JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 1, pp. 1-11, January 2020
https://doi.org/10.9708/jksci.2020.25.01.001

A Study on Efficient Executions of MPI Parallel Programs in Memory-Centric Computer Architecture

Je-Man Lee*, Seung-Chul Lee*, Dongha Shin**

*Student, Dept. of Computer Science, Sangmyung University, Seoul, Korea
*Student, Dept. of Computer Science, Sangmyung University, Seoul, Korea
**Professor, Dept. of Electronics, Sangmyung University, Seoul, Korea

[Abstract]

In this paper, we present a technique that executes MPI parallel programs, that are developed on processor-centric computer architecture, more efficiently on memory-centric computer architecture without program modification. The technique we present here improves performance by replacing low-speed data communication over the network of MPI library functions with high-speed data communication using the property called fast large shared memory of memory-centric computer architecture. The technique we present in the paper is implemented in two programs. The first program is a modified MPI library called MC-MPI-LIB that runs MPI parallel programs more efficiently on memory-centric computer architecture preserving the semantics of MPI library functions. The second program is a simulation program called MC-MPI-SIM that simulates the performance of memory-centric computer architecture on processor-centric computer architecture. We developed and tested the programs on distributed systems environment deployed on Docker based virtualization. We analyzed the performance of several MPI parallel programs and showed that we achieved better performance on memory-centric computer architecture. Especially we could see very high performance on the MPI parallel programs with high communication overhead.

▸ **Key words**: Computer Architecture, Memory-centric Computer Architecture, Parallel Programming, MPI, Simulation

[요 약]

본 논문에서는 프로세서 중심 컴퓨터 구조에서 개발된 MPI 병렬 프로그램을 수정하지 않고 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행시키는 기술을 제안한다. 본 연구에서 제안하는 기술은 메모리 중심 컴퓨터 구조가 가지는 빠른 대용량 공유 메모리 특징을 이용하여 MPI 표준 라이브러리 함수가 수행하는 네트워크 통신을 통한 느린 데이터 전달을 공유 메모리를 통한 빠른 데이터 전달로 대체하여 효율성을 얻는다. 본 연구에서 제안한 기술은 두 개의 프로그램에 구현되었다. 첫 번째 프로그램은 MC-MPI-LIB라고 불리는 수정된 MPI 라이브러리인데 이는 기존 MPI 표준 라이브러리 함수의 의미를 유지하면서 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행한다. 두 번째 프로그램은 MC-MPI-SIM이라고 불리는 시뮬레이션 프로그램인데 이는 프로세서 중심 컴퓨터 구조 상에서 메모리 중심 컴퓨터 구조의 수행을 시뮬레이션한다. 본 논문에서 제안한 기술은 도커 가상화 상에서 구현된 분산 시스템 환경에서 개발하고 시험하였다. 다수의 MPI 병렬 프로그램을 이용하여 제안한 기술의 성능을 측정한 결과 메모리 중심 컴퓨터 구조에서 더 높은 성능으로 수행 가능함을 보였으며, 특히 통신 오버헤드 비율이 높은 MPI 병렬 프로그램의 경우 매우 높은 성능으로 수행 가능하다는 점을 확인하였다.

▸ **주제어**: 컴퓨터 구조, 메모리 중심 컴퓨터 구조, 병렬 프로그래밍, MPI, 시뮬레이션

• First Author: Je-Man Lee, Corresponding Author: Dongha Shin
*Je-Man Lee (frad0211@naver.com), Dept. of Computer Science, Sangmyung University
*Seung-Chul Lee (chewin9@naver.com), Dept. of Computer Science, Sangmyung University
**Dongha Shin (dshin@smu.ac.kr), Dept. of Electronics, Sangmyung University
• Received: 2019. 09. 30, Revised: 2019. 11. 22, Accepted: 2019. 11. 25.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

최근 들어 컴퓨터가 양적으로 급증하는 데이터를 효율적으로 처리하지 못하는 현상이 발생함에 따라[1][2] 기존의 프로세서 중심 컴퓨터 구조를 대체하는 메모리 중심 컴퓨터 구조에 대한 개발이 시작되고 있다[3][4].

메모리 중심 컴퓨터 구조의 가장 큰 특징은 빠른 대용량 공유 메모리이다[1][2]. 이는 기존 네트워크 통신을 통한 느린 데이터 전달을 공유 메모리를 통한 빠른 데이터 전달로 대체하여 병렬 프로그램에서 많이 사용하는 MPI(Message Passing Interface)를 프로세서 중심 컴퓨터 구조에서 수행했을 때 보다 더 효율적으로 수행할 수 있는 환경을 제공한다.

메모리 중심 컴퓨터 구조가 등장하더라도, 프로세서 중심 컴퓨터 구조에서 개발된 MPI 병렬 프로그램[5]은 메모리 중심 컴퓨터 구조의 특징을 활용하여 개발된 것이 아니기 때문에 재작성되거나 수정되어야 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행될 수 있다는 문제점이 있다. 본 연구에서는 기존 프로세서 중심 컴퓨터 구조에서 개발된 MPI 병렬 프로그램을 수정하지 않고 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행시키는 기술과 얼마나 더 효율적으로 수행 가능한지 미리 알아볼 수 있는 기술을 제안한다.

본 연구에서 제안하는 기술은 크게 두 가지로 구성된다. 첫 번째는 기존 MPI 표준 라이브러리 함수[6][7]의 의미를 유지하면서 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행하는 수정된 MPI 라이브러리 함수(MC-MPI-LIB)의 개발이다. 두 번째는 프로세서 중심 컴퓨터 구조상에서, MPI 병렬 프로그램에 대하여, 메모리 중심 컴퓨터 구조의 수행을 시뮬레이션하는 기술(MC-MPI-SIM)이다. 첫 번째 기술은 메모리 중심 컴퓨터 구조에서 MPI 병렬 프로그램을 더 효율적으로 수행하는 기술이고, 두 번째 기술은 메모리 중심 컴퓨터 구조상에서 MPI 병렬 프로그램이 얼마나 더 효율적으로 수행하는지 알아볼 수 있는 기술이다. 본 연구에서는 이들 기술을 구현하는 소프트웨어를 도커[8][9][10]를 이용한 분산 시스템상에서 개발하였다.

MC-MPI-LIB은 기존 MPI 함수 내의 네트워크 통신을 통한 데이터 전달[5][11] 부분을 공유 메모리를 통한 데이터 복사로 수정하여 효율적인 수행을 얻는다. 이를 위하여 각 MPI 함수에 대응하는 수정된 MPI 함수를 개발하였다. 사용자는 이 라이브러리를 사용하여 기존 프로세서 중심 컴퓨터 구조에서 개발된 MPI 프로그램을 수정하지 않고 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행할 수 있다. 현재 MC-MPI-LIB에는 6개의 수정된 주요 MPI 함수가 시험적으로 구현되었다.

