시화할 수 있도록 하는 오픈 소스 프로그램이다[1]. 본 기술문서에서는 ParaView를 이용해서 테라 스케일의 거대용량 로터 시뮬레이션 데이터를 가시화하기 위한 방법에 대해 살펴본다. 가시화 분야를 로터 시뮬레이션 데이터에 한정지었지만 2장에서 설명하는 ParaView에 대한 설명은 ParaView가 지원하는 모든 응용 분야의 데이터에 대해서도 적용이 가능하다.

2. ParaView

가. 개요

ParaView는 Kitware사에서 개발한 가시화 툴로써 동사에서 개발한 VTK를 기반으로 한 오픈소스 가시화 프로그램이다. 대용량 데이터를 빠르게 가시화하기 위해 병렬 가시화를 가능하게 하였고 그 이름에도 병렬(Parallel)을 뜻하는 Para를 붙였다. ParaView는 대형 데이터에 대해 interactive한 가시화를 가능하게 하기 위해 LOD(level of detail) 모델을 생성하고 이를 이용해서 원격 가시화를 가능하게 했다. 또, ParaView는 공유 메모리 컴퓨터, 분산 메모리 클러스터 등 여러 종류의 병렬 컴퓨팅은 물론 일반적인 PC까지 지원한다.

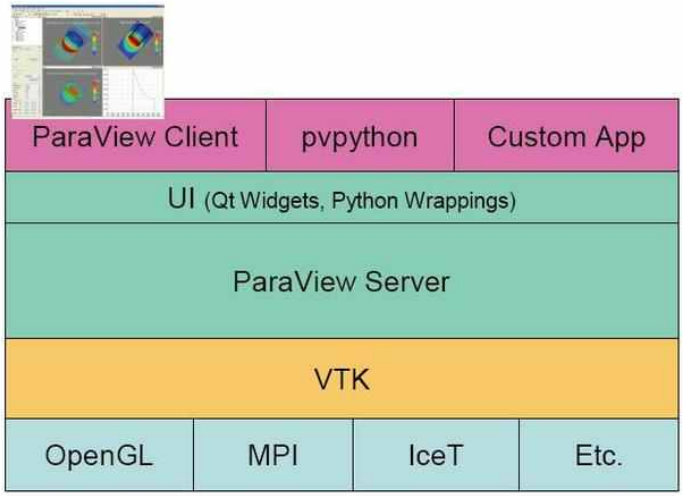
ParaView는 여러 컴퓨팅 노드를 이용한 병렬 가시화뿐만 아니라 타일 디스플레이를 이용한 가시화 또한 지원한다. 이는 IceT 라이브러리를 기반으로 가능했다. 그러나 ParaView는 GPU 컴퓨팅을 전혀 지원하지 못하는 단점이 있는데, 이는 VTK의 단점이기도 하다.



[그림 2-1] ParaView

나. 구조

ParaView는 그림 2-2와 같은 구조를 가진다.



[그림 2-2] ParaView 구조

그림에서 보듯이 ParaView는 OpenGL, MPI, IceT, VTK 등을 기반으로 구성된다. OpenGL은 표준 그래픽스 라이브러리이고 MPI는 병렬 처리를 위한 메시지 패싱 라이브러리, IceT는 타일 디스플레이를 위한 Sort-Last 컴포지션 라이브러리 그리고 VTK는 ParaView의 핵심인 가시화 라이브러리이다. ParaView Server는 실제 가시화 작업이 진행되는 서버로써 데이터 서버와 렌더 서버로 나뉠 수 있다. 마지막으로 유저 인터페이스는 두 가지가 제공되는데, 하나는 Qt 기반의 GUI 인터페이스이고 나머지 하나는 pvpython이라 부르는 Python wrapper이다.

다. 가시화 모드

ParaView는 “독립(stand alone)”, “클라이언트/서버”, “클라이언트/데이터 서버/렌더 서버”의 3가지 모드를 지원한다[2,3]. 본 기술문서에서 다루고자 하는 데이터는 1.2TB의 거대용량으로 단일 컴퓨팅 노드에서 처리가 불가능하므로 병렬 가시화를 지원하는 클라이언트/서버 모드나 클라이언트/데이터 서버/렌더 서버 모드로 처리해야 한다. ParaView에서 가시화하는 데이터가 병렬 파일 형식인 경우 여러 대의 ParaView 서버를 이용해서 읽으면 ParaView가 자동으로 이를 각 서버에 분산, 작업함으로써 클라이언트로의 부하를 줄일 수 있을 뿐만 아니라 보다 빠른 속도로 가시화를 수행할 수 있다.

클라이언트/서버 또는 클라이언트/데이터 서버/렌더 서버 모드로 ParaView를 구동하기 위해서는 먼저 ParaView GUI와는 별개인 ParaView 서버인 pvserver 실행파일을 MPI 작업으로 여러 클러스터 노드들에 걸쳐 실행시켜야 한다. 이렇게 실행된 pvserver들은 ParaView GUI에서 별도의 명령으로 연결할 수 있다.

1) 클라이언트/서버 모드

클라이언트/서버 모드는 별도의 ParaView 서버에서 데이터 로드(load), 필터링(filtering), 렌더링(rendering) 작업들을 다 수행하고 ParaView GUI에서는 이를 디스플레이만 하도록 하는 모드이다. 데이터의 크기가 단일 노드에서 실행하기에는 너무 클 경우 여러 노드에서 서버를 병렬로 실행함으로써 작업량이 많은 여러 작업들을 빠르게 처리하도록 해서 전체적인 가시화 성능을 높일 수 있다. 이때, 만약 ParaView GUI의 렌더링 성능이 충분히 빠른 경우 서버에서는 필터링 작업까지만 수행하고 생성된 폴리곤을 GUI가 실행되는 클라이언트로 전송 받아서 GUI에서 렌더링되도록 할 수도 있다. 이는 ParaView GUI에서 설정이 가능하다. 클라이언트/서버 모드로 ParaView를 구동하기 위해서는 먼저 pvserver를 MPI 작업으로 구동해야 한다.

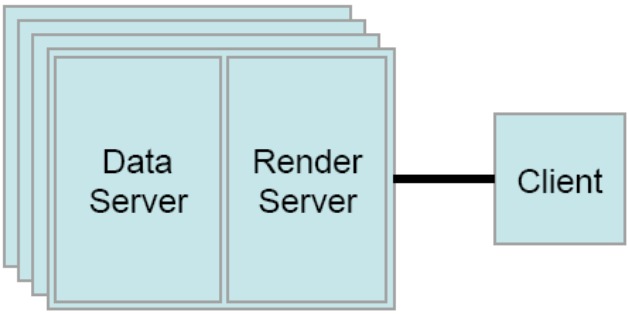
```
mpirun_rsh -ssh -np 90 -hostfile hostfile /usr/local/ParaView/bin/pvserver
```

[그림 2-3] pvserver MPI 구동 명령

물론, 여기서 -np 뒤에는 실행하고자 하는 작업 노드의 개수를 넣고 -hostfile 뒤에는 작업 노드의 host 이름을 넣으면 된다. 이때, 각 pvserver와 ParaView GUI는 포트번호 11111로 연결되는데, 이때 다른 번호의 포트를 이용하고자 할 경우에는 pvserver 명령 뒤에 -sp=포트번호 를 넣으면 된다. 이렇게 기본이 아닌 포트 번호를 이용할 경우에는 ParaView GUI에서 서버에 연결할 때에도 따로 설정해야 한다.

