ISBN 978-89-6211-611-3

GDM : GLOVE Data Manager

구 기 범 (voxel@kisti.re.kr)

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

ISBN 978-89-6211-611-3

제목 차례

ISBN 978-89-6211-611-3

ISBN 978-89-6211-611-3

표 차례

그림 차례

ISBN 978-89-6211-611-3

소스 차례

ISBN 978-89-6211-611-3

1. 개요

Scientific visualization의 핵심은 데이터 관리에 있다. 특히 슈퍼컴퓨팅 환경에서 만들어지는 시뮬레이션 데이터는 그 크기가 TB(테라바이트) 수준을 넘고 있으며, PFLOPS 급 성능을 갖춘 시스템의 등장과 LHC의 본격적인 가동 등 앞으로 우리가 보게 될 데이터의 크기가 더 빠르게 증가할 것이라는 점은 쉽게 짐작할 수 있다.

한편, 슈퍼컴퓨터를 이용한 시뮬레이션은 대부분의 경우 수시간~수주에 이르는 긴 실행시간을 필요로 한다. 그리고 슈퍼컴퓨터의 성능/용량이 증가할수록 시뮬레이션 역시 더 높은 해상도의 메시(mesh)와 더 많은 데이터를 다루도록 요구받기 때문에, 시스템의 성능향상이 시뮬레이션 실행시간의 대폭적인 단축으로 이어지지는 않는다. 계산과학자도 이런 상황을 잘 인지하고 있기 때문에 시뮬레이션의 실행시간 단축에 대한 요구수준은 data visualization과 비교하면 상대적으로 낮은 편이다.

하지만 data visualization은 최소 interactive time(5~15 frames/sec)에 준하는 속도의 data manipulation이 실현되지 않으면 사용자들이 금방 불편을 느끼게 된다는 특징이 있다. 바로 이 지점에서 visualization의 큰 주제 중 하나인 '데이터 관리'가 등장한다. 즉, 시뮬레이션 단계에서 오랜 시간동안 만들어진 대용량 데이터를 실시간으로 가공해서 사용자에게 보여주기 위한, 말하자면 실시간 데이터 관리기법이 필요한 것이다.

이 보고서에서는 KISTI 슈퍼컴퓨팅본부에서 개발 중인 어플리케이션인 GLOVE(GLObal Virtual Environment for data visualization)의 데이터 관리 모듈에 대해 집중적으로 설명한다. GLOVE는 대용량 로터 동역학(rotor dynamics) 시뮬레이션 데이터를 실시간으로 가상현실 환경에서 분석하기 위한 visualization 어플리케이션으로, 기본적으로 테라바이트급 데이터를 다루기 위한 구조로 설계되어있다. GLOVE는 분산 메모리 구조를 갖는 클러스터 환경에서 대용량 데이터를 효과적으로 가공하기

ISBN 978-89-6211-611-3

위해 어플리케이션 레벨의 DSM(Distributed Shared Memory) 툴킷인 Global Arrays(이하 GA)를 이용하고 있다. GA는 여러 대의 클러스터 노드가 갖고 있는 메인 메모리를 가상의 대형 메모리로 보이도록 함으로써 대용량 메모리를 다루는 어플리케이션 개발을 쉽게 만들어준다는 장점을 갖고 있으며, 오랜 시간동안 개발된 만큼 충분한 안정성도 확보하고 있다.

이 보고서는 다음과 같이 구성되어 있다. 먼저 2장에서는 GA에 대해 간략하게 소개한다. 3장에서는 GLOVE의 데이터 관리자(GLOVE Data Manager, 이하 GDM)의 일반적인 특징에 대해 설명한다.

ISBN 978-89-6211-611-3

2. Global Arrays

가. 개요

Global Arrays(이하 GA)는 Pacific Northwest National Lab에서 개발한 소프트웨어로, 물리적으로 분산된 메모리 구조를 갖는 병렬 컴퓨터의 메모리를 하나의 큰 공유 메모리처럼 쓸 수 있도록 해주는 역할을 담당한다. 이 소프트웨어는 주로 양자화학 분야에서 사용되었으며, 이를 이용하는 프로젝트에는 NWChem, MOLPRO, UTChem, MOLCAS, TURBOMOLE 등이 있다.

GA는 공유 데이터(shared data)의 global indexing이 가능하며, 클러스터의 노드가 추가 되는대로 공유 메모리의 크기를 늘릴 수 있다는 장점이 있다. 또한 병렬 프로그램을 개발할 때 많이 사용하는 MPI와의 혼용이 가능하기 때문에 기존의 어플리케이션이 쉽게 공유 메모리를 사용할 수 있다는 점도 큰 장점으로 작용한다.

