KIPS Tr. Software and Data Eng.
Vol.5, No.6 pp.273~282 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.6.273

GLOVE: Distributed Shared Memory Based Parallel Visualization Tool for Massive Scientific Dataset

Joong-Youn Lee† · Min Ah Kim†† · Sehoon Lee† · Young Ju Hur†

ABSTRACT

Visualization tool can be divided by three components – data I/O, visual transformation and interactive rendering. In this paper, we present requirements of three major components on visualization tools for massive scientific dataset and propose strategies to develop the tool which satisfies those requirements. In particular, we present how to utilize open source softwares to efficiently realize our goal. Furthermore, we also study the way to combine several open source softwares which are separately made to produce a single visualization software and optimize it for realtime visualization of massiv espatio-temporal scientific dataset. Finally, we propose a distributed shared memory based scientific visualization tool which is called "GLOVE". We present a performance comparison among GLOVE and well known open source visualization tools such as ParaView and VisIt.

Keywords : Scientific Visualization, Parallel Visualization, Distributed Shared Memory, Visualization System, Open Source Software

GLOVE: 대용량 과학 데이터를 위한 분산공유메모리 기반 병렬 가시화 도구

이 중 연† · 김 민 아†† · 이 세 훈† · 허 영 주†

요 약

가시화 도구는 데이터 입출력, 시각적 변환, 상호작용적인 렌더링의 세 구성요소로 구분할 수 있다. 본 논문에서는 거대용량의 과학 데이터를 실시간으로 가시화하기 위해 가시화 도구의 세 구성요소에 대한 요구사항을 분석, 정의하고 이를 만족시키기 위한 방안을 제시하고자 한다. 특히, 효율적인 가시화 도구의 개발을 위해 공개 소프트웨어 도구를 최대한 활용하고자 하였으며, 서로 다른 용도로 개발된 각 공개 소프트웨어 도구를 통합하여 하나의 가시화 도구로 개발하는 방안과 시공간적인 과학 데이터의 실시간 가시화를 위한 최적화 방법에 대해 논한다. 이를 통해 분산공유메모리 기반의 과학 데이터 병렬 가시화 도구인 GLOVE를 제안하였으며, 유동해석 분야 과학 데이터를 이용한 실험을 통해 GLOVE와 다른 데이터 가시화 소프트웨어와의 성능을 비교 분석했다.

키워드 : 과학적 가시화, 병렬 가시화, 분산공유메모리, 가시화 시스템, 공개 소프트웨어 도구

1. 서 론

컴퓨팅 자원의 성능 향상과 다양한 과학 분야에서의 데이터 획득방법의 발전(CT 및 MRI 스캐너, 위성 촬영 등)으로 인해 과학 데이터의 크기는 테라 바이트 단위를 넘어 페타 및 엑사 바이트 단위로 증가하고 있다. 단일 PC의 처리 능력을 뛰어넘는 거대용량의 데이터가 지속적으로 생산되는 반면, 이를 분석하고 시각화하기 위한 데이터 가시화 및 해석 도구의 성능은 상대적으로 정체되어 있다. 대용량 데이터를 실시간으로 분석하기 위해 클러스터를 이용한 병렬 컴퓨팅, GPU 컴퓨팅 기술을 가시화 도구에 접목시키고자 하는 노력이 지속되고 있지만 많은 연구자들은 여전히 병렬처리를 지원하지 않는 기존의 가시화 도구를 사용하고 있다. 이것은 기존의 도구들이 분산/병렬 컴퓨팅 환경에서 원활하게 동작하도록 하기 위해 사용자가 설정해야 할 과정이 매우 복잡하여 비숙련자가 사용하기에는 어려울 뿐만 아니

† 정 회 원 : 한국과학기술정보연구원 가시화기술개발실 선임연구원
†† 정 회 원 : 한국과학기술정보연구원 가시화기술개발실 실장
Manuscript Received : May 3, 2016
Accepted : May 17, 2016
* Corresponding Author : Joong-Youn Lee(jylee@kisti.re.kr)

라, 그 성능이 기대에 못 미치는 경우가 많기 때문이다. 데이터 가시화 및 분석 작업의 특성 상 가시화 도구는 실시간 렌더링 및 상호작용적인 조작을 지원해야만 하는데, 대용량의 데이터에 대해서는 이러한 성능을 보이기가 어려운 것이 하나의 원인인 것으로 판단된다.

본 논문에서는 데이터 가시화 도구를 데이터 입출력(Data I/O), 데이터의 시각적 변환(Visual Transformation) 그리고 상호작용적인 렌더링(Interactive Rendering)의 세 가지 요소로 구성하고, 각 구성요소 별 요구사항을 제안하고자 한다. 단일 PC의 처리 능력을 뛰어넘는 대용량 데이터에 대해 실시간 가시화를 구현하기 위해서는 각 요소에서 모두 높은 성능을 만족하여야 하는데, 한정된 시간에 한정된 인력으로 모든 요소의 성능을 극대화시키는 것은 쉽지 않은 문제이다. 이러한 문제를 해결하고 대용량 과학 데이터를 위한 실시간 가시화 도구를 효율적으로 개발하기 위해 기존에 개발된 공개 소프트웨어 도구를 활용하는 방안을 사용하고자 한다. 이를 위해 다음과 같은 접근방법을 사용했다

1. 각 구성요소별 요구사항 도출
2. 각 요구사항을 만족하는 공개 소프트웨어 도구 선정
3. 각 공개 소프트웨어 통합 구현
4. 실시간 가시화 응용에 최적화

특히, 일반적인 PC의 처리능력을 벗어나는 크기의 대용량 데이터를 빠르게 처리하기 위해 분산 및 병렬 컴퓨팅을 활용한다. 분산 메모리 구조의 병렬 컴퓨팅 환경에서는 높은 성능을 보장하면서 데이터를 서로 공유하고 작업을 분배하는 것이 어려운 문제인데, 본 논문에서는 분산 메모리 환경에서 각 분산 노드의 지역 메모리를 연동하여 하나의 공유 메모리처럼 사용하도록 하는 분산공유메모리(Distributed Shared Memory) 기술을 사용하여 이 문제를 해결하고자 했다. 공개 분산공유메모리 라이브러리를 사용해서 전체 컴퓨팅 노드의 지역 메모리를 묶은 뒤, 이를 하나의 거대한 공유메모리로 활용하도록 했으며, 최종적으로 가시화 및 실시간 렌더링 개발도구와 연동하여 통합하는 방안을 제시한다. 또한, 가시화 알고리즘을 그 특징에 따라 크게 두 종류로 구분하고 각 종류에 따라 서로 다른 데이터 공급 정책을 사용하여 I/O 성능을 최적화하는 방안에 대해 논한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 관련 연구를 살펴보고, 3장에서는 대용량 실시간 가시화 소프트웨어를 개발하기 위한 세 가지 구성요소와 각 구성요소별 요구사항을 설명한다. 4장에서는 본 논문에서 제안하는 대용량 실시간 가시화 소프트웨어인 GLOVE(GLObal Visualization Environment)의 전체 구조를 논하고, 5장, 6장 및 7장에서는 GLOVE의 세 구성요소인 데이터 관리자, 병렬 가시화 엔진 그리고 렌더링 클라이언트의 설계 및 구현방안에 대해 차례로 설명한다. 8장에서는 실제 유체역학 시뮬레이션 데이터를 이용한 실험 결과를 보이고, 마지막으로 9장에서 토의 및 결론으로 논문을 마무리 한다.