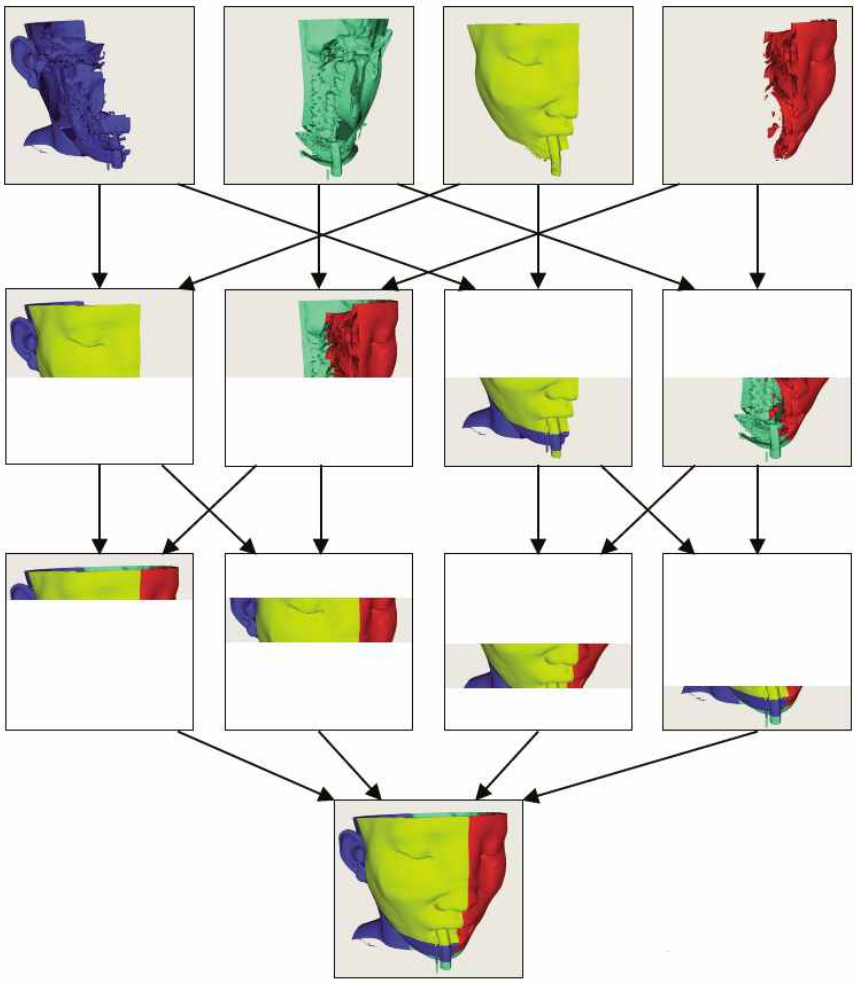
그림 2-4는 클라이언트/서버 모드로 구동하는 ParaView의 구조이다. 그림에서 Data Server/Render Server로 되어 있는 부분이 ParaView 서버인데, 그림에서와 같이 ParaView 서버는 하나의 서버로 구동되지만 내부에는 데이터 서버와 렌

더 서버로 구분된다. ParaView 서버가 구동되는 컴퓨터의 그래픽스 성능이 충분하지 못하거나 여러 가지 이유로 그 컴퓨터의 그래픽스 하드웨어를 사용하지 못할 경우, 또는 데이터 서버에서 생성한 폴리곤 데이터의 크기가 작아서 클라이언트에서 가시화하는 것이 더 빠를 경우에는 렌더 서버는 동작하지 않고 데이터 서버로만 동작한다.



[그림 2-4] ParaView 클라이언트/서버 모드 구조

만약 데이터 서버에서 가시화한 폴리곤 데이터가 굉장히 크다면 이 폴리곤 데이터를 클라이언트로 전송해서 렌더링하도록 하면 두 가지 문제가 발생한다. 하나는 폴리곤 크기가 클라이언트의 그래픽스 성능을 초과해서 매우 느린 렌더링 성능을 보일 수 있다는 것이고 다른 하나는 서버에서 클라이언트로 폴리곤을 전송하는데 시간이 오래 걸린다는 것이다. 따라서 이런 경우에는 데이터 서버와 물리적으로 붙어있는 렌더 서버에서 렌더링까지 수행한 후 결과 이미지를 클라이언트로 전송하는 것이 낫다. 물론 만약 사용자가 데이터를 지속적으로 움직이면 렌더 서버에서 계속 렌더링을 수행, 이미지를 매 프레임마다 전송해야 하는 문제가 발생한다. 반면에 만약 데이터 서버에서 가시화한 폴리곤 데이터의 크기가 그다지 크지 않다면 위와 같은 렌더 서버에서의 지속적인 이미지 전송 문제를 피하기 위해 폴리곤 데이터 자체를 클라이언트로 전송한 후 클라이언트에서 렌더링하도록 할 수 있다. 이런 경우에는 처음 데이터 서버에서 생성한 폴리곤을 전송하는데 시간이 걸릴 수 있지만 한번 전송이 된 뒤에는 더 이상의 네트워크 작업이 필요 없기 때문에 더 나은 성능을 보일 수 있다. ParaView는 데이터 서버가 가시화한 폴리곤 데이터를 렌더 서버와 클라이언트 중 어느 쪽에서 렌더링할 것인지를 사용자가 선택할 수 있도록 한다. 폴리곤 데이터의 크기에 따라서 설정한 크기보다 크면 렌더 서버에서, 작으면 클라이언트에서 처리하도록 하는 것인데, 이는 ParaView GUI의 Settings에서 설정이 가능하다.

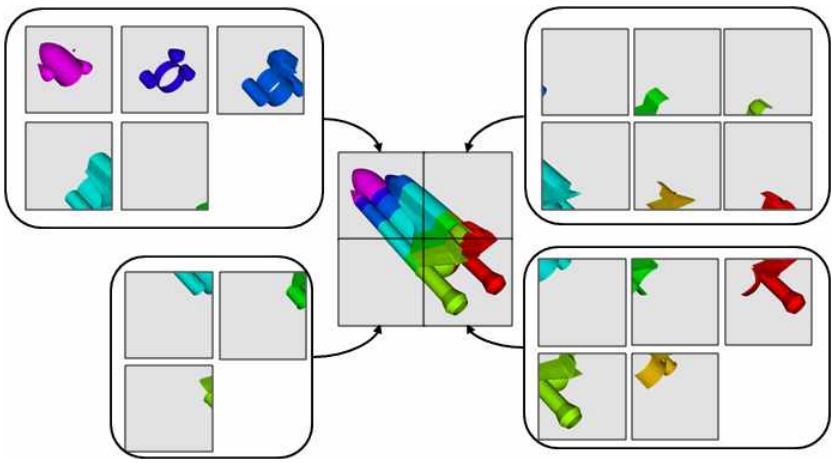


[그림 2-5] ParaView 병렬 가시화

2) 타일 디스플레이 모드

만약 pvserver가 실행되는 노드가 타일 디스플레이에 연결되어 있다면 각 노드에서 가시화한 결과를 타일 디스플레이에 띄울 수 있다. 이는 IceT(Image Composition Engine for Tiles) 라이브러리를 이용해서 가능하다. IceT 라이브러리는 고성능의 sort-last 병렬 렌더링 라이브러리로 샌디아(Sandia) 국립 연구소에서 개발했다. 이 라이브러리는 여러 노드에서 병렬로 렌더링한 이미지를 타일 디스플레이에서 디스플레이할 수 있도록 했는데, binary swap compositing을 지원해서 빠른 성능을 보인다. 그림 2-6은 IceT의 타일 디스플레이로의 이미지 컴포지션 예이다. 총 8개의 노드에 데이터가 분산된 상태에서 타일 디스플레이에 컴포지션을 하는 것이다. 분산된 각 노드별로 서로 다른 색깔로 데이터가 색칠되어 있는데, 각 노드에서 렌더링한 결과는 실제 디스플레이된 타일을 담당하는 노드

로 전송되고 각 타일 디스플레이를 담당하는 노드는 전송된 결과의 깊이 정보를 이용해서 컴포지션한다.



