Open MPI 를 이용한분산/병렬 교통 시뮬레이션 프로그램 개발에 관한 연구

조민규, 경민기, 신인수, 민덕기[1]
건국대학교 컴퓨터공학과
e-mail: { whalsrb1226, moonend, readen, dkmin}@kokkuk.ac.kr

A Study on Developing Distributed and Parallel Traffic Simulation Program with Open MPI

Min-Kyu Cho, MinGi Kyung, In-soo Shin, Dug-Ki Min*
College of Computer Science,
Konkuk University, South Korea

요 약

교통 시뮬레이션 시스템은 현실 세계의 교통 및 차량 관련 데이터를 기반으로 미래의 차량 움직임을 예측하는 프로그램으로, 다양한 교통문제를 해결을 위한 도구가 될 수 있다. 시뮬레이션 스케일을 전국단위로 확장하기 위해서 분산/병렬 시스템을 도입해야 하는데, 이 논문에서는 병렬/분산 과정에서 핵심이 되는 Open MPI 기반의 데이터 교환에 대한 방법을 제안하고자 한다. 공통된 하나의 커뮤니케이션 모듈을 기반으로 분산된 노드의 데이터 교환에 대한 문제를 해결하여 생산성을 높이고, 시뮬레이션 과정에서 소요되는 커뮤니케이션 타임을 줄여줄 것으로 예상된다.

1. 서론

최근 다양한 도심의 교통문제를 해결하기 위한 연구가 각 분야에서 활발히 진행되고 있다. 본 연구팀은 교통정보 데이터를 통합하고 활용하여 도심의 트래픽을 예측하고, 클라우드 인프라의 확장성을 갖춘 아키텍처를 도입하여 교통 시스템 및 교통 정책을 사전 검증할 수 있는 트래픽 예측 시뮬레이션 기술 개발을 목표로 연구를 진행하고 있다. 연구 초기에는 하나의 구에 대한 시뮬레이션을 목표로 진행되어 하나의 노드(컴퓨터)에서 시뮬레이션을 하는 것이 가능하였다. 하지만 이제는 현실 세계를 반영하여 전국 단위의 빅데이터를 사용하는 효율적인 시뮬레이션 시스템을 구축해야 하는 단계이다. 그리고 이러한 시스템을 구축하기 위해서는 분산/병렬 시스템에 대한 연구가 불가피하다.

이 논문에서는 교통 시뮬레이션 시스템 구축을 위해 사용되는 실제 데이터를 바탕으로 분산/병렬 시뮬레이션의 필요성에 대해 서술하고, 분산/병렬 노드 간의 통신을 위해 작성된 Open MPI 기반의 차량 데이터 교환 모듈에 대한 내용을 다룬 후에, 이 모듈을 도입함으로써 얻을 수 있는 시뮬레이션 타임 스텝에 대한 소요 시간 감소의 가능성을 확인한다.

2. 교통 시뮬레이션을 위한 데이터

교통 시뮬레이션을 하기 위해서는 현실 세계의 데이터가 필요하다. 본 논문에서 사용한 데이터는 크게 3 가지로, 각각은 차량의 정보를 담고 있는 Vehicle, 교차로와 연결 없는 도로 그 자체인 Link, 그리고 교차로에서 이루어지는 링크 연결 기능을 수행하는 Connection Cell 이다. 이 논문에서 사용하는 각 데이터의 개수와 크기는 표 1 에서 자세하게 확인할 수 있다.

	강동구	강남 4 구(통합)
Vehicle 개수	3568 개	195193 개
Vehicle 데이터 크기	5MB	673MB
Link 개수	12448 개	62441 개
Link 데이터 크기	364MB	2333MB
Connection Cell 개수	16096 개	61383 개
Connection Cell 데이터 크기	72MB	2822MB

<표 1 – 시뮬레이션 데이터>

[1] 교신저자. 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. 2017-0-00121, 도시 교통 문제 개선을 위한 클라우드 기반 트래픽 예측 시뮬레이션 SW 기술 개발)