2. 기존 연구

2.1 대용량 데이터 가시화 도구

ParaView는 Kitware 사에서 개발한 병렬 가시화 도구로써 VTK(Visualization Tool Kit)[1]를 기반으로 개발된 오픈소스 소프트웨어이다. ParaView는 다양한 분야의 과학 데이터에 대한 범용 가시화 기능을 지원하며 클라이언트/서버 구조로 병렬 및 분산 환경에서의 고성능 가시화를 제공하여 많은 사용자를 확보하고 있다[2]. 그러나 가시화 알고리즘에 대한 병렬화만 지원할 뿐 데이터 관리는 병렬화하지 않아 정적 데이터 분산(Static Data Distribution)만을 제공하여 부하 균형(Load Balancing)이 제대로 이루어지지 않는 문제가 있다. 특히, 벡터 데이터 가시화의 경우에는 이러한 문제가 두드러진다. VisIt은 DOE (Department of Energy) 산하의 ASCI(Advanced Simulation and Computing Initiative)에서 개발한 소프트웨어로, 주로 테라 스케일 이상의 시뮬레이션 결과를 가시화할 용도로 개발되었다. VisIt 역시 ParaView와 마찬가지로 클라이언트/서버 구조이고 분산 환경에서의 고성능 가시화를 지원한다[3]. VisIt은 벡터 데이터 가시화 등 특정 가시화 기법에 대해서는 동적 데이터 분산을 지원하지만, 사용자가 사용할 알고리즘이 이를 지원하는지를 파악하고 초기 실행 시에 미리 사용할 것인지를 지정하지 않으면 동적 데이터 분산이 이루어지지 않는다. 만약 이를 지원하지 않는 알고리즘에 사용할 경우 잘못된 가시화 결과가 도출될 수도 있다. 사용자 환경 측면에서도 VisIt은 병렬 가시화를 설정하고 소프트웨어를 설치하기 위한 절차가 매우 복잡하며, 병렬 처리에 대한 지식이 없는 비숙련자가 사용하기 쉽지 않은 인터페이스를 가지고 있다. Ensight는 유체 동역학 데이터의 처리를 위한 전문적인 상용 가시화 도구로써 가시화와 렌더링을 위한 병렬 서버를 별도로 구성하여 대용량 데이터를 처리할 수 있다[4]. 그러나 Ensight는 유체 동역학 데이터의 처리만 가능할 뿐 범용 가시화가 불가능한 단점이 있을 뿐만 아니라 고가의 상용 소프트웨어로써 연구자들이 접근하기 쉽지 않다는 문제가 있다.

2.2 분산공유메모리

성능 및 속도가 중요한 HPC 분야는 전통적으로 병렬 머신의 최대 성능을 얻기 위해 프로그래머가 직접 제어하는 프로그래밍 모델이 주를 이루었다. 그러나 멀티코어, 매니코어, GPGPU, 클러스터 등으로 복잡해진 하드웨어 아키텍처는 프로그래머가 NUMA(Non-Uniform Memory Access) 등의 많은 이슈를 고려하게 하였고, HPC 알고리즘 및 코드들은 관리가 힘들 정도로 복잡해지고 있다. 이로 인해 프로그래머의 생산성이 점차 중요해지면서 최근 대부분의 병렬 프로그래밍 모델은 생산성과 성능을 함께 고려하는 PGAS

(Partitioned Global Address Space) 모델을 채택하고 있다[5]. PGAS 모델은 MPI와 같이, 각 프로세스가 개별 메모리를 가지고 서로 메시지를 교환하는 메시지-패싱 모델과 OpenMP로 대표되는, 다수의 스레드가 메모리를 공유하는 공유메모리 모델의 특징을 모두 가진다. 물리적으로 분리되어 있는 메모리들을 묶어 하나의 메모리처럼 사용할 수 있도록 전역 주소 공간(Global Address Space)을 제공하여 프로그래머의 생산성을 향상시키는 한편, 로컬 및 원격 데이터에 대한 접근을 구분하여 프로그래머가 직접 성능을 최적화하고, 확장성(Scalability)을 확보할 수 있도록 한다.

90년대 후반, 기존 프로그래밍 언어에 PGAS 모델을 통합한 CAF(Co-Array-Fortran)[6], UPC(Unified Parallel C)[7], Titanium[8]가 등장했다. CAF는 Fortran, UPC는 C, Titanium은 Java를 확장했다. 2004년 페타플롭 시스템을 만들기 위해 추진된 DARPA의 HPCS(High Productivity Computing Systems) 프로젝트의 결과로, Cray의 Chapel[9], IBM의 X10[10], Sun Microsystems의 Fortress[11]가 탄생했다. 이들은 기존 언어를 확장하지 않고 새로운 언어로 설계되었다. Global Arrays Toolkit(GA)[12]는 1994년 PNNL (Pacific Northwest National Laboratory)에서 개발된 고수준 라이브러리이다. PGAS 언어들과는 달리 컴파일러에 의존하지 않고, 다른 프로그래밍 언어와 함께 동작할 수 있는 라이브러리로 구현되어 기존의 코드에 쉽게 접목할 수 있는 장점이 있다. GA는 양자화학, 분자동역학, 계산유체역학, 대기과학, 천체물리학 등 다양한 계산과학 분야에서 활용되고 있을 뿐만 아니라, 과학 데이터의 시각화 분야에서도 사용되고 있다. PiCEIS[13]는 실시간 이미지 처리를 위해 GA를 활용해 복잡한 메시지-패싱 대신 간단한 구현으로 분산 데이터를 관리했고, CUMULVS[14]는 가시화 및 시뮬레이션 Steering을 위한 데이터 관리를 위해 GA를 도입했다. Zippy 프레임워크[15]는 GPU 클러스터에서 MPI 대신 GA를 활용하여 sort-last 볼륨 렌더링, 등치면(iso-surface) 추출 및 렌더링, 유동 시뮬레이션 및 가시화를 수행했다.

3. 대용량 실시간 가시화 도구의 구성요소

실시간 가시화 도구는 가시화 작업의 특징으로 인해 비록 대용량의 데이터를 다룬다고 하더라도 인터랙티브한 반응을 보여야 한다. 본 논문에서는 실시간 가시화 도구를 크게 세 구성요소로 나누고, 각 구성요소 별로 거대용량의 데이터에 대해 인터랙티브 성능을 달성하기 위한 요구 사항을 제시한다. Fig. 1은 본 논문에서 제안하는 대용량 실시간 가시화 도구의 구성요소들이다. 대용량 실시간 가시화 도구는 데이터 입출력(Data I/O), 시각적 변환(Visual Transformation), 상호작용적인 렌더링(Interactive Rendering)으로 구분될 수 있다. 각 구성요소는 순차적으로 실행되어야 하며, 이전 구성요소의 성능이 느리면 후속 구성요소의 성능에도 영향을 미친다. Table 1은 가시화 도구의 세 구성요소의 대표적인 요구사항이다.