나. Picasso에의 설치

GA를 설치하기 위해서는 여러 가지 환경병수를 지정해야 하는데, Picasso는 클러스터 노드 간 네트워크로 인피니밴드를 사용하기 때문에 환경병수를 다음과 같이 설정한 후 GA를 설치해야 했다

환경변수	내용
TARGET	LINUX64
MPI_INCLUDE	/usr/local/MPI/include
MPI_LIB	/usr/local/MPI/lib
ARMCI_NETWORK	OPENIB
IB_INCLUDE	/usr/include/infiniband
IB_LIB	/usr/lib64

<표 2-1> GA를 설치하기 위한 환경변수

ISBN 978-89-6211-611-3

다. 대표적인 GA API

(1) Create

GA 영역을 생성하기 위해 필요한 함수들은 다음과 같다. 분산 메모리 구조 시스템에서 실행되는 것을 주목적으로 하기 때문에 노드 사이의 데이터 분산을 자세히 제어할 수 있다는 특징이 있다.

```
Int NGA_Create( int type, int ndim, int dims[], char *name, int chunk[] )
Int NGA_Create_irreg( int type, int ndim, int dims[], char *name, int nblock
[], int map[] )
Int GA_Create_handle()
Void GA_Set_data( int g_a, int ndim, int dims[], int type )
Void GA_Set_array_name( int g_a, char *name )
Void GA_Set_chunk( int g_a, int chunk[] )
Void GA_Set_irreg_dist( int g_a, int map[], int nblock[] )
Int GA_Allocate( int g_a )
Int GA_Duplicate( int g_a, char *name )
```

<소스 2-1> GA 영역의 생성을 위한 API

(2) 단방향 통신

서로 다른 노드 사이의 데이터 전송을 위해 마련된 API는 다음과 같다. 여기서 NGA_Read_inc같은 함수는 atomic operation이라는 특징이 있으며, NGA_Gather와 NGA_Scatter는 collective operation을 구현한다.

```
Void NGA_Put( int g_a, int lo[], int hi[], void *buf, int ld[] )
Void NGA_Scatter(int g_a, void *v, int indices[], int n)
Void NGA_Get( int g_a, int lo[], int hi[], void *buf, int ld[] )
Void NGA_Gather (int g_a, void *v, int indices[], int n)
Void NGA_Acc(int g_a, int lo[], int hi[], void *buf, int ld[], void *alpha)
Void NGA_Read_inc (int g_a, int subscript[], long inc)
```

<소스 2-2> GA에서의 단방향 통신을 위한 API

ISBN 978-89-6211-611-3

(3) Utility

Utility 관련 함수는 데이터 입/출력과는 직접 관련이 없지만 어플리케이션의 실행과 관련해서 유용한 기능을 구현한 함수들이다. 특히 프로그램의 디버깅 등을 위한 메시지 출력 관련 함수들은 주의해서 볼 필요가 있다.

```
int GA_Nodeid ()
int GA_Nnodes ()
int NGA_Locate (int g_a, int subscript[])
int NGA_Locate_region (int g_a, int lo[], int hi[], int map[], int proc[])
Void NGA_Distribution (int g_a, int iproc, int lo[], int hi[])
Void GA_Print (int g_a)
Void GA_Print_patch (int g_a, int ilo, int ihi, int jlo, int jhi, int pretty)
Void GA_Print_stas (void)
Void GA_Print_distribution (int g_a)
Void GA_Summarize (int verbose)
```

<소스 2-3> GA 유틸리티 함수

(4) Collective operation

여러 노드에 분산 저장되어 있는 데이터의 초기화, 대량 복사 등을 위해 준비한 API 집합이다.

```
Void GA_Zero( int g_a )
Void GA_Fill( int g_a, void *value )
Void GA_Scale( int g_a, void *value )
Void GA_Copy( int g_a, int g_b )
Void GA_Zero_patch( int g_a, int lo[], int hi[] )
Void GA_Fill_patch( int g_a, int lo[], int hi[], void *value )
Void GA_Scale_patch( int g_a, int lo[], int hi[], void *alpha )
Void GA_Copy_patch( char trans, int g_a, int alo[], int ahi[], int g_b, int blo
[], int bhi[] )
```