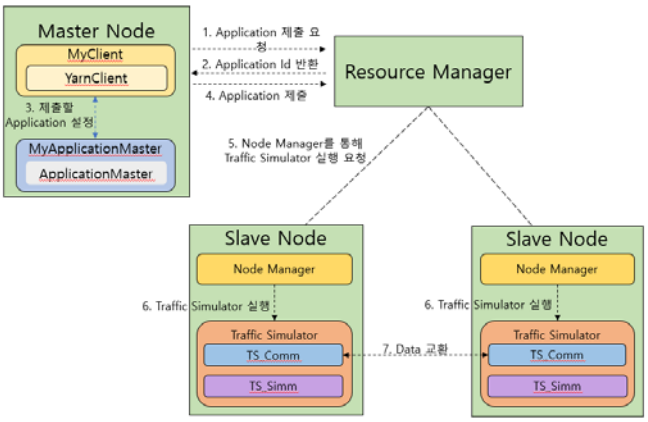
위 표는 이 논문에서 사용되는 현실 세계의 데이터에 대한 크기와 개수를 보여준다. Vehicle, Link, Connection Cell 데이터는 SUMO 기반의 교통 데이터에서 분산/병렬 과정을 위해 partitionOwner 라는 개념이 추가된 SALT 기반의 데이터로부터 시뮬레이션 연구를 위해 만들어낸 데이터이다.
또한 위의 표에서 보이는 강남 4 구 데이터는 서초구, 강남구, 송파구, 강동구를 통합한 데이터이다.

3. 분산/병렬 시뮬레이션의 필요성

초기의 교통 시뮬레이션 프로그램 개발에 대한 연구는 여러 개의 구들 중 하나인 강동구를 목표로 진행이 되었다. 강동구에 대한 데이터는 <표 1> 에서 확인할 수 있듯이 Vehicle 데이터가 5MB, Link 데이터가 364MB, Connection Cell 데이터가 72MB 정도로, 작지는 않은 규모지만 1 대의 컴퓨터로 처리가 가능한 수준이기 때문에 분산/병렬 컴퓨팅에 대한 연구의 필요성은 부각되지 않았었다.
하지만 교통 시뮬레이션에 대한 연구가 해를 거듭하면서 2019 년에 3 차년도까지 이르게 되었고, 이제는 도심 단위의 교통 시뮬레이션으로 확장하게 되었다. 하여 2019 년도에는 강남 4 구 데이터를 사용하는데, <표 1>에서 확인할 수 있듯이 Vehicle 데이터가 673MB, Link 데이터가 2333MB, Connection Cell 데이터가 2822MB 정도로 총합 약 5828MB 정도가 되는 크기로 1 대의 컴퓨터로 처리하기에는 버거운 양이 되었다.
Link 의 개수를 늘려가면서 시뮬레이션을 실행한 결과 Link 의 개수가 2000 개일 때는 시뮬레이션 클록 당 소요되는 시간이 약 50 초 정도이고, Link 의 개수가 10000 개일 때는 시뮬레이션 클록 당 소요되는 시간이 약 410 초 정도임을 확인할 수 있었다. 즉, Link 의 개수가 증가함에 따라 시뮬레이션에 소요되는 시간이 급증하고 있음을 실험을 통해 확인하였다.
이러한 실험 결과를 통해 교통 시뮬레이션을 수행하려고 할 때, 도로의 수와 차량 수에 따라 시뮬레이션 클록 당 소요되는 시간이 점차 증가하기 때문에, 시뮬레이션 면적을 적절히 조절하여 분산 처리를 수행할 필요성이 부각되었고, 분산/병렬 시뮬레이션 컴퓨팅에 대한 연구를 시작하게 되었다.