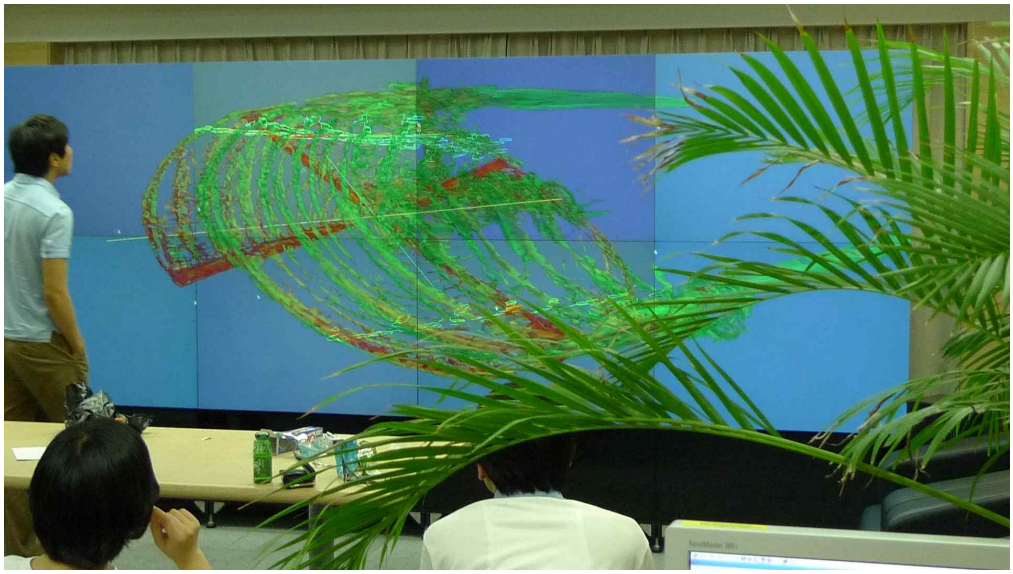
[그림 2-6] IceT를 이용한 타일 디스플레이에서의 컴포지션

pvserver를 타일 디스플레이 모드로 실행하기 위해서는 다음과 같이 타일 디스플레이에 대한 정보를 주어야 한다.

```
mpirun_rsh -ssh -np 90 -hostfile hostfile /usr/local/ParaVi
ew/bin/pvserver -tdx=4 -tdy=2 -display :0.0
```

[그림 2-7] pvserver 타일 디스플레이 모드 구동 명령

여기서 tdx는 타일 디스플레이의 가로(width) 방향의 디스플레이 개수, tdy는 세로(height) 방향의 디스플레이 개수로 위 예제에서는 4 × 2의 타일 디스플레이이다. 이때, hostfile의 host는 row major로 저장되어 있어야 한다.



[그림 2-8] KISTI 가시화실의 타일 디스플레이에서의 ParaView

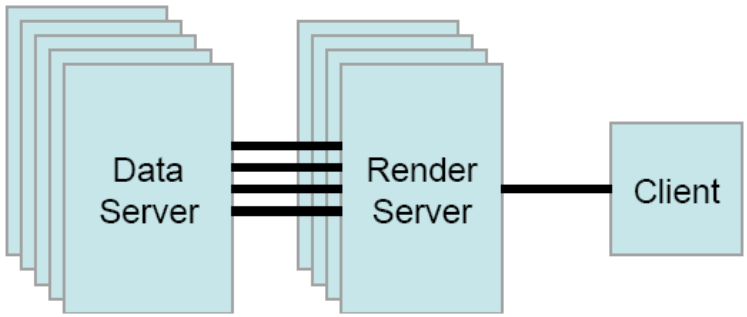
3) 클라이언트/데이터 서버/렌더 서버 모드

ParaView 서버는 데이터 서버와 렌더 서버를 물리적으로 나누어서 실행할 수도 있다. 여기서 데이터 서버는 데이터를 로드하고 필터링을 통해 폴리곤을 생성하는 서버이고 렌더 서버는 데이터 서버에서 생성한 폴리곤을 렌더링하는 서버이다. 만약 특정 노드에는 대용량 스토리지가 연결되고 멀티 코어의 빠른 CPU가 장착되어 있을 경우 이들 노드들은 데이터 서버로서 적합하다. 그러나 빠른 렌더링을 위한 GPU가 장착되어 있지 않다면 렌더 서버로는 적합하지 못한 것이다. 반대로 어떤 노드에 빠른 GPU와 고해상도 모니터가 장착되어 있다면 렌더 서버로 적합하지만 대용량 저장장치나 빠른 CPU가 없다면 데이터 서버로는 적합하지 못하다. ParaView에서는 이와 같이 특성이 다른 여러 노드들을 데이터 서버와 렌더 서버로 분리하여 실행하도록 함으로써 시스템 성능에 최적화된 작업을 수행할 수 있도록 한다. 이때, 렌더 서버의 개수는 반드시 데이터 서버의 개수보다 적거나 같아야 한다. 또, 렌더 서버에는 타일 디스플레이가 장착되어 데이터 서버에서 필터링한 데이터를 타일 디스플레이에 렌더링될 수 있도록 한다.

```
mpirun_rsh -ssh -np 90 -hostfile hostfile /usr/local/ParaVi
ew/bin/pvdataserver
 mpirun_rsh -ssh -np 90 -hostfile hostfile /usr/local/ParaV
iew/bin/pvrenderserver -tdx=4 -tdy=2 -display :0.0
```

[그림 2-9] pvdataserver와 pvrenderserver의 MPI 구동 명령

그림 2-10은 ParaView를 데이터 서버/렌더 서버/클라이언트로 나누어서 실행할 경우의 전체 구조이다.



[그림 2-10] ParaView 데이터 서버/렌더 서버/클라이언트 모드 구조

데이터 서버는 그래픽스 하드웨어를 필요로 하지 않기 때문에 일반적인 계산 자원에서도 구동이 가능하다. 데이터가 매우 큰 경우에 계산 자원에서 가시화 자원으로 데이터를 옮기는 것은 매우 오랜 시간을 필요로 할 뿐만 아니라 일반적으로 규모가 큰 계산 자원에서 생성한 데이터를 가시화 자원에서 처리하기에는 크기가 너무 큰 경우가 많다. 이러한 이유로 계산 자원 자체를 ParaView 데이터 서버로 활용해서 대용량의 원시 데이터를 가시화 자원으로 전송하지 않고 데이터 서버에서 한번 가시화 처리를 한 보다 크기가 작은 폴리곤 데이터만을 가시화 자원(렌더 서버)로 전송해서 렌더링한다면 보다 효율적으로 가시화 작업을 처리할 수 있다.

따라서 ParaView는 렌더 서버의 개수를 반드시 데이터 서버 개수보다 적거나 같다록 제한한다. 렌더 서버의 개수가 데이터 서버 개수보다 많더라도 대부분의 경우 데이터 서버에서 다루는 데이터가 렌더 서버에서 다루는 데이터 보다 크기가 훨씬 크기 때문에, 이는 일반적인 환경에서는 비효율적인 구성이 될 수 밖에 없

다.