MC-MPI-SIM은 MPI 병렬 프로그램이 메모리 중심 컴퓨터 구조에서 수행될 때 수행 시간을 예측하는 프로그램이다. 이 프로그램을 사용하면 어떤 MPI 병렬 프로그램의 프로세서 중심 컴퓨터 구조에서 수행 시간과 메모리 중심 컴퓨터 구조에서 수행 시간을 비교해 볼 수 있다. 이 프로그램은 MPI 병렬 프로그램의 수행 시간을 통신 수행 시간과 계산 수행 시간으로 나누어 비교할 수도 있고, 각 MPI 함수가 수행한 수행 시간을 구분하여 알아볼 수도 있다.

본 연구는 다음과 같은 의의가 있다. 첫 번째, 기존 개발된 많은 MPI 병렬 프로그램을 수정 없이 메모리 중심 컴퓨터 구조상에서 더 효율적으로 수행할 수 있다. 두 번째, 메모리 중심 컴퓨터 구조가 아직 구현되어 있지 않은 현실에서 메모리 중심 컴퓨터 구조에서 수행시켰을 때 MPI 병렬 프로그램이 어느 정도의 높은 성능을 보일지 예측할 수 있다. 세 번째, 본 연구에서 개발한 도커 기반 분산 시스템은 매우 유연하여 본 연구와 유사한 연구를 위한 분산 병렬 프로그래밍 개발 환경으로 사용할 수 있다.

본 논문의 2장에서는 본 연구 설명에 필요한 사전 지식으로, 프로세서 중심 컴퓨터 구조와 메모리 중심 컴퓨터 구조의 특징에 대하여 먼저 설명하고, 주요 MPI 함수의 기능 및 통신 오버헤드에 대하여 설명하며, 도커를 이용한 분산 병렬 시스템 구축에 대하여 설명한다. 3장 및 4장에서는 본 연구가 제안하는 주요 기술인 MC-MPI-LIB과 MC-MPI-SIM의 구현 아이디어 및 구현 기술에 대하여 설명한다. 5장에서는 다수의 MPI 병렬 프로그램들을 본 연구에서 개발한 MC-MPI-LIB 및 MC-MPI-SIM을 사용하여 성능 측정한 결과에 대하여 기술한다. 여기서 MPI 병렬 프로그램 중 통신 오버헤드가 높은 프로그램일수록 메모리 중심 컴퓨터 구조에서는 더 효율적으로 수행됨을 확인할 수 있었다. 6장에서는 본 논문의 결론으로 본 연구의 의의와 앞으로의 연구 방향에 대하여 설명한다.

II. Preliminary research

1. Processor-Centric Computer Architecture

프로세서 중심 컴퓨터 구조(Processor-Centric Computer Architecture, PC)는 Fig. 1과 같이 프로세서와 프로세서 자신만 접근 가능한 로컬 메모리로 이루어진 노드들이 통신 네트워크에 병렬로 연결된 컴퓨터 구조이다. 현재의 분산 시스템, 클러스터 및 슈퍼컴퓨터 등은 이 구조로 되어 있다[5]. 지난 40년 동안 IT 산업은 프로세서의 속도를 증가 시켜 컴퓨터의 계산 능력을 향상시키는 프로세서 중심 컴퓨터 구조를 사용해왔다[12]. 하지만 프로세서의 속도 증

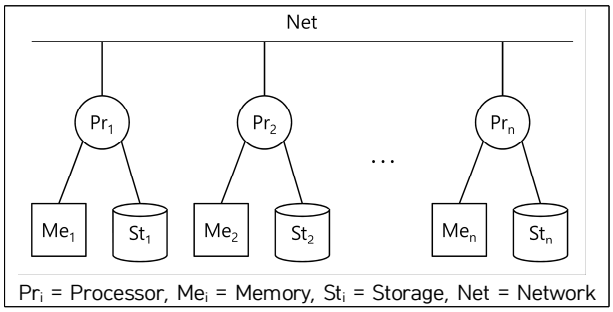


Fig. 1. Processor-centric computer architecture

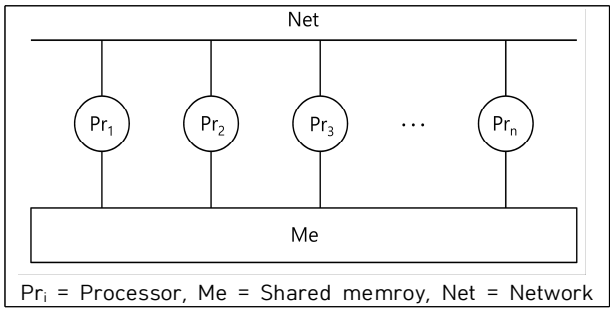


Fig. 2. Memory-centric computer architecture

가는 최근 들어 무어의 법칙이 더 이상 적용되지 않으면서 한계에 이르게 되었고, 메모리 응답의 속도 증가는 프로세서의 속도 증가에도 훨씬 미치지 못하고 있다[13]. 이는 프로세서와 메모리 두 장치 사이의 속도 불균형이 더욱더 커지게 하였다. 프로세서 중심 컴퓨터 구조는 이 속도 불균형으로 인해 최근 들어 양적으로 급증하는 데이터를 효율적으로 처리하지 못하고 있다. 이러한 프로세서 중심 컴퓨터 구조의 문제점이 대용량 데이터의 효율적인 처리와 프로세서와 메모리 두 장치 사이의 속도 불균형을 극복할 수 있는 메모리 중심 컴퓨터 구조로의 변화를 요구하고 있다.

2. Memory-Centric Computer Architecture

프로세서 중심 컴퓨터 구조의 문제점을 해결하기 위해 등장한 메모리 중심 컴퓨터 구조(Memory-Centric Computer Architecture, MC)는 Fig. 2와 같이 다수의 프로세서 노드들이 빠른 대용량 공유 메모리를 공유하는 구조를 가진다. HP에서 최근 프로토타입으로 개발한 The Machine은 이 구조로 되어 있다[2][14]. 메모리 중심 컴퓨터 구조의 가장 큰 특징은 빠른 대용량 공유 메모리이다. 이를 활용하면 프로세서 중심 컴퓨터 구조의 네트워크 통신을 통한 느린 데이터 전달을 메모리 중심 컴퓨터 구조의 공유 메모리를 통한 빠른 데이터 복사로 대체할 수 있다. 이는 MPI 병렬 프로그램을 메모리 중심 컴퓨터 구조에서 더욱 효율적으로 수행할 수 있는 환경을 제공한다. 하지만 기존 MPI 병렬 프로그램은 메모리 중심 컴퓨터 구조의 특징을 활용하여 개발된 것이 아니기 때문에 메모리 중심 컴퓨터가 등장하더라도 MPI 병렬 프로그램을 메모리 중심 컴퓨터 구조의 특징을 활용하여 재작성되거나 수정되어야 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행될 수 있다.

