GOVP 12009321

98-NF-03-04-A-01

초고속 컴퓨터 기반 소프트웨어 및 응용 기술

초고속 컴퓨터의 효율적인 프로세서간 통신함수 설계 및 개발

Design of communication functions for high-performance multicomputers

고려대학교 정보통신연구소

과 학 기 술 부

제 출 문

과학기술부 장관 귀하

본 보고서를 "**초고속 컴퓨터 기반 소프트웨어 및 응용 기술** 에 관한 연구" 과제 (세부과제 " **초고속 컴퓨터의 효율적인 프로세서간 통신함수 설계 및 개발** 에 관한 연구") 의 보고서로 제출합니다.

주관연구기관명 : 고려대학교

주관연구책임자 : 김 동 승

요 약 문

Ⅰ. 제목

초고속 컴퓨터의 효율적인 프로세서간 통신함수 설계 및 개발

(Design of communication functions for high-performance multicomputers)

Ⅱ. 연구개발의 목적 및 필요성

프로그램 실행시간을 단축하기 위해서는 고속 프로세서를 장착한 수퍼컴퓨터를 활용하거나, 수십-수백개의 프로세서를 내장한 병렬컴퓨터를 이용해야 한다. 다수의 프로세서가 상호연결망으로 결합된 분산메모리형 병렬컴퓨터에서는 프로세서간 통신성능이 전체 시스템 성능을 좌우하게 된다. 특히 분산된 작업간의 상호 데이터 교환량이나 통신빈도가 클수록 전체 실행시간은 통신시간의 크기에 따라 결정되게 된다. 본 연구에서는 빠른 통신 하드웨어 설치에 의해 성능이 향상되는 것과는 별도로, 소프트웨어적으로 성능을 개선하고자 기존의 메시지 전달형 함수를 재설계하여 적용하도록 함으로써 작업시간 단축을 꾀하고자 시도되었다.

초고속 컴퓨터에서 활용되는 프로세서 상호간의 통신을 분석하고 응답 속도를 향상시키고 작업 효율을 높이도록 통신 함수 library를 개선한다. 이는 구조의 의존성을 배제하고 일대일 통신은 물론, 집단화 통신, 방송기능(broadcast)등을 실현하도록 하고, 초고속형 대상 컴퓨터에 구현하고자 한다. 메시지 분할을 통한 성능개선 효과는 병렬 컴파일러, 통신함수 개발의 핵심요소로 활용하게 된다.

III. 연구개발의 내용 및 범위

- 1차년도 ('96)
  - 병렬프로그램 모델링- PRAM모델, LogP 모델, BSP 모델
  - 메시지 교환방식 연구- wormhole routing, active message
  - MPI의 통신함수 구현 분석
  - IBM SP2 시스템 통신 하드웨어, 소프트웨어 구조 연구

- 2차년도 ('97)
  - 일대다수 통신방식의 모델링 및 설계- k-segment broadcast scan, scatter, reduce
  - 통신 유형별 모델링: 점대점 통신, 집단화 통신
  - 연산 시간 모델링
  - 알고리즘 실행시간 모델링- 바이토닉 정렬, FFT, 행렬의 곱셈 실험 결과

- 3차년도 ('98)
  - 집단화 통신 고속화 연구실험
  - 고속 통신함수 라이브러리 개발
  - PC cluster 개발 및 실험
  - 알고리즘적 성능개선 연구
  - 병렬 분산환경 완성 공개

Ⅳ. 연구개발결과

- 1차년도 ('96)
  - 병렬 프로그램 요구사항 분석
  - 메시지 교환 메카니즘 분석
  - 집단화 통신 파악
  - MPI 구현 계층구조 연구

- 2차년도 ('97)
  - MPI 라이브러리에서의 집단화 통신함수 분석 완료
  - edge-disjoint broadcast 분석 및 구현실험 수행
  - k-segment broadcast 설계, 구현 및 실험 수행
  - 프로그램 수행시간 모델링 및 실험 수행
  - PC cluster 설계, 개발진행

- 3차년도 ('98)
  - MPI 라이브러리에서의 집단화 통신함수 수정 및 성능개선
  - 통신과 입출력의 최적화 실험연구
  - 집단화 통신 고속화 방식 제시
  - 고속 통신함수 라이브러리 개발
  - PC 클러스터 개발 기술 공개

Ⅴ. 연구개발결과의 활용계획

- 고속 병렬 분산 컴퓨팅에서 전체 실행시간중 통신시간의 단축이 실행시간 단축에 지대한 기여를 하는 분야에서 속도 개선에 이용할수 있음
- 분산공유메모리 등의 메시지 전달형 함수를 대치하여 사용함
- MPI-2 환경하에 적용실험/ 성능개선 측정예정

SUMMARY

This projects is aimed at developing fast and efficient interprocessor communication schemes on high-performance parallel computers for science and engineering, knowledge processing, and internet services. Collective communications can be sped up by dividing the message into equal sized segments and transmitting them in multiple disjoint paths in parallel that utilizes communication links. "Segmented broadcast" thus greatly improves the performance compared to generic MPI library function MPI_bcst. Other collective communication functions are also modified using segmentation and message pipelining to achieve better performance. Since the methods does not assume any machine communication architecture, they can be applied to various parallel machines for performance if they support message send and receive functions.

CONTENTS

목 차

제 1 장 서 론

컴퓨터의 성능은 날로 발전하고 있지만, 성능이 향상되면 될수록 그에 따라 항상 더 많은 계산능력을 필요로 하게 되었다. 고속의 컴퓨터로도 처리시간이 오래 걸리는 많은 프로그램이 있기 때문에 이의 해결을 위하여 최근에는 여러 프로세서들이 하나의 프로그램을 분할하여 동시에 처리하는 병렬처리기술이 널리 사용하되 있다. 이러한 병렬처리를 효과적으로 하기 위하여 일반적으로 사용되는 두 가지의 방법이 있는데 그중 하나는 프로세서들 사이에 공유메모리를 두어서 서로 데이터를 공유하는 공유메모리 방식이고, 나머지 하나는 프로세서마다 지역메모리를 가지고 있으면서 상호 연결된 통신망을 이용하여 데이터를 주고받는 메시지 전달방식이다. 이 두 방식은 서로 장단점을 가지고 있는데 특히 확장성이 좋다는 메시지 전달방식의 장점으로 인하여 최근의 병렬컴퓨터들은 대부분 메시지 전달방식에 의해 만들어졌다.

메시지 전달 방식의 병렬컴퓨터는 서로 메시지를 주고받는 하드웨어와 소프트웨어를 기반으로 하여 구현이 된다. 그 중에서 소프트웨어는 서로 다른 시스템간에도 프로그램 코드의 변경 없이 수행하게 하기 위해 제정된 표준이 있는데 그 표준으로 제안된 것이 MPI(Message Passing Interface)[9,10,12]이다. 일반적으로 하드웨어는 기본적인 send와 receive를 할 수 있는 것만이 제공되게 된다. 그렇기 때문에 기본적인 send/receive외의 기능들은 소프트웨어로 구현이 되게 된다.

본 연구에서는 메시지 전달형 병렬 컴퓨터에서 많이 쓰이는 함수인 broadcast와 reduce 그리고 gather와 scatter와 같은 collective communication 함수를 메시지 분할 전송에 의해 성능을 개선하기 위한 알고리즘과 그 실험결과를 제시하였다. 연구는 점대점 통신을 모델링하고 기존의 알고리즘을 분석하여 이보다 성능을 향상시킨 알고리즘을 제안, MPI 환경의 병렬 컴퓨터에서 구현하였다.

제 2 장 Communication model

2.1 선형 모델

SP2에서는 점대점 통신에 걸리는 시간은 프로세서의 인덱스와는 무관하기 때문에 점대점 통신은 준비시간(startup time) ts, 메시지크기 m, 점근적 대역폭(asymptotic bandwidth) B로 다음과 같이 모델링 될 수 있다.

$$T = t_s + \frac{m}{B}$$

실험에 쓰인 시스템들은 크기가 m인 메시지를 주고받는데 걸리는 시간은 다음의 각각 수식으로 표현된다.

SP2: $T_{send}(m) = 46 + 0.035m\ [\mu\text{sec}]$

T3E: $T_{send}(m) = 18 + 0.006m\ [\mu\text{sec}]$

Cluster: $T_{send}(m) = 176 + 0.105m\ [\mu\text{sec}]$

이 실험은 Ping-pong method 방법으로 실험되었다. 이 방식은 한 프로세서가 메시지를 다른 프로세서로 보내고 다시 보낸 메시지를 되돌려 받는데 걸리는 시간을 2로 나눈 시간을 측정한 것이다. 이후 수식은 SP2에 대한 값을 기록한다.

2.2 LogP

LogP[3] 모델은 점대점 통신을 하는 분산 메모리형 병렬컴퓨터의 성능을 모델링을 하기 위한 방법으로 네트워크의 구조에 관계없이 interconnection network의 성능의 특징을 기술할 수 있는 것이다. 이 모델의 주요 파라미터는 다음과 같다.

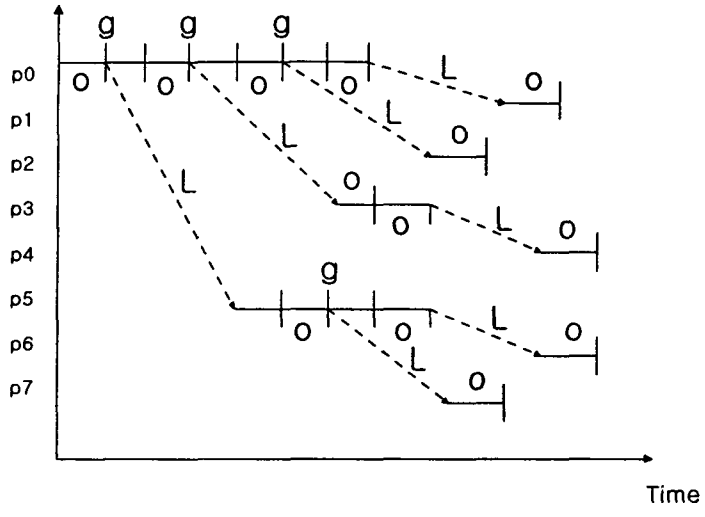


그림 2.1 LogP model

L: 메시지를 보내는 프로세서에서 메시지를 수신하는 노드로 적은 수의 메시지를 보낼 때 생기는 latency의 upperbound이다.

o: 프로세서가 통신을 위해서 참여하는 시간을 나타낸다. 이 시간 동안 프로세서는 다른 일을 할 수 없다.

g: 연속적으로 메시지를 보낼 때 전송할 수 있는 두 메시지 사이의 간격을 나타낸다. g의 역수는 프로세서당 통신 대역폭에 해당한다.

P: 프로세서/메모리 모듈의 수

이 모델은 기본적으로 적은 크기의 메시지의 통신을 모델링 할 때 사용되어진다. 현재 대부분의 병렬 컴퓨터가 DMA장치와 같은 통신 전용 하드웨어를 이용하여 긴 메시지의 전송하는 것이 일반적이기 때문에 짧은 메시지를 여러 번 보내는 것 보다 긴 메시지로 묶어서 보냄으로서 알고리즘의 성능을 개선하려는 시도가 많이 있다. 만일 긴 메시지를 주고받는 경우 LogP 모델로는 한계가 있기

때문에 여기서 설명한 파라미터 이외의 값이 필요하게 된다.

2.3 LogGP

LogP모델이 적은 메시지에만 적용이 되는 단점을 보완하기 위해 G라는 파라미터를 추가한 모델이다

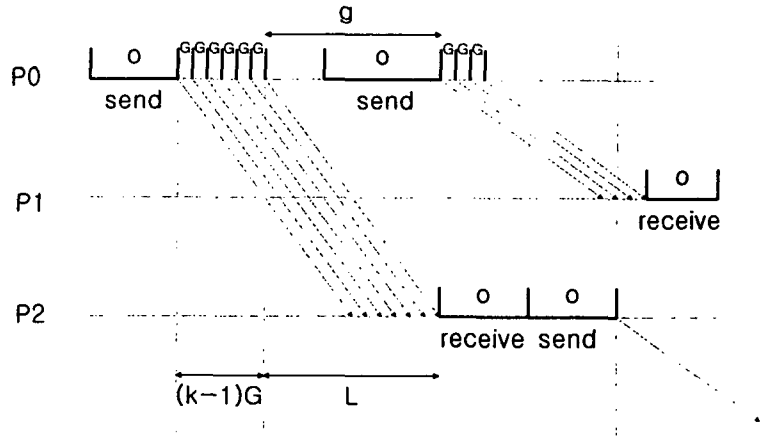


그림 2.2 LogGP model

G: 큰 메시지에서 byte당 Gap을 나타낸다.