이러한 데이터 서버/렌더 서버 간의 폴리곤 데이터 전송을 위해 ParaView에서는 M to N 병렬 폴리곤 데이터 전송 알고리즘을 사용했다. 이 알고리즘은 M개의 프로세스에서 가지고 있는 데이터를 N개의 프로세스에 재분배하는 알고리즘으로 두 개의 $\log_2 P$ (P는 프로세스 개수) 스테이지로 처리가 가능하다. 이 알고리즘은 두 부분으로 나뉘는데, 첫 번째 부분은 폴리곤 데이터를 균등하게 배분하는 것이고 두 번째 부분은 균등하게 배분한 폴리곤 데이터를 최종 N개의 프로세스에 분배하는 것이다. 자세한 알고리즘은 다음과 같다.

1. 초기 배분 상태 A를 균등 배분 상태로 재분배하는 전송 스케줄 생성
2. 균등 배분 상태에서 최종 배분 상태 B로 재분배하는 전송 스케줄 생성
3. 전송자(send)와 수신자(receive)를 바꾸고 스케줄 순서를 거꾸로 해서 2번 스케줄의 역 스케줄 생성
4. 데이터에 스케줄 1을 적용
5. 역(逆) 스케줄 적용

이 알고리즘에서 4, 5번째 스텝이 각각 $\log_2 P$ 스테이지를 가진다.

3. 데이터

본 기술 문서에서 가시화하고자 하는 데이터는 총 1.2TB의 용량을 차지하는 테라 스케일 로터 시뮬레이션 데이터로 모두 98개의 타입 스텝으로 이루어져있다.

로터 시뮬레이션 데이터는 로터가 회전할 때 블레이드에 가해지는 pressure와 force, moment 그리고 블레이드 인근 유동의 velocity, vorticity 등을 계산한 것으로 처음 계산 코드로부터 생성된 파일은 tecplot 형식이다. 이러한 tecplot 데이터는 ASCII 파일 형식으로 총 4TB의 용량을 차지하는데, 최종적으로 이진(binary) 형식의 VTK 데이터로 저장할 경우 1.21TB로 줄어든다.

가. 데이터 개요

이 로터 시뮬레이션 데이터는 총 98개의 타임 스텝으로 이루어져있다. 각 스텝의 각도 차는 1도인데, 실제 시뮬레이션은 0.2도 단위로 수행했으나 데이터 파일의 크기가 너무 커져서 1도 단위로만 저장했다. 따라서 데이터 파일의 numbering은 5 단위로 되어 있다. 한 타임 스텝은 총 200개의 유동장 블록(flow field block)과 16개의 블레이드 블록(blade block)으로 구성되어 있다. 전체 타임 스텝의 데이터 크기는 약 1.2TB이고 한 타임 스텝의 데이터 크기는 약 12.61GB이다.

나. 블레이드 데이터

- 격자 데이터

블레이드 데이터는 구조화된 격자(structured grid) 구조로 구성되어 있다.

[표 3-1] 블레이드 데이터 격자 구조 정보

Block ID	블록 개수	격자 크기	스텝 개수
짝수	8	1 × 79 × 81	98
홀수	8	1 × 79 × 49	98
합계	16	-	98

- Attributes

[표 3-2] 블레이드 데이터 attribute 정보

이름	형식	타입
Postion	벡터	float
Pressure	스칼라	double

- 데이터 크기

[표 3-3] 블레이드 데이터 크기 정보

Block ID	블록 개수	point 개수	스텝 개수	point 당 attribute 크기	전체 크기
짝수	8	6,399	98	20Bytes	96.59MB
홀수	8	3,871	98	20Bytes	57.89MB
합계	16		98	20Bytes	154.47MB

다. 유동장 데이터

- 격자 데이터

유동장 데이터는 구조화된 격자(structured grid) 구조로 구성되어 있다.

[표 3-4] 유동장 데이터 격자 구조 정보

Block ID	블록 개수	격자 크기	스텝 개수
타입 1	32	4 × 381 × 381	98
타입 2	152	5 × 381 × 381	98
타입 3	16	49 × 103 × 81	98
합계	200		98

- Attributes

[표 3-5] 유동장 데이터 attribute 정보

이름	형식	타입
Postion	벡터	float
Blanking	스칼라	double
Density	스칼라	double
Pressure	스칼라	double
Vorticity	스칼라	double
Q-criteria	스칼라	double
Om	벡터	double
Velocity	벡터	double

- 데이터 크기

[표 3-6] 유동장 데이터 크기 정보

Block ID	블록 개수	point 개수	스텝 개수	point 당 attribute 크기	전체 크기
타입 1	32	580,644	98	100Bytes	169.58GB
타입 2	152	725,805	98	100Bytes	1006.9GB
타입 3	16	408,807	98	100Bytes	59.70GB
합계	16		98	100Bytes	1.21TB

라. 데이터 변환

기존의 legacy VTK 파일의 경우 ParaView를 병렬 모드로 실행한다고 해도 데이터가 적절히 각 노드에 분산되어 저장되지 않고 중복하게 저장되어 병렬 가시화의 장점을 살리지 못한다. 특히, 본 기술문서에서 다루는 로터 데이터의 경우 전체 크기가 1.2TB로 이를 모든 노드에 중복해서 로드하는 것은 현재 환경에서는 불가능하다. 또, 기존의 legacy VTK 파일을 그대로 읽어 들일 경우 사용자는 각 파일 별로 하나씩 차례대로 데이터를 읽어야 한다. 앞 장에서 설명했듯이 본 기술문서에서 가시화하고자 하는 데이터는 각 타임 스텝 당 216개의 블록이고 총 98스텝으로 모두 합쳐 21,168개의 블록으로 구성되어 있다. 각 블록은 모두 별개의 파일로 저장되어 있으므로 모두 21,168번의 데이터 로드를 해야 하는 것이다. ParaView는 새로운 병렬(parallel - 또는 partitioned) VTK 형식을 지원하는데, 이 형식으로 저장된 데이터는 병렬 모드의 ParaView에서 읽을 경우 자동으로 각 노드에 분산해서 데이터를 저장하고 단 한 번의 데이터 로드 명령으

로 모든 데이터를 읽어 들이는 것이 가능하다. 또, 시변환 데이터를 ParaView에 로드하고자 할 때에는 병렬 VTK 형식과 유사한 Paraview 데이터 형식으로 저장해야 하는데, 이 형식을 이용해야만 데이터 세트를 시변환 데이터로 인식해서 ParaView에서 애니메이션이 가능해진다.

1) 병렬 VTK 데이터 형식

병렬 VTK 데이터 형식은 XML 형식으로 되어 있으며 실제로는 일반 직렬(serial) VTK 데이터 파일을 병렬로 구성, 읽기 위한 메타 데이터이다. Legacy VTK 데이터 파일에 비해 병렬 VTK 데이터 파일은 여러 가지 기능을 제공한다[4].

1. 데이터 스트리밍
2. 병렬 I/O
3. 데이터 압축
4. big endian / little endian 바이트 지원
5. 랜덤 액세스

이러한 병렬 VTK 데이터는 다음과 같은 여러 종류의 파일 확장자로 구분된다.