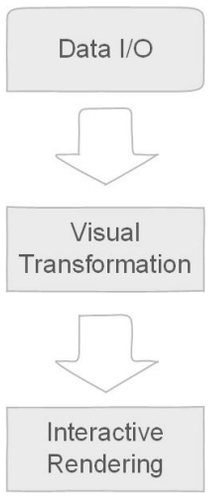


Fig. 1. Three Major Components of Data Visualization Tools

Table 1. Requirements of Three Major Components of Data Visualization Tools for Massive Scientific Dataset

Components	Requirements
Data I/O	Semantic data access to spatio-temporal scientific dataset (access via parameters such cas timestep, physical variable, etc.)
	Real-time sharing of whole dataset across entire computing nodes
Visual Transformation	Providing various visualization methods for spatio-temporal scientific dataset
	Supporting a parallelism on both of a single multi-core machine and a large distributed computing system
Interactive Rendering	Real-time rendering on massive visual representations (usually polygons)
	Data management of visual representations based on scene graph or similar data structure
	Portability : Integration with windows toolkit such as (Qt, GLUT, WxWidget, and MFC)

3.1 데이터 입출력

최근 컴퓨팅 기술이 급속도로 발전하고 있지만, 프로세스 성능이나 메모리 크기의 발전에 비해 데이터 입출력 성능은 상대적으로 개선 속도가 느리다. 이에 따라 대용량 데이터를 다루는 대부분의 응용에서 데이터 입출력 성능이 전체 성능을 저하시키는 경우가 많다. 대용량 실시간 가시화 도구 역시 최대한 빠른 데이터 입출력 성능을 필요로 한다. 이를 위해 물리적인 스토리지의 입출력 성능을 개선시키는 것뿐만 아니라, 별도의 소프트웨어 기술이 필요하다.

데이터 입출력은 시각적 변환 및 사용자 인터페이스 구성요소를 실행하기 위해 반드시 요구되는 구성요소이다. 시각적 변환 및 사용자 인터페이스 구성요소의 효율적인 구현을 위해 사용자 친화적이고 대용량 데이터로의 편리한 접근을

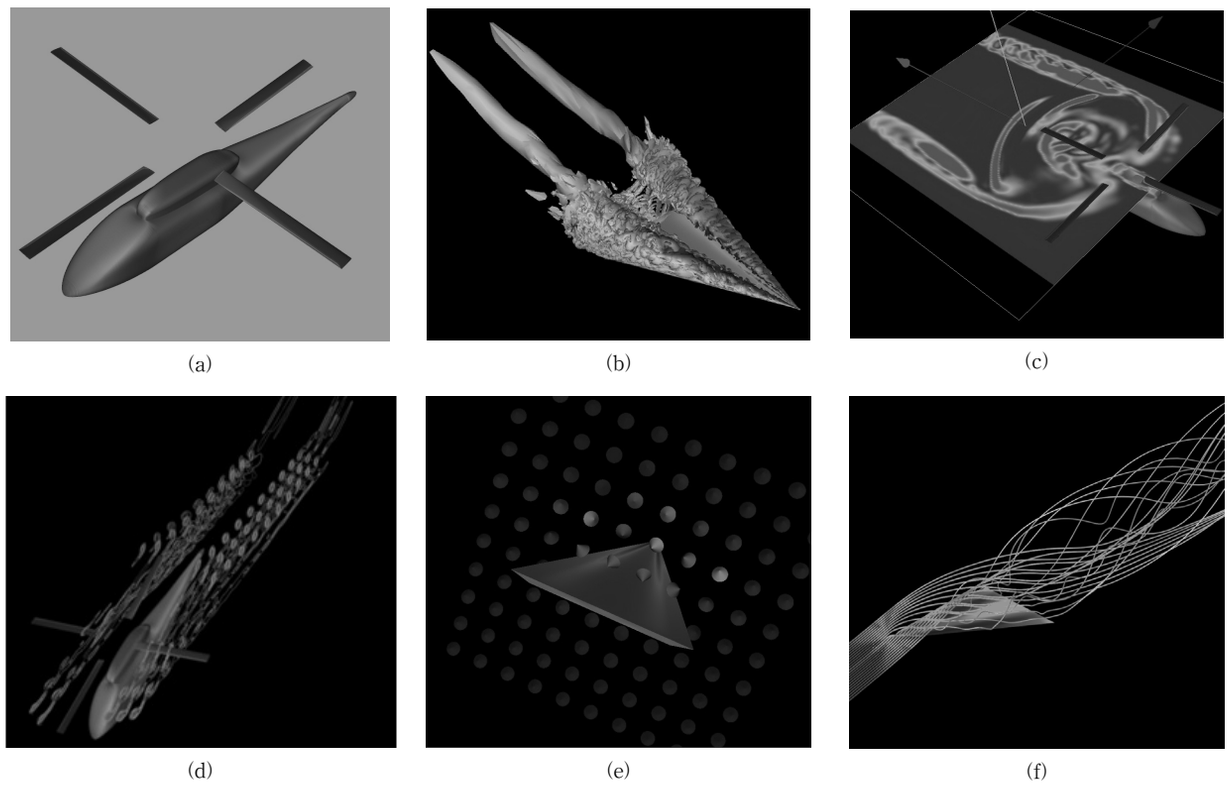


Fig. 2. Basic visualization methods on spatio-temporal scientific dataset. (a) Surface display (b) Iso-surface (c) cutting slice (d) contour line (e) vector glyphs (f) streamlines

가능하게 하는 인터페이스 함수를 제공해야 한다. 이를 위해 시공간적(Spatio-Temporal) 과학 데이터에서 타임스텝, 물리 변수, 데이터 좌표값 등의 의미론적 파라미터로 데이터에 접근할 수 있는 API를 제공해야 한다.

3.2 시각적 변환 (Visual Transformation)

시각적 변환은 원시 데이터(Raw Data)에서 시각적인 표현을 추출하는 과정으로 일반적으로 3차원 볼륨 데이터로부터 폴리곤 데이터를 생성한다. 시각적 변환 방법은 원시 데이터에서 변수의 차원에 따라 스칼라 데이터 (변수가 1차원인 경우), 벡터 데이터(변수가 2차원 이상인 경우) 가시화 방법으로 구분할 수 있으며, 가시화 응용에 따라 요구되는 가시화 방법이 서로 다르다. Fig. 2는 시공간적 과학 데이터를 다루는 많은 가시화 도구에서 지원하는 대표적인 가시화 방법들이다. (a)~(d) 까지는 모두 스칼라 데이터 가시화 방법이고 (e)와 (f)는 벡터 데이터 가시화 방법이다.

또한, 대용량 실시간 가시화 도구에서는 대용량 데이터의 빠른 처리를 위한 병렬 및 분산 가시화 기능이 요구된다. 멀티 코어 및 분산 메모리 환경에서의 효율적인 병렬처리를 위해 부하 균형(Load Balancing)과 확장성(Scalability) 등의 효율적인 병렬화 기법들 또한 필요하다.

3.3 상호작용적인 렌더링

상호작용적인 렌더링은 시각적 변환 과정에서 생성되는 대용량의 폴리곤 데이터의 실시간 렌더링이 가능해야 한다. 이를 위해 그래픽스 하드웨어 가속을 지원해야 하고, 장면 그래프 등 3차원 그래픽스에 특화된 자료구조를 지원하여 전체 장면의 3차원 객체들의 조작을 용이하게 할 수 있어야 한다. 또한 그래픽 렌더링은 호환성이 보장되지 않는 경우가 많으므로, 다양한 운영체제와 윈도우 툴킷 환경으로의 높은 이식성을 보장하여 다양한 환경으로의 포팅을 지원해야 한다.