(G의 역수 값이 큰 메시지의 통신 대역폭 값이 된다.)

LogGP 모델은 실제로 큰 메시지의 경우 선형모델과 같은 방식의 모델링 방법이다. 다시 말하면, 적은 메시지는 LogP 모델로 표현하면서 큰 메시지의 경우 선형모델을 같은 파라미터를 이용해서 표현하려는 시도인 것이다.

2.4 Parameterized Communication Model

LogP와 같은 모델이 파라미터 값의 측정에 어려움이 있기 때문에 모델을 단순화하여 모델 값을 쉽게 구할 수 있는 모델이다. 이 모델 또한 LogP모델과 마찬

가지로 적은 크기의 메시지일 경우에 해당하는 것으로 점대점 통신을 송신 오버헤드와 네트워크 지연, 그리고 수신 오버헤드 시간으로 나누고, 연속적인 메시지를 보낼 수 있는 시간을 hold time이라고 하여 한 프로세서에서 다른 프로세서로 연속적으로 메시지를 보낼 때 소요되는 시간을 모델링 할 수 있다.

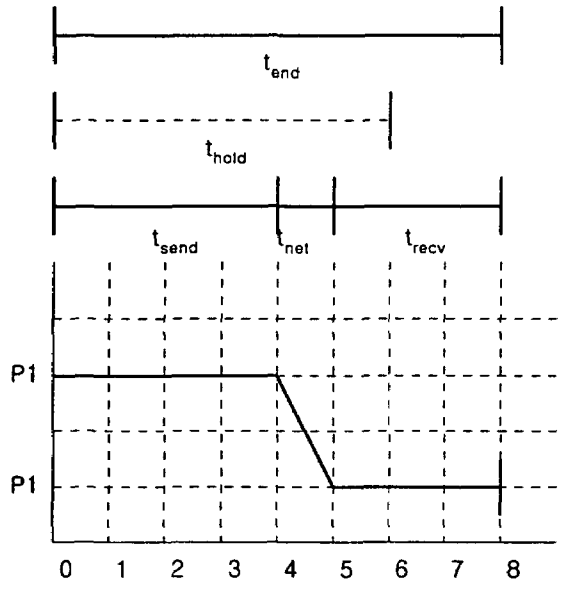


그림 2.3 parameterized communication model

본 연구에서 구현하고자 하는 알고리즘들은 주로 상대적으로 긴 메시지를 다루고 있고, 또 LogGP의 파라미터 값을 구하는 어려움 때문에 선형모델을 적용해서 알고리즘을 분석하고 실험결과를 예측해서 실제 실험 결과와 비교하도록 할 것이다.

제 3 장 MPI환경에서 통신 함수의 성능개선

3.1. MPI

Message passing interface(MPI)[9,10,12]는 MPP와 이기종 네트워크 컴퓨터의 병렬 프로그래밍을 지원하기 위한 도구로서, 성능이 좋고 이식성이 뛰어나기 때문에 메시지 패싱 프로그래밍 모델의 표준으로 정착되고 있고 활용율도 급격하게 증가하고 있다.

MPI는 메시지 패싱 모델에만 제한되어 있기 때문에 Hi Performance Fortran(HPS)에서와 같이 모든 프로세서에 의해 공유되는 전역주소 개념이 제공되지 않는다

MPI에서 기본적인 두 가지 개념은 프로세스와 메시지이며, 일대일 통신과 집합통신(collective communication)은 가장 중요한 핵심 요소이다. 그 외에 프로세스 그룹, 가상 토폴로지(virtual topology)등의 개념이 있다.

메시지:

메시지는 실제 보내는 내용인 body와 부가정보를 담고 있는 envelop으로 구성이 된다. envelop은 메시지를 어떤 프로세스가 수신할 것인가와 같은 정보를 포함하게 된다. MPI에서는 메시지 교환만이 프로세스간에 데이터를 접근하게 할 수 있는 유일한 방법이다.

일대일 통신기능:

MPI는 MPI_Send와 MPI_Recv의 두 가지 기본함수가 있다. 이 두 가지 모두 blockling 과 non-blocking의 모드를 제공한다. blocking이란 함수 호출 시 그 함수의 기능이 모두 완료된 후 함수가 리턴 되는 것을 말하며, non-blocking이란 함수가 수행될 준비만 한 후 바로 종료 되고 background로 함수의 기능이 수행이 되는 방식을 말한다. non-blocking기능을 이용하여 통신과 연산을 중첩하여

수행 할 수 있게 된다. MPI_Send함수는 동기, 버퍼, 표준, 준비(Ready)의 네 가지 모드를 제공한다.

동기화:

동기화는 프로세스간에 동기를 맞추어 프로그램을 수행하기 위해 사용이 되며, MPI에서는 MPI_Barrier란 함수를 제공한다.

집단화 통신

병렬 프로그램에서 단순히 두 개의 프로세스간의 메시지 교환뿐만 아니라 다수의 프로세서가 참여하여야 하는 경우가 많이 있다. 예를 들어 여러 프로세스에 분산되어있는 데이터의 값을 더한다든지, 분산되어있는 행렬의 곱셈을 계산할 때 관련된 프로세스간의 통신이 필요하게 된다. 이런 집단화 통신으로 broadcast, gather, scatter, reduce, scan등의 연산이 있다.

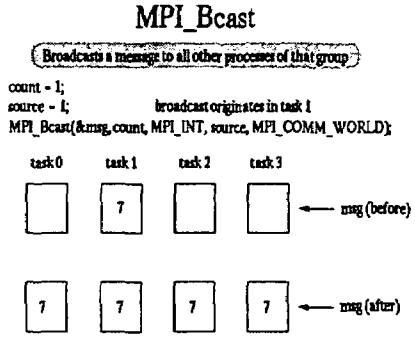


그림 3.1 Broadcast

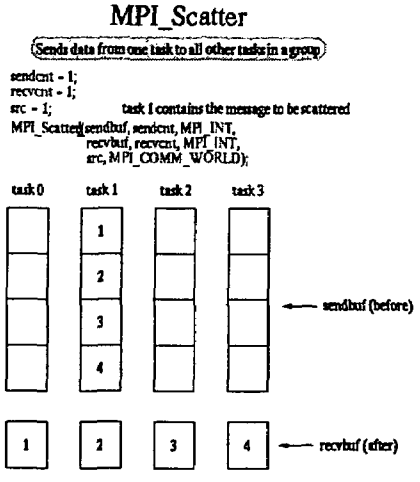


그림 3.2 Scatter

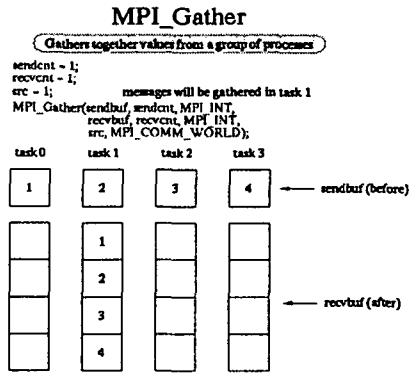


그림 3.3 Gather

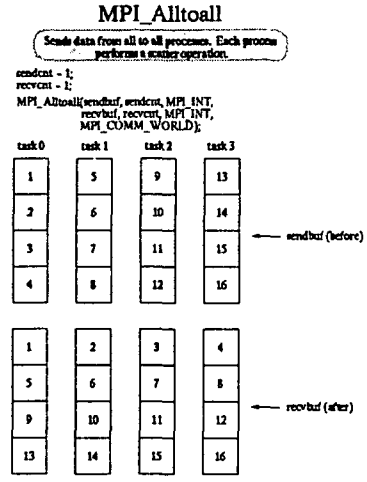


그림 3.4 Alltoall

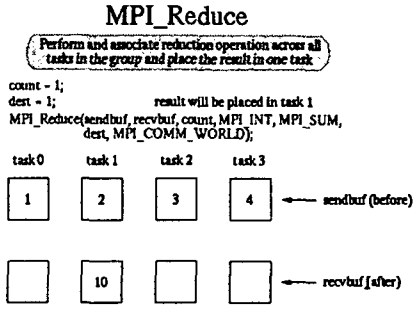


그림 3.5 Reduce

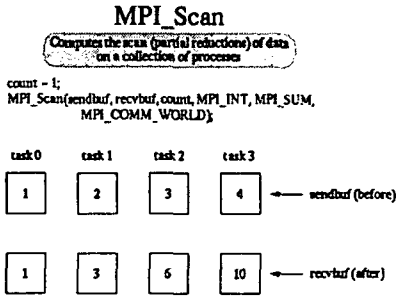


그림 3.6 Scan

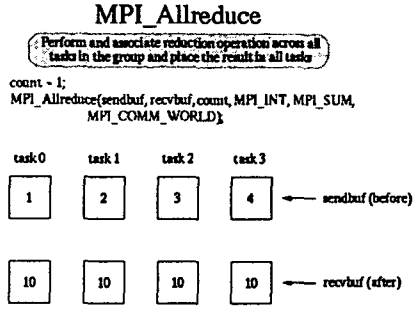


그림 3.7 Allreduce

3.2. 집단화 통신 함수의 유형 분석

라이브러리의 성능을 개선하기 위해서는 각 함수의 유형을 분석하고 각 유형별로 개선점을 찾는 과정이 필요하다. MPI의 집단화 통신함수는 broadcast형, gather/scatter형, all-to-all형, 그리고 reduce/scan형으로 크게 4가지의 유형으로 분석이 될 수 있다.

1) broadcast형

집단화 통신 함수 중에서 가장 많이 쓰이는 함수 중 하나인 broadcast의 경우 메시지를 broadcast하고자 하는 노드가 다른 노드로 메시지를 보낼 때 동일한 내용의 메시지가 다른 모든 노드로 보내주어야 하기 때문에 성능이 알고리즘에 따라서 많은 차이를 보이게 된다. 통신함수의 성능 개선을 위해서 고려할 사항은 메시지를 받은 노드가 받은 메시지를 다른 노드로 다시 전송해주어 source 노드의 부담을 덜어주는 방식의 개선방식과 큰 메시지를 broadcast할 경우 큰 메시지가 다른 노드로 전송되는 시간 동안 나머지 노드들의 idle time을 줄이는 방법을 생각할 수 있다. idle time은 큰 메시지를 적절한 크기의 작은 메시지로 분할하여 보내는 방법으로 줄일 수 있다.

2) gather/scatter형

gather와 scatter의 경우는 각 프로세서에서 보내거나 받는 메시지가 노드마다 서로 다르기 때문에 broadcast와 같은 방식으로는 성능을 개선할 여지가 없다. 다시 말해서 gather의 경우 전체 시간이 데이터가 모아지는 프로세서가 받는 데이터의 전체 크기에 의해 좌우되는데 알고리즘적으로 그 데이터의 크기를 줄일 수 없기 때문에 해결할 수 없고, 마찬가지로 scatter의 경우 데이터를 내보내는 프로세서의 총 데이터의 크기가 고정되어 있기 때문에 역시 알고리즘적 해결 방안이 없다. 전체적으로 보내는 회수를 줄여서 send와 receive시 발생하는 startup overhead를 줄일 수 있으나 큰 메시지의 경우 상대적으로 전체 시간에서 차지하는 비중이 무시할 정도이므로 큰 성능의 개선을 기대 할 수 없다.

3) all-to-all형

all-to-all형의 통신 함수는 전체 시간동안 모든 프로세서가 데이터의 통신에 참여하기 때문에 gather/scatter형과 마찬가지로 알고리즘적으로 성능을 개선할 수 없는 형태의 함수들이다. 이 형태의 함수들은 프로그램적으로 blocking이나 nonblocking send/receive 함수를 적절히 사용하여 성능을 향상시킬 수는 있지만 크게 성능개선을 할 수는 없다.