<소스 2-4> GA의 collective operation 함수

ISBN 978-89-6211-611-3

3. 일반사항

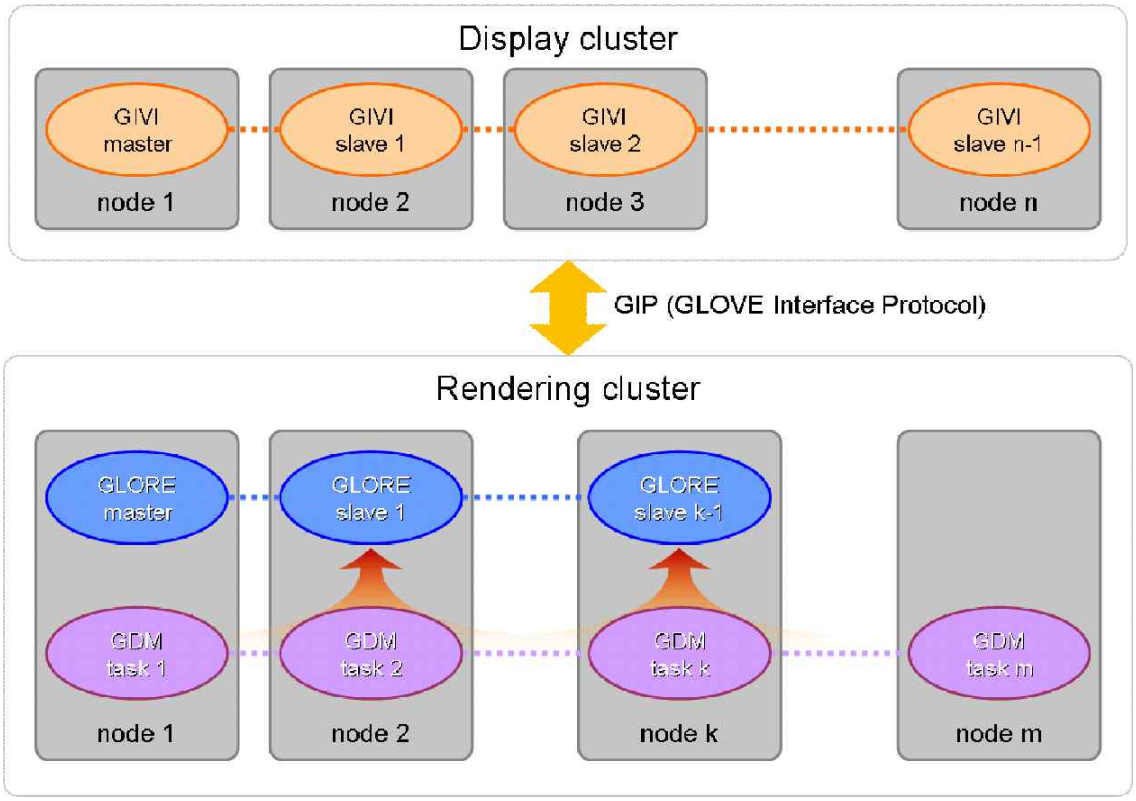
가. GDM에 대한 요구사항

(1) 전제조건

- GDM은 다른 GLORE 어플리케이션(GLORE, GIVI 등)과는 별도로, 데몬 프로세스와 비슷하게 백그라운드 작업으로 실행되어야 한다.
- GLORE에서는 컴파일할 때 별도로 링크할 라이브러리와 API가 필요하다. 여기서는 그 라이브러리를 DM library라고 부른다.
- GLORE로부터의 명령은 메시지 큐(IPC message queue)를 이용해서 전달하고, 데이터 전송은 공유 메모리(IPC shared memory)를 통한다. 단, 명령어를 전달할 때 파라미터(parameter)가 많으면 공유 메모리를 이용할 수도 있다.
- GDM과 GLORE 모두 MPI 기반의 병렬 프로그램인데, GDM을 구성하는 MPI 태스크의 수는 최소한 GLORE를 구성하는 MPI 태스크보다 많아야 한다.
- GLORE로부터의 모든 명령은 티켓(ticket, 추후 설명)을 포함한다.
- 저수준의 데이터 관리는 GA를 이용한다.
- DM library는 C++ 인터페이스를 갖고 있다.
- GDM이 GLORE로 데이터를 전달할 때 최종 형태는 vtkDataSet으로 주어진다.

ISBN 978-89-6211-611-3

나. GDM의 역할



<그림 3-1> GLOVE 내에서의 GDM의 역할

다. 설정 파일

GDM의 설정파일은 XML 형태로 저장되며, GDM은 물론 DM library도 동일한 설정 파일을 읽어야 한다. GDM 설정 파일은 다음의 내용을 담고 있다.

Field name	Content
number of nodes	node 의 개수

ISBN 978-89-6211-611-3

message queue key_send	gdm에서 dmlib으로 보내는 message queue key
message queue key_recv	dmlib에서 gdm으로 보내는 message queue key
shared memory key_check alive	check alive를 수행하는 shared memory key
shared memory size_check alive	check alive를 수행하는 shared memory size
shared memory key_load	load 명령의 field 값을 받는 shared memory key
shared memory size_load	load 명령의 field 값을 받는 shared memory size
shared memory key_ticket	issue ticket 명령의 field 값을 받는 shared memory key
shared memory size_ticket	issue ticket 명령의 field 값을 받는 shared memory size
shared memory key_mesh	mesh data 명령의 field 값을 받는 shared memory key
shared memory size_mesh	model data 명령의 field 값을 받는 shared memory size
shared memory key_field block	field block 명령의 field 값을 받는 shared memory key
shared memory size_field block	field block 명령의 field 값을 받는 shared memory size
shared memory key_field position	field position 명령의 field 값을 받는 shared memory key
shared memory size_field position	field position 명령의 field 값을 받는 shared memory size
shared memory key_return mesh	return 되는 mesh data를 받는 shared memory key
shared memory size_return mesh	return 되는 mesh data를 받는 shared memory size
shared memory key_return field block	return 되는 field block을 받는 shared memory key
shared memory size_return field block	return 되는 field block을 받는 shared memory size
shared memory key_return field position	return 되는 field position을 받는 shared memory key
shared memory size_return field position	return 되는 field position을 받는 shared memory size

<표 3-1> GDM 설정파일의 내용

ISBN 978-89-6211-611-3

라. 데이터 접근 모드

GLOVE가 수행하는 여러가지 visualization 작업에 대해 데이터의 접근 방법은 다음의 세 가지 패턴으로 정리할 수 있다.

① 한 번의 visualization 작업을 위해 전체 데이터를 살펴본다. 예컨데 iso-surface extractio이 이에 해당한다. 이 경우 GDM이 자체 카운터를 갖고 데이터의 어떤 부분을 접근했는지 기록하고 있어야 한다.

② 데이터 블록 단위의 접근 : visualization 작업이 전체 데이터를 필요로 하지는 않는 대신 특정 영역을 대상으로 할 경우에 해당하는데, GLOVE에서 대상으로 하는 데이터가 multi-block으로 구성되어 있으므로 해당 특정 블록만 GLORE로 넘겨주면 된다.

③ 특정 position이 주어졌을 때, 그 position을 포함하는 블록의 접근 : 전송하는 데이터의 형태는 앞의 ②번과 동일하지만 앞의 경우는 어플리케이션이 블록을 지정하는 경우이고, 여기서는 3차원 공간상의 지점이 주어졌을 때 그 지점을 포함하는 블록을 검색해서 전달해준다는 차이가 있다.