4. 분산/병렬 교통 시뮬레이션 아키텍처와 문제점

분산/병렬 교통 시뮬레이션을 위해 설계된 아키텍처는 아래의 그림 1 과 같다. 전체 시뮬레이션 시스템은 YARN 을 이용하여 자원을 관리하고, Application 을 제출한 후에 MPI 기반으로 통신을 한다.
Master Node 는 Map 데이터를 분할하고 전체 시뮬레이션을 구동하는 등의 역할을 한다. 그리고 시뮬레이션을 구동하는 Slave Node 는 실제 차량의 움직임을 GPU 에서 시뮬레이션 하는 TS_Simm 와 시뮬레이션 전/후에 이동한 차량을 전송하거나 결과를 출력하고 CPU 의 메모리에 있는 차량 데이터를 시뮬레이션을 위해 GPU 로 올려주는 등의 역할을 하는 TS_Comm 으로 구성된다. 즉 TS_Comm 에서 시뮬레이션을 위해 필요한 데이터를 CPU 의 메모리에 가지고 있으며, 데이터 교환 등의 작업을 하고, 다시 시뮬레이션을 해야 할 때 CPU 에 있는 차량 데이터를 GPU 로 올려준다. TS_Simm 에서는 GPU 에 올라간 차량 데이터를 바탕으로 실제 시뮬레이션을 구동한다.
이러한 구조로 작성된 전체 시뮬레이션에서 소요되는 시간을 분석한 결과를 TS_Simm 모듈에서 GPU 로 실제 시뮬레이션을 수행하는 Computation Time 은 8% 정도밖에 되지 않으며, 전체 시뮬레이션 소요시간의 92%에 해당하는 Communication Time 은 Link 와 Link 사이의 데이터를 전달하는 과정에서 소요되고 있다. 시스템을 설계하고 구현하여 실제 시뮬레이션을 구동한 결과, 노드와 노드 사이에 데이터를 교환하는 과정이 상당히 많은 시간을 소요함을 알 수 있었고, 빠르고 정확한 차량 데이터 교환을 위한 Open MPI 기반의 커뮤니케이션 모듈의 필요성이 대두되었다
실제 교통 시뮬레이션 연구 과정에는 다양한 데이터 전달 관련 알고리즘 방법에 대한 연구 역시 포함되어 있지만, 이 논문에서는 Open MPI 기반의 차량 데이터 교환 모듈에 관한 내용만을 다룬다.



(그림 1) 교통 시뮬레이션 아키텍처 사진

그림 1 은 전체 교통 시뮬레이션 시스템의 흐름도를 보여준다. Master Node 는 시뮬레이션을 실행하기 이전에 필요한 작업들(맵 데이터 분할, 시뮬레이션 노드 초기화 등)을 수행한 후에 Application 을 Resource Manager(YARN)로 제출한다. Master Node 는 Resource Manager 가 반환한 Application Id 를 받은 후에 제출할 Application 을 설정하여 실제 Application 을 제출한다. 이러한 과정 후에 Resource Manager 는 실행가능한 노드들을 확인한 후에 자원을 할당하여 시뮬레이션을 구동할 Slave Node 들에게 Simulation 실행 요청을 전달한다. Slave Node 의 Node Manager 는 요청을 받은 후에 Traffic Simulator 를 실행한다.
TS_Comm 모듈에서는 데이터 교환, 경계 부분의 데이터 처리, 결과 로그 남기기 등의 역할을 CPU 에서 수행한다. 반면에 TS_Simm 모듈에서는 일정한 클록 단위로 GPU 에서 실제 시뮬레이션을 수행한다.

5. Open MPI 기반의 차량 데이터 교환 모듈

구조체 형태의 자료구조를 갖는 차량 정보를 Open MPI 기반으로 교환하기 위해서는 MPI_Datatype 과 typeLen 을 지정하는 방법 등 다양한 방법이 있지만, 개발의 효율성과 데이터 교환의 정확성을 위해 Byte 단위로 Serialize 하여 데이터를 교환하는 방법을 고안하였다.

Byte 단위로 차량 데이터를 전송하기 위해서는 MPI 함수에 차량 구조체의 크기와 전달할 차량의 수를 곱하여 전달할 크기를 명시해주어야 한다. 또한 Byte 단위로 차량 데이터를 수신하기 위해서는 수신할 차량의 데이터만큼 메모리의 영역을 확보해야 한다.

전송을 하기 위한 데이터의 크기를 Byte 단위로 명시함으로써 필요한 만큼의 메모리를 확보하여 효율성을 높일 수 있고, 전달하겠다고 받은 크기가 구조체의 크기로 나누어 떨어지지 않을 경우 전송할 크기를 다시 요청함으로써 정확한 데이터 교환을 확보할 수 있다. 이러한 내용을 바탕으로 Open MPI 기반의 차량 데이터 교환 모듈인 TS_Comm 을 개발하였다.