4) reduce/scan형

reduce/scan형의 함수는 통신의 양상이 broadcast와 마찬가지로 중간에 받은 노드가 다른 노드로 메시지를 보내주거나 노드의 idle time을 줄이는 방식의 알고리즘을 사용하여 성능을 개선 할 수 있다. 뿐만 아니라 reduce와 scan의 경우 통신뿐만 아니라 중간에 프로세서가 참여하는 연산과정이 필요로 하는데, 현재의 대부분의 병렬 컴퓨터가 통신에 프로세서가 참여하는 시간을 줄일 수 있도록 여러 가지 하드웨어적인 기법들을 사용하므로 연산과 통신의 중첩이라는 방법을 통해서 성능을 개선 할 수 있다.

3.3 Broadcast함수의 성능개선

1) MPI library의 broadcast(binomial tree)

기존의 MPI의 성능 개선을 위해서 먼저 MPI library의 broadcast의 알고리즘을 분석해보면 recursive doubling이라고도 불리는 binomial tree방식을 사용함을 알 수 있다.

이 알고리즘은 매 단계마다 두 배씩 전달 되게 되어 P개의 프로세서가 있다면 LogP 단계에 broadcast를 진행 할 수 있어 naive한 방식 보다 성능의 개선이 있다. 다시 말하면 broadcast를 가장 간단하게 구현할 수 있는 알고리즘은 데이터

를 가진 프로세서가 나머지 프로세서들에게 차례대로 보내는 것이다. 그러나 이 알고리즘은 통신에 참여하는 프로세서의 수가 증가할수록 통신에 걸리는 시간도 선형적으로 증가하여 많은 수의 프로세서가 broadcast 통신에 참여하게 되면 사용하기에 부적절하다. 이보다 성능을 향상시킨 것이 바로 트리를 이용한 방식의 알고리즘이다. 이 알고리즘의 도식도가 그림 3.8에 있다. 이 그림은 16개의 프로세서에서 broadcast를 log16 즉 4번의 단계로 수행하는 모습을 도식화한 것이다. 그림에서 보듯이 이 알고리즘은 먼저 데이터를 가진 프로세서가 나머지 프로세서 중 하나에 데이터를 보내고 또 다른 프로세서에게 보내는 동안 먼저 받은 프로세서는 메시지를 받지 못한 다른 프로세서에게 메시지를 보내는 식으로 모든 프로세서가 데이터를 가질 때까지 반복하도록 하여 구현된다. 만약 P개의 프로세서가 통신에 참여한다면 broadcast에 걸리는 단계 수는 log P가 된다.

이 알고리즘을 위에서 제시한 점대점 통신 방식의 모델 값으로 적용하면 다음과 같은 수행 시간이 표현된다.

$$T(m,P)=(t_0+\frac{m}{B})\log P$$

IBM SP2상에서는 이것이 다음과 같이 모델링 된다.

$$T(m,P)=(46+0.035\,m)\log P$$

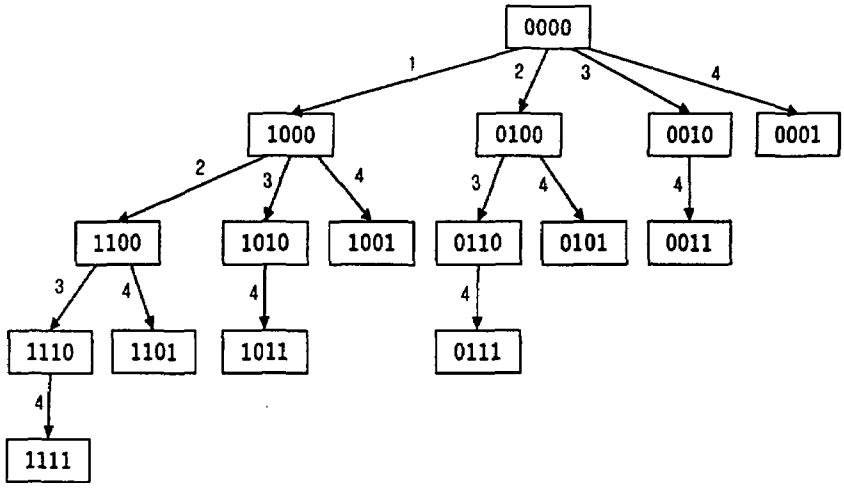


그림 3.8 binomial tree

2) edge-disjoint broadcast

edge-disjoint broadcast는 메시지를 logP개로 나누어 각각을 disjoint한 경로를 통해서 binomial tree 방식으로 전달하는 방법이다. 이 방법을 사용하면 메시지를 조각으로 나누어 통신을 함으로써 큰 메시지를 보낼 때 쉬는 프로세서의 시간을 줄일 수 있으므로 통신 시간의 개선이 일어나고 P가 늘어나도 통신 시간에 영향을 덜 줄 수 있게 되어있다.

edge-disjoint 방식의 소요시간은

$$T(m,P) = \log P \times T_{send}(\frac{m}{\log P}) + \log P \times T_{send/recv}(\frac{m}{\log P})$$

와 같이 모델링 되어진다. IBM SP2는 실제로 bidirectional one port 구조로 되어 있으나 실험 결과 MPI환경에서 send와 receive를 동시에 중첩해서 실행하게 되면 성능의 감소가 생기는 것을 확인하였다. 그러므로 $T_{send/recev}$ 는 $2\times T_{send}$로 모델링 될 수 있다. IBM SP2상에서의 모델링 값은

$$\begin{aligned} T(m,P) &= \log P(46 + 0.035\frac{m}{\log P}) + 2\log P(46 + 0.035\frac{m}{\log P}) \\ &= 3\log P(46 + 0.035\frac{m}{\log P}) \end{aligned}$$

와같이나타내어진다.

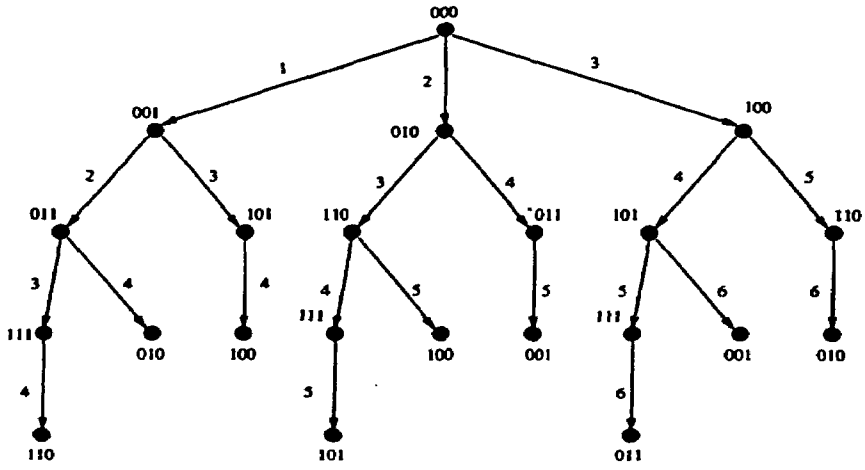


그림 3.9 edge-disjoint binomial spanning tree

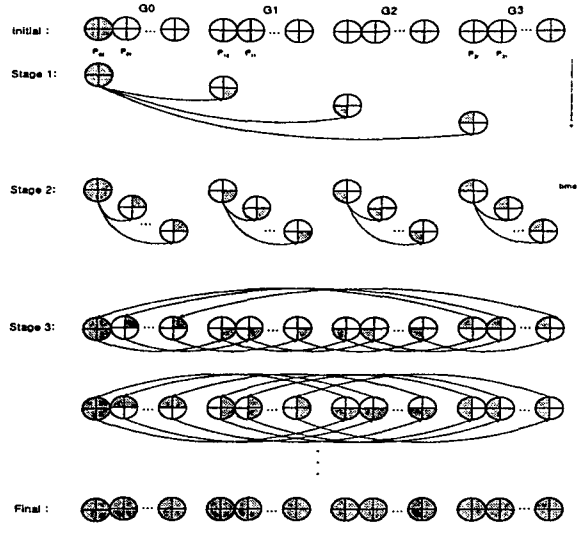


그림 3.10 k-segment broadcast

3) k-segment broadcast[18]

Edge-disjoint 알고리즘과 마찬가지로 메시지를 분할하여 통신량을 모든 프로세

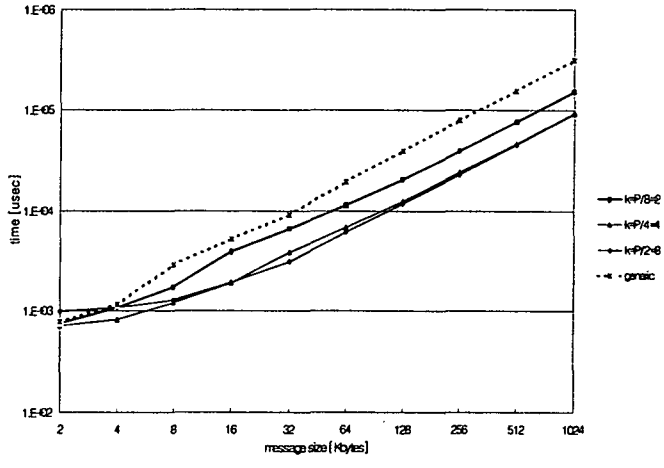


그림 3.11 segment수에 따른 broadcast 시간
(P=16)

서에 분산시킨 후 각 프로세서들이 병렬로 진행하게 하여 통신량의 균등화를 통

해 성능을 개선하는 알고리즘이다. Broadcast 함수를 향상시키기 위하여 먼저 메시지를 적절한 개수(k)의 조각으로 나누고 메시지를 가지고 있는 루트가 그 조각들을 프로세서들에게 균등하게 배분한 후 병렬로 메시지를 전송한다. 아래 그림은 8개의 프로세서와 메시지를 4개의 조각으로 나누었을 경우에 대해 이를 도식한 것이다. 전체 broadcast에 걸리는 시간은 다음과 같다.

$$T_{broadcast}(k,m,P) = (k-1)T_{send}(\frac{m}{k})+(\frac{P}{k}-1)T_{send}(\frac{m}{k})+ 2(k-1)T_{send}(\frac{m}{k})$$
$$= (3k+\frac{P}{k}-4)\,T_{send}(\frac{m}{k})$$

위의 수식을 미분해서 최소 값을 구하면 k는 $P/2$, 즉 $k=P/2$ 일 때 $T_{broadcast}$의 값이 최소가 된다.

그림 3.11은 실제로 IBM SP2에서 16개의 프로세서를 가지고 segment수에 따른 수행시간을 측정해본 결과이다. 이 실험으로 $k=P/2$ 일 때 최소 시간이 걸리는 것을 확인 할 수 있다.

$$T_{broatcast}(m,P)=46\times(\frac{3}{2}P-2)+0.035\times(3-\frac{4}{P})\times m \quad [\mu\text{sec}]$$

4) Pipeline Broadcast

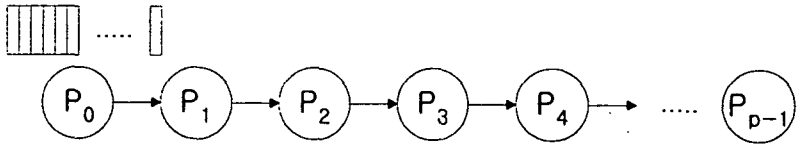


그림 3.12 pipeline broadcast

이 방법은 메시지를 작은 메시지로 분할하여 컴퓨터 CPU설계 방식인 pipeline 기법을 이용하여 선형적으로 배치한 프로세서들 사이에 메시지를 보내는 방식이다. 이 방법 역시 앞의 edge-disjoint와 k-segment알고리즘과 마찬가지로 메시지를 분할하여 보냄으로써 메시지의 크기가 큰 경우 프로세서의 수의 영향을 줄일

수 있는 방법이다. 수행시간은 다음과 같이 모델링 되어진다.

$$T_{pbcast}(M,P) = (k+P-2)T_{send/recv}(\frac{M}{k})$$
$$= (k+P-2)2T_{send}(\frac{M}{k})$$

이 식에 SP2의 점대점 통신성능의 모델값을 대입하여 정리하면

$$T_{pbcast}(M,P) = 2(k+P-2)(46+0.035\frac{M}{k})$$

수행시간을 최소화하는 k값을 구해보면

$$\frac{d}{dk}T_{pbcast}(M,P) = 2(46+0.035\frac{M}{k}) - 2(k+P-2)(0.035\frac{M}{k^2}) = 0$$

식을 정리하면

$k=\sqrt{\frac{(P-2)0.035M}{46}}$ 일 때 최소의 수행시간이 나옴을 알 수 있다.