마. 티켓

티켓은 GDM에 연결하는 GLORE와, GLORE가 수행하는 작업을 식별하기 위해 사용하는 32-bit 정수값이다. 티켓은 데이터 ID와 태스크 ID의 두 부분으로 나누어져서 상위 16-bit에는 데이터 ID, 하위 16-bit에는 해당 데이터에 대한 태스크 ID가 저장된다.

ISBN 978-89-6211-611-3

4. dmLib

가. 클래스

GLOVE의 렌더링 엔진인 GLORE에서 GDM에 연결하기 위하여 별도의 인터페이스가 필요한데, 이를 dmLib이라는 클래스로 구현했다. GLORE는 dmLib 클래스의 멤버 함수를 호출함으로써 GDM으로 하여금 필요한 명령을 수행하도록 지정할 수 있다. 또한 그리고 dmLib은 fetch된 data를 GLORE에서 사용할 수 있도록 vtkDataSet 타입으로 변환하는 역할도 담당한다.

나. dmlibManager 클래스

dmlibManager 클래스는 GLORE와 같은 외부 어플리케이션이 dmLib을 사용하기 위한 인터페이스를 모두 구현하고 있다.

(1) Private member functions

이 클래스의 private 멤버 함수의 대부분은 설정 파일을 읽기 위해 사용된다. 그리고 GDM으로부터 가져온 데이터를 vtkDataSet 타입으로 변환하기 위한 루틴(ConvertData)을 구현했다.

```
/// Parse XML-based configuration file
void ParseIPC( xmlDocPtr doc, xmlNodePtr cur );
void ParseQueue( xmlDocPtr doc, xmlNodePtr cur );
void ParseShm( xmlDocPtr doc, xmlNodePtr cur );
void ParseShmKey( xmlDocPtr doc, xmlNodePtr cur );
void ParseShmSize( xmlDocPtr doc, xmlNodePtr cur );
vtkDataSet *ConvertData (double *buffer)
```

<소스 4-1> dmlibManager 클래스의 private member function

ISBN 978-89-6211-611-3

(2) Public member functions

이 클래스의 public 멤버 함수가 바로 GLORE에 의해 호출되는 함수들로, GDM의 실행을 제어하는 데에 사용된다. 여기에는 GDM 설정 파일을 읽거나 GDM으로 하여금 실제 데이터를 읽도록 명령하는 함수가 있으며, 가장 중요한 함수로 GDM으로부터 데이터를 가져오도록 명령하는 Fetch 함수가 구현되어 있다.

```
int LoadConfig (char *filename);
int SetManager ();
unsigned int Load (char *filename);
unsigned int IssueTicket (unsigned int ticketId, int taskType);
int Unload (unsigned int ticketId);
vtkDataSet *Fetch(unsigned int ticket, int field, int ts);
vtkDataSet *Fetch (unsigned int ticket, int field, int ts, int element, int block);
vtkDataSet *Fetch (unsigned int ticket, int field, int ts, double x, double y, double z);
vtkDataSet *Fetch (unsigned int ticket, int field, int ts, double position[3]);
```

<소스 4-2> dmlibManager 클래스의 public member function

다. dmlibQueue

dmlibQueue는 dmLib와 GDM 사이의 통신 채널을 구현하고 있다. 앞에서 설명했듯이 dmLib와 GDM 사이의 명령어 전송에는 IPC message queue를 이용하고, 명령에 대한 파라미터와 데이터 전송은 IPC shared memory로 구현한다. 이 클래스의 함수들은 모두 dmLib 내부에서 호출하고, GLORE 와 같은 외부 어플리케이션이 사용하지는 않는다.

(1) Private member functions

이 클래스의 private 멤버 함수는 IPC message queue와 IPC shared memory를 만들거나 없애기 위한 용도로만 호출된다.

ISBN 978-89-6211-611-3

```
int OpenMsgQ (key_t key);
int RecvMsgQ (int id, void *msg, size_t size, long type, int flag);
int SendMsgQ (int id, void *msg, size_t size, int flag);
int CloseMsgQ (int msgqid);
int OpenShm (gipTransactionDB **db, char * tableName, key_t key, unsigned int size);
int OpenShm (gipVariableTrDB **db, char * tableName, key_t key, unsigned int size);
```

<소스 4-3> dmlibQueue 클래스의 private member function

(2) Public member functions

이 클래스에 구현되어 있는 public 멤버 함수는 IPC message queue와 IPC shared memory의 작동특성을 설정하는 함수가 대부분을 차지하고 있다. 이 함수들은 dmLib와 GDM 사이에 주고받는 데이터의 종류 별로 서로 다른 shared memory 채널을 만들도록 구성되어 있다.