4. GLOVE 개요

3장에서 설명한 대용량 실시간 가시화 도구의 각 구성요소 및 요구사항에 맞춰 새로운 대용량 실시간 가시화 도구인 GLOVE(GLObal Visualization Environment)를 제안한다. GLOVE는 대용량의 3차원 과학 데이터를 빠르게 가시화하고 분석할 수 있게 하기 위해 고안한 실시간 가시화 도구이다. Fig. 3은 GLOVE의 전체 구조를 보여준다. GLOVE의 각 컴포넌트는 3장에서 설명한 구성요소들과 1:1 대응이 된다. GLOVE는 크게 병렬 가시화 서버와 렌더링 클라이언트로 구분되며, 일반적인 클라이언트/서버 구조로 구동된다. 병렬 가시화 서버는 가시화 엔진과 데이터 관리자로 구성되고 각각 시각적 변환과 데이터 입출력에 대응된다. 렌더링 클라이언트는 병렬 가시화 서버와는 별개의 컴퓨터에서 구

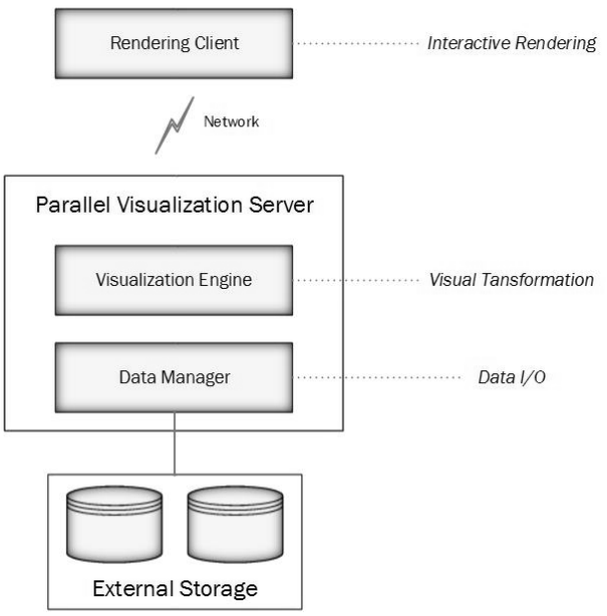


Fig. 3. Overall Architecture of GLOVE

동되며 병렬 가시화 서버와 네트워크를 통해 통신한다.

GLOVE는 효율적인 개발 및 구현을 위해 공개 소프트웨어 도구를 최대한 활용했다. Table 2는 각각의 구성요소 별로 활용한 공개 소프트웨어들이다. GLOVE에서 사용한 각 공개 소프트웨어들은 모두 독립적으로 개발된 소프트웨어이기 때문에 이들 소프트웨어들을 통합하기 위한 노력이 필요했다. 또한 GLOVE의 개발에 사용한 공개 소프트웨어들이 모두 실시간 가시화의 용도로 개발된 것은 아니기 때문에 추가적인 최적화 작업을 통해 성능을 높이고자 했다.

Table 2. List of Open Source S/W Utilized in GLOVE

Component	Open Source S/W	Note
Data I/O	GA	Distributed Shared Memory
Visual Transformation	VTK	Visualization
	MPI	Distributed Memory Parallelism
	OpenMP	Multi-Core Parallelism
Interactive User Interface	OpenSceneGraph	Data Rendering
	Qt	Window Toolkit

5. 데이터 관리자

실시간 가시화 도구를 위한 데이터 관리자의 요구사항을 만족하기 위해 본 논문에서는 공개 분산공유메모리 라이브러리인 GA (Global Arrays)를 활용하여 데이터 관리자를 구현한다. GA는 분산 메모리 구조의 병렬 컴퓨팅 환경에서 지역 메모리를 하나로 묶어서 전역 주소(Global Address)로 접근할 수 있도록 한다. 전역 주소는 분산 공유 메모리에 참여하는 모든 노드들이 공통적으로 사용하는 주소로 이 주소를 통해 다른 노드에 저장된 데이터에 접근이 가능하다. 이를 통해 각 병렬 노드의 지역 메모리 크기보다 큰 규모의 대용량 데이터에 대한 효율적인 접근 및 공유가 가능하다.

5.1 의미론적인 접근

일반적으로 시공간적 과학 데이터는 여러 구분에 따라 의미론적(semantic)으로 나눌 수 있다. 이러한 구분에는 1) 데이터를 구성하는 물리적 단위 요소(ex. 대기(air), 수중(water), 엔진내부 등)와 2) 속도, 압력과 같은 물리 값을 표현하는 변수, 3) 시간의 흐름에 따른 타임스텝 그리고 4) 데이터의 공간적 좌표 범위 등이 있다. 본 논문에서는 이러한 의미론적 요소를 파라미터로 데이터에 접근하고자 하는 요청을 의미론적 접근 요청이라고 한다. 과학 데이터를 위한 가시화 도구는 사용자의 의미론적 요구에 따라 데이터를 실시간으로 접근할 수 있어야 한다.

GLOVE에서는 이러한 의미론적인 접근을 최적화하기 위해 과학 데이터를 의미론적 구분에 따라 미리 나누어 분산 공유메모리에 분산 저장함으로써 의미론적 접근 요청 시 필요한 원시데이터로의 접근 시간을 최소화하도록 했다. Fig. 4는 이를 위해 구성한 과학 데이터의 구조 트리이다. GLOVE의 구조 트리는 루트 노드에서부터 순서대로 E#, V#, T# 그리고 B#으로 구성되는데, E#은 물리적 단위 요소, V#은 변수, T#은 타임스텝 그리고 B#은 공간 영역을 나눈 데이터 블록의 번호이다(#은 임의의 정수). GA 분산공유메모리에서는 모든 데이터를 배열로 관리한다. 구조 트리에서 GA에 실제로 저장되는 부분은 검정색 데이터 블록 노드이고, 파란색 노드는 메타 데이터 형태로 지역 메모리에서 관리된다. 데이터 관리자는 데이터에의 의미론적인 접근을 위

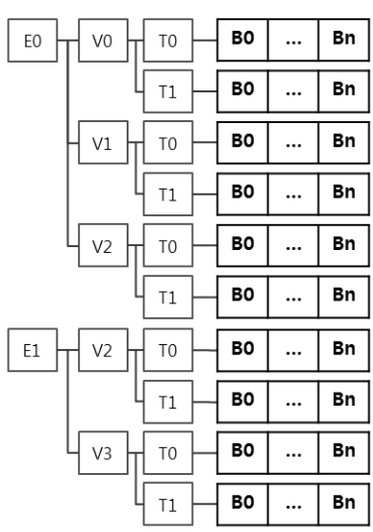


Fig. 4. Structure tree of GLOVE which is optimized for semantic access of spatio-temporal scientific dataset

한 인터페이스 함수를 제공한다. 이를 통해 가시화 엔진은 분산공유메모리에서 데이터의 실제 주소를 알지 못하더라도 데이터에 접근할 수 있다. 이를 위해 의미론적 접근 파라미터로부터 실제 데이터가 저장된 배열의 주소를 알 수 있어야 한다. GLOVE 데이터 관리자는 각 병렬 노드의 지역 메모리에 의미론적인 파라미터로부터 실제 배열의 분산공유메모리 상의 전역 주소를 얻을 수 있는 테이블을 저장하여 빠르게 전역 주소를 얻을 수 있다. GA는 요청된 전역주소에 위치한 배열을 요청한 병렬 노드의 지역메모리로 복사한다.

5.2 실시간 가시화를 위한 성능 최적화

GLOVE 데이터 관리자는 데이터 입출력 성능을 최적화하기 위해 최대한 데이터 통신량을 줄이는 방법을 사용한다. 이를 위해 크게 두 가지 최적화 방안을 제안한다.