이 알고리즘의 특징은 하드웨어의 통신시간을 모델링한 결과 값이 알고리즘의 k 값을 좌우하므로 각 하드웨어에 최적인 성능을 낼 수 있다는 특징이 있다. SP2의 경우 send와 receive를 동시에 진행하면 수행시간이 2배로 늘어남을 앞에서도 언급했다. 위의 파이프라인 구조를 보면 source 노드가 메시지를 옆의 노드로 보내면 받은 노드가 자신의 옆 노드로 메시지를 전달하는 동안 source 노드는 메시지를 보내지 못하고 멈추어 있음을 알 수 있다. 이 idle time을 없애는 방법으로 전체 P개의 프로세서가 있는 파이프를 P/2개씩 둘로 나누어서 전송을 하면 source 노드의 idle time을 줄여서 수행시간을 단축할 수 있다.

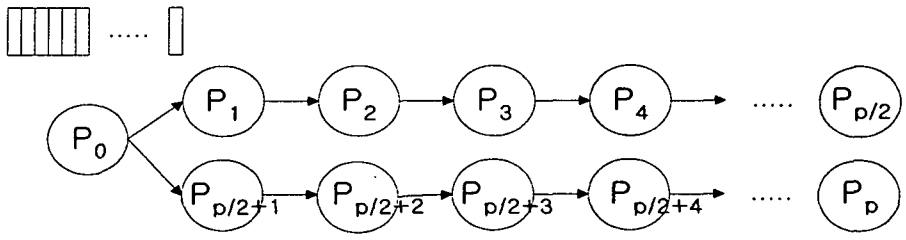


그림 3.13 pipeline broadcast with 2 path

이와 같은 방식으로 broadcast를 수행하면 수행시간은 다음과 같이 모델링 되어진다.

$$T_{pbcast}(M,P) = (k+\frac{P}{2}-2)T_{send/recv}(\frac{M}{k})$$
$$= (k+\frac{P}{2}-2)2T_{send}(\frac{M}{k})$$

이 알고리즘은 일반 파이프라인 보다 프로세서의 수가 많아질 때 더 많은 성능 개선의 효과를 얻을 수 있다.

3.4 Reduce함수의 성능개선

reduce란 각 프로세서가 가지고 있는 데이터를 가지고 최대 값, 최소 값, 곱셈, 덧셈 등의 연산을 수행하는 것을 말한다.

1) binomial 알고리즘

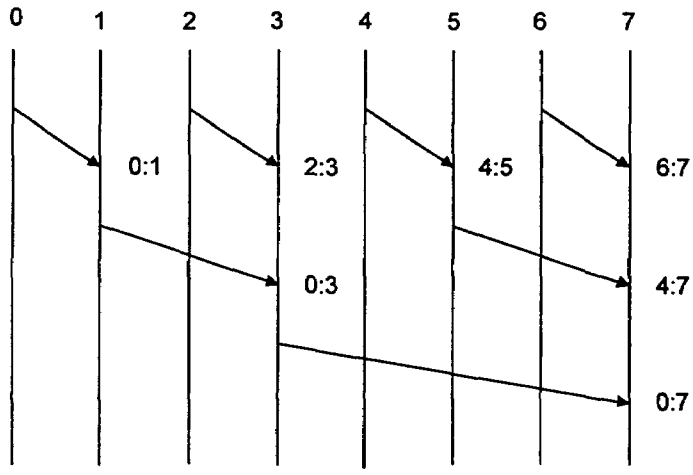


그림 3.14 binomial reduce

일반적인 reduce의 알고리즘은 all-to-all gather형태로 데이터를 모은 후 모은

데이터를 가지고 연산을 수행하는 형태로 되어 있다. 수행시간은 다음과 같이 모델링이 된다.

$$T(M,P) = \log P(t_s + (\frac{1}{B} + t_{op})M)$$

여기서 t_s는 startup 시간이고, B는 asymptotic bandwidth, 그리고 t_{op}는 연산에 필요한 시간이 된다.

2) reduction of segmented message

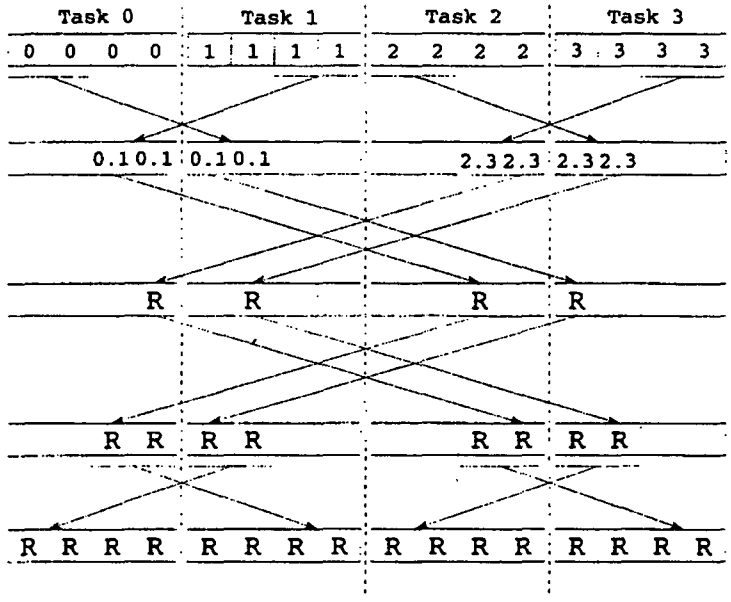


그림 3.15 reduce with segmented message

만일 연산이 교환법칙과 결합법칙이 성립이 되는 것이라면 모든 데이터를 모으지 않고 일부 연산이 수행된 값을 모아서 연산 회수와 데이터의 이동을 줄일 수 있는 방식으로 구현 할 수 있다. 위의 그림은 일반적인 logP단계의 reduce보다 단계 수는 2배가 많게 되나 연산의 회수와 데이터의 이동이 줄어듦을 알 수 있다. 첫 번째 logP단계에서는 매 단계마다 $M/2^{step}$개의 데이터를 이동하여 reduction을 수행하여 각 프로세서가 각각 1개씩의 reduction 결과를 가지게 된 후 다음 logP단계동안 결과를 broadcast하는 형식으로 진행이 된다. 수행시간은

다음과 같이 나타내어진다.

$$T(M,P) = (2T_{send}(\frac{M}{2}) + T_{comp}(\frac{M}{2})) + (2T_{send}(\frac{M}{4}) + T_{comp}(\frac{M}{4})) + \cdots + (2T_{send}(\frac{M}{2^{\log P}}) + T_{comp}(\frac{M}{2^{\log P}})) + 2T_{send}(\frac{M}{2}) + 2T_{send}(\frac{M}{4}) + \cdots + 2T_{send}(\frac{M}{2^{\log P}})$$

$$T(M,P) = 4T_{send}(\frac{M}{2}) + 4T_{send}(\frac{M}{4}) + \cdots + 4T_{send}(\frac{M}{2^{\log P}}) + T_{comp}(\frac{M}{2}) + T_{comp}(\frac{M}{4}) + \cdots + T_{comp}(\frac{M}{2^{\log P}})$$

이 식을 정리하면

$$T(M,P) = 4(t_s + \frac{M}{B}(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\log P}})) + t_{op}M(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\log P}}) = t_s 4\log P + (\frac{4}{B} + t_{op})(1 - \frac{1}{2^{\log P}})M$$

결과적으로 모델링 시간은 다음과 같다.

$$T(M,P) = t_s 4\log P + (\frac{4}{B} + t_{op})M\frac{P-1}{P}$$

M의 크기가 작아서 startup 시간의 영향이 크지 않다면 이 알고리즘의 성능이 뛰어남을 알 수 있다.

3.5 Scan함수의 성능개선

scan이란 프로세서들이 가지고 있는 데이터를 프로세서의 인덱스까지의 데이터들만 가지고 연산을 수행하는 과정을 말한다. 여기에서는 세가지 알고리즘을 제시한 후 그 성능을 예측하고 비교해보도록 하겠다.

1) Pipeline Scan

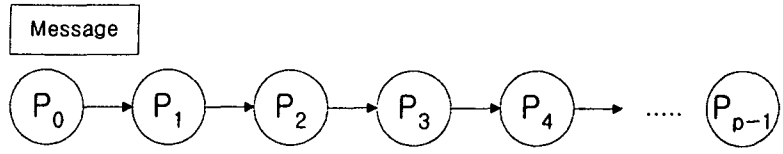


그림 3.16(a) linear array scan

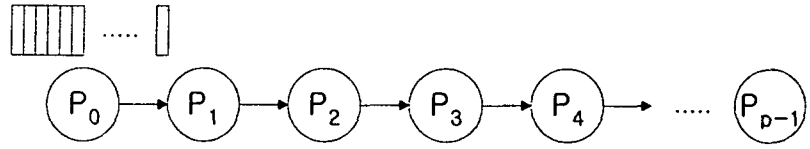


그림 3.16(b) pipeline scan

그림 6.9의 a)와 b)는 프로세서를 linear array 형태로 배열한 후 왼쪽에서 오른쪽으로 메시지를 보내는 방식으로, 각 프로세서는 수신한 메시지를 가지고 연산을 수행한 후 다른 프로세서로 송신을 하는 방식으로 진행된다. a) 의 경우 프로세서의 수에 비례하여 scan연산의 시간이 결정되기 때문에 많은 프로세서로 작업을 수행할 경우 시간이 많이 소요되는 알고리즘이다. 연산 시간은 다음과 같이 모델링 된다.

$$\begin{aligned} T_{scan}(M,P) &= P(T_{send/recv}(M) + T_{comp}(M)) \\ &= P(2T_{send}(M) + T_{comp}(M)) \end{aligned}$$

a)의 단점을 보완하기 위해서 b)와 같은 pipelining 방식을 사용하면 큰 메시지를 작은 메시지들로 분할하여 송신하기 때문에 프로세서의 idle시간을 줄일 수 있기 때문에 프로세서 수와 무관한 연산 시간을 얻을 수 있다. 하지만, 메시지를 너무 많이 분할할 경우에는 send와 receive에 소요되는 startup overhead인 t_s의 영향으로 성능이 오히려 저하 될 수 있으므로 메시지 크기에 따른 최적의 segment의 수를 찾아 연산시간을 최소화 할 수 있다.

$$\begin{aligned} T_{scan}(M,P) &= (k+P)(T_{send/recv}(\frac{M}{k}) + T_{comp}(M)) \\ &= (k+P)(2T_{send}(\frac{M}{k}) + T_{comp}(M)) \end{aligned}$$

2) Brent & Kung 알고리즘[15]

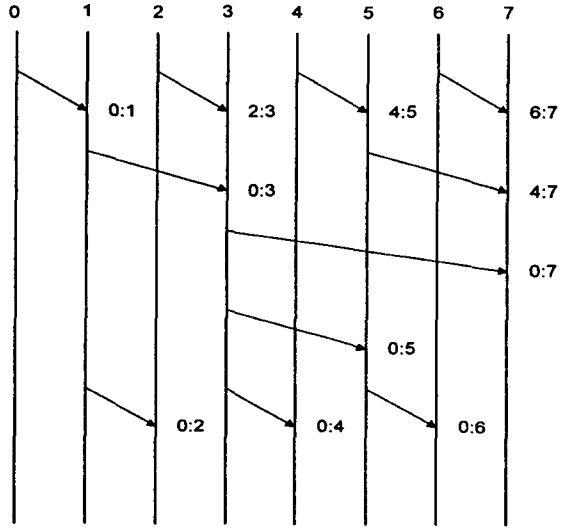


그림 3.17 Brent & Kung scan

이 알고리즘은 scan연산을 2logP-1 단계로 수행할 수 있는 알고리즘이다. 밑의 그림을 보면 먼저 제일 첫 번째 단계에서 짝수 인덱스를 가진 프로세서가 홀수 인덱스의 옆 노드로 메시지를 보내고, 맨 마지막 단계에서 홀수 인덱스의 프로세서가 짝수 인덱스의 옆 프로세서로 보내는 방식이다. 그리고 그 사이에 홀수 인덱스를 가진 프로세서만 가지고 위와 같은 방식으로 반복적으로 네트워크를 구성하면 된다.