TS_Comm 모듈은 여러 노드들에서 MPI 를 사용하기 위해 size 와 rank 를 초기화하는 Init_Communicator, 차량 데이터를 송/수신하는 Send_Vehicle, Recv_Vehicle, 차량 데이터를 Scatter, Gather, Broadcast 하는 Sactter_Vehicle, Gather_Vehicle, Broadcast_Vehicle 그리고 수신할 차량의 수를 Broadcast 하는 Broadcast_Num 으로 구성되며 이와 관련한 정보는 그림 2 에서 확인할 수 있다.

```
/*-----------------------------------------------------------------*/
/// @brief   Function to initialize MPI rank and size
/*-----------------------------------------------------------------*/
void Init_Communicator();


/*-----------------------------------------------------------------*/
/// @brief   Function to braodcast Number
/*-----------------------------------------------------------------*/
void Broadcast_Num(int* num, int rank);


/*-----------------------------------------------------------------*/
/// @brief   Function to send vehicle data to another node
/*-----------------------------------------------------------------*/
void Send_Vehicle(vector<vehicle>* send_v, int rank);


/*-----------------------------------------------------------------*/
/// @brief   Function to receive vehicle data from another node
/*-----------------------------------------------------------------*/
void Recv_Vehicle(vector<vehicle>* recv_v, int rank);


/*-----------------------------------------------------------------*/
/// @brief   Function to scatter vehicle data
/*-----------------------------------------------------------------*/
void Scatter_Vehicle(vector<vehicle>* send_v, vector<vehicle>* recv_v, int rank);


/*-----------------------------------------------------------------*/
/// @brief   Function to scatter vehicle data
/*-----------------------------------------------------------------*/
void Gather_Vehicle(vector<vehicle>* send_v, vector<vehicle>* recv_v, int rank);


/*-----------------------------------------------------------------*/
/// @brief   Function to Broadcast vehicle data
/*-----------------------------------------------------------------*/
void Broadcast_Vehicle(vector<vehicle>* v, int rank);
```

(그림 2) TS_Comm 헤더 파일

그림 2 는 TS_Comm 의 헤더 파일로, 작성된 차량 교환 모듈의 함수들을 보여준다. 교환할 차량들은 c++ STL 인 vector 자료구조에 저장하고 있다. MPI 기반으로 차량을 송/수신하기 위해 초기화 관련 함수를 제외한 모든 함수의 파라미터에는 rank 가 포함된다.

6. Open MPI 기반의 테스트 예제

6.1 실험 환경

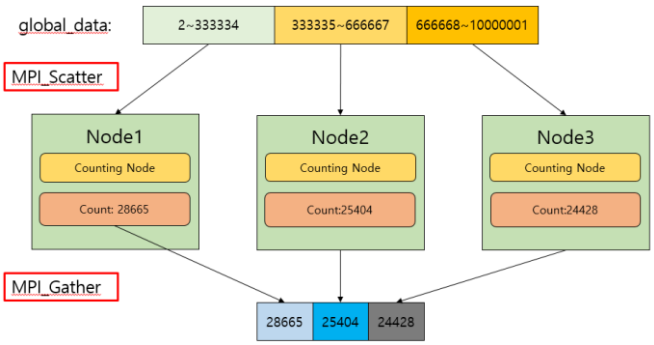
Open MPI 기반의 테스트 코드를 구동하기 위해 Ram 2GB 를 갖는 Ubuntu 16.04 환경의 물리 머신을 1 대부터 3 대까지 사용하였다. 실행한 테스트 코드는 2~1000001 사이에 존재하는 1000000 개의 정수에서 소수의 개수를 구하는 예제이고 Machine 의 개수를 1 개에서부터 시작하여 3 개까지 늘린 성능을 비교하였다.

6.2 실험 결과

2~1000001 까지의 데이터를 Machine 들이 나누어 갖기 위해서는 Scatter 를 사용하였고, 각각의 Machine 에서 구한 소수의 개수를 병합하기 위해서는 Gather 를 사용하였다. 3 개의 Machine 을 사용하여 위의 예제를 실행한 결과는 그림 3 과 같다. 일정한 크기 간격으로 분할받은 데이터에서 소수의 개수를 구하고 Master Node 로 결과를 반환하여 Master Node 가 최종 소수의 개수를 취합한다.

표 2 를 참고하면 1 개의 Machin 부터 3 개의 머신을 사용한 결과를 확인할 수 있는데, 여러 개의 머신을 사용할수록 시간이 기하급수적으로 감소하는 이유는 병렬처리가 되기 때문이다. 병렬처리가 되면서 시간이 줄어드는 자세한 이유는 그림 4 에서 확인할 수 있다.