3. Major MPI Functions and Communication Overhead

본 절에서는 주요 MPI 함수의 기능과 통신 오버헤드를 설명한다. 주요 MPI 함수는 Open MPI 3.1.2를 기준으로 설명한다. 아래에서 사용하는 측정 단위인 t_s는 통신을 위한 startup 시간, t_c는 통신 네트워크를 통한 word(4-bytes)당 전송 시간, p는 프로세스의 개수이며 m은 데이터의 크기이다.

3.1 MPI_Send & MPI_Recv

MPI_Send 및 MPI_Recv는 one-to-one 통신으로 두 개의 프로세스가 데이터를 주고받는 함수이다. 단일 프로세스는 데이터를 다른 프로세스에게 전송하고, 다른 프로세스는 데이터를 수신한다. 두 프로세스가 m-words 크기의 데이터를 one-to-one 통신할 때 발생하는 통신 오버헤드는 $(t_s+m.t_c)$이다.

3.2 MPI_Bcast

MPI_Bcast는 one-to-all broadcast 통신으로 단일 프로세스가 자기 자신을 포함한 다른 모든 프로세스에게 동일한 데이터를 전송하는 함수이다. MPI_Bcast 함수 수행 전에는 단일 프로세스만 데이터를 가지고 있지만, MPI_Bcast 함수 수행 후에는 모든 프로세스가 동일한 데이터를 복사하여 가지고 있다. MPI_Bcast 함수는 recursive doubling을 사용하여 $\log_2{}^p$ 단계로 수행된다. p개의 프로세스가 m-words 크기의 데이터를 one-to-all broadcast 통신할 때 발생하는 통신 오버헤드는 $((t_s+m.t_c)\log_2{}^p)$이다.

3.3 MPI_Reduce

MPI_Reduce는 all-to-one reduce 통신으로 모든 프로세스가 단일 프로세스에게 데이터를 전송하고 단일 프로세스는 데이터들을 수신하고 주어진 특정 연산자를 통해 데이터를 결합하는 함수이다. MPI_Reduce 함수 수행 전에는 모든 프로세스가 데이터를 가지고 있지만, MPI_Reduce 함수 수행 후에는 단일 프로세스만 주어진 결합 연산자를 통해 결합된 데이터를 가지고 있다. MPI_Reduce 함수는 recursive doubling을 사용하여 $\log_2{}^p$ 단계로 수행된다. p개의 프로세스가 m-words 크기의 데이터를 all-to-one reduce 통신할 때 발생하는 통신 오버헤드는 $((t_s+m.t_c)\log_2{}^p)$이다.

3.4 MPI_Scatter

MPI_Scatter는 one-to-all personalized 통신으로 단일 프로세스가 자기 자신을 포함한 다른 모든 프로세스에게 각각 다른 m-words 크기의 데이터를 전송한다. MPI_Scatter 함수 수행 전에는 MPI_Bcast와 다르게 하나의 프로세스가 p개의 데이터를 가지고 있다. MPI_Scatter 함수 수행 후에는 모든 프로세스가 단일 프로세스에 있었던 p개의 데이터를 나눠 가지고 있다. MPI_Scatter 함수는 recursive doubling을 사용하여 $\log_2^p$ 단계로 수행된다. One-to-all personalized 통신할 때 데이터의 양은 p-1개이므로 발생하는 통신 오버헤드는 $(\log_2^p.t_s+m.(p-1).t_c)$이다.

3.5 MPI_Gather

MPI_Gather는 all-to-one concatenation 통신으로 단일 프로세스가 자기 자신을 포함한 모든 프로세스에서 m-words 크기의 데이터를 수집한다. 수집한 데이터들은 프로세스 순서대로 정렬되어 단일 프로세스에 저장된다. 다른 프로세스들은 수집된 데이터들을 사용하지 못한다. MPI_Gather 함수 수행 전에는 모든 프로세스가 데이터를 가지고 있지만, MPI_Gather 함수 수행 후에는 단일 프로세스가 프로세스 순서대로 정렬된 데이터들을 가지고 있다. MPI_Gather 함수는 recursive doubling을 사용하여 $\log_2^p$ 단계로 수행된다. All-to-one concatenation 통신할 때 데이터의 양은 p-1개이므로 통신 오버헤드는 $(\log_2^p.t_s+m.(p-1).t_c)$이다.

4. Distributed Parallel System Development Using Docker

본 논문에서는 컨테이너 기반의 오픈소스 가상화 플랫폼인 도커를 이용하여 연구를 위한 분산 병렬 시스템을 구축하였다. 도커는 하드웨어를 소프트웨어로 유연하게 만들 수 있어 손쉽게 다수의 노드를 구성할 수 있다. 또한, 도커는 사용하기 쉬우며 적은 비용으로 HPC 환경과 유사한 분산 병렬 시스템을 구축할 수 있다. 본 논문에서는 이러한 이유로 도커를 이용하여 분산 병렬 시스템을 구축하였다.

III. MC-MPI-LIB Design and Implementation

본 논문에서는 프로세서 중심 컴퓨터 구조에서 개발된 MPI 병렬 프로그램을 수정하지 않고 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행하도록 하는 라이브러리인 MC-MPI-LIB을 개발하였다. MC-MPI-LIB은 프로세서 중심 컴퓨터 구조의 네트워크 통신을 통한 느린 데이터 전달을 메모리 중심 컴퓨터 구조의 공유 메모리를 이용한 빠른 데이터 전달로 대체하여 MPI 병렬 프로그램을 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행하도록 한다.

본 논문에서는 프로세서 중심 컴퓨터 구조의 네트워크 통신을 통한 느린 데이터 전달을 메모리 중심 컴퓨터 구조의 공유 메모리 간 복사를 통한 빠른 데이터 전달로 대체하기 위해 size가 1인 고정 크기의 메세지를 개발하였다. Size가 1인 고정 크기의 메세지는 메모리 중심 컴퓨터 구조의 공유 메모리 간 복사를 통해 빠른 데이터 전달을 위한 실제 데이터의 절대 주소와 실제 데이터의 크기의 정보를 가지고 있다. MC-MPI-LIB은 MPI 함수가 네트워크를 통해 모든 데이터를 통신하는 것을 이 고정 크기 메시지만 통신하는 것으로 수정한다. 고정 크기의 메세지를 통신한 프로세서는 고정 크기의 메세지가 가지고 있는 실제 데이터의 주소와 크기를 이용하여 실제 데이터를 메모리 중심 컴퓨터 구조의 공유 메모리를 통해 복사한다.