이 알고리즘의 수행 단계는

$$BK(P) = 2 + BK(\frac{P}{2}) \quad , \quad BK(2) = 1$$

위의 식을 정리하면 2logP-1단계가 된다. 그러므로 수행 시간은 다음과 같이 모델링 되어진다.

$$T_{scan}(M, P) = (2\log P - 1)(T_{send}(M) + T_{comp}(M))$$

이 알고리즘 또한 reduce에서와 마찬가지로 통신시간과 연산시간의 중첩으로 성능을 좀더 개선시킬 수 있다.

3) RootP 알고리즘

이 알고리즘은 위에서 설명한 Brent&Kung의 알고리즘 보다는 단계가 더 많은 $2\sqrt{P}+1$로 나타난다. 하지만 Brent&Kung의 알고리즘은 한 프로세서에 통신작업 시간이 집중되어 있기 때문에 메시지를 작게 나누어 보내는 파이프라인과 같은 방식이 구현되기 힘든 반면 RootP 알고리즘은 프로세서들의 작업이 Brent&Kung에 비해 고르게 분배되어 있기 때문에 작게 메시지를 나누어서 보내면 노드의 idle time을 줄일 수 있기 때문에 통신시간 개선의 효과를 얻을 수 있다.

이 알고리즘의 수행 시간은 다음과 같이 모델링 된다.

$$T_{scan}(M,P) = 2(\sqrt{2P}-3)(T_{send}(M)+T_{comp}(M))$$

그림 6.11의 0번 프로세서에서 출발하는 메시지 경로를 살펴보면 각 step마다 하나씩 거리가 더 떨어진 노드로 전송이 되고 있다. 즉 s를 0번에서 출발하는 메시지 경로의 단계 수라고 하면 s번째 단계에 $\frac{s(s+1)}{2}$ 개의 프로세서 노드로 메시지를 보낼 수 있게 된다. 다시 말해서 대략적으로 $s = \sqrt{2P}-1$ 의 값으로 나타나게 된다. 0번 노드에서 보내지는 경로에 s단계까지 메시지가 이동한 후 밑에 다시 s-1번의 단계가 필요하기 때문에 전체 수행시간은 위에 설명한 것과 같은 형태로 나오게 된다.

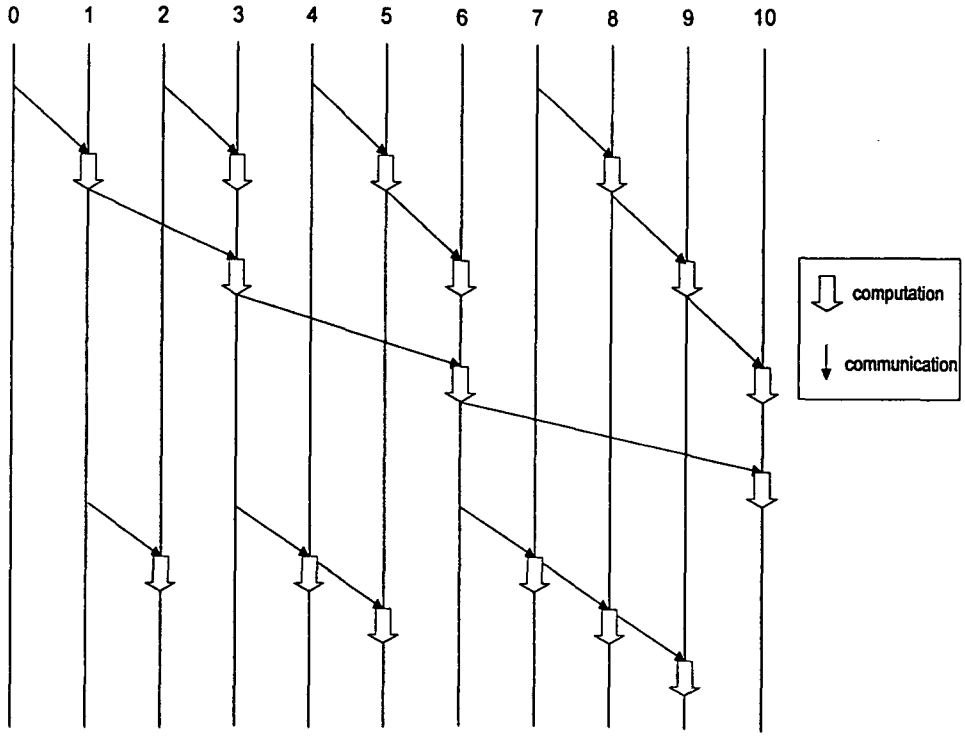


그림 3.18 $\sqrt{P}$ 알고리즘

3.6 Gather/Scatter함수의 성능개선

앞장의 통신 유형 분석에서도 설명했듯이 gather와 scatter의 경우는 통신 시 메시지의 이동이 한 프로세서에 집중되어 있는 현상이 있기 때문에 메시지 수신이나 송신 사이의 지연이 없다면 알고리즘적으로 개선할 수 있는 방법이 없다. 실험결과 두 연산을 단계를 줄이는 방식과 가장 단순한 방식과의 실행 시간의 차이가 없음을 확인하였다. gather와 scatter의 경우 같은 알고리즘을 non-blocking과 blocking의 서로 다른 통신 방식을 이용하여 구현하였을 때 성능의 차이가 많이 나타나는 현상이 있음이 확인되었다.

3.7 All-to-all 형태의 성능 개선

모든 프로세서가 통신에 참여하게 되는 all-to-all형태의 통신에서는 앞에서 제시한 broadcast와 같이 통신량의 분배로 인한 개선 효과는 찾을 수가 없다. all-to-all 형태의 통신은 통신의 단계를 줄임으로서 얻어지는 Startup 시간의 감소나 각 단계가 끝난 후 다음 단계로 진행되는 알고리즘상의 문제를 개선함으로서 성능의 향상을 꾀할 수 있다. 하지만 실제로 단계를 줄이는 방법으로의 통신시간의 개선은 미미한 것으로 실험결과 나타났다. 알고리즘적 해결 방법이 아닌 통신함수의 종류에 따라서는 성능에 많은 차이가 나타난다. 통신 실험에서 blocking send/receive 와 non-blocking send/receive로 같은 알고리즘을 구현했을 때 차이가 나타나는 것으로 확인하였다.

제 4 장 실험 결과

4.1 실험에 사용된 시스템

1) IBM SP2

IBM SP2[2]는 multistage interconnection network 구조이며 wormhole routing을 이용한 스위치를 사용하고 있다. 또한 통신 전용 프로세서가 있어서 메인 프로세서가 통신에 관여하는 것을 최소화하였다.
IBM SP2는 분산 메모리형태의 컴퓨터이다. 즉, 프로세서(노드)들이 상호연결망(interconnection network)을 통하여 연결되어 있는 형태이다.

• Network Topology

네트워크는 두 가지 형태의 장치, 즉 노드와 스위칭 장치를 통신 선으로 연결하는 것으로 이루어지게 된다. 여기서 노드는 스위칭과 관련 없는 프로세서나 I/O 노드 와 같은 것을 말하고, 스위칭 장치는 입력포트에 도착한 데이터를 적절한 출력포트로 보내주는 다수의 입력과 출력포트로 구성되어 있다. SP2의 네트워크는 bidirectional multistage interconnection networks (MIN's)의 구조이다. bidirectional MIN은 각 link가 양방향으로 데이터 이동할 수 있는 두 개의 channel로 구성이 되어 있다. MIN형태의 네트워크는 노드의 수가 많아지더라도 bisection bandwidth를 늘릴 수 있기 때문에 많은 노드가 있더라도 Bus형태와 같은 병목현상 등으로 인한 네트워크의 성능 감소가 거의 나타나지 않는다. HPS에서 메시지는 패킷(Packet)으로 분할되어 노드 사이에 전송이 되게 된다. 모든 패킷은 패킷이 가야할 경로 정보를 가지고 있어서 스위치는 그 정보를 가지고 패킷을 목적하는 노드로 도착할 수 있도록 해준다. 흐름 제어를 하는 최소단위를 flit(flow-control digit)이라고 하는데 SP2의 스위치에서는 1byte를 말한다. SP2

에서 패킷전송은 wormhole routing 방식으로 진행된다. wormhole routing은 패킷의 flit들이 스위치에 도착하는 대로 스위치가 적절한 출력포트로 전송을 시켜주는 방식이다. wormhole routing과 대비되는 방식으로 store-and-forward방식이 있는데 이 방식은 패킷 전체가 스위치에 도착 된 후 다음 단계로 전송하는 방식으로 데이터를 주고받는 노드사이의 스위치의 개수에 비례하는 시간이 소요되기 때문에 wormhole routing에 비해 성능이 떨어진다. wormhole routing의 경우에 만일 들어온 flit을 내보낼 출력포트가 다른 일을 하느라고 사용할 수 없는 경우에는 버퍼에 잠시 flit을 보관하였다가 출력포트를 다시 사용할 수 있게 되면 다시 전송을 재개하는 방식을 사용한다. SP2의 경우 일반적인 wormhole routing과 약간의 차이가 있다. SP2에서는 flit이 blocking된 입력포트에 버퍼링을 시키는 일반적인 방식과 달리 모든 입력포트에서 blocking된 flit을 Central Queue에 함께 보관하는 방식을 사용한다.

• SP2 processor 통신 어댑터

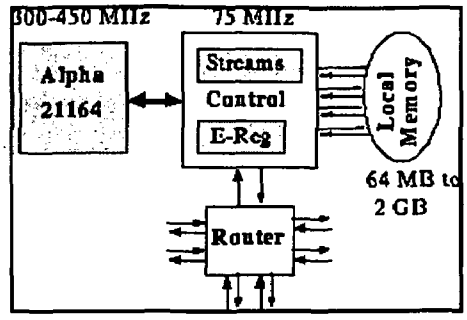
SP2의 통신 어댑터는 대역폭(bandwidth)을 늘리고 노드 프로세서의 통신에 대한 부담을 줄이기 위해서 많은 기법들을 사용하고 있다. SP2는 micro-channel의 bus master와 streaming기능을 이용하여 80MB/s의 최고속도를 나타낸다. 또한 micro- channel뿐만 아니라 Intel의 i860XR 64bit 마이크로 프로세서가 통신 전용프로세서로 사용되며, CRC 발생과 같은 데이터 checking 기능을 하드웨어로 구현하였다.

본 연구에서는 MHPCC의 SP2시스템을 이용하여 실험을 하였다.

2) Cray T3E

T3E는 Shared memory 방식의 MPP로써 DEC Alpha 21164 프로세서를 3D torus 구조로 연결하였으며, scalable한 성능을 얻기 위해서 여러 가지 많은 기법들을 사용하고 있다.

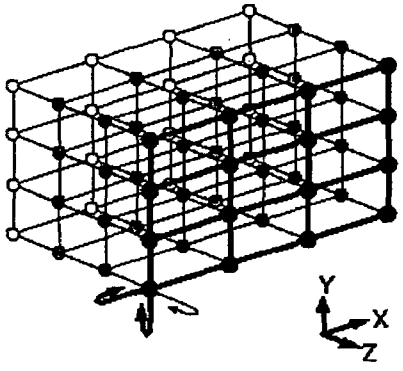
• Processing Element



T3E node

각 노드는 300-450MHz의 Alpha 21164의 프로세서와 Shell이라고 불리는 회로로 구성이 되어있다. Shell회로는 local memory와 control칩, 그리고 라우터 칩으로 이루어 진다. Shell회로는 75MHz로 동작을 한다. control칩에 있는 Stream과 E-register는 latency를 줄일 수 있게 고안된 방법으로 T3E에서 노드의 수가 많아지게 되어도 성능의 둔화가 생기지 않게 해준다. 64M에서 2G까지의 지역 메모리를 갖을 수 있으며, 1.2GB/s의 peak 대역폭을 나타낸다. 라우터 칩은 7개의 bidirectional port들로 구성이 되어있어서 1개는 PE에 연결되며 나머지 6개는 Torus network에 연결이 되는 구조를 가지고 있다. T3E의 PE는 보드 차원의 캐쉬 없이 프로세서 내의 2-Level(1-level: 8KB, 2-level: 96KB) 캐쉬를 이용한다.