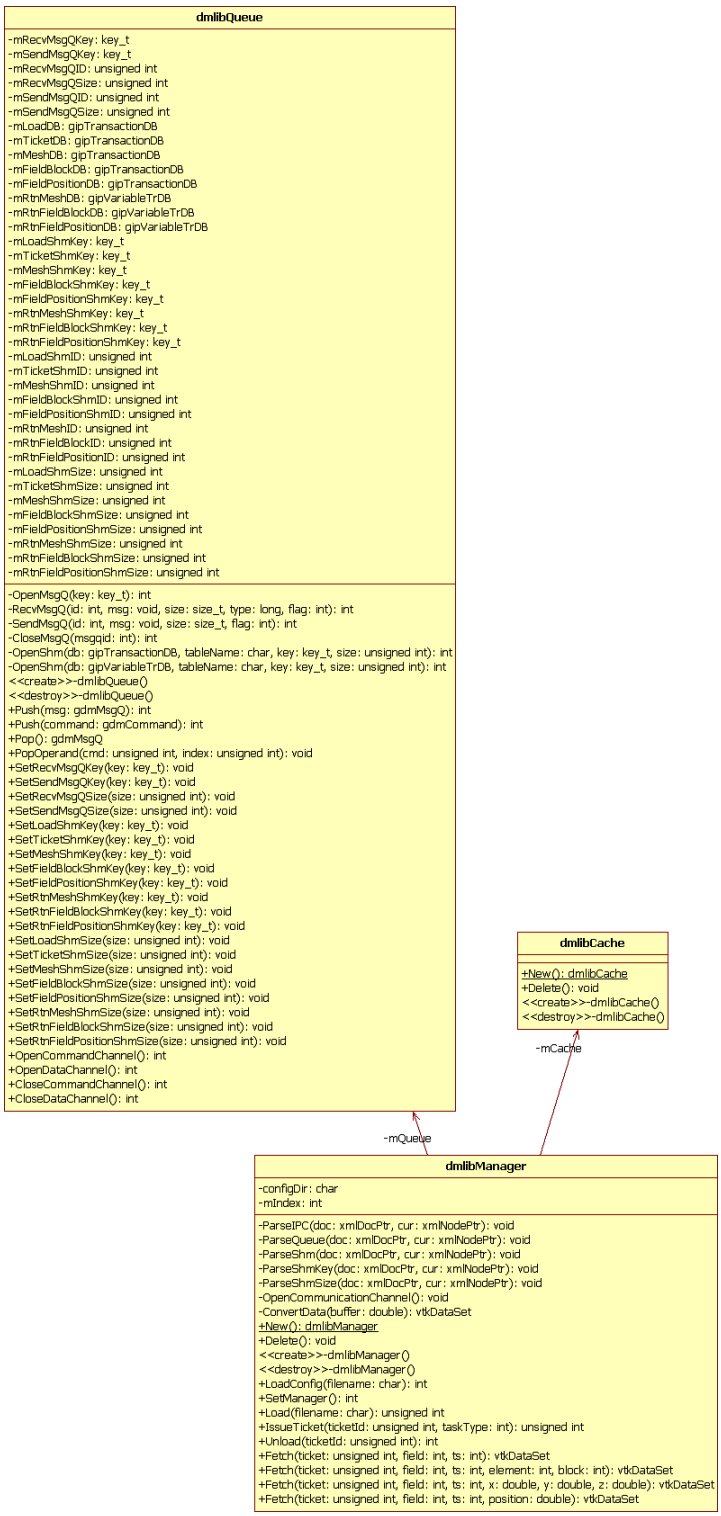
그리고 GDM으로 명령어를 보내거나 GDM으로부터 데이터를 받기 위한 함수가 구현되어 있다.

```
int Push( gdmMsgQ* msg ); /// Queueing Functions
int Push( gdmCommand * command );
gdmMsgQ* Pop();
void* PopOperand( unsigned int cmd, unsigned int index );
void SetRecvMsgQKey( key_t key );
void SetSendMsgQKey( key_t key );
void SetLoadShmKey( key_t key );
void SetTicketShmKey( key_t key );
void SetMeshShmKey( key_t key );
void SetFieldBlockShmKey( key_t key );
void SetFieldPositionShmKey( key_t key );
void SetRtnMeshShmKey( key_t key );
void SetRtnFieldBlockShmKey( key_t key );
void SetRtnFieldPositionShmKey( key_t key );
void SetLoadShmSize( unsigned int size );
void SetTicketShmSize( unsigned int size );
void SetMeshShmSize( unsigned int size );
void SetFieldBlockShmSize( unsigned int size );
void SetFieldPositionShmSize( unsigned int size );
void SetRtnMeshShmSize( unsigned int size );
void SetRtnFieldBlockShmSize( unsigned int size );
void SetRtnFieldPositionShmSize( unsigned int size );
int OpenCommandChannel();
```

<소스 4-4> dmlibQueue 클래스의 public member function

ISBN 978-89-6211-611-3

라. 클래스 다이어그램



<그림 4-1> dmLib의 클래스 다이어그램

ISBN 978-89-6211-611-3

5. GDM의 프로세스 구성

GDM의 프로세스는 크게 dmMessenger와 dmCore의 두 개로 구성되어 있다. GDM의 구현 초기단계에는 dmTicketMgr가 별도의 프로세스로 존재했는데 이제는 dmMessenger의 일부로 포함되어서 두 개의 프로세스로 줄일 수 있었다.

가. dmMessenger

(1) 개요

dmMessenger 프로세스는 dmTicketMgr로부터 명령을 받고 이 명령이 다른 노드들에게 필요한 명령일 경우 MPI_Bcast를 통해 다른 노드에 전달한다. 그리고 최종적으로 dmCore로 전송한다.

(2) gdmMessengerManager

gdmMessengerManager는 gdmMessenger의 실행을 전체적으로 관리하는 클래스로, IPC 관련 채널의 open/close, 설정 파일 읽기, 다른 프로세스로부터 명령을 받아서 전달하는 역할을 담당한다.