현재 MC-MPI-LIB에는 1개의 구조체, 7개의 도우미 함수와 이들을 이용한 6개의 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수가 구현되어있다. 이 장에서는 MC-MPI-LIB의 MC_MESSAGE 형, 도우미 함수들 및 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수에 대해 설명한다.

1. MC_MESSAGE Type

본 논문에서는 size가 1인 고정 크기의 메세지를 표현하기 위한 MC_MESSAGE 형을 개발했다. 고정 크기의 메세지를 표현하는 형인 MC_MESSAGE는 Table 1과 같이 실제 메세지의 절대 주소를 저장하는 addr와 메세지의 byte 크기를 저장하는 size로 구성되는 구조체 형이다. 이 형은 뒤에 나오는 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수에서 사용된다.

Table 1. MC_MESSAGE type

```
typedef struct {
  void *addr;
  int size;
} MC_MESSAGE
```

2. Helper Functions

기존 프로세서 중심 컴퓨터 구조에서 개발된 MPI 병렬 프로그램을 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행되도록 하기 위해서는 아래와 같은 도우미 함수들이 필요하다. Table 2에 설명된 도우미 함수들은 뒤에 나오는 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수에서 사용된다.

Table 2. Helper functions

Function	Description
MC_Address(addr)	Convert the current process's virtual address "addr" to the absolute address.
MC_Memcpy(to, from, size)	Copy the "size" bytes from the absolute address "from" to the current process's virtual address "to".
MC_Post(rank)	Send a signal to a process that has a given "rank".
MC_Pend(rank)	Wait for the signal from the process with the given "rank".
MC_Post_all(comm)	Send a signal to all ranks (except the current rank) that make up the given "comm".
MC_Pend_all(comm)	Wait for the signal from all ranks (except the current rank) that make up the given "comm"
MC_Op(op)	Operator for MC_MESSAGE corresponding to each reduce "op"(e.g., MPI_MAX, MPI_MIN, MPI_SUM, etc.)

3. Modified MPI Functions for Memory-Centric Computer Architecture

기존 MPI 함수는 메모리 중심 컴퓨터 구조의 특징을 활용하여 개발된 것이 아니기 때문에 메모리 중심 컴퓨터 구조에서 수행되었을 때 더 높은 성능으로 수행될 수 없다. 따라서 본 논문에서는 기존 MPI 함수에 대응하는 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수를 개발했다. 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수는 기존 MPI 함수를 수정하여 프로세서 중심 컴퓨터 구조의 네트워크를 통한 느린 데이터 전달 부분을 메모리 중심 컴퓨터 구조의 공유 메모리를 통한 빠른 데이터 복사로 수정하여 메모리 중심 컴퓨터 구조에서 수행되었을 때 더 높은 성능으로 수행한다. 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수는 앞서 설명한 것과 같이 MC_MESSAGE 형과 도우미 함수들을 이용한다. MC-MPI-LIB에는 MPI_Send, MPI_Recv, MPI_Bcast, MPI_Reduce, MPI_Scatter 및 MPI_Gather 총 6개의 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수가 구현되어 있으며, Table 3과 Table 4는 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수 중 MPI_Send와 MPI_Recv를 보여준다.

Table 3. MPI_Send

Existing function	MPI_Send(buffer, count, datatype, dest, tag, comm);

Modified function	{
	MC_MESSAGE mc_buffer;
	mc_buffer.addr = MC_Address(buffer);
	mc_buffer.size = count * sizeof(datatype);
	MPI_Send(&mc_message, 1,
	MPI_MC_MESSAGE, dest, tag, comm);
	MC_Pend(dest);
	}

Table 4. MPI_Recv

Existing function	MPI_Recv(buffer, count, datatype, source, tag, comm, status);

Modified function	{
	MC_MESSAGE mc_buffer;
	MPI_Recv(&mc_message, 1,
	MPI_MC_MESSAGE, source, tag,
	comm, status);
	MC_Memcpy(buffer, mc_buffer.addr,
	mc_buffer.size);
	status->count = mc_buffer.size /
	sizeof(datatype);
	MC_Post(source);
	}

4. Communication Overhead Comparison of Existing MPI Functions and Modified MPI Functions for Memory-Centric Computer Architecture

본 절에서는 2장 3절에서 설명한 주요 MPI 함수와 3장 3절에서 설명한 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수의 통신 오버헤드를 비교한다. 앞서 설명한 것과 같이 프로세서 중심 컴퓨터 구조의 네트워크를 통한 느린 데이터 전달 부분을 메모리 중심 컴퓨터 구조의 공유 메모리를 통한 빠른 데이터 복사로 수정하기 때문에 메모리 중심 컴퓨터를 위한 수정된 MPI 함수의 통신 오버헤드는 기존 MPI 함수에 비해 감소한다. 아래에서 사용하는 측정 단위인 t_s는 통신을 위한 startup 시간, t_c는 통신 네트워크를 통한 word(4-bytes)당 전송 시간, t_m은 메모리를 통한 복사 시간, p는 프로세스의 개수이며 m은 데이터의 크기이다.

4.1 MPI_Send & MPI_Recv

기존 MPI_Send 및 MPI_Recv의 통신 오버헤드는 $(t_s+m.t_c)$이다. 메모리 중심 컴퓨터 구조를 위한 수정된 MPI_Send 및 MPI_Recv 함수의 오버헤드는 size가 1인 고정 크기의 메시지 통신 오버헤드인 $(t_s+1.t_c)$와 메모리 복사 시간인 $(m.t_m)$을 합친 $(t_s+1.t_c)+(m.t_m)$이다.

4.2 MPI_Bcast & MPI_Reduce

기존 MPI_Bcast와 MPI_Reduce의 통신 오버헤드는 $((t_s+m.t_c)\log_2{}^p)$이다. 메모리 중심 컴퓨터 구조를 위한 수정된 MPI_Bcast와 MPI_Reduce 함수의 오버헤드는 size가 1인 고정 크기의 메시지 통신 오버헤드인 $((t_s+1.t_c)\log_2{}^p)$와 메모리를 복사 시간인 $(m.t_m)$을 합친 $((t_s+1.t_c)\log_2{}^p)+(m.t_m)$이다.

4.3 MPI_Scatter & MPI_Gather

기존 MPI_Bcast와 MPI_Reduce의 통신 오버헤드는 $(\log_2{}^p.t_s+m.(p-1).t_c)$이다. 메모리 중심 컴퓨터 구조를 위한 수정된 MPI_Bcast와 MPI_Reduce 함수의 오버헤드는 size가 1인 고정 크기의 메시지 통신 오버헤드인 $(\log_2{}^p.t_s+1.(p-1).t_c)$와 메모리 복사 시간인 $(m.t_m)$을 합친 $(\log_2{}^p.t_s+1.(p-1).t_c)+(m.t_m)$이다.