• Interconnection Network



3D Torus

T3E는 3D torus network를 이용하여 low-latency, high-bandwidth의 통신을 제공한다. Interconnection network는 매 시스템 클럭(13.3ns)마다 64bit의 word를 6개의 link에서 전송할 수 있어서 512 PE의 시스템에서 122GB/s 이상의 bisection bandwidth를 갖는다. 통신은 통신양이 많은 부분을 피하여 최단경로로 갈 수 있는 알고리즘이 적용이 되어 있다.

• Operating System

T3E는 self-host 시스템으로 운영체제는 Cray의 64bit 운영체제로 사용되는 UNICOS를 변형한 UNICOS/mk를 사용한다. UNICOS/mk는 single system image를 kernel level에서 지원하는 분산 운영체제이다.

본 연구에서는 SERI의 T3E시스템을 사용하여 실험하였다.

3) PC Cluster

프로세서의 성능이 계속 발전하면서 적은 비용으로 빠른 성능의 연산을 행하려는 시도로 최근에 PC Cluster분야가 새롭게 각광을 받고 있다. Cluster는 일반 PC를 Fast Ethernet등의 빠른 network장치로 연결을 하여 linux등의 운영체제에서 MPI, PVM등을 사용하여 병렬 처리를 하는 것이다.
본 연구에서는 16개의 PentiumII PC를 이용하여 Linux환경에서 MPI라이브러리를 이용하여 실험 하였다.

실험은 세가지 시스템에서 같은 프로그램을 각각 수행하였다. 기존의 MPI 라이브러리는 최적화 되어 있다고 생각하여 모든 프로그램은 -O3 옵션으로 최적화하여 컴파일 하였다. 최적화 여부가 성능에 많은 차이가 나타남을 확인 하였다. 특히 계산량이 많은 reduce와 scan의 경우 최적화 할 경우 많은 성능의 개선이 생겼다. 실험결과는 수행시간을 log scale로 나타 냈다.

4.2 Broadcast

본 연구에서 실험한 k-segment, nEBST, pipeline의 세가지 알고리즘은 모두 프로세서의 수가 늘어나도 늘어난 프로세서의 수가 수행시간에 영향을 적게 줄 수 있도록 고안된 알고리즘들로, 세 알고리즘 모두 큰 메시지 영역에서는 좋은 성능을 보이고 있다. 기본적으로 메시지를 분할 전송함으로써 프로세서의 idle time을 없애고 있다. 실험결과를 보면 같은 알고리즘이라도 시스템에 따라서 다른 양상을 보이고 있음을 알 수 있다. 실험결과를 통해서 각 시스템의 특성을 파악할 수 있다.

Cluster의 경우 다른 두 시스템에 비해 실험 결과에 많은 차이가 있음을 알 수 있다. 가장 많은 이유는 먼저 통신 성능이 특정 메시지 크기에서는 운영체제의 영향으로 통신성능이 급격하게 감소하게 된다. 8Kbytes 이상의 영역부터 이와같은 현상이 나타나는데 메시지가 아주 큰 영역에서도 같은 현상이 나타나지만 통신시간 또한 늦어지기 때문에 영향이 별로 없이 다시 선형적인 모습을 보여주고 있다.

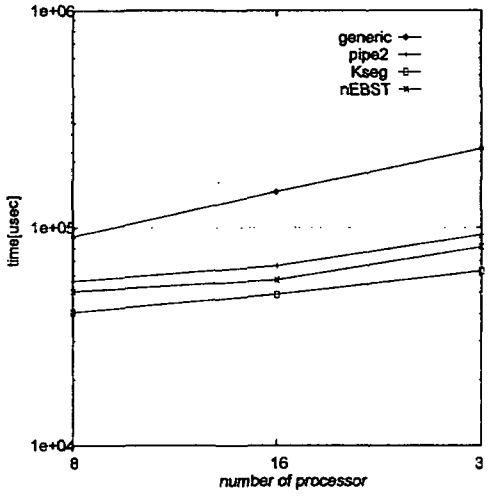


그림 4.1 프로세서 수에 따른 성능 (M=512Kbytes)

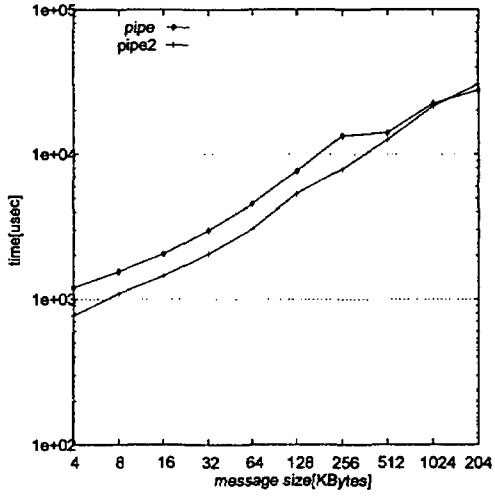


그림 4.2 2-path pipeline와 pipeline의 비교(SP2 P=16)

그림 4.1은 K-segment, nEBST, pipeline 모두 프로세서의 수가 늘어나더라도 실행시간은 많이 증가되지 않아 모델링에서 예상한 결과가 나왔음을 알 수 있다. pipeline방식의 broadcast의 성능은 메시지 크기가 작은 영역에서는 메시지의 조

각의 수가 너무 작기 때문에 실제 pipelining의 성능을 얻을 수 없으나 메시지 크기가 큰 영역에서 좋은 성능이 나타남을 알 수 있다. 그림 4.2는 send와 receive를 중첩해서 실행할 수 없는 구조를 위한 2-path pipelining 알고리즘은 작은 메시지 크기에 대해서는 pipeline보다 많은 성능의 개선이 나타남을 보여준다.

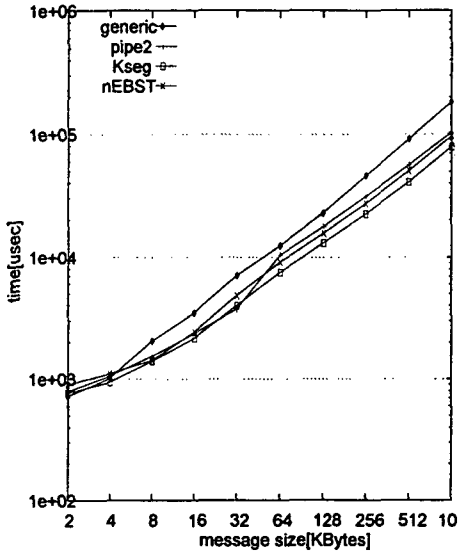


그림 4.3 SP2 broadcast (P=8)

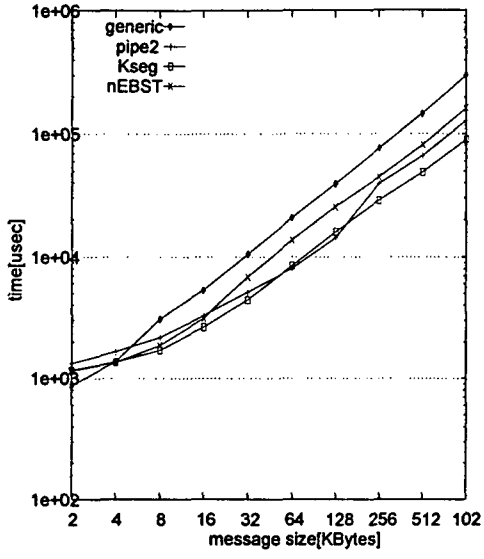


그림 4.4 SP2 broadcast (P=16)

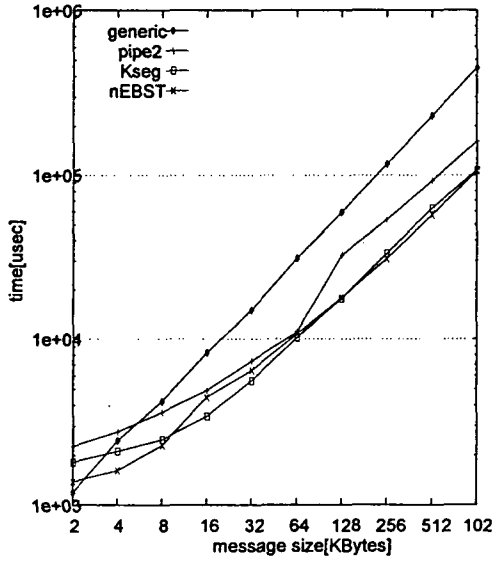


그림 4.5 SP2 broadcast (P=32)

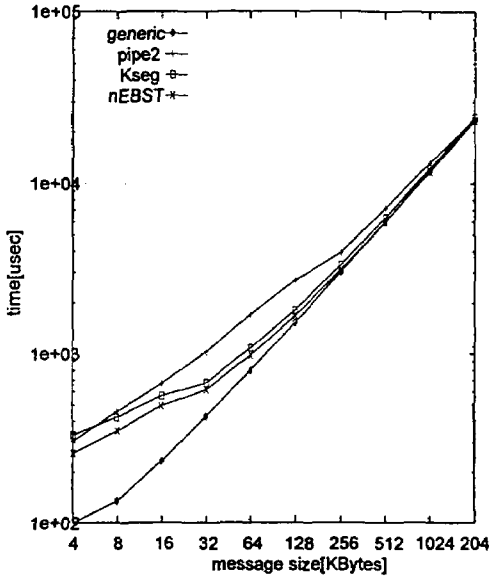


그림 4.6 T3E broadcast (P=8)

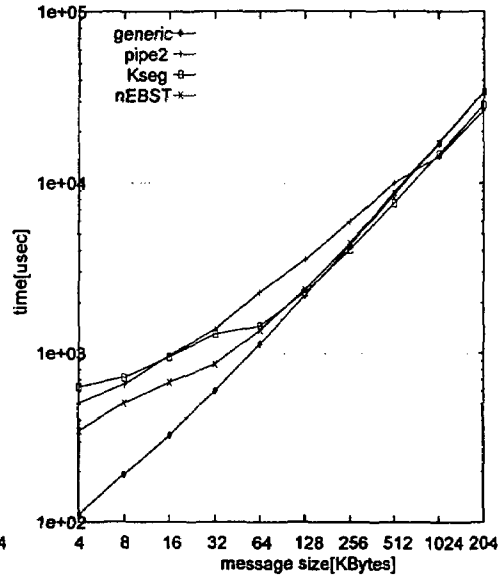


그림 4.7 T3E broadcast (P=16)

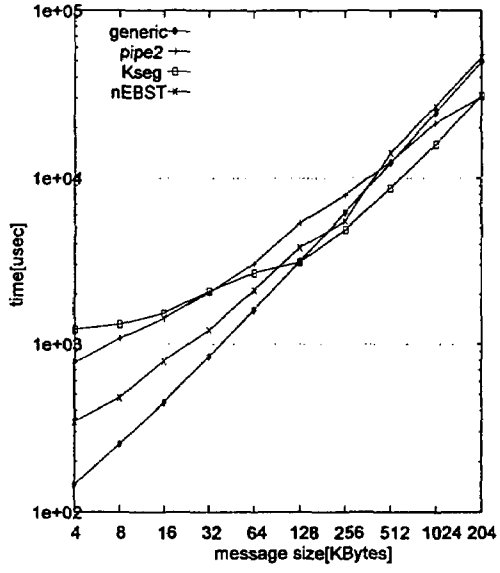


그림 4.8 T3E broadcast (P=32)

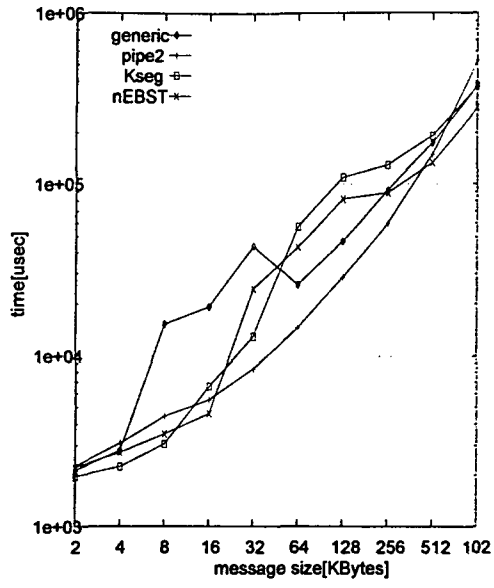


그림 4.9 Cluster broadcast (P=8)

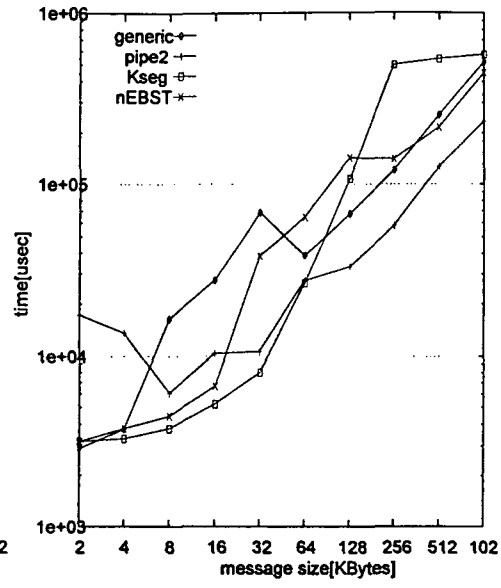


그림 4.10 Cluster broadcast (P=16

SP2에서 Pipeline의 경우 프로세서의 수가 많아지거나 메시지 크기가 커지게 되면 성능의 감소가 나타나는데 이것은 통신의 성능을 linear하게 모델링을 하였지만 실제로는 큰 메시지를 보낼 때 더 큰 대역폭을 갖기 때문에 메시지를 분할하면 성능이 떨어지는 것을 알 수 있다. T3E에서는 전체적으로 성능이 좋지 않고 큰 메시지 영역에서 약간 좋은 성능이 나타나게 되는데 이것은 T3E의 통신 속도가 빠르기 때문에 Startup time이 상대적으로 대역 폭에 비하여 큰 값을 갖기 때문에 작은 크기의 메시지 영역에서는 메시지 분할을 많이 할수록 성능이 나빠지기 때문이다.