1) 게으른 데이터 페치 (Lazy Data Fetch)

일반적인 가시화 도구에서는 데이터를 파일 단위로 접근한다. 따라서 한 파일에 저장된 모든 데이터를 한꺼번에 지역 메모리로 읽는데, 한 데이터를 구성하는 많은 물리 값 변수들이 사용되지도 않으면서 입출력 통신(I/O Communication) 대역폭의 상당부분을 차지하게 된다. GLOVE에서는 게으르게 데이터를 페치하는 방법을 통해 입출력 통신량을 줄임으로써 입출력 성능을 높이고자 했다. 즉, 명시적으로 요청되지 않은 변수나 타임스텝의 데이터는 페치하지 않는 것이다.

2) 데이터 캐시 (Data Cache)

데이터 관리자는 분산공유메모리를 통해 전역 주소로 데이터 접근을 가능하도록 하지만 원격 노드의 메모리에 위치한 데이터에 접근할 때는 지역 메모리의 데이터에 접근할 때보다 속도가 느릴 수밖에 없다. 이러한 문제를 최대한 보완하기 위해 데이터 캐시를 도입하여 속도를 향상시키도록 했다. 본 논문에서는 분산공유메모리에서 데이터 블록을 페치할 때, 의미론적인 접근을 통해 데이터를 페치하므로 데이터 캐시를 저장할 때의 파라미터 역시 의미론적인 파라미터를 사용한다. 본 논문에서 데이터 캐시는 (E#, V#, T#, B#) - (단위요소, 변수, 타임스텝, 블록번호)의 조합을 열쇠로 설정한다. GA를 통해 분산공유메모리의 데이터에 접근하기 전에 캐시를 먼저 살펴본 뒤, 데이터가 캐시 내에 이미 있으면 캐시의 데이터를 먼저 가져오는 방법을 통해 데이터 접근 속도를 개선시킬 수 있다. 캐시 정책은 일반적인 LRU (Least Recently Used) 정책을 사용한다. 이를 이용해서 동일한 배열을 여러 번 접근할 경우 데이터 접근 속도를 향상시킬 수 있다. 가시화 알고리즘의 성격에 따라 다르지만, 스트림라인과 같은 일부 가시화 알고리즘은 멀티코어 프로세스에서 지역 응집성을 이용하여 캐시 히트율을 높이는 방법을 통해 입출력 통신량을 급격히 감소시키는 것이 가능하다. 이 문제에 대해서는 6장에서 자세히 다룬다.

6. 가시화 엔진

시공간적 과학 데이터의 실시간 가시화를 위한 가시화 엔진은 여러 종류의 3차원 가시화 기법을 지원해야 한다. 3장에서 제시한 3차원 가시화 기법들의 구현을 위해 본 논문에서는 공개 소프트웨어 도구인 VTK를 사용한다. VTK는 ParaView나 VisIt 등 대표적인 3차원 가시화 도구에서 사용하는 가시화 툴킷으로 Fig. 2의 3차원 가시화 기법을 모두 지원한다[1]. VTK는 기본적인 3차원 가시화 기능들을 지원하지만 멀티코어 또는 분산 메모리 환경에서의 병렬화 부분은 미흡하다. 본 논문에서는 VTK 라이브러리와 MPI, OpenMP 등의 병렬처리 라이브러리 그리고 GLOVE 데이터 관리자를 이용해서 병렬 가시화 엔진을 구현하였다. Fig. 5는 본 논문에서 제안하는 병렬 가시화 엔진이다. 각 가시화 프로세스는 할당된 CPU 코어 수만큼의 스레드를 가지면서 가시화 작업을 병렬로 처리한다. 데이터 관리자는 GA 분산공유메모리를 통해 하나의 전역 주소를 통한 데이터 접근을 가능하게 한다. 가시화 스레드와 데이터 관리자 사이에는 데이터 캐시가 존재하여 데이터 입출력 속도를 향상시킨다.

본 논문에서는 GA 기반의 데이터 관리자와 VTK 기반의 가시화 엔진을 통합하는 방법과 데이터 관리자를 통해 GA 분산공유메모리 상의 데이터에 접근할 때, 가시화 응용의 특성을 고려하여 데이터 입출력을 최적화하는 방안에 대해 논한다.

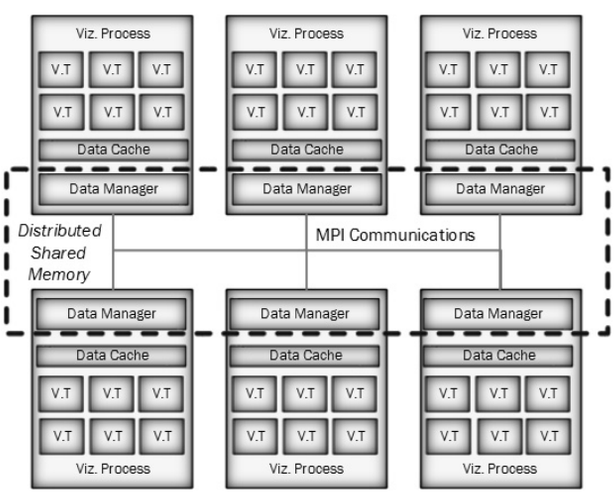


Fig. 5. Architecture of Parallel Visualization Engine (V.T : Visualization Thread)

6.1 데이터 관리자 - 가시화 엔진 인터페이스

GA는 데이터를 단순한 배열 형태로 관리하는 반면 VTK는 여러 배열들을 하나의 클래스로 묶은 vtkDataSet 또는 자식 클래스 형식의 데이터를 필요로 한다. 이를 위해 GLOVE에서는 GA의 배열들을 이용하여 vtkDataSet 형태로 변환하여 VTK 기반의 가시화 엔진에 입력으로 넣는다. vtkDataSet은 vtkStructuredGrid, vtkUnstructuredGrid 등 VTK에서 다루는 대부분의 데이터 형식의 부모 클래스로, 메쉬 정점

(mesh points), 변수 배열(variable arrays)의 포인터를 멤버로 가지는 클래스이다. 이러한 포인터에 GA에서 페치한 배열의 주소를 할당하고, 메쉬 정점의 개수, 변수의 개수 및 이름 등을 설정하면 vtkDataSet이 생성된다. 이때, 대용량 데이터에서의 배열 복사를 최소화하기 위해 vtkDataSet을 구성할 때 GA를 통해 페치한 배열로의 포인터 할당으로 처리하여 메모리 낭비를 줄이고 속도를 향상시키는 것이 가능했다. 배열들로부터 VTK 데이터를 생성하는 방법은 [16]에서 찾을 수 있다.

6.2 가시화 엔진의 병렬화

GLOVE는 MPI를 이용해서 분산 환경에서의 병렬 가시화를 지원함과 동시에 OpenMP를 통해 멀티 코어 환경에서의 병렬 가시화를 지원한다. MPI와 OpenMP 하이브리드 병렬화를 위해 데이터 관리자에서는 Fig. 4와 같이 데이터를 공간 영역(spatial space)에서 여러 개의 작은 블록으로 나누어 저장한다. 각 병렬 노드에서는 미리 나누어 놓은 데이터 블록 단위로 가시화 작업을 수행하는데, 데이터 관리자에서 제공하는 의미론적 접근 인터페이스를 통해 분산공유메모리 상에 위치한 데이터로의 접근이 가능하다. 3장에서 언급한 바와 같이 대용량 데이터를 처리할 때, 대부분의 경우 데이터 입출력 부분의 성능 문제로 인해 전체 성능이 저하되게 된다. 이러한 문제를 해결하기 위해 본 논문에서는 가시화 작업을 스칼라 가시화 작업과 벡터 가시화 작업으로 구분하고, 각각의 작업에 최적화된 데이터 입출력 기법을 적용함으로써 성능을 향상시키고자 한다.