(가) Private member functions

gdmMessengerManager 클래스의 private 멤버 함수는 GDM 설정 파일을 읽기 위한 함수로만 구성되어 있다.

```
void ParseNode( xmlDocPtr doc, xmlNodePtr cur );
void ParseIPC( xmlDocPtr doc, xmlNodePtr cur );
void ParseQueue( xmlDocPtr doc, xmlNodePtr cur );
void ParseShm( xmlDocPtr doc, xmlNodePtr cur );
void ParseShmKey( xmlDocPtr doc, xmlNodePtr cur );
void ParseShmSize( xmlDocPtr doc, xmlNodePtr cur );
```

<소스 5-1> dmMessenger 클래스의 private member function

ISBN 978-89-6211-611-3

(나) Public member functions

Public 멤버 함수는 외부로부터 받은 명령어를 받아서 dmCore에 전달하는 함수 위주로 구성되어 있다.

```
int GetNumberOfNodes();
int GetRank();
bool IsManager();
void SetManager( bool isManager );
void OpenCommunicationChannel();
void CloseCommunicationChannel();
int LoadConfig( char * filename );
int FetchCommand();
int DispatchCommand();
int CollectCommand();
int ForwardCommand();
int ForwardLoadCommand (unsigned int cmd, unsigned int ticket, char * filename );
int ForwardTicketCommand (unsigned int cmd, unsigned int ticket, unsigned int tasktype );
```

<소스 5-2> dmMessenger 클래스의 public member functions

(3) gdmMessengerQueue

이 클래스는 gdmMessenger가 외부와의 데이터 교환을 위해 사용하는 IPC message queue와 IPC shared memory를 관리한다. 그 중에서도 실제로 데이터를 보내거나 받기 위해 필요한 루틴이 구현되어 있다.

(가) Private member functions

Private 멤버 함수는 가장 저수준(low-level)의 IPC 관련 I/O 함수들을 구현하고 있다.

```
int OpenMsgQ (key_t key);
int RecvMsgQ (int id, void *msg, size_t size, long tye, int flag);
int SendMsgQ (int id, void *msg, size_t size, int flag);
int CloseMsgQ (int msgqid);
int OpenShm (gipTransactionDB **db, char * tableName, key_t key, unsigned int size);
```

<소스 5-3> gdmMessengerQueue 클래스의 private member function

ISBN 978-89-6211-611-3

(나) Public member functions

Public 멤버 함수에서는 명령어의 타입에 따라서 서로 다른 communication channel을 만들거나 제거하는 루틴과, dmLib으로부터 전달받은 명령어/데이터를 다른 프로세스에게 전달하기 위한 루틴(Push)으로 분류할 수 있다.

```
int Push( gdmMsgQ* msg ); /// Queueing Functions
int Push( gdmCommand * command );
gdmMsgQ* Pop();
void* PopOperand( unsigned int cmd, unsigned int index );
void SetRecvMsgQKey( key_t key );
void SetLoadShmKey( key_t key );
void SetLoadShmSize( unsigned int size );
void SetTicketShmKey( key_t key );
void SetTicketShmSize( unsigned int size );
int OpenCommandChannel();
int OpenDataChannel();
int CloseCommandChannel();
int CloseDataChannel();
```

<소스 5-4> gdmMessengerQueue의 public member functions

ISBN 978-89-6211-611-3

(4) Class diagram

<그림 5-1> dmMessenger의 클래스 다이어그램

ISBN 978-89-6211-611-3

나. dmCore

(1) 개요

dmCore는 GLORE로부터의 명령을 최종적으로 전달받는 프로세스이며, 데이터 관리에 필요한 전반적인 GDM 명령을 처리한다. GDM에서 실제 데이터가 저장되는 곳은 GA 영역이기 때문에 dmCore는 GA에 대한 handle만 관리한다.

(2) gdmCoreManager

(가) Private member functions

gdmCoremanager의 private 멤버 함수는 모두 GDM 설정 파일을 읽는 함수로만 구성되어 있다.

```
void ParseNode( xmlDocPtr doc, xmlNodePtr cur );
void ParseIPC( xmlDocPtr doc, xmlNodePtr cur );
void ParseGA( xmlDocPtr doc, xmlNodePtr cur );
void ParseQueue( xmlDocPtr doc, xmlNodePtr cur );
void ParseShm( xmlDocPtr doc, xmlNodePtr cur );
```

<소스 5-5> gdmCoreManager의 private member functions

(나) Public member functions

gdmCoreManager의 멤버 함수는 다른 프로세스와의 통신 채널 제어, 실제로 전달받은 명령어 처리, GA 영역으로부터 데이터를 전달받아서 GLORE로 전달 등 실제 데이터 관리 및 입/출력을 위한 제반 API를 구현하고 있다.

ISBN 978-89-6211-611-3

```
int GetNumberOfNodes();
int GetRank();
void SetManager( bool isManager );
bool IsManager();
void OpenCommunicationChannel();
void CloseCommunicationChannel();
int LoadConfig( char * filename );
int FetchCommand (void);
int Process (void);
int Fetch (int data, int ts, int field);
int Fetch (int data, int ts, int field, int element, int block);
int Fetch (int data, int ts, int field, double x, double y, double z);
int Fetch (int data, int ts, int field, double position[3]);
```

<소스 5-6> gdmCoreManager의 public member function

(3) gdmCoreQueue

이 클래스는 gdmCoreManager가 외부와의 데이터 교환을 위해 사용하는 IPC message queue와 IPC shared memory를 관리한다. 그 중에서도 실제로 데이터를 보내거나 받기 위해 필요한 루틴이 구현되어 있다.