[표 3-7] 병렬 VTK 데이터 확장자

데이터 타입	설명
이미지 데이터 (.vti)	vtkImageData
폴리곤 데이터 (.vtp)	vtkPolyData
직교 격자 데이터 (.vtr)	vtkRectilinearGrid
구조화된 격자 데이터 (.vts)	vtkStructuredGrid
비구조화된 격자 데이터 (.vtu)	vtkUnstructuredGrid
병렬 이미지 데이터 (.pvti)	병렬 vtkImageData
병렬 폴리곤 데이터 (.pvtp)	병렬 vtkPolyData
병렬 직교 격자 데이터 (.pvtr)	병렬 vtkRectilinearGrid
병렬 구조화된 격자 데이터 (.pvts)	병렬 vtkStructuredGrid
병렬 비구조화된 격자 데이터 (.pvtu)	병렬 vtkUnstructurdGrid

```
<?xml version="1.0"?>
<VTKFile type="Collection" version="0.1" byte_order="LittleEndian" compres
sor="vtkZLibDataCompressor">
  <Collection>
    <DataSet timestep="0" part="0" file="flo_1800/flo_1800_0.vts"/>
    <DataSet timestep="0" part="1" file="flo_1800/flo_1800_1.vts"/>
    <DataSet timestep="0" part="2" file="flo_1800/flo_1800_2.vts"/>
    <DataSet timestep="0" part="3" file="flo_1800/flo_1800_3.vts"/>
    <DataSet timestep="0" part="4" file="flo_1800/flo_1800_4.vts"/>
    <DataSet timestep="0" part="5" file="flo_1800/flo_1800_5.vts"/>
    <DataSet timestep="0" part="6" file="flo_1800/flo_1800_6.vts"/>
    <DataSet timestep="0" part="7" file="flo_1800/flo_1800_7.vts"/>
    <DataSet timestep="0" part="8" file="flo_1800/flo_1800_8.vts"/>
    <DataSet timestep="0" part="9" file="flo_1800/flo_1800_9.vts"/>
    <DataSet timestep="0" part="10" file="flo_1800/flo_1800_10.vts"/>
    <DataSet timestep="0" part="11" file="flo_1800/flo_1800_11.vts"/>
    <DataSet timestep="0" part="12" file="flo_1800/flo_1800_12.vts"/>
  </Collection>
</VTKFile>
```

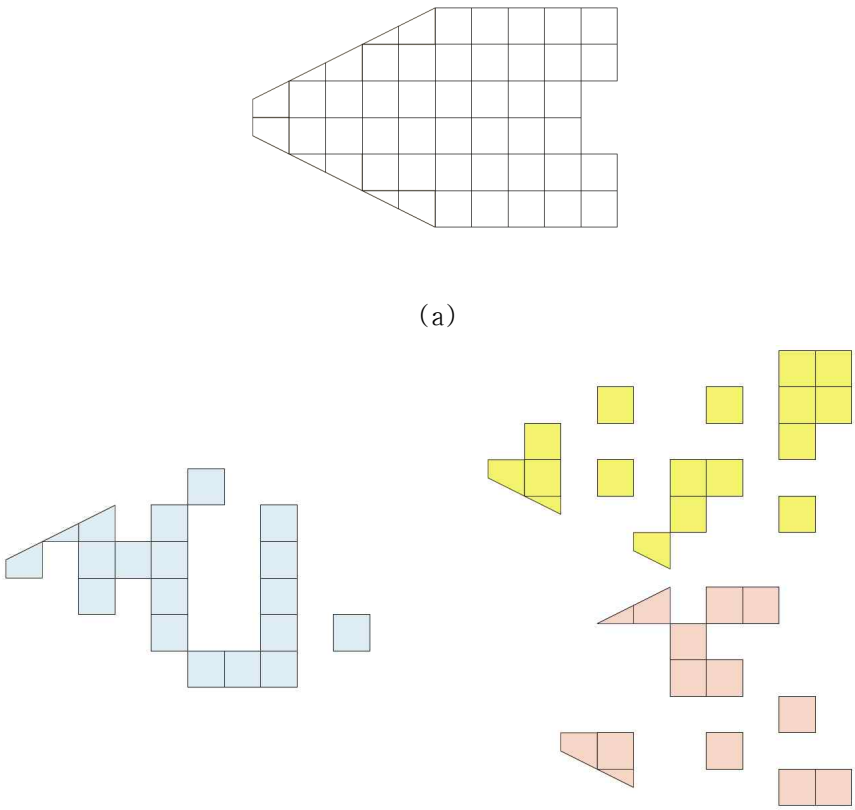
[그림 3-1] 병렬 VTK 데이터 예

2) 병렬 VTK 데이터 분산 로딩

병렬 VTK 데이터를 여러 개의 ParaView 서버로 로드할 때, 데이터가 구조화된 격자 구조(structured grid)일 경우 ParaView가 자동으로 데이터를 분산, 로드하도록 한다. 데이터를 인접한 블록끼리 묶어서 로드하도록 하면 실제 가시화 때 일부 노드에 작업이 몰릴 가능성이 많아 작업이 균등화되지 못해서 작업 효율이 떨어질 수 있다. 반면에 너무 분산시켜서 로드하도록 하면 병렬 가시화 작업에서 반드시 필요한 고스트 셀(ghost cell)이 많아져서 중복되어 로드되는 데이터가 많아지게 되므로 메모리 사용이 비효율적으로 된다. 따라서 ParaView에서는 데이터를 랜덤하게 분산하되 인접한 데이터가 너무 많이 나뉘어 고스트 셀이 많아지지 않도록 자동으로 적절히 데이터를 분산시킨다. 이는 데이터가 구조화된 격자 구조일 경우 가능한 것으로 비구조화된 격자(unstructured grid)일 경우에는 사용자가 수동으로 설정해야 한다.

그림 3-2는 임의의 구조화된 격자 구조의 데이터에 대해 3개의 노드로 분산 로드하는 그림이다. (a)는 원 데이터 구조이고 (b)는 (a)의 데이터가 색깔에 따라

서로 다른 노드에 로드된 그림인데, 데이터가 적절히 분산되었음을 알 수 있다.



(a)

(b)

[그림 3-2] 구조화된 격자 데이터의 분산 로드

ParaView에는 입력된 데이터를 자동으로 재분할하는 필터가 있다. 이 필터의 이름은 D3인데, 임의의 형식의 VTK 데이터를 받아서 무조건 비구조화된 격자 구조(unstructured grid)로 변환한 뒤 분할한다. 각 분할된 데이터들은 최대한 프로세스의 로드가 분산되도록 동일한 개수의 셀을 가지도록 되어있고 대부분의 가시화 알고리즘에서 필요로 하는 고스트 셀을 자동으로 생성함은 물론이다. 이 필터는 "Distributed Data Decomposition"의 약자로 D3라는 이름이 붙었으며, 실재로는 VTK의 vtkDistributedDataFilter를 사용해서 구현되었다.

이렇게 분할한 데이터는 비구조화된 격자 구조를 가지는데, ParaView나 VTK에서 비구조화된 격자 구조의 데이터를 기반으로 하는 알고리즘들은 대부분 그 속도가 매우 느리다. 그 이유는 전체 셀에서 포인트 위치을 구하는 연산에 매우 많

은 시간이 필요하기 때문이다. VTK에서는 이러한 점을 해결하기 위해서는 BSP tree나 kd-tree 등을 사용해야 하며 일부 알고리즘에서는 실제로 사용하도록 구현되었다. 특히, D3 필터등을 이용해서 데이터가 여러 노드들에 분할되어 저장되어있는 경우 가시화 알고리즘에 따라 인접 블럭과 데이터를 교환해야 할 필요가 있는데, 이를 빠르게 처리하기 위해 전체 블럭에 대한 BSP tree가 생성되어 빠르게 인접 블럭을 찾을 수 있도록 할 수 있다.