IV. MC-MPI-SIM Design and Implementation

MPI 병렬 프로그램의 성능을 측정하는 많은 방법이 존재한다[15][16][17]. 하지만 이들 중 MPI 병렬 프로그램이 메모리 중심 컴퓨터 구조에서 얼마나 더 효율적으로 수행하는지 미리 알아볼 수 있는 방법은 개발되어 있지 않다.

본 논문에서는 프로세서 중심 컴퓨터 구조에서 수행한 MPI 병렬 프로그램의 성능을 측정하고 메모리 중심 컴퓨터 구조에서 MPI 병렬 프로그램의 수행 시간을 예측하는 프로그램인 MC-MPI-SIM을 개발하였다. 앞서 설명한 MC-MPI-LIB과 이 장에서 설명할 MC-MPI-SIM은 하나의 프로그램으로 결합되어 있으며 이를 이용하면 기존 MPI 병렬 프로그램을 성능 측정에 용이한 프로세서 중심 컴퓨터 구조용(PC용) MPI 병렬 프로그램과 메모리 중심 컴퓨터 구조용(MC용) MPI 병렬 프로그램으로 자동 변환 컴파일할 수 있다. PC용 MPI 병렬 프로그램은 기존 MPI 함수를 사용한 MPI 병렬 프로그램이고 MC용 MPI 병렬 프로그램은 기존 MPI 함수가 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수로 변환된 프로그램이다.

본 논문에서는 현재 메모리 중심 컴퓨터가 없기 때문에 PC용 MPI 병렬 프로그램과 MC용 MPI 병렬 프로그램을 모두 기존의 프로세서 중심 컴퓨터에서 성능을 측정한다. 아래는 본 연구에서 개발한 MC-MPI-SIM이 성능 측정을 위해 가정하고 있는 부분이다.

(1) MPI 함수가 사용하는 size가 m인 가변 크기의 메세지를 size가 1인 고정 크기 메세지로 변환함

(2) size가 1인 고정 크기의 메세지를 전달하는 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수는 기존 통신 네트워크를 사용한다고 가정함

(3) 메모리 중심 컴퓨터의 메모리 접근 속도는 기존 프로세서 중심 컴퓨터의 메모리 접근 속도와 동일하다고 가정함

(4) MPI 함수 변환 시 동기화를 위해 추가되는 함수들의 수행 시간은 무시함

(5) MPI 함수 변환 시 추가되는, 가상 주소를 절대 주소로 변환하는, 함수인 MC_Address의 수행 시간은 무시함

(6) 성능 측정에 있어 (4)와 (5) 같이 수행시간을 무시하는 동작들은 메모리 중심 컴퓨터 구조가 구현되었을 때, 메모리 중심 컴퓨터 구조에서 사용될 운영체제에서 제공한다고 가정함.

이 장에서는 PC용 MPI 병렬 프로그램과 MC용 MPI 병렬 프로그램의 성능 측정 방법과 MC-MPI-SIM의 구현 방법을 설명한다.

1. How to Measure Performance

MC-MPI-SIM은 성능 측정을 위해, MPI 병렬 프로그램의 wall time을 측정하는 함수인 MPI_Wtime를 이용한 MPI wall clock time 로깅 코드를 삽입한다. MPI wall clock time 로깅 코드 삽입 위치는 측정하고자 하는 부분의 직전과 직후이다. 측정하고자 하는 부분 직전의 호출한 MPI wall time 로깅 코드의 수행 값과 측정하고자 하는 부분 직후의 호출한 MPI wall time 로깅 코드의 수행 값 차이로 그 부분의 수행 시간을 측정한다.

1.1 Measurement of overall execution time of MPI parallel programs for PC and MC

전체 수행 시간 측정을 위해 MPI 병렬 프로그램의 시작 부분인 함수 MPI_Init 호출 직후와 MPI 병렬 프로그램의 끝 부분인 함수 MPI_Finalize 호출 직전에 각각 MPI wall clock time 로깅 코드를 삽입한다. MPI 병렬 프로그램의 전체 수행 시간은 이 두 MPI wall clock time 로깅 코드의 수행 값의 차이다.

1.2 Measurement of execution time of MPI parallel program communication part for PC

PC용 MPI 병렬 프로그램 통신 부분의 수행 시간 측정을 위해 PC용 MPI 병렬 프로그램의 모든 MPI 함수 호출 직전과 직후에 각각 MPI wall clock time 로깅 코드를 삽

입한다. 각 MPI 함수의 수행 시간은 이 두 MPI wall clock time 로깅 코드의 수행 값의 차이이고, PC용 MPI 병렬 프로그램 통신 부분의 수행 시간은 모든 MPI 함수의 수행 시간의 합이다.

1.3 Measurement of execution time of MPI parallel program communication part for MC

MC용 MPI 병렬 프로그램 통신 부분의 수행 시간도 PC용 MPI 병렬 프로그램 통신 부분의 수행 시간과 동일한 방법으로 측정한다. 하지만 현재 메모리 중심 컴퓨터가 존재하지 않기 때문에 MC용 MPI 병렬 프로그램의 성능은 앞서 설명한 것과 같이 기존 프로세서 중심 컴퓨터 구조에서 측정한다. 이를 위해 메모리 중심 컴퓨터 구조를 위한 수정된 MPI 함수에는 프로그램 수행을 위한 기존 MPI 함수가 존재한다. 따라서 MC용 MPI 병렬 프로그램 통신 부분의 수행 시간은 측정 후 프로그램 수행을 위한 기존 MPI 함수의 수행 시간을 빼주어야 한다.

1.4 Measurement of execution time of MPI parallel program calculation part for PC and MC

계산 부분의 수행 시간은 MPI 병렬 프로그램의 전체 수행 시간에서 통신 부분의 수행 시간을 뺀 값이다.

2. Insert Code for Storing Performance Measurement Data

MPI wall clock time 로깅 데이터를 저장하기 위한 코드는 MPI 병렬 프로그램이 수행되는 동안 측정한 각 프로세스의 전체 수행 시간, 통신 부분의 수행 시간 및 계산 부분의 수행 시간을 저장한다. 통신 부분의 수행 시간은 각 프로세스의 통신 부분의 수행 시간과 각 MPI 함수별 수행 시간으로 저장한다. 각 프로세스는 측정한 수행 시간들을 루트 프로세스에게 전달하고 수행 시간들을 전달받은 루트 프로세스는 이 시간들을 모두 합하여 프로그램의 전체 수행 시간, 통신 부분의 전체 수행 시간 및 계산 부분의 전체 수행 시간을 구한다.