1) 스칼라 가시화 기법의 병렬화

스칼라 데이터 가시화는 전체 셀에 걸쳐 동일한 가시화 연산을 수행하고, 한 프로세스를 처리할 때 다른 프로세스가 처리하는 데이터에 접근할 필요가 없다[17]. 따라서 각 병렬 노드에서는 분산공유메모리로부터 서로 다른 데이터 블록을 페치하여, 독립적으로 가시화 작업을 수행하는 방법으로 병렬 가시화를 구현한다. 이러한 작업 분산 기법을 데이터 블록 분산 기법이라고 한다. 본 논문에서는 각 병렬 노드에서 분산공유메모리로부터 데이터를 페치할 때, 가용한 CPU 코어의 개수만큼의 데이터 블록을 한번에 페치하여 병렬로 처리함으로써 멀티 코어 병렬화를 구현하였다.

2) 벡터 가시화 기법의 병렬화

스트림라인 생성과 같은 적분 기반 가시화는 스칼라 데이터 가시화 기법과 달리 한 스트림라인 생성 작업을 여러 노드에 분산시켜서 병렬 처리할 수 없다. 이는 스트림라인 생성 계산이 라인을 따라 순차적으로 이루어지기 때문이다[18]. 따라서 작업을 데이터 블록 기준으로 분산하는 스칼라 데이터 가시화 알고리즘의 작업 분산 기법은 좋은 병렬화 효율을 얻을 수 없다.

본 논문에서는 스트림라인 생성 가시화 기법의 병렬 성능의 향상을 위해 병렬 계산 작업의 분배를 스칼라 데이터 가시화 방법과는 달리 씨드 포인트(seed point)를 기준으로 나눈다. 이를 위해 데이터 관리자는 가시화 엔진에서 요청한 포인트 좌표를 기반으로 데이터 블록을 동적으로 공급할 수 있도록 한다. 병렬 노드의 각 스레드는 하나의 데이터 캐시를 공유하는데, 캐시 히트율을 높이기 위해 씨드 포인트를 분배할 때 지역 응집성을 고려하여 분배한다. 지역적으로 인접한 씨드 포인트는 동일한 데이터 블록 내에 위치할 가능성이 높고, 이를 통해 여러 스레드에서 데이터 블록을 공유하도록 하여 캐시 히트율을 높이는 것이다. 한편, 각 병렬 노드는 최대한 동일한 개수의 씨드 포인트(seed point)를 할당받아 부하가 전체 노드에 고르게 분산되도록 해야 한다. 즉, 이를 정리하면 이 문제는 총 n개의 포인트가 3차원 공간에 정의되었을 때, 지역적으로 가까운 포인트들끼리 군집으로 묶되, 각 군집이 동일한 개수의 포인트(군집의 개수가 k개라면 n/k개)를 가지도록 하는 군집화(clustering) 문제이다. 이 문제를 해결하기 위해 본 논문에서는 잘 알려진 데이터 군집화 기법인 k-평균 알고리즘[19]을 사용하였다. K-평균 알고리즘에서 최악의 군집화 결과를 방지하고자 씨드 선정에 k-평균++ 알고리즘[20]을 사용하였으며, 군집화 과정에서 군집 내 멤버의 개수가 n/k보다 커지면 해당 군집은 더 이상 멤버를 받지 못하도록 제약하는 방법으로 각 군집의 멤버 개수를 동일하도록 강제했다. 이러한 제약은 군집화의 품질을 저하시킬 수 있지만, 스트림라인 생성에서는 부하균형이 캐시 히트율 향상보다 전체 성능에 더욱 큰 영향을 줄 것이라는 판단으로 각 군집의 멤버 포인트 개수를 균등하게 유지하여 전체 병렬 노드의 부하를 최대한 균등하게 하도록 했다. 이를 통해 부하를 전체 병렬 노드에 고르게 분산시키면서 캐시 히트율을 높이는 효과를 얻고자 했다.

7. 렌더링 클라이언트

GLOVE의 마지막 컴포넌트는 렌더링 클라이언트로 실시간 과학데이터 가시화 도구에서 상호작용적인 렌더링 구성요소에 대응된다. 본 논문에서는 3장에서 제시한 상호작용적인 렌더링의 요구사항을 만족하는 대표적인 공개 렌더링 도구인 OpenSceneGraph(이하 OSG)를 사용하여 렌더링 클라이언트를 구현한다. 가시화 엔진에서 사용하는 VTK 역시 렌더링 기능을 제공하지만, 렌더링 성능이 느려서 대용량 폴리곤 데이터의 실시간 렌더링에는 적합하지 못하다.

OSG는 비주얼 시뮬레이션, 가상/증강현실, 의료/과학 데이터 가시화, 교육 및 게임 등 다양한 분야의 소프트웨어 개발에 사용 가능한 실시간 3D 그래픽스 라이브러리이다. 장면 그래프(Scene Graph) 기반으로 구성되었으며 대용량 폴리곤 데이터에서의 실시간 렌더링을 위한 다양한 3D 렌더링 가속 기법을 지원한다[21].

GLOVE 가시화 엔진은 VTK로 구현되어 최종 가시화 결과를 vtkPolyData 형식으로 저장한다. 반면 OSG는 osg::Drawable 형식의 데이터만을 장면 그래프에 등록하고 렌더링할 수 있다. 따라서 vtkPolyData의 osg::Drawable 데

Table 3. Experiment Data (Tera-scale Rotor Blade Simulation Dataset)

Block ID	Number of Blocks	Number of Mesh Points	Timesteps	Number of Variables	Data Type	Total Size (GB)
Type 1	32	580,644	98	8 (2 vectors, 6 scalars)	double	169.58
Type 2	152	725,805				1006.9
Type 3	16	408,807				59.7
Total	200	-				1,236.18

Table 4. Experimental Environment

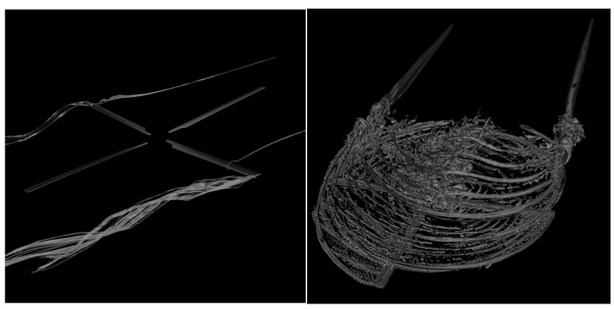
Number of Nodes	64
CPU Core / Node	4 (Xeon X5450 3.0GHz)
Total CPU Cores	256
Memory Size / Node	32 GB
Total Memory Size	2048 GB

이터로의 변환이 필요하다. OSG에서는 사용자 맞춤형 Drawable을 지원하는데, osg::Drawable을 상속받는 새로운 클래스를 생성한 후 drawImplementation() 멤버 함수와 computeBound() 멤버 함수를 구현하면 된다[21]. drawImplementation() 함수에서는 다양한 OpenGL 함수를 사용할 수 있으므로 vtkPolyData 내의 각 폴리곤 및 삼각형 리스트의 정점 배열(vertex array)를 받아서 OpenGL의 정점 배열 연산(vertex array operation)을 사용하여 렌더링하면 VTK 데이터를 빠르게 렌더링하는 것이 가능해진다.