이러한 MPI 기반의 실험 결과를 통해 분산/병렬 컴퓨팅 기반의 성능 향상을 확인할 수 있으며, 실제 교통 시뮬레이션 차량 교환에 관련한 성능 향상을 기대할 수 있다.



(그림 3) 3 개의 Machine 에서 소수 구하는 도식

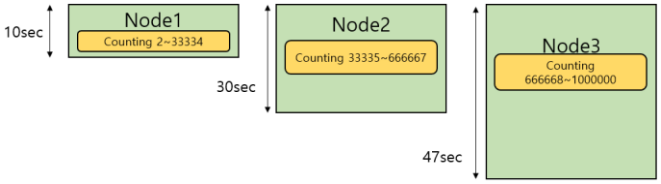
그림 3 은 3 개의 Machine 에서 소수를 나누어 구하는 과정을 도식으로 보여주고 있다. 각각의 Machine 들은 일정한 양의 데이터를 할당받게 되고, 소수의 개수를 구해서 Master Node 에게 결과를 반환하고 있다. 그리

고 Master Node 는 모든 결과를 받은 후에 최종 결과를 더하여 전체 소수의 개수를 갖게 된다.

	1 Machine		2 Machine		3 Machine	
	탐색 범위	시간	탐색 범위	시간	탐색 범위	시간
탐색	2~1000001	89.0702초	2 ~ 500002	23.41 초	1~333334	10.84초
			500002 ~ 1000001	65.76초	333335 ~ 666667	30.00초
					666668 ~ 1000001	47.04초
총 시간	89.070229초		67.678947 초		47.123734 초	

<표 2> 여러 개의 Machine 에서 소수 구하기 결과

표 2 는 소수를 구하는 예제를 1 개의 컴퓨터부터 3 개의 컴퓨터를 활용한 결과를 보여주고 있다. 표 2 의 결과를 통해 분산/병렬 시뮬레이션 프로그램을 구성하여 얻을 수 있는 시간적 효율성을 확인할 수 있다.



(그림 4) 여러 개의 Machine 에서 소수 구하기 결과

그림 4 는 3 개의 Machine 을 활용하여 소수를 구하는 과정을 병렬처리 하였을 때 시간적 효율성이 발생하는 과정을 보여주고 있다. 동일한 시간에 여러 개의 컴퓨터가 동시에 소수를 구하는 일을 하므로 처리 속도가 확연히 줄어듬을 확인할 수 있다. 이러한 결과를 통해 동시에 여러 컴퓨터에서 시뮬레이션을 하면 처리 속도를 상당히 높여줄 것을 기대할 수 있다.

7. 결과

본 논문에서는 Open MPI 기반의 차량 데이터 교환 모듈에 대한 내용을 소개하였다. 한 대의 구 시뮬레이션에서 도심 단위의 시뮬레이션으로 확장됨에 따라 분산/병렬 시뮬레이션에 대한 연구의 필요성이 대두되었고, 빠르고 정확한 데이터 교환을 위해 Open MPI 를 사용하게 되었다. 다양한 Open MPI 의 데이터 교환 방법들 중에서 Byte 단위로 Serialize 하는 방법을 선택하여, 빠르고 정확한 차량 데이터 교환을 위한 TS_Comm 모듈을 개발하였다. 향후 연구는, 해당 모듈을 실제 분산/병렬 시뮬레이션에 도입하여 노드들을 동기화(Synchronize)하고 Time-Step 을 맞추어주는 방향으로 진행할 필요성이 있다.

참고문헌

[1] MPI-Forum Documentation.
https://www.mpi-forum.org/docs/

[2] Denio MPI, The Greate and Terrible implementation of MPI-2.
https://mpi.deino.net

[3] MPITutorials, A Comprehensive MPI Tutorial Resource.
https://mpitutorial.com

[4] Derived Data Types in MPI, Dilip Angom, 2005, Theorical Physics and Complex Systems Physical Research Laboratory.

[5] Open MPI.
https://www.open-mpi.org/

[6] netlib, mpi-book.
http://www.netlib.org/utk/papers/mpi-book/

[7] Apache Hadoop Yarn, Apach Software Foundation.
https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[8] edureka, Hadoop Yarn Tutorial.
https://www.edureka.co/blog/hadoop-yarn-tutorial/

[9] YARN Essentials, Nirmal Kumar, Amol Fasale, 2015