V. Performance Measurement Result

이 장에서는 기존 MPI 프로그램을 3장과 4장에서 설명한 기술들을 통해 PC용과 MC용으로 변환 컴파일한 뒤 Dell PowerEdge T620 Server, Intel(R) Xeon(R) CPU E5-2640 @ 2.50GHz, 12-Core * 2, Memory 192G와 Ubuntu 18.04.1 LTS Server OS 환경 및 도커를 이용한 분산 병렬 시스템상에서 수행 시켜 성능을 측정한 결과에 대하여 기술한다.

본 연구에서는 성능 측정을 위해 Ping-pong, Pi-Calculation, Matrix-vector multiplication, Sorting 및 All-pairs shortest paths의 MPI 병렬 프로그램을 사용하였다. Table 5는 성능 측정에 사용한 프로그램의 간단한 설명이다.

Table 5. Program description

Ping-pong	- Two processes send and receive messages 1,000 times - Message size (N=words) is given as an argument - Run in 2 processes
Pi-Calculation	- Calculate pi values using integral - Number of integral sections (N=Number of divisions) is given as an argument - The larger the N, the more precise the Pi value. - Run in 8 processes
Matrix-vector multiplication	- Multiply the matrix of size N*N and the vector of size N - The size of the matrix and vector (N=Matrix rows or columns) is given as an argument - Run in 8 processes
Sorting	- Arrange an array of given integers - Size of array (N=size of array) is given as an argument - Run in 8 processes
All-pairs shortest paths	- Calculate all-pair shortest path from a given graph - The distance between nodes on a given graph is expressed as an integer value - Number of nodes in the graph (N=number of nodes) is given as an argument - Run in 8 processes

1. Ping-pong

Ping-pong 프로그램은 프로그램 내에 계산은 거의 없고 통신이 대부분을 차지하는 프로그램이다. Fig. 3의 (a-1)을 보면, PC용 MPI 병렬 프로그램의 경우 메세지의 크기인 N이 증가함에 따라 전체 수행 시간이 비례하여 증가함을 알 수 있고, MC용 MPI 병렬 프로그램의 경우 N에 상관없이 전체 수행 시간이 고정 시간임을 알 수 있다. Fig. 3의 (a-2)를 보면, 계산 부분의 수행 시간이 PC용과 MC용 병렬 프로그램 모두 N에 상관없이 고정 시간임을 알 수 있다. 여기서 MC용 병렬 프로그램의 계산 부분의 수행 시간이 PC용 병렬 프로그램의 계산 부분의 수행 시간보다 더 큰 이유는 MC용 병렬 프로그램 수행 시 통신

네트워크를 통한 통신 시간은 줄어들지만, 메모리 복사를 위한 함수 MC_Memcpy 수행으로 인한 계산 시간은 늘어나기 때문이다. Fig. 3의 (a-3)을 보면, 통신 부분의 수행 시간이 PC용 병렬 프로그램의 경우 통신 네트워크를 통해 모든 데이터를 통신하기 때문에 N에 비례하여 증가하나, MC용 병렬 프로그램의 경우 고정 크기의 메시지만 통신하기 때문에 매우 작은 고정 크기임을 알 수 있다.

2. Pi-Calculation

Pi-Calculation 프로그램은 프로그램 내에 계산은 많고 통신은 아주 작은 프로그램이다. Fig. 3의 (b-1)을 보면 정해진 적분 구간을 나눈 수인 N이 증가함에 따라 수행 시간이 비례하여 증가함을 알 수 있는데 여기서 MC용 MPI 병렬 프로그램의 전체 수행 시간이 PC용 MPI 병렬 프로그램의 전체 수행 시간보다 조금 빠름을 알 수 있다. 그 이유는 이 프로그램이 매우 작은 통신 시간을 가지기 때문이다. Fig. 3의 (b-2)를 보면, 계산 부분의 수행 시간이 PC용과 MC용 MPI 병렬 프로그램이 거의 같음을 알 수 있다. 그리고 Fig. 3의 (b-3)을 보면, 통신 부분의 수행 시간이 PC용 MPI 병렬 프로그램의 경우 통신 네트워크를 통해 모든 데이터를 통신하기 때문에 N이 증가하면서 비례하여 증가하지만, MC용 MPI 병렬 프로그램의 경우 고정 크기의 메시지만 통신하기 때문에 N에 상관없이 작은 고정값임을 알 수 있다.

3. Matrix-vector multiplication

Matrix-vector multiplication 문제를 병렬로 푸는 여러 방법이 있지만 본 프로그램은 가장 기초적인 1-D 분할 방법을 사용한 문제 풀이이다. 이 방법을 사용한 병렬 프로그램은 계산 시간과 더불어 통신 시간도 매우 큰 프로그램이다. Fig. 3의 (c-1)을 보면, 전체 수행 시간이 PC용 MPI 병렬 프로그램의 경우 행렬의 행 혹은 열의 크기인 N이 증가함에 따라 수행 시간이 2차 곡선을 그리며 증가함을 알 수 있다. 이는 문제 풀이를 위한 계산의 양이 N의 제곱이기 때문이다. MC용 MPI 병렬 프로그램의 경우에도 N이 증가함에 따라 수행 시간이 2차 곡선을 그리며 증가함을 알 수 있다. 이는 계산 부분의 수행 시간인 Fig. 3의 (c-2)를 보면 더 확실히 알 수 있다. Fig. 3의 (c-2)를 보면 더 확실히 알 수 있다. Fig. 3의 (c-3)을 보면, 통신 부분의 수행 시간이 PC용 MPI 병렬 프로그램의 경우 통신 네트워크를 통해 모든 데이터를 전달하기 때문에 N이 증가함에 따라 2차 곡선을 그리며 크게 증가하지만, MC용 MPI 병렬 프로그램의 경우는 고정 크기의 메시지만 통신하기 때문에 아주 작게 증가함을 알 수 있다.

4. Sorting

Sorting 프로그램은 주어진 정수 배열을 여러 프로세스가 나누어 병렬로 quick 정렬한 후 여러 프로세스가 같이 병렬로 merge 하면서 최종 정렬한다. Fig. 3의 (d-1)을 보면, 전체 수행 시간이 PC용 MPI 병렬 프로그램의 경우가 MC용 MPI 병렬 프로그램의 경우 보다 좀 더 시간이 걸리는 것을 알 수 있다. Fig. 3의 (d-2)를 보면 계산 부분의 수행 시간은 PC용과 MC용 MPI 병렬 프로그램이 거의 같은 수행 시간을 가진다. Fig. 3의 (d-3)을 보면 통신 부분의 수행 시간은 PC용 MPI 병렬 프로그램의 경우 통신 네트워크를 통해 모든 데이터를 통신하기 때문에 N이 증가함에 따라 크게 증가하지만, MC용 MPI 병렬 프로그램의 경우에는 고정 크기의 메시지만 통신하기 때문에 아주 느리게 증가함을 알 수 있다.