3) 병렬 I/O

위에서 설명했듯이 ParaView에서 병렬로 데이터를 읽기 위해서는 데이터를 병렬 파일 형식으로 저장해야 한다. ParaView의 병렬 파일 리더는 이 병렬 파일에 기술된 정보에 따라 데이터를 분할해 각 렌더링 노드에 올리게 된다. 이때, 각 렌더링 노드에 작업을 균등하게 배분하기 위해서 각 분할 데이터의 크기를 최대한 동일하게 나누어야 하는데, 이는 D3 필터를 이용하면 된다.

ParaView에서는 D3 필터 등을 이용해서 데이터를 분산하기는 하지만 실제로 데이터를 읽을 때는 모든 노드에서 동시에 모든 데이터를 다 읽은 후 필요 없는 부분은 버리는 형식으로 진행된다. 따라서 데이터 로딩 시의 속도는 매우 느려질 수 밖에 없다. 이러한 문제는 PV_USE_TRANSMIT 환경변수를 True로 저장하면 일정부분 해결될 수 있는데, 이 환경변수를 설정하면 전체 노드들 중 첫 번째 노드에서 데이터를 일괄적으로 읽고 각 노드에서 필요로 하는 데이터를 전송해 주게 된다. 하지만 이 경우에는 실제로는 데이터를 병렬로 읽는 것도 아니고 렌더링 노드가 많아질수록 데이터를 읽는 노드에 부하가 많이 걸려서 전체적인 속도는 여전히 느려지는 문제가 발생한다.

한편, ParaView에서는 MPI-I/O를 이용한 병렬 I/O를 부분적으로 지원하는데, 이는 HDF5의 MPI-I/O 기능을 이용해서 구현되었다. 따라서 ParaView에서 병렬 I/O를 구현하기 위해서는 데이터가 HDF5와 호환되어야 하고 파일 시스템 역시 HDF5의 MPI-I/O와 호환되는 GPFS나 Lustre 등의 파일 시스템이어야 한다.

4. 가시화 결과

가. 가시화 내용

일반적으로 로터 시뮬레이션 데이터에서는 vortex 영역의 가시화가 중요하다. Vortex는 유동 데이터에서 유동이 급격하게 회전하는 부분을 말하며 회전하는 정도를 vorticity라고 한다. 일반적으로 vortex 영역은 vorticity가 크지만 vorticity가 크다고 전부 vortex인 것은 아니다. 많은 관련 연구자들이 vortex의 정의를 찾기 위해 노력하고 있지만 명확한 정의는 아직까지 만들어지지 않았다. 본 기술문서에서 다루는 로터 시뮬레이션 데이터에서는 다음과 같은 방법으로 vorticity와 vortex 영역을 생성했다.

1) Vorticity

Vorticity는 vector의 회전하려는 정도를 나타내는 값으로 수학적으로 curl로 표현된다. 이는 vector가 *(u, v, w)*라고 할 때, 다음과 같은 수식으로 표현할 수 있다[5].

$$vorticity = curl\, v = \nabla \times v = \begin{vmatrix} i & j & k \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ u & v & w \end{vmatrix} = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}\right) i + \left(\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}\right) j + \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right) k$$

[수식 4-1] Vorticity 계산 수식

여기서, vorticity vector는 $\left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)$ 이고, vorticity magnitude는 $\sqrt{\left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}\right)^2 + \left(\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)^2}$ 이다.

2) Vortex 영역

Vortex의 영역을 찾는 가장 간단한 알고리즘은 vorticity의 크기가 큰 영역을 vortex로 정의하는 것이다. Vorticity는 velocity의 curl로 정의가 된다. 즉, $\nabla \times v$ 이다. 또, 이는 지역 유동의 회전을 표현하기도 한다. 물론 vortex 영역의 vorticity 크기는 항상 크다. 하지만 역은 성립하지 않는다. 즉, vorticity가 크다고 항상

vortex인 것은 아니다. 따라서 vorticity가 큰 영역을 vortex로 정의하는 것은 계산량이 적은 장점이 있지만 상대적으로 부정확하다. Vortex 영역을 찾는 또 다른 알고리즘으로는 vorticity 대신 helicity를 사용하는 것이다. Helicity란 vorticity vector를 velocity vector에 투영한 것이다. 즉, $(\nabla \times v) \cdot v$ 이다. 이 방법을 사용하면 velocity에 수직인 vorticity component는 모두 제거된다. 또 다른 Vortex 영역을 찾는 알고리즘으로 Jeong과 Hussain이 제안한 방법이 있다[6]. 이 방법은 전체 유동장에서 vorticity가 strain보다 큰 부분을 vortex라고 정의한다. 계산식은 다음과 같다.

$$Q = \Omega^2 - S^2 = -\frac{1}{2}\left(\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial z}\frac{\partial w}{\partial x} + \frac{\partial v}{\partial z}\frac{\partial w}{\partial y}\right)$$

[수식 4-2] Q-Criteria 계산 수식

이 방법은 Q-Criteria라는 이름으로 알려져 있다.

3) 가시화 내용

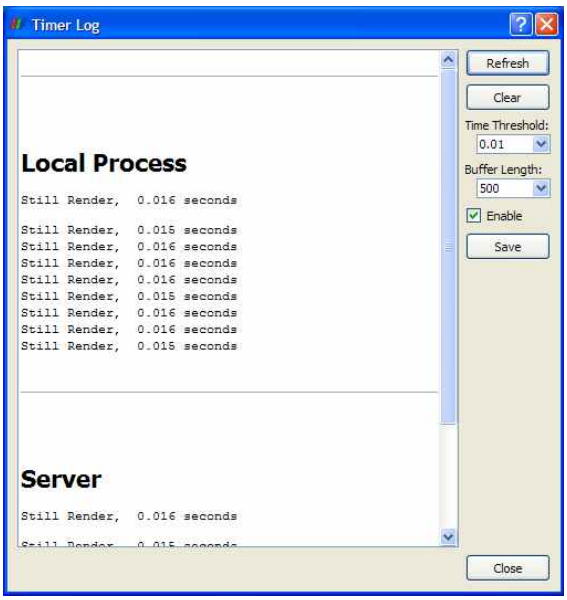
본 기술문서에서 가시화하고자 하는 데이터에 대해서도 Q-Criteria 방식으로 vortex 영역을 정의하고, vorticity가 높은 블레이드 양 끝단에서 cutting plane을 배치, vorticity의 contour line을 표시함으로써 블레이드 주변의 vorticity 분포를 가시화하고자 했다. 이 데이터에서 Vortex 영역을 가시화하기 위한 Q 값은 0.001로 기존 coarse나 fine grid 데이터에서의 0.0001이나 0.00001 보다 큰 Q 값을 사용했다. 이는 데이터가 너무 거대해서 0.001보다 작은 값으로 가시화를 할 경우 너무 많은 영역이 표시되어 Vortex 영역을 구분하기가 어려웠기 때문이다.

3장에서 언급했듯이 본 기술문서에서 가시화하고자 하는 데이터는 총 98 스텝의 시변환 데이터이다. 각 스텝은 1도 간격으로 되어 있는데, 데이터의 특성상 90도 간격으로 동일한 현상이 반복되기 때문에 처음 90도만 가시화하면 된다.

각 타임 스텝마다 동일한 크기의 Q 값으로 Q-Criteria iso-surface를 생성하고 동일한 블레이드 끝단에 cutting plane을 배치하여 시간에 따른 vortex와 vorticity의 변화를 살펴보았다. 가시화하고자 하는 로터 블레이드 데이터의 블레이드는 시간에 따라 회전하기 때문에 cutting plane의 위치도 블레이드를 따라 회전하도록 했다.