(가) Private member functions

Private 멤버 함수는 가장 저수준(low-level)의 IPC 관련 I/O 함수들을 구현하고 있다.

```
int OpenMsgQ( key_t key );
int RecvMsgQ( int id, void *msg, size_t size, long type, int flag );
int SendMsgQ( int id, void *msg, size_t size, int flag );
int CloseMsgQ( int msgqid );
```

<소스 5-7> gdmCoreQueue의 private member functions

(나) Public member functions

이 클래스의 public member function은 데이터 전달을 위해 만들어지는 IPC communication channel의 특성을 제어하거나 실제로 명령어/데이터를 외부로부터 받거나 외부로 내보내기 위해 구현된 함수로 구성되어

ISBN 978-89-6211-611-3

있다. 이 중 gipVariableTrDB라는, transaction database에 대한 I/O 함수는 대용량 데이터를 GLORE에 전달하기 위해 사용된다.

```
int Push( gdmMsgQ* msg ); /// Queueing Functions
int Push( gdmCommand * command );
gdmMsgQ* Pop();
void* PopOperand( unsigned int cmd, unsigned int index );
void SetRecvMsgQKey( key_t key );
void SetSendMsgQKey( key_t key );
unsigned int GetSendMsgQID();
void SetLoadShmKey( key_t key );
void SetTicketShmKey( key_t key );
void SetMeshShmKey( key_t key );
void SetFieldBlockShmKey( key_t key );
void SetFieldPositionShmKey( key_t key );
void SetRtnMeshShmKey( key_t key );
void SetRtnFieldBlockShmKey( key_t key );
void SetRtnFieldPositionShmKey( key_t key );
void SetLoadShmSize( unsigned int size );
void SetTicketShmSize( unsigned int size );
void SetMeshShmSize( unsigned int size );
void SetFieldBlockShmSize( unsigned int size );
void SetFieldPositionShmSize( unsigned int size );
void SetRtnMeshShmSize( unsigned int size );
void SetRtnFieldBlockShmSize( unsigned int size );
void SetRtnFieldPositionShmSize( unsigned int size );
gipVariableTrDB* GetRtnMeshDB();
gipVariableTrDB* GetRtnFieldBlockDB();
gipVariableTrDB* GetRtnFieldPositionDB();
```

<소스 5-8> gdmCoreQueue의 public member functions

(4) gdmLoader 클래스

gdmLoader는 디스크에 저장된 데이터를 읽어서 gdmDatapool에 저장한다. 이 클래스의 특징은 parallel filesystem에 여러 개의 파일로 나누어 저장되어있는 데이터를 여러 노드가 동시에 읽어서 gdmDatapool에 등록하도록 구현되어 있다는 점이다.

(가) Private member functions

이 클래스의 private 멤버 함수는 크게 두 가지로 분류할 수 있다. 하나

ISBN 978-89-6211-611-3

는 GDM 설정 파일을 읽는 것으로 단순한 XML 파서를 구현한 것이다. 또 다른 하나는 파일 시스템에서 데이터를 읽기 위해 구현한 함수들인데, TraverseDirectory와 ComputeTarget의 두 함수로 나누어서 구현되어 있다. 전자는 데이터가 저장된 디렉터리와 그 하위 디렉터리를 모두 추적해서 전체 데이터를 불러들이기 위해 필요한 파일이 몇 개가 있는지 확인한다. 이 확인이 끝나면 ComputeTarget 함수에서 GDM을 구성하고 있는 MPI 태스크의 수를 확인해서 각 태스크가 비슷한 개수의 파일을 읽도록 노드별로 분배해주는 역할을 담당한다.

```
void ParseFilename (char *fullpath);
/// Metadata
int ParseMetadata (char *filename, gdmMetadata *meta); /// Load metadata
void ParseGrid (xmlDocPtr doc, xmlNodePtr cur);
int ParseTimeStep (xmlDocPtr doc, xmlNodePtr cur);
int ParseElementCount (xmlDocPtr doc, xmlNodePtr cur);
void ParseElement (xmlDocPtr doc, xmlNodePtr cur, gdmMetadata *meta);
int ParseID (xmlDocPtr doc, xmlNodePtr cur);
void ParseName (xmlDocPtr doc, xmlNodePtr cur);
int ParseBlockCount (xmlDocPtr doc, xmlNodePtr cur);
int ParseValueCount (xmlDocPtr doc, xmlNodePtr cur);
int ParseBlock (xmlDocPtr doc, xmlNodePtr cur, META_BLOCK *block);
void ParseDimension (xmlDocPtr doc, xmlNodePtr cur, int dim[3]);
int ParseX (xmlDocPtr doc, xmlNodePtr cur);
int ParseY (xmlDocPtr doc, xmlNodePtr cur);
int ParseZ (xmlDocPtr doc, xmlNodePtr cur);
void ParseValue (xmlDocPtr doc, xmlNodePtr cur, gdmMetadata *meta);
int ParseType (xmlDocPtr doc, xmlNodePtr cur);
/// Get list of data files + compute distribution factor
int TraverseDirectory (void);
int ComputeTarget (gdmMetadata *meta);
```

<소스 5-9> gdmLoader 클래스의 private member functions

(나) Public member functions

Public 멤버 함수는 메타데이터를 읽는 함수와 실제 데이터를 읽는 함수 두 가지만 구현되어 있다. 데이터를 읽는 함수 Load는 앞에서 설명한, ComputeTarget 함수에서 분배해준 파일 목록을 갖고 디스크로부터 데이터를 읽도록 구현되어 있다.