4.3 All-Reduce

segmented message방식은 모델링 값에서도 알 수 있듯이 프로세서의 수의 증가에 둔감하게 동작되므로 프로세서의 수가 증가 될 수록 기존의 함수보다 성능이 개선됨을 알 수 있다.

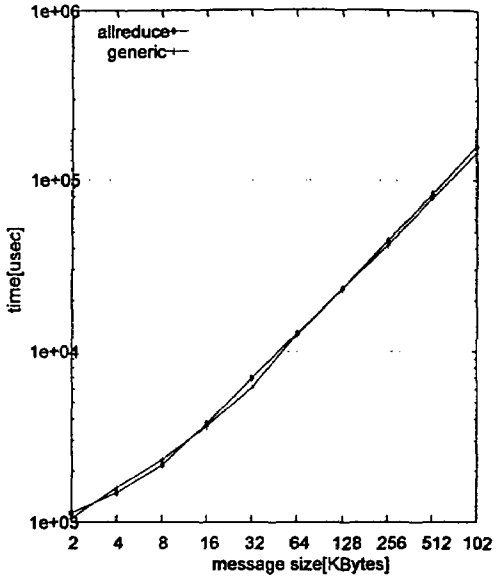


그림 4.11 all-reduce on SP2 (P=8)

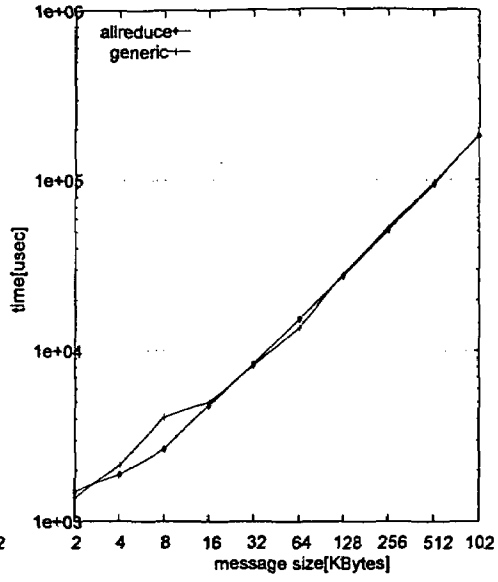


그림 4.12 all-reduce on SP2 (P=16)

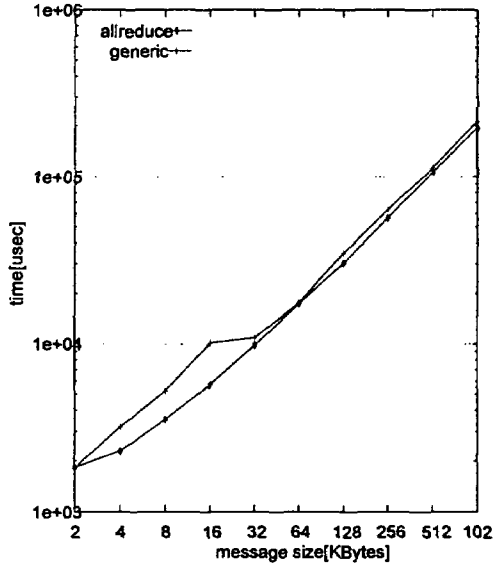


그림 4.13 all-reduce on SP2 (P=32)

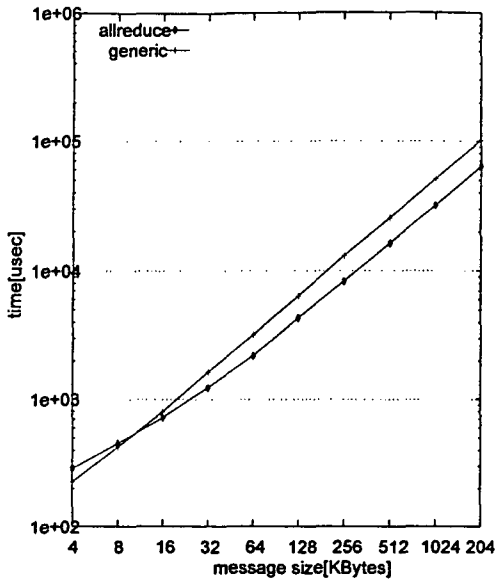


그림 4.14 all-reduce T3E (P=8)

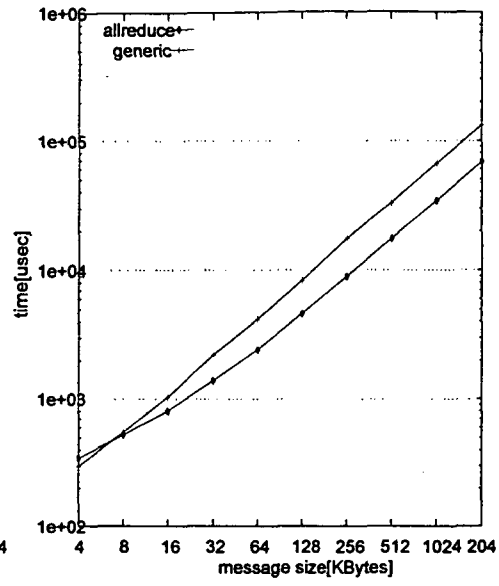


그림 4.15 all-reduce T3E (P=16)

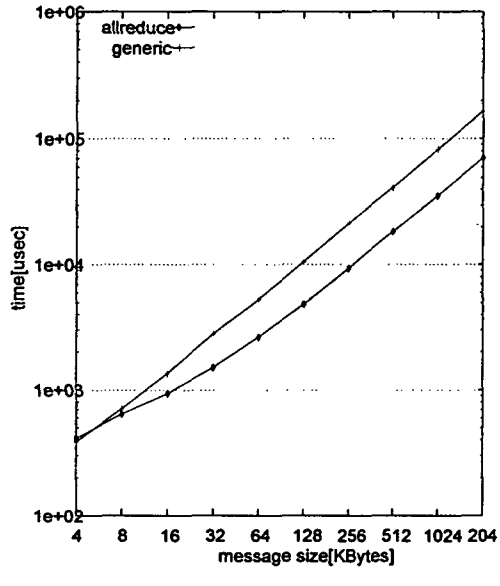


그림 4.16 all-reduce on T3E (P=32)

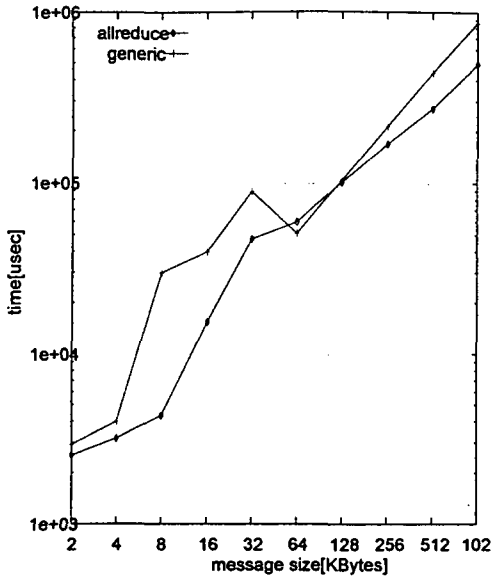


그림 4.17 all-reduce on Cluster(P=8)

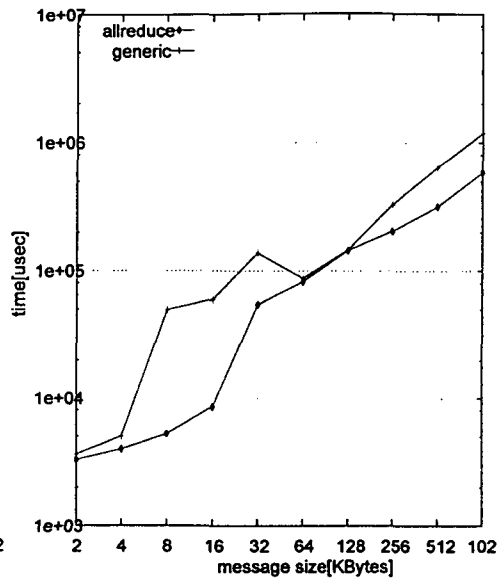


그림 4.18 all-reduce on Cluster(P=16

4.4 Scan

본 실험에서는 RootP, Brent&Kung 그리고 이 두가지를 작은 메시지로 나누어서 반복하여 수행하는 알고리즘과 pipeline 그리고 pipeline의 통신과 연산을 non-blocking 함수를 이용하여 중첩시키는 알고리즘의 모두 6가지의 함수를 기존의 MPI함수와 비교하여 실험하였다.

단계별로 수행되는 RootP와 Brent&Kung알고리즘은 거의 비슷한 성능을 보이며 큰 메시지를 여러개의 작은 메시지로 분리하여 RootP와 Brent&Kung을 연속적으로 수행하여 Scan하는 알고리즘에서는 RootP 알고리즘이 한 프로세서에 집중적으로 작업이 모여있는 Brent&Kung알고리즘에 비하여 모델링에서 예상한 것과 같이 많은 성능의 개선을 보여준다.

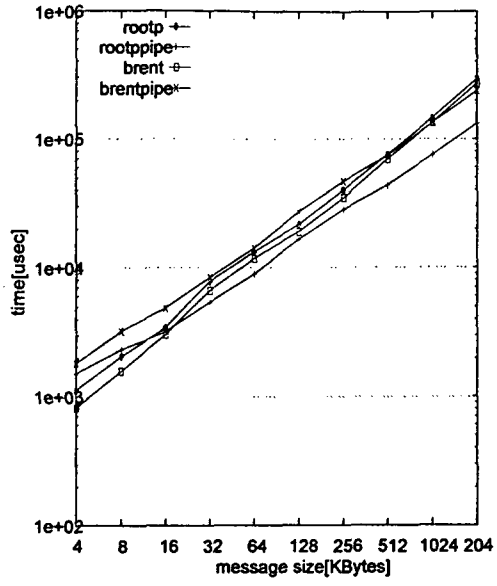


그림 4.19 작은 메시지로 분할 할 때의
RootP 알고리즘의 성능개선

또한 pipeline scan을 통신과 계산을 non-blocking 함수를 이용하여 중첩 시키는 실험을 하였는데 T3E의 경우는 공유메모리 구조이기 때문에 중첩을 하면 오히려 성능이 나빠지는 현상을 볼 수 있다.

전체적으로 pipeline방식으로 scan을 하는 알고리즘이 모든 시스템에서 가장 좋은 성능을 보임을 확인 할 수 있다.