나. 가시화 성능

테라 스케일의 로터 데이터에 대해 이를 로드(load)하는데 걸리는 시간과 0.001의 Q값으로 iso-surface를 생성하는데 걸리는 시간 그리고 블레이드 끝단에 cutting plane을 하는 시간을 단일 노드와 8개의 타일 디스플레이 모드, 90개의 pvserver를 이용한 서버/클라이언트 모드를 사용하여 각각 측정했다. 측정 방법은 ParaView GUI의 Timer Log 메뉴를 이용했다. 이 메뉴는 ParaView에서 실행한 모든 작업의 수행 시간 기록을 보여준다. ParaView는 VTK를 기반으로 개발되었고 내부 프로세스는 대부분 VTK 필터를 사용하는데, Timer Log는 이러한 VTK 필터의 수행 시간을 로컬 프로세스(local process)와 서버 프로세스 별로 보여준다.



[그림 4-1] Timer Log 메뉴

[표 4-1] 테라 스케일 로터 데이터에 대한 ParaView 가시화 성능 (단위 : 초)

		Load	Cutting Plane	Iso-Surface
단일 스텝	Single node	221.1	11.93	14.24
	8 nodes	76.5	1.14	6.85
	90 nodes	18.2	1.79	10.11
전체 스텝	Singe node	N/A	232	235
	8 nodes	N/A	82.1	89.43
	90 nodes	N/A	23.87	31.1

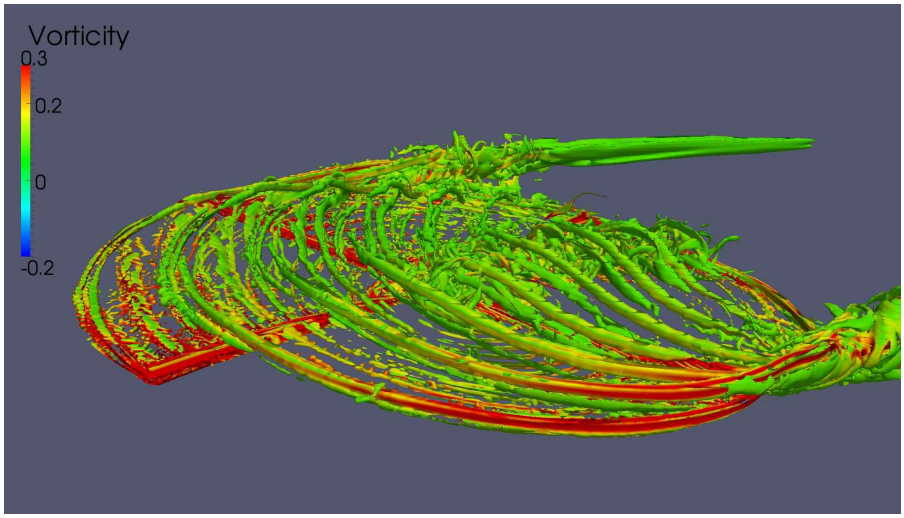
표 4-1은 테라 스케일 로터 데이터에 대한 ParaView 가시화 성능을 측정한 결과이다. 단일 스텝은 한 타임 스텝에 대한 가시화 성능 측정 결과이고 전체 스텝은 90개 스텝에 대한 애니메이션 성능 측정 결과이다. Single node는 ParaView를 독립 모드로 구동한 것이고 8 nodes는 8대의 타일 디스플레이 노드를 이용한 타일 디스플레이 모드, 90 nodes는 90대의 ParaView 서버를 이용한 클라이언트/서버 모드이다.

ParaView가 전체 스텝에 대해 미리 로드를 한 후 가시화할 것을 기대했는데 각 스텝별로 로드를 해서 전체 스텝에 대한 로드 시간은 측정하지 못했다. 또, 애니메이션 시 각 타임 프레임별로 그때 그때 데이터를 로드했기 때문에 전체 스텝에 대한 시간 측정은 결국 데이터 로드 시간 + 실제 가시화 알고리즘 수행 시간이 되어 전체적인 성능이 떨어졌다.

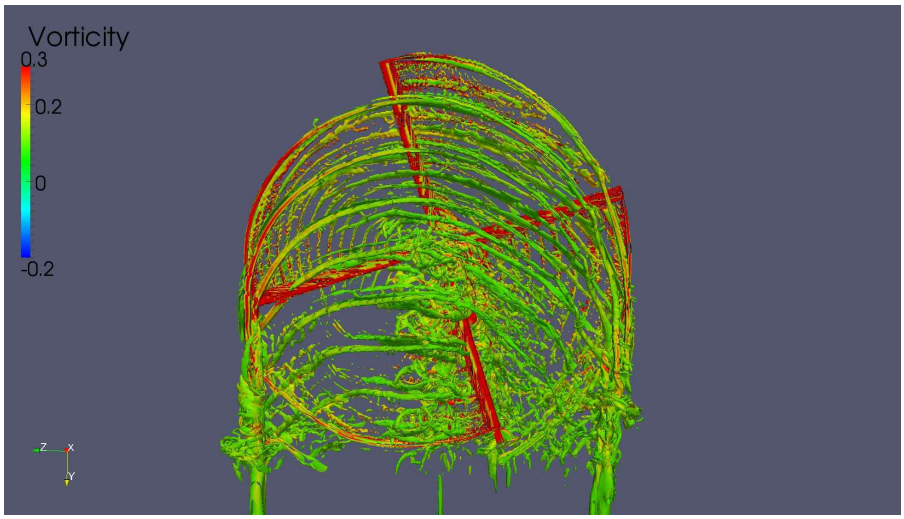
8대의 컴퓨팅 노드를 이용한 타일 디스플레이 모드가 90대의 노드를 이용한 클라이언트/서버 모드보다 더 빨랐다. 이는 8대의 노드를 이용한 경우 각 노드 간 통신이 그리 많이 필요하지 않지만 90대의 노드를 이용한 경우에는 노드 간 통신이 많이 필요하여 오랜 시간이 걸렸던 것으로 판단된다.

다. 가시화 결과

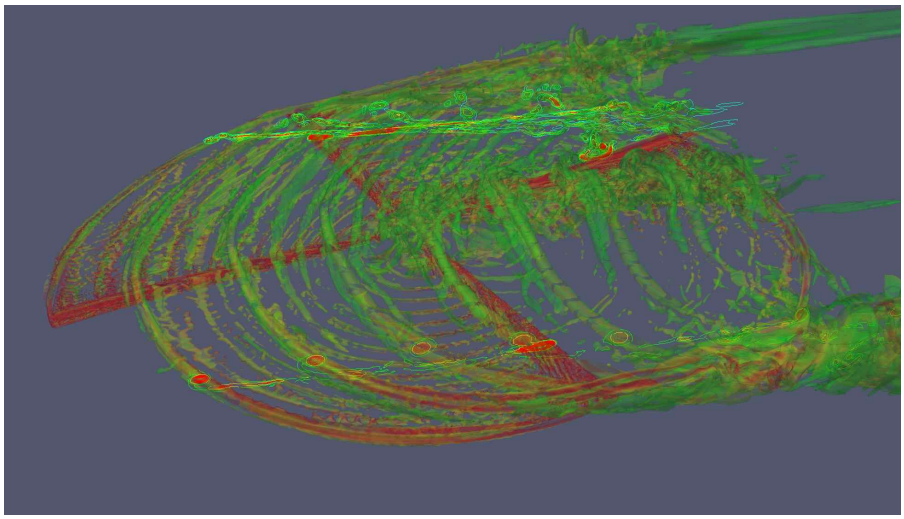
로터 시뮬레이션 데이터에 대한 가시화 결과는 그림 4-2와 같다. 그림 4-2의 (a)와 (b)는 0.001의 Q값에 대한 Q-Criteria Iso-surface를 수행하고 그 surface에서의 vorticity값에 대해 색을 넣은 것이다. (c)는 동일한 Q-Criteria Iso-surface를 반투명하게 처리하고 블레이드의 양 끝단에 cutting plane을 배치한 뒤 vorticity contour line을 그린 그림이다. 그림 4-3은 이렇게 가시화한 결과를 타일 디스플레이에 가시화한 사진이다.