ISBN 978-89-6211-611-3

```
/// Load metadata & actual dataset
int LoadMetadata (char *filename, gdmDatapool *pool);
int Load (gdmDatapool *pool);
```

<소스 5-10> gdmLoad 클래스의 public member functions

(5) gdmDatapool 클래스

하나의 gdmDatapool 클래스는 하나의 시뮬레이션 데이터를 저장/관리하기 위해 사용한다.

(가) Private member functions

Private 멤버 함수로 구현되어 있는 CreateGA와 DestroyGA는 나중에 설명할 gdmData 클래스의 동일한 명칭을 갖는 함수에 대응된다.

```
void CreateGA (void);
void DestroyGA (void);
int GetGA( int handle, int lo[], int hi[], int ld[] );
```

<소스 5-11> gdmDatapool 클래스의 private member functions

(나) Public member functions

gdmDatapool이 전체 데이터셋을 관리하는만큼 다양한 종류의 멤버 함수가 구현되어 있다. 이 중 가장 중요한 함수들은 Fetch와 Register 관련 함수다. Fetch는 dmLib이 지정하는 데이터를 GA 영역으로부터 가져와서 dmLib에 돌려주는 역할을 담당한다. 이 함수는 다양한 형태로 데이터를 특정할 수 있도록 구현되어 있다. 그리고 Register 함수는 디스크에 저장되어 있는 데이터를 읽어서 GA 영역에 저장할 때 사용된다.

ISBN 978-89-6211-611-3

```
void CopyMetadata (void);
gdmMetadata *GetMetadata (void);
/// Register metadata & dataset
int Register (gdmMetadata *meta);
int Register (std::string name, int ts, int field, int element, int block, doub
le *value);
/// Fetch dataset - applications should use 'Fetch'
int Fetch (int ts, int field);
int Fetch (int ts, int field, int element, int block);
int Fetch (int ts, int field, double x, double y, double z);
/// Set space for shared memory
void SetRtnMeshDB( gipVariableTrDB *db, int index );
void SetRtnFieldBlockDB( gipVariableTrDB *db, int index );
void SetRtnFieldPositionDB( gipVariableTrDB *db, int index );
/// Set message queue id
void SetSendMsgQID( unsigned int id );
```

<소스 5-12> gdmDatapool 클래스의 public member functions

(6) gdmElement 클래스

gdmElement 클래스는 데이터의 hierarchy를 표현하기 위해 구현되어 있는 클래스다.

(가) Public member functions

이 클래스의 멤버 함수는 모두 gdmData의 public 멤버 함수에 대한 wrapper와 유사한 역할을 담당한다, 그렇기 때문에 gdmData 클래스와 거의 동일한 함수들이 구현되어 있다.

```
int Key (void);
// Set properties
void SetType (int field, int type);
void SetDimension (int field, int dim);
void SetTimeStep (int count);
void SetDataCount (int count);
void SetBlockCount (int count);
void SetBlockDimension (int block, int x, int y, int z);
// Register dataset
void Register (std::string name, int ts, int field, int block, double *data);
// Fetch dataset
double *Fetch (int ts, int field, int block);
```

<소스 5-13> gdmElement 클래스의 public member functions

ISBN 978-89-6211-611-3

(7) gdmData 클래스

gdmData 클래스는 GDM의 상위 계층 루틴이 보기에는 시뮬레이션 데이터를 제공해주지만 실제로는 GA 영역의 생성과 제거, 데이터 저장 등 GA 영역을 대상으로 하는 모든 종류의 작업을 수행하는 클래스다.

(가) Private member functions

GA 영역을 생성/제거하기 위한 함수가 구현되어 있다.

```
/// Management of global array space
void CreateGA ();
void DestroyGA ();
```

<소스 5-14> gdmData 클래스의 private member functions

(나) Public member functions

gdmData 클래스의 public 멤버 함수는 GA 영역에 저장되는 데이터의 특성을 정의하기 위한 목적으로 사용된다. 여기에는 timestep의 수, 멀티 블록 데이터의 경우 블록의 수, 블록의 해상도 등 실제로 데이터를 접근하기 위해 필요한 모든 종류의 함수가 구현되어 있다.

```
// Set properties
void SetDimension (int dim);
void SetTimeStep (int count);
void SetBlockCount (int count);
void SetName (std::string name);
/// Block-wise information
void SetBlockDimension (int block, int x, int y, int z);
/// Load dataset to global array space
/// value is a (logically) 3D array of (x, y, z)
void Register (int ts, int block, double *value);
/// Fetch dataset
double *Fetch (int ts, int block);
```

<소스 5-15> gdmData 클래스의 pubilc member functions

ISBN 978-89-6211-611-3

6. 결론

지금까지 GLOVE의 데이터 관리자인 GDM(GLOVE Data Manager)에 대해 설명했다. GDM은 분산 메모리 구조를 갖는 클러스터에서 큰 공유 메모리 공간을 만들어줌으로써 visualization 어플리케이션이 손쉽게 대용량 데이터를 다룰 수 있도록 해준다는 장점이 있다. 앞으로 다양한 플랫폼에서의 성능 평가 등을 통해 데이터의 입/출력 부분을 더욱 최적화하고, 이를 기반으로 더 빠른 응답시간을 갖는 GLOVE를 개발할 예정이다.