8. 실험 결과 및 토의

본 논문에서는 테라 스케일의 로터 동역학 데이터[22]를 이용해서 GLOVE와 ParaView 그리고 VisIt의 성능을 비교 테스트하였다. 세 가시화 도구는 모두 VTK를 이용해서 구현되었기 때문에 동일 환경에서 지역 메모리에 동일 데이터가 적재되었을 때, 단일 프로세스를 사용한 가시화 성능은 모두 동일하다고 가정한다. 실험에 사용된 로터 동역학 데이터는 정렬 격자(structured grid)로 구성된 시변환(time-varying) 데이터로, 격자의 정점마다 속도, 압력, Q-기준(criterion) 등의 스칼라 또는 벡터 변수가 저장되어 있다. Table 3은 실험에 사용한 데이터의 정보이고, 실험 환경은 Table 4에서 확인할 수 있다. Fig. 6은 스칼라와 벡터 가시화 성능 실험을 통한 가시화 결과 그림이다.

GLOVE와 ParaView 그리고 VisIt 각각에 대해서 전체 실험 데이터를 모두 올린 뒤 등치면(iso-surface) 및 스트림라인 생성 실험을 하였다. 전체 타임스텝 중 랜덤하게 타임



(a) Stream line (b) Iso-surface

Fig. 6. Visualization Results on Two Experiments

스텝을 선택하여 총 100번의 실험을 한 후 평균을 구했는데, GLOVE 등치면 생성의 경우 캐시를 끈 상태에서 50번, 캐시를 켠 상태에서 50번을 실험하였고, GLOVE 스트림라인 생성 실험의 경우 캐시를 끈 상태에서 34번, 캐시를 켠 상태에서 33번, k-평균 군집화를 통한 씨드 포인트 선정을 사용한 상태에서 33번 실험하였다. 또한, 스트림라인 생성의 경우 총 100개의 씨드 포인트를 사용했다. 실험 결과는 Table 5에서 볼 수 있다.

등치면 생성 실험과 스트림라인 생성 실험 모두 캐시를 사용할 때와 사용하지 않을 때의 성능을 비교하여 캐시의 효용 여부를 비교했다. 스트림라인 생성 실험의 경우 씨드 포인트를 지역 응집성을 고려한 씨드 포인트를 할당하는 경우와 무작위로 할당하는 경우도 함께 비교 실험했다. 실험 결과 등치면 생성과 스트림라인 생성 모두 GLOVE가 ParaView 및 VisIt의 성능을 능가했다. 등치면 생성의 경우, GLOVE의 캐시를 사용할 때 ParaView 대비 약 4.77배의 성능 향상을 보였다. 이는 데이터 용량이 지역 메모리의 크기를 능가할 경우 데이터를 지역 메모리에 미리 적재하지 않고, 필요할 때마다 스토리지로부터 실시간으로 데이터를 적재하는 ParaView의 데이터 관리 특성 때문인 것으로 보인다. ParaView는 실행 시간(run-time)에 실시간으로 데이터를 메모리에 적재하기 때문에 데이터 입출력 시간으로 인해 성능에 급격한 저하가 오게 된다. 반면 데이터를 미리 모든 병렬 노드의 지역 메모리에 분산 저장하였다가 네트워

Table 5. Experimental Results (Unit: sec)

	ParaView	VisIt	GLOVE		
			Cache Off	Cache On	Cache & Clustering On
Iso-surface	10.11	5.54	3.37	2.99	N/A
Streamline	72.71	13.74	12.94	8.46	6.54

크를 통해 데이터를 공유하는 VisIt은 ParaView 대비 GLOVE에 조금 더 가까운 성능을 보였다. VisIt은 GLOVE에 비해 다양하고 강력한 부하 분산 기법을 제공하지만 전체 성능은 GLOVE가 더 좋았는데, 이는 게으른 데이터 페치 정책 덕분에 GLOVE의 네트워크 데이터 전송량이 VisIt 대비 작았기 때문인 것으로 판단된다. 한편, GLOVE에 캐시를 사용하였을 경우 사용하지 않았을 경우에 비해 약 1.13배의 성능 향상을 보였다. 일반적으로 등치면 생성과 같은 스칼라 데이터 가시화 기법에서는 한번 페치한 데이터 블록을 다시 사용하는 경우가 드물기 때문에 캐시의 효용성이 높지 않았던 것으로 보인다. 그러나 사용자의 사용 패턴에 따라 동일한 데이터 블록을 여러 번 페치할 경우 캐시의 히트율이 높아져서 성능이 향상될 것으로 보인다.

스트림라인 생성의 경우, 등치면 생성에 비해 더욱 극적인 성능 차이를 보였다. ParaView는 GLOVE나 VisIt에 비해 최소 5.29배, 최대 11.12배의 성능 저하를 보였다. 스트림라인 생성의 경우 등치면 생성에 비해 데이터 입출력량이 훨씬 컸기 때문에 성능 저하가 보다 뚜렷했던 것으로 보인다. 캐시를 사용하지 않을 경우 GLOVE와 VisIt은 거의 유사한 성능을 보였다. 게으른 데이터 페치 정책에도 불구하고 적극적인 부하 균형과 데이터 공유 정책을 사용하는 VisIt의 성능이 매우 좋았던 것으로 판단된다[23]. 스트림라인 생성 계산에서는 등치면 생성에 비해 캐시의 효용성이 높았다. 씨드 포인트를 각 노드에 랜덤하게 배정했을 경우, 캐시를 사용할 때가 사용하지 않았을 때에 비해 1.53배의 성능 향상이 있었다. 그러나 k-평균 군집화 기법을 통해 지역적으로 가까운 씨드 포인트 끼리 묶어서 배정할 경우 1.98배의 성능 향상이 있었다. 이를 통해 지역적으로 응집되게 씨드 포인트를 배정할 경우 캐시 히트율이 올라감을 알 수 있었다.

9. 결 론

본 논문에서는 대용량 데이터 가시화 도구의 세 가지 구성요소를 데이터 입출력, 데이터의 시각적 변환, 상호작용적인 렌더링으로 정의하고, 각 구성요소 별 요구사항을 제시하였다. 모든 요구사항을 만족하는 가시화 도구를 효율적으로 개발하기 위한 방안으로 공개 소프트웨어 도구를 사용하여 통합하고 가시화 응용에 최적화하는 접근방법을 제안하였으며, 실제 공개 소프트웨어들을 활용하여 분산공유메모리 기반 대용량 데이터 가시화 도구인 GLOVE를 제안하였다. GLOVE의 각 구성요소 별로 데이터 입출력은 GA, 시각적 변환은 VTK, 상호작용적인 렌더링은 OSG 공개 소프트웨어 도구를 사용하였으며, 이를 통합하여 병렬 가시화 도구를 개발하는 방법과 데이터 호환 방안을 논의하였다. 또한 각각의 공개 소프트웨어 도구를 단순히 통합하는데 그치지 않고, 가시화 작업의 특성을 이용하여 각 구성요소를 최적화하는 방법도 제안하였다. 또한 GLOVE와 ParaView, VisIt 등의 대용량 과학 데이터 가시화 도구에 대해 실제 전산유체역학 데이터를 이용하여 등치면 생성과 스트림라인 생성 성능을 비교 분석하였다. 이를 통해 제안한 여러 최적화 기법들이 유효하였음을 밝혔다.