(a)

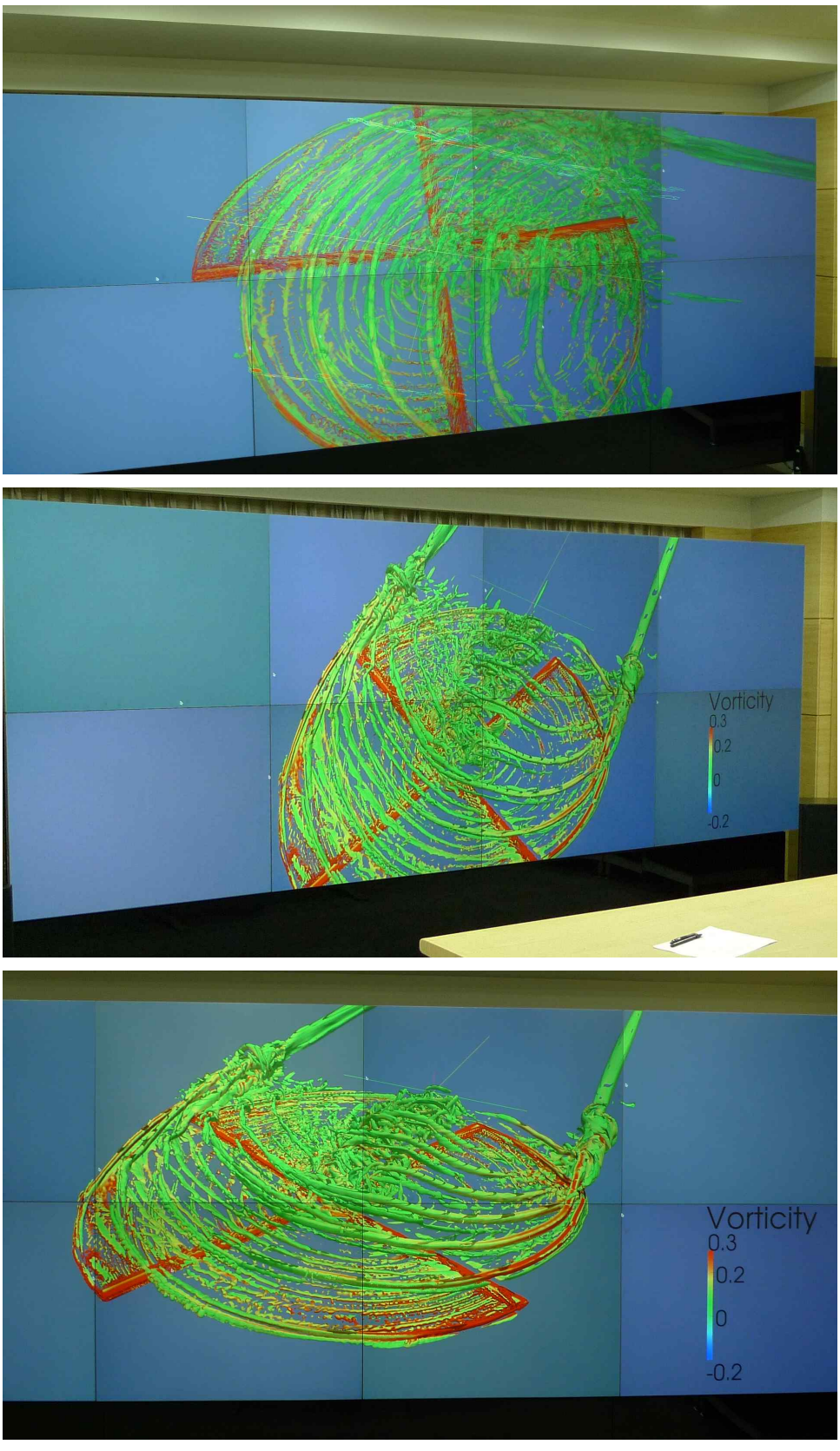


(b)



(c)

[그림 4-2] Q-Criteria Iso-surface 및 vorticity contour line 가시화



[그림 4-4] 타일 디스플레이 가시화

5. 결론

본 기술문서에서는 테라 스케일의 로터 시뮬레이션 데이터를 ParaView를 이용, 가시화하는 방법에 대해 기술했다. ParaView는 대용량 데이터를 여러 노드의 클러스터 컴퓨터를 이용, 빠르게 가시화할 수 있도록 하는 오픈 소스 프로그램으로 본 프로그램을 이용, 대용량의 로터 시뮬레이션 데이터를 가시화할 수 있었다.

총 1.2TB 크기의 시변환 로터 시뮬레이션 데이터를 빠르게 로드하고 가시화할 수 있도록 원래 tecplot 형식이었던 데이터를 병렬 VTK 형식으로 변환하였으며 이를 90대로 구성된 클러스터를 이용해서 가시화했다.

단일 스텝의 데이터에 대해서는 각 가시화 알고리즘에 따라 1~3초의 빠른 속도로 가시화하는 것이 가능했으나 여러 스텝의 데이터에 대한 애니메이션의 경우에는 기대만큼의 빠른 속도를 보이지 못했다. 이는 ParaView가 시변환 데이터에 대해 전체 스텝을 모두 읽고 처리하는 방식이 아닌, 현재 가시화해야 하는 단일 스텝에 대해서만 읽어서 처리하는 일종의 out-of-core 방식을 사용하기 때문인 것으로 판단된다. 데이터가 크지 않을 때는 필요할 때에만 읽어 들이는 방식이 가능하지만 본 기술문서에서 다룬 테라 스케일의 데이터에 대해서는 한 타임 스텝을 읽는데 18초 가량이 걸렸기 때문에 전체 성능에 영향을 주어 문제가 되었다.

ParaView는 매우 뛰어난 성능의 병렬 가시화 소프트웨어로 로터 시뮬레이션 데이터 뿐만 아니라 다양한 종류의 과학 데이터에 대한 병렬 가시화를 지원한다. 또한, 현재도 지속적으로 개발, 발전되고 있는 소프트웨어로, 최근 기반으로 하는 VTK에서 시변환 데이터를 기본으로 지원하기 위한 프로젝트가 진행 중이므로 향후 시변환 데이터에 대해서도 빠른 성능을 보일 수 있을 것으로 기대된다.

6. 참고문헌

[1] Kitware Paraview, http:://www.paraview.org

[2] Andy Cedilnik, Berk Geveci, Kenneth Moreland, James Ahrens, and Jean Favre. "Remote Large Data Visualization in the ParaView Framework." In Eurographics Parallel Graphics and Visualization 2006, pg. 163-170, May 2006

[3] Amy Henderson Squillacote, "The ParaView Guide", Kitware, Inc.

[4] "The VTK User's Guide", Kitware, Inc.

[5] Wikipedia, http://en.wikipedia.org/wiki/Vorticity

[6] J. Jeong and F. Hussain, "On the Identification of a Vortex", Journal of Fluid Mechanics, 285, pp.69−94, 1995