원형 쉬프트 통신의 중첩 효과 분석

김 정 환† · 노 정 규†† · 송 하 윤†††

요 약

통신과 계산 작업을 중첩 수행함으로써 통신 시간의 감춤 효과를 얻는 것은 일반적인 병렬 프로그램 최적화 방법 중의 하나이다. 본 논문에서는 데이타 병렬 프로그램에서 자주 사용되는 군집 통신(collective communication)의 하나인 원형 쉬프트(circular shift) 통신에 대해 중첩 효과를 실험하고 고찰하였다. 이더넷 스위치로 연결된 클러스터 시스템에서 원형 쉬프트 통신을 수행할 때, 중첩으로 얻을 수 있는 최대 이득과 중첩할 수 없는 시간을 측정하였다. 각 플랫폼 별로 이러한 측정값들을 얻어 최적화 컴파일러의 입력으로 활용할 수 있을 것이다. 한편 기존의 성능 모델을 통해 최적화하는 것은 크게 두가지 문제를 갖고 있다. 하나는 기본적인 점대점 통신에 입각한 모델을 제공하기 때문에 통신 라이브러리의 함수를 사용할 때의 종합적인 효과, 특히, 군집 통신과 같은 경우에는 적용하기 어렵다는 것이다. 다른 하나는 군집 통신의 성능은 분석은 가능하지만, 중첩 효과는 분석할 수 없다는 것이다. 본 논문에서는 이러한 기존 모델의 단점을 보완하여 확장하였다. 또한, 원형 쉬프트 통신에 대한 실험 결과를 토대로 확장된 모델의 매개 변수 값들을 추출하여 예제 프로그램을 통해 분석하였다.

Overlapping Effects of Circular Shift Communication and Computation

Junghwan Kim† · Jungkyu Rho†† · Hayoon Song†††

ABSTRACT

Many researchers have been interested in the optimization of parallel programs through the latency hiding by overlapping the communication with the computation. We analyzed overlapping effects in the circular shift communication which is one of the collective communications being frequently used in many data parallel programs. We measured the time which can be possibly overlapped and the time which cannot be overlapped in over all circular shift communication period on an Ethernet switch-based clustered system. The result from each platform may be used for the input of optimizing compilers. The previous performance models usually have two kinds of drawbacks : one is only based on point-to-point communication, so it is not appropriate for analyzing the overall effects of collective communications. The other provides the performance of collective communication, but no overlapping effect. In this paper we extended the previous models and analyzed the experimental results of the extended model.

키워드 : 군집 통신(collective communication), 성능 모델(performance model), 원형 쉬프트(circular shift), 중첩(overlapping), 데이타 병렬(data parallel), 클러스터(cluster)

1. 서 론

지연(latency)과 동기화(synchronization) 문제는 병렬 컴퓨팅에서 매우 중요한 이슈이다. 근래에 NOW[1]나 Beowoulf [2] 등과 같이 다수의 워크스테이션이나 PC 등을 연결한 클러스터 시스템이 주목받고 있는데, 이러한 클러스터 시스템에서도 지연과 동기화는 중요한 이슈이다. 특히, 노드 간 통신 지연 시간은 지역적인 계산 시간에 비해 상대적으로 커서 통신의 최적화는 클러스터 시스템의 효용성을 결정짓는 중요한 요소이다.

통신 지연시간을 프로세서의 계산 작업과 중첩함으로써 통신 시간 감춤 효과를 얻는 것은 대표적인 최적화 방법의 하나이다. 이 방법에는 메시지 파이프라이닝[3]과 반복 재순서화[4] 등이 있다. 통신 작업과 계산 작업의 중첩, 또는 통신 작업 간의 중첩을 효과적으로 하기 위해서는 몇가지 시스템의 변수들을 이용할 필요가 있다. LogP 모델[5]은 병렬 시스템의 변수를 물리적 지연시간, 프로세서 오버헤드, 통신 대역폭, 프로세서 수 등으로 구분함으로써 단순하면서도 현실적인 병렬 모델을 나타내고 있다. 통신 시간은 프로세서 오버헤드와 같이 다른 계산 작업과 중첩될 수 없는 부분과 물리적 지연시간처럼 중첩될 수 있는 부분으로 구

† 정 회 원 : 건국대학교 컴퓨터·응용과학부 교수
†† 정 회 원 : 서경대학교 컴퓨터과학과 교수
††† 정 회 원 : 홍익대학교 정보·컴퓨터공학부 교수
논문접수 : 2002년 3월 7일, 심사완료 : 2002년 6월 3일

분할 수 있는데, 이 모델은 통신 작업간의 중첩 또는 통신과 계산의 중첩을 효과적으로 하는 알고리즘의 개발 및 성능 예측에 유용하다.