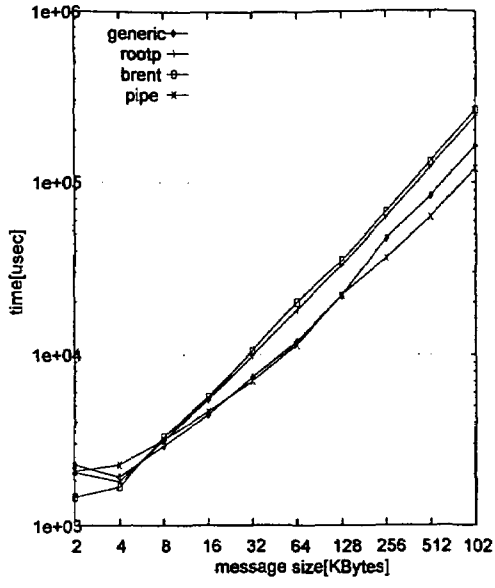


그림 4.20 Scan on SP2 (P=8)

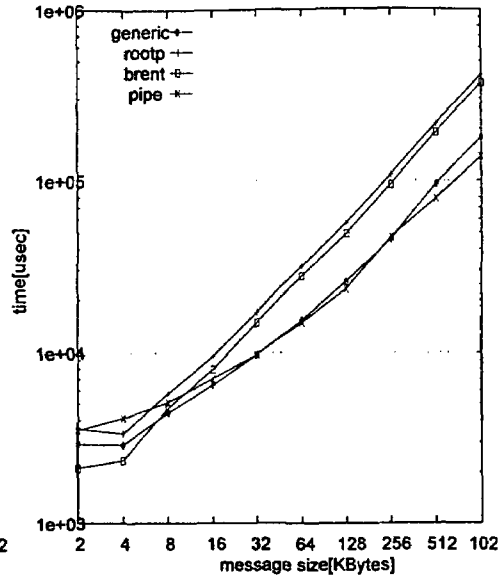


그림 4.21 Scan on SP2 (P=16)

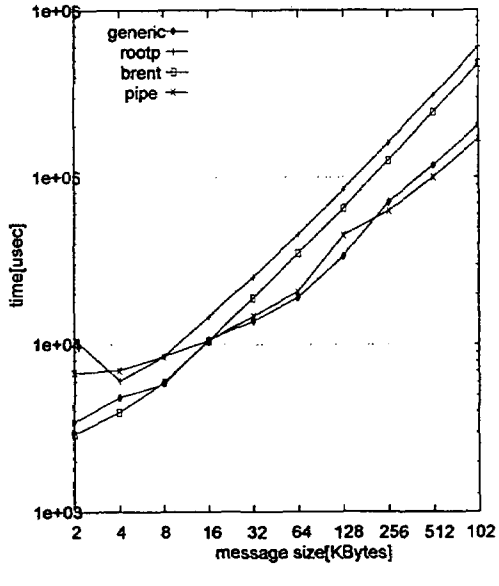


그림 4.22 Scan on SP2 (P=32)

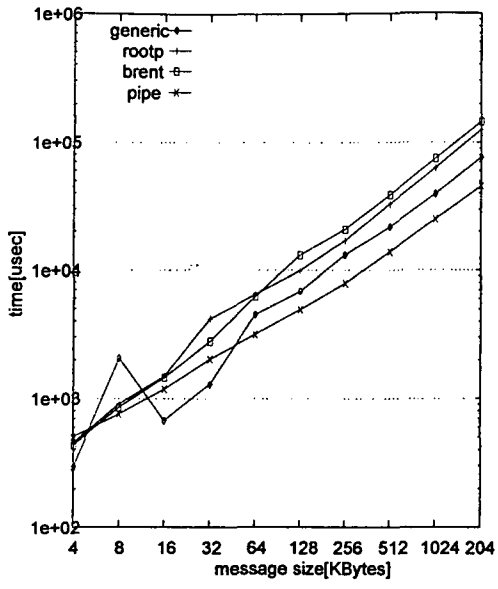


그림 4.23 Scan on T3E (P=8)

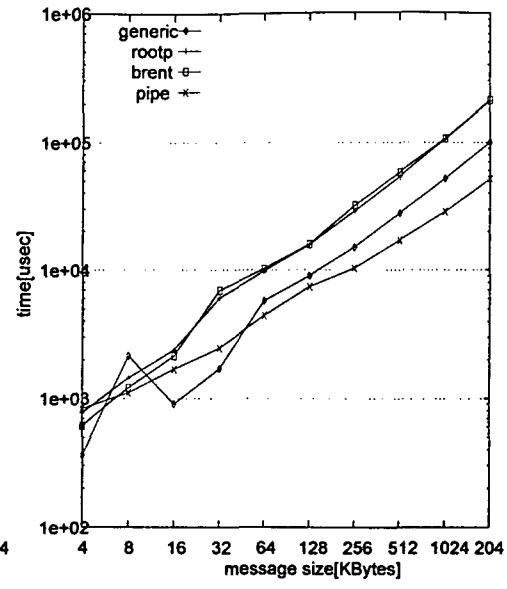


그림 4.24 Scan on T3E (P=16)

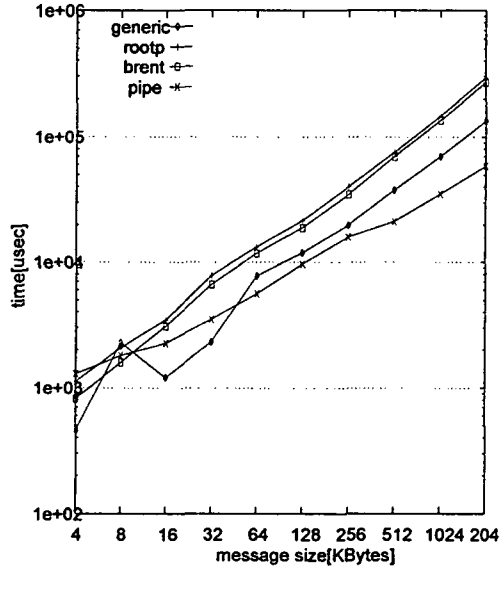


그림 4.25 Scan on T3E (P=32)

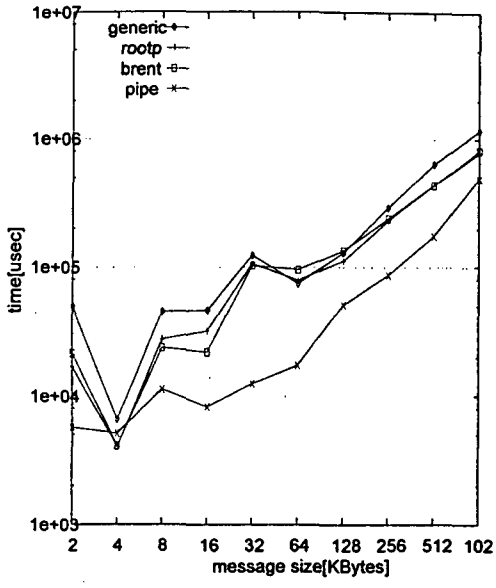


그림 4.26 Scan on Cluster (P=8)

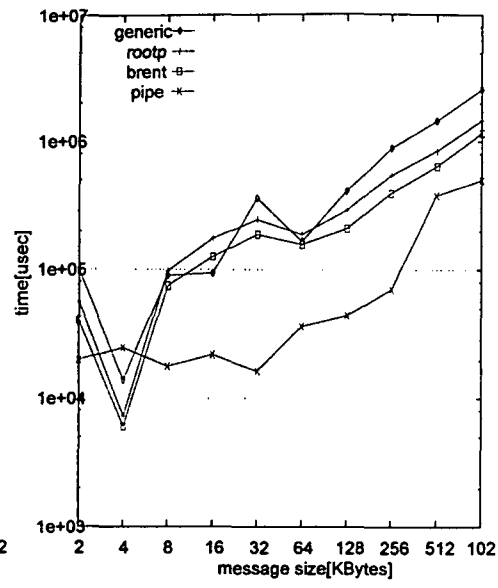


그림 4.27 Scan on Cluster(P=16)

제 5 장 결론

본 연구에서는 초고속 컴퓨터의 성능을 높이도록 집단화 통신을 메시지 분할 전송을 통하여 통신 시간을 단축시킬 수 있는 방법을 제시하였다. 즉 메시지를 분할하여 프로세서의 idle time을 줄여 통신량을 적절히 분배함으로써 성능이 개선되었다. 또한 제안된 알고리즘을 구현하고 실험을 통해서 기존의 MPI 라이브러리보다 성능이 우수함을 확인하였다.

Broadcast의 경우 SP2에서는 edge-disjoint, k-segment 그리고 pipeline 모두 MPI 라이브러리보다 200%정도의 성능개선을 보였고, 프로세서의 수가 증가하더라도 성능에는 큰 영향을 주지 않았기 때문에 프로세서의 수가 많아질수록 성능은 더욱 좋아졌다. 특히 pipeline을 이용한 broadcast와 scan의 경우는 하드웨어의 성능을 ping-pong method 실험을 통하여 구해진 모델링한 파라미터 값만을 이용하여 알고리즘의 segment의 수를 조정함으로써 상당한 성능의 개선됨을 확인하였다.

Reduce와 scan의 경우는 대부분의 병렬 컴퓨터가 지원하는 통신 전용 하드웨어의 기능을 이용하여 통신과 연산의 중첩을 통해서 기존의 라이브러리보다 좋은 성능을 얻게 되었다.

참고 문헌

[1] Blaise M. Barney, "The Message Passing Interface", HPC Asia '97, 1997

[2] Craig B. Stunkel et al., "The SP2 Communication Subsystem", 1994

[3] D. E. Culler, R. M. Karp, D. A. patterson,"LogP: Towords a Realistic Model of Parallel Computation", In fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, May 1993

[4] Hong Xu, Yadong Gui and Lionel M. Li, "Optimal Software Multicastin Wormhole-Routed Multistage Networks", IEEE Transactions on Parallel and Distributed Systems Vol. 8, No. 6, June 1997, pp 597-607

[5] "IBM Parallel Environment for AIX: Operation and Use", IBM Corp.

[6] "IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference", IBM Corp.

[7] J. Miguel, A. Arruabarrena, R. Beivide and J. A. Gregorio, " Assessing the Performance of the New IBM SP2 Communication Subsystem", IEEE Parallel & Distributed Technology, Winter 1996, pp 12-22

[8] Kai Hwang, Zhiwei Xu and Masahiro Arakawa, "Benchmark Evaluation of the IBM SP2 for Parallel Signal Processing", IEEE Transactions on Parallel and Distributed Systems Vol. 7, No. 5, May 1996, pp 522-535

[9] LoK Tin Liu, David E. Culer and Chad Yoshikawa, " Benchmarking Message Passing Performance using MPI", International Conference

on Parallel Processing '96, 1996, pp. 101-110

[10] "MPI: A Message-Passing Interface Standard", MPI Forum, 1995

[11] J.-Y. L. Park, H.-A. Choi, N. Nupairoj, and L. M. Ni, "Construction of Optimal Multicast Trees Based on the Parameterized Communication Model", In Proceedings of the 1996 International Conference on Parallel Processing, pages 180-187, Aug. 1996

[12] Peter S. Pacheco, "Parallel Programming with MPI", Morgan Kaufmann Publishers, 1997

[13] Philip K. McKinley, Yih-jia Tsai and David F. Robinson, "Collective Communication in Wormhole-Routed Massively Parallel Computers",

[14] Ronald Marz, "Reducing the Variance of Point-to-Point Tranfers for Parallel Real-Time Programs", IEEE Parallel & Distributed Technology, Winter 1994, pp 20-31

[15] S. Lakshmivarahan nad S. K. Dhall, "Parallel Computing Using the Prefix Problem", Oxford University Press, 1994

[16] Y. Tanaka, K. Kubota, M. Matsuda, M. Sato and S. Sekiguchi, "A Comparison of Data-Parallel Collective Communication Performance and Its Application", Proc. HPC Asia'97, 1997, pp. 137-144

[17] Zhiwei Xu and Kai Hwang, "Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2", IEEE Parallel & Distributed Technology, Spring 1996, pp 9-23

[18] 윤일홍, 김동승, "MPI환경에서의 브로드캐스트의 모델링 및 성능향상 기법", 정보과학회 논문지, 1999