5. All-pairs shortest paths

Fig. 3의 (e-1), (e-2)와 (e-3)을 보면, All-pair shortest path 프로그램도 앞의 sorting 프로그램과 유사한 수행 시간을 보여주고 있다. 일반적으로 병렬 프로그램 중 통신의 비중이 큰 프로그램일수록 MC용 MPI 병렬 프로그램에서 더 좋은 성능을 보여줄 수 있다.

VI. Conclusion

지난 40년 동안 IT 산업에서 사용한 프로세서 중심 컴퓨터 구조는 최근 들어 양적으로 급증하는 데이터를 효율적으로 처리하지 못하고 있다. 이에 따라 기존의 프로세서 중심 컴퓨터 구조의 문제점을 해결할 수 있는 메모리 중심 컴퓨터 구조에 대한 개발이 시작되고 있다. 하지만 메모리 중심 컴퓨터 구조가 등장하더라도, 기존 MPI 병렬 프로그램은 재작성되거나 수정되어야 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행될 수 있다는 문제점이 있다.

본 연구에서는 기존 프로세서 중심 컴퓨터 구조에서 개발된 MPI 병렬 프로그램을 수정하지 않고 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행시키는 기술인 MC-MPI-LIB과 메모리 중심 컴퓨터 구조에서 얼마나 더 효율적으로 수행가능한 지 미리 알아볼 수 있는 기술인 MC-MPI-SIM을 개발하였다. MC-MPI-LIB은 프로세서 중심 컴퓨터 구조의 네트워크 통신을 통한 느린 데이터 전달을 메모리 중심 컴퓨터 구조의 공유 메모리 간 복사를 통한 빠른 데이터 전달로 대체하여 MPI 병렬 프로그램을

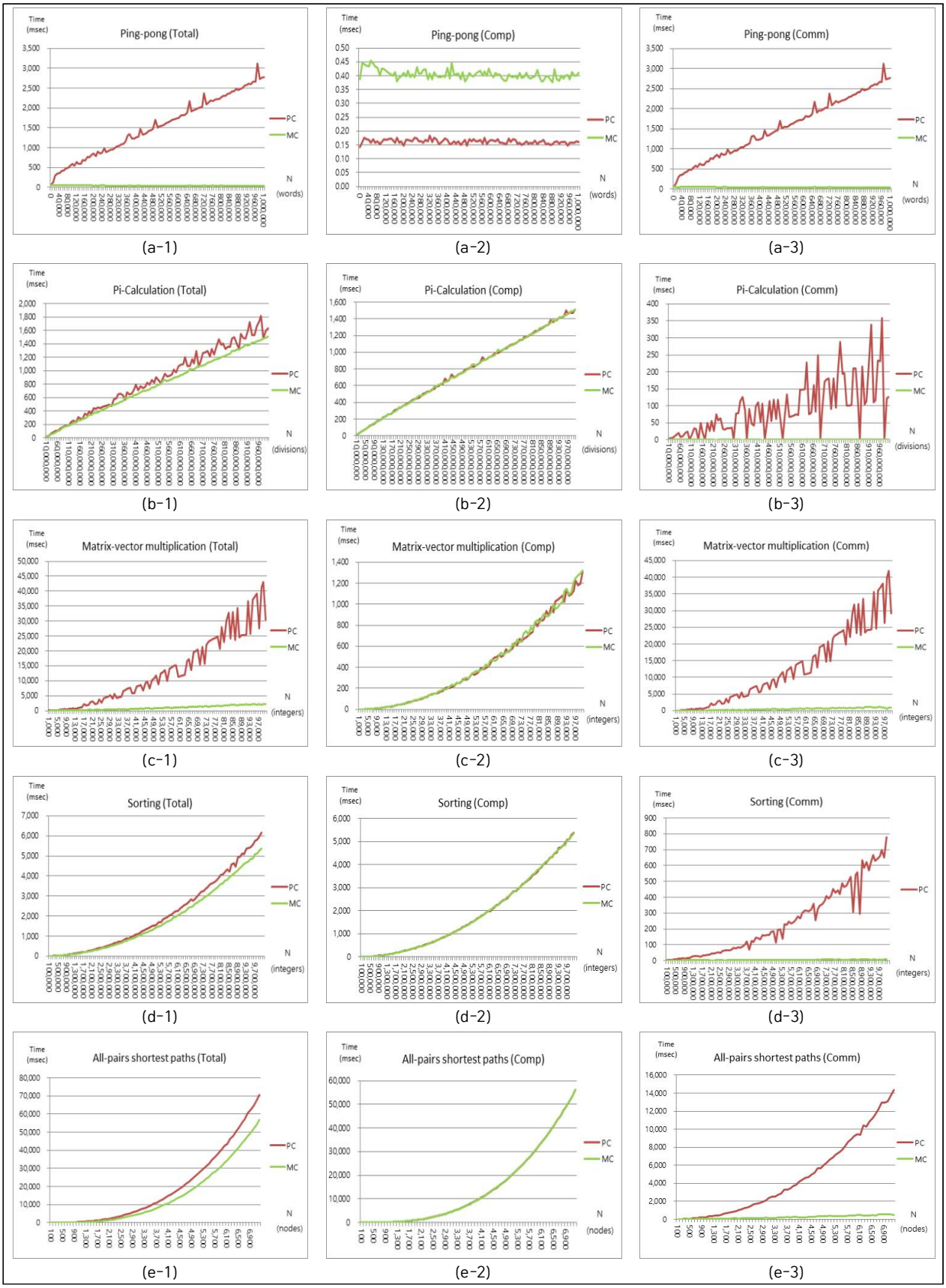


Fig. 3. Performance comparison of MPI parallel program on PC and MC

메모리 중심 컴퓨터 구조에서 더 효율적으로 수행하도록 설계하였다. MC-MPI-LIB을 설계한 내용은 구현하기 위해 본 논문에서는 MC_MESSAGE형, 도우미 함수들과 기존 MPI 함수에 대응하는 수정된 MPI 함수를 개발하였다. MC_MESSAGE형은 공유 메모리를 통한 데이터 복사에 필요한 데이터의 실제 주소와 크기를 가지고 있는 구조체이다. 도우미 함수들은 기존 개발된 MPI 병렬 프로그램을 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행되도록 하기 위해 필요한 함수들이다. 그리고 기존 MPI 함수에 대응하는 수정된 MPI 함수는 프로세서 중심 컴퓨터 구조의 네트워크를 통한 느린 데이터 전달 부분을 메모리 중심 컴퓨터 구조의 공유 메모리를 통한 빠른 데이터 복사로 수정한 MPI 함수이다. MC-MPI-SIM은 MPI 병렬 프로그램의 MPI wall clock time 로깅 코드를 삽입하여 수행 시간을 측정할 수 있도록 개발하였다. MC-MPI-SIM으로 측정한 성능은 각 프로그램의 총 수행 시간이다. 각 프로그램의 총 수행 시간은 다시 계산 부분의 수행 시간 및 통신 부분의 수행 시간을 나누어진다. 이 나누어진 수행 시간을 비교 관찰하여 주어진 프로그램의 특성을 예측할 수 있다.