본 논문에서 제안한 GLOVE는 모든 데이터를 분산공유메모리에 올리고 게으른 데이터 페치를 하기 때문에 데이터 입출력 성능이 향상되어 기존의 가시화 도구인 ParaView나 VisIt에 비해 높은 성능을 보였다. 그러나 데이터의 크기가 분산공유메모리의 크기보다도 클 경우에 데이터 가시화를 수행할 수 없고, GPU 병렬화가 안된다는 단점이 존재한다. 또한 스칼라와 벡터 데이터 가시화 기법만 제공하고 텐서 데이터 가시화 기법을 제공하지 못할 뿐만 아니라 볼륨 렌더링과 같이 가시화 결과가 이미지로 생성되는 가시화 기법을 제공하지 못하는 문제가 있다. 이러한 단점에 대한 보완이 필요할 것으로 생각된다. 한편, 본 논문에서 사용한 여러 최적화 기법들에 대한 보다 깊이 있는 추가 연구가 필요하다. 본 논문에서는 게으른 데이터 페치, k-평균 군집화를 이용한 최적의 씨드 포인트 분배 기법, 데이터 캐시를 사용한 데이터 입출력 속도 향상 등 다양한 최적화 기법이 제안되고 적용되었고, 실험 데이터에 대해 성능 향상에 효과가 있었음을 알 수 있었다. 그러나 다양한 실험 데이터에서 보다 체계적인 실험 시나리오를 통해 좀 더 깊이 있는 효용성 판단과 추가적인 성능 개선이 가능할 것으로 판단된다.

References

[1] W. Schroeder, K. Martin, and B. Lorensen, “The Visualization Toolkit(4th ed.),” Kitware, ISBN 978-1-930934-19-1.

[2] J. Ahrens, B. Geveci, and C. Law, “ParaView: An End-User Tool for Large Data Visualization,” Visualization Handbook, Elsevier, 2005, ISBN-13: 978-0123875822.

[3] H. Childs et al., “VisIt: An End-User Tool For Visualizing and Analysizing Very Large Data,” High Performance Visualization-Enabling Extreme-Scale Scientific Insight, Oct 2012, pp.357-372.

[4] Ensight [Internet], http://www.ensight.com.

[5] M. De Wael, S. Marr, B. De Fraine, T. Van Cutsem, and W. De Meuter, “Partitioned Global Address Space Languages,” *ACM Computing Surveys*, Vol.47, Issue 4, Article No.62, 2015.

[6] R. W. Numrich and J. Reid, “Co-Array Fortran for Parallel Programming,” *ACM FORTRAN FORUM,* Vol.17, Issue 2, 1998.

[7] T. El-Ghazawi, W. Carlson, T. Sterling, and K. Yelick, “UPC: Distributed Shared Memory Programming,” Hoboken, NJ: Wiley, 2005.

[8] K. A. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. N. Hilfinger, S. L. Graham, D. Gay, P. Colella, and A. Aiken, “Titanium: A High-performance Java Dialect,” *Concurrency: Practice and Experience*, Vol.10, Issue 11-13, pp.825-836, 1998.

[9] B. L. Chamberlain, D. Callahan, and H. P. Zima, “Parallel Programmability and the Chapel Language,” *International Journal of High Performance Computing Applications,* Vol.21, No.3, pp.291-312, 2007.

[10] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar, "X10: An Object-Oriented Approach to Non-Uniform Cluster Computing," in *Proc. 20th annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05),* ACM, NY, USA, pp. 519-538, 2005.

[11] E. Allen, D. Chase, J. Hallett, V. Luchangco, J.-W. Maessen, S. Ryu, Guy L. Steele, and S. Tobin-Hochstadt. "The Fortress Language Specification," Technical Report. Sun Microsystems, Inc., Version 1.0, 2008.

[12] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. "Global Arrays: A Portable 'Shared-Memory' Programming Model for Distributed Memory Computers," in *Proceedings Supercomputing '94,* pp.340-349, 1994.

[13] D. R. Jones, E. R. Jurrus, B. D. Moon, and K. A. Perrine, "Gigapixel-size Real-time Interactive Image Processing with Parallel Computers," *Proceedings of Workshop on Parallel and Distributed Processing Symposium,* 7, 2003.

[14] J. A. Kohl, T. Wilde, and D. E. Bernholdt, "CUMULVS: Interacting with High-Performance Scientific Simulations, for Visualization, Steering and Fault Tolerance," *International Journal of High Performance Computing Applications*, Vol.20, p.255, 2006.

[15] Z. Fan, F. Qiu, and A. E. Kaufman, "Zippy: A framework for computation and visualization on a gpu cluster," *Computer Graphics Forum*, Vol.27, No.2, pp.341-350, 2008.

[16] "The VTK User's Guide 11th Edition," Kitware Inc., pp. 105-117, 2010.

[17] M. Kim, Y. Hur and J.-Y. Lee, "InVis: An Interactive Visualization Framework for Massive Data supporting Multiple Users," *Journal of KIISE: Computing Practices and Letters*, Vol.18, No.1, 2012.

[18] L. Chen and I. Fujishiro, "Optimizing Parallel Performance of Streamline Visualization for Large Distributed Flow Datasets," in *2008 IEEE Pacific Visualization Symposium*, pp.87-94, 2008.

[19] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability,* University of California Press, pp.281-297, 1967.

[20] D. Arthur and S. Vassilvitskii, "K-means++: the Advantages of Careful Seeding," *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics Philadelphia*, PA, USA, pp.1027-1035, 2007.

[21] R. Wang and X. Qian, "OpenSceneGraph 3.0: Beginner's Guide," PACKT Books, ISBN 9781849512824, 2010.

[22] J. Kim, J. Sa, S. Park, J. Park, S. Jung, Y. Yoo, and K. Cho, "Parallel CFD Computation for Vortex Flow Field around HART II Rotor Blades with Prescribed Blade Deformation," *Proceedings of 22nd International Conference on Parallel Computational Fluid Dynamics*, 2010.

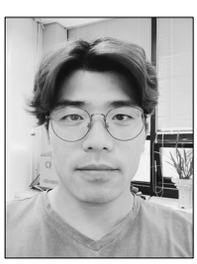
[23] D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy, "Streamline Integration Using MPI-Hybrid Parallelism on a Large Multicore Architecture," *IEEE Transactions on Visualization and Computer Graphics*, Vol.17, No.11, pp. 1702-1713, 2011.

이 중 연

e-mail : jylee@kisti.re.kr
1998년 서강대학교 전자계산학과(학사)
2000년 서강대학교 컴퓨터학과(석사)
2011년~현 재 KAIST 전산학부 박사과정
2000년~현 재 한국과학기술정보연구원 가시화기술개발실 선임연구원
관심분야 : Computer Graphics, Scientific Visualization, Parallel Visualization, Visual Analytics, In-situ Visualization, Exa-scale Visualization

김 민 아

e-mail : petimina@kisti.re.kr
1996년 부산대학교 전자계산학과(학사)
1998년 부산대학교 전자계산학과(석사)
1998년~2000년 (주) LG 정보통신
2000년~2003년 (주) 필링크 책임연구원
2004년~2005년 동명대학교 초빙교수
2005년~2012년 한국과학기술정보연구원 가시화기술개발실 선임연구원
2013년~현 재 한국과학기술정보연구원 가시화기술개발실 실장
관심분야 : Deep Learning, Virtual Reality, Visual Analysis, Exa-scale Visualization, In-situ Visualization



이 세 훈

e-mail : sehooi@kisti.re.kr
2003년 KAIST 전산학과(학사)
2006년 KAIST 전산학과(석사)
2000년~현 재 한국과학기술정보연구원 가시화기술개발실 선임연구원
관심분야 : Visual Computing, Parallel Data Processing

허 영 주

e-mail : popea@kisti.re.kr
1997년 성균관대학교 정보공학과(학사)
2002년 성균관대학교 컴퓨터공학과(석사)
2002년~현 재 한국과학기술정보연구원 가시화기술개발실 선임연구원
관심분야 : Data Visualization, Scientific Visualization