본 논문에서 개발한 기술들은 도커를 이용한 분산 병렬 시스템에서 개발하고 성능을 측정하였다. 기존에 개발된 다수의 MPI 병렬 프로그램의 성능을 측정한 결과 메모리 중심 컴퓨터 구조에서 더 높은 성능으로 수행 가능함을 보였으며, 특히 통신 오버헤드 비율이 높은 MPI 병렬 프로그램의 경우 메모리 중심 컴퓨터 구조 에서 매우 높은 성능으로 수행 가능하다는 점을 확인하였다.

본 연구는 다음과 같은 의의가 있다. 첫 번째, 메모리 중심 컴퓨터가 상용화되었을 때 기존에 개발된 MPI 병렬 프로그램을 수정하지 않고 메모리 중심 컴퓨터 구조에서 더 효율적으로 수행시킬 수 있는 기술을 제공한다. 두 번째, 메모리 중심 컴퓨터가 아직 구현되지 않은 현실에서 메모리 중심 컴퓨터 구조에서 MPI 병렬 프로그램을 수행시켰을 때 어느 정도의 높은 성능을 보일지 예측할 수 있는 기술을 제공한다. 세 번째, 본 연구에서 개발한 도커 기반 분산 시스템은 매우 유연하여 본 연구와 유사한 연구를 위한 분산 병렬 프로그램이 개발 환경으로 사용할 수 있다.

본 연구에서는 성능을 직접 측정하여 MPI 병렬 프로그램의 성능을 비교하지만, 향후 연구에서는 MPI 병렬 프로그램의 실행 싸이클 횟수 및 통신에 필요한 정보를 통해 성능을 계산하여 비교한다면 보다 유용한 연구로 발전시킬 수 있을 것으로 기대된다.

ACKNOWLEDGEMENT

This work was partially supported by the Core Technology Research on Fabric Memory Computing (18ZS1220) conducted by the Electronics and Telecommunications Research Institute (ETRI) grant funded by the Ministry of Science and ICT (MSIT).

REFERENCES

[1] P. Faraboschi, K. Keeton, T. Marsland, and D. Milojicic, "Beyond processor-centric operating systems", Proceedings of the 15th USENIX conference on Hot Topics in Operating Systems, pp. 17-17, Switzerland, May 2015.

[2] K. Keeton, "Memory-Driven Computing," Keynote at USENIX FAST, Santa Clara, CA, USA, Feb. 2017.

[3] K. M. Bresniker, S. Singhal, and R. S. Williams, "Adapting to thrive in a new economy of memory abundance," Computer, Vol. 48, No. 12, pp. 44-53, Dec. 2015.

[4] D. Efnusheva, G. Dokoski, A. Tentov and M. Kalendar, "A Novel Memory-centric Architecture and Organization of Processors and Computres", International Conference on Applied Innovations in IT, Vol. 3, No. 1, pp. 47-53, Koethen, Germany, March 2015.

[5] A. Grama, A. Gupta, G. Karypis, and V. Kumar, "Introduction to Parallel Computing," Second Edition, Addison Wesley, 2003.

[6] W. Gropp, E. Lusk, and A. Skjellum, "Using MPI Portable Parallel Programming with the Message-Passing Interface," Third Edition, The MIT Press, 2014.

[7] MPI Forum, https://www.mpi-forum.org

[8] M. T. Chung, N. Quang-Hung, M. Nguyen, and N. Thoai, "Using Docker in High Performance Computing Applications", IEEE International Conference on Communications and Electronics, pp. 52-57, Ha Long, Vietnam, July 2016.

[9] Docker Documentation, https://docs.docker.com

[10] K. Matthias and S. P. Kane, "Docker: Up and Running," O'Reilly, 2015.

[11] G. Hager, and G. Wellein, "Introduction to High Performance Computing for Scientists and Engineers," CRC Press, 2010.

[12] K. Takeuchi, "Memory system architecture for the data centric computing", Japanese Journal of Applied Physics, Vol. 55, No. 4S, pp. 04EA02, Feb. 2016.

[13] J. McCalpin, "Memory Bandwidth and System Balance in HPC Systems", Salt Lake City, Utah, USA, Invited talk at ACM/IEEE Supercomputing Conference, Dec. 2016.

[14] The Machine, https://www.labs.hpe.com/memory-driven-computing

[15] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI Applications: The SMPI Approach," IEEE Transactions on Parallel and Distributed Systems, Vol. 28, No. 8, pp. 2387-2400, Aug. 2017.

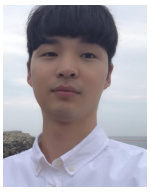
[16] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim-Simulating Large-Scale Applications in the LogGOPS Model", Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, Illinois, USA, pp. 597-604, June 2010.

[17] C. Janssen, H. Adalsteinsson, S. Cranford, J. Kenny, A. Pinar, D. Evensky and J. Mayo, "A Simulator for Large-scale Parallel Computer Architectures," International Journal of Distributed Systems and Technologies, Vol. 1, No. 2, pp. 57-73, April 2010.

[18] Je-Man Lee, Seung-Chul Lee, and Dong-Ha Shin, "Efficient Executions of MPI Parallel Programs in Memory-Centric Computer Architecture," Proceedings of The Korea Society of Computer and Information Summer Conference, Vol. 27, No. 2, pp. 257-258, Jeju, South Korea, July 2019.

Authors

Je-Man Lee received the B.E. degree in Computer Science from Sangmyung University in 2018. He is currently a M.S. student of Department of Computer Science at Graduate School of Sangmyung University. His research interests include Parallel Programming, Computer Architecture, Memory-Centric Computer Architecture, MPI, Simulation.



Seung-Chul Lee received the B.E. degree in Computer Science from Sangmyung University in 2018. He is currently a M.S. student of Department of Computer Science at Graduate School of Sangmyung University. His research interests include Parallel Programming, Computer Architecture, Memory-Centric Computer Architecture, MPI, Simulation.

Dongha Shin received the B.S. degree in Computer Engineering from Kyungpook National University in 1980, the M.S. degree in Computer Engineering from Seoul National University in 1982 and the Ph.D. degree in Computer Science from University of South Carolina in 1994. During 1982-1996, he stayed in ETRI as a technical staff to study expert systems, word processing systems, file systems and language processing systems. During 1997-current, he is a professor of Dept. of Electronics in Sangmyung University. His research interests include real-time kernels, kernel protection, embedded computing and compiler implementation.