한편, 사용자 수준에서 노드 간의 메시지 전달은 여러 소프트웨어 계층을 거치는 MPI[6]나 PVM[7]과 같은 메시지 라이브러리를 사용하는 것이 일반적이다. 이러한 수준의 메시지 전달 함수를 하드웨어 매개변수들을 사용하여 분석하기는 어렵다. 특정한 통신 패턴의 메시지 전달 함수를 사용할 때의 통신 대역폭과 물리적인 통신 대역폭은 차이가 있다. 군집 통신(collective communication)은 하나의 그룹 내에 속한 프로세스들 사이의 집단적인 통신 형태로, 데이타 병렬 프로그램 등에서 자주 사용하는 형태이다. 이들의 종류로는 Broadcast, Scatter, Gather, Total Exchange, Circular Shift, Reduction, Scan 등이 있다. 이러한 군집 통신을 여러 개의 점대점(point-to-point) 통신으로 분해한 후 중첩 효과를 분석하는 것은 효과적인 방법이 될 수 없다. 왜냐하면, 여러 개의 점대점 통신과 계산 작업의 중첩은 매우 유동적인 상황을 가져오기 때문에 의미있는 분석이 어렵다. 분석할 수 있다고 해도 사용자 코드 수준에서 라이브러리 함수 내부까지 고려한 최적화는 불가능하며 라이브러리 함수의 변동에 영향을 받게 된다.

Zhiwei Xu와 Kai Hwang은 낮은 수준에서의 분석 대신에 MPI 함수(또는 MPL[8] 함수) 수준에서의 군집 통신에 대해 분석하였다[9]. 이들은 IBM SP2[10]에서 각 병렬 라이브러리 별로 통신 소요시간을 분석하여 모델을 제시하였다. 이 모델은 개발하고자 하는 병렬 프로그램의 특성에 적합한 병렬 라이브러리를 선택하거나 또는 병렬 프로그램의 성능 예측에 필요한 척도를 제공한다. 그러나, 이 연구는 LogP 모델과는 달리 하나의 통신 진행 과정을 단계별로 구분한 매개변수를 제공하지 않기 때문에 중첩 효과 분석에 활용할 수 없다.

본 논문에서는 사용자 코드 수준의 실험을 통해 군집 통신 중의 하나인 원형 쉬프트(circular shift) 통신에 대해 분석하였으며 이를 설명할 수 있는 모델을 제안하였다. 모델과 실험 데이타는 원형 쉬프트 통신과 계산의 중첩 시에 얻을 수 있는 성능을 예측하거나 프로그램을 최적화하는데 사용될 수 있다.

본 논문의 2절에서 관련 연구를 소개하고 3절에서는 사용자 코드 수준에서의 실험과 그 결과를 분석하였다. 4절에서는 이를 토대로 확장된 모델을 제안하고 Jacobi 예제를 통해 비교 분석하였다. 마지막으로 5절에서는 결론 및 향후 연구에 대해 기술하였다.

2. 관련 연구

이 절에서는 병렬 컴퓨터 시스템의 통신과 관련된 기존 모델들을 기술한다.

2.1 Hockney의 모델

Roger Hockney는 점대점 통신 시간에 대해 유용한 모델을 제안하였다[11].

$$t = t_0 + \frac{m}{r_\infty} \tag{1}$$

여기서 t는 통신에 소요된 시간, t_0는 지연시간(latency) 혹은 시작 시간(startup time), m은 메시지의 길이(바이트 수), 그리고 r_∞는 점근 대역폭(asymptotic bandwidth)을 각각 나타낸다. t_0는 0-바이트 메시지를 보내는데 소요되는 시간이며 r_∞는 메시지 길이가 무한대일 때 얻어지는 최대 대역폭을 의미한다. m-바이트 메시지를 보낼 때의 순수 전송 지연(transmission delay)은 시작 시간 t_0를 제외한 m/r_∞로 계산된다.

여기에 두 개의 매개변수가 부가되는데, 하나는 $m_{1/2}$이고 다른 하나는 π_0이다. $m_{1/2}$은 점근 대역폭의 절반에 도달하기 위한 메시지 길이를 나타내고 π_0는 작은 길이의 메시지에 대한 대역폭을 나타낸다. 이들 매개변수 사이에는 또한 다음과 같은 관계가 성립한다.

$$t_0 = \frac{m_{1/2}}{r_\infty} = \frac{1}{\pi_0} \tag{2}$$

Hockney 모델은 단순하면서도 측정치와 근접하는 우수한 특성을 갖는다. 특히, 이 모델은 매개변수 값은 달라지지만 통신 하드웨어 구조에 독립적이며, 메시지 길이 m에 비례하는 선형 함수를 제공하고 몇 개의 기본적인 값(t_0와 r_∞)으로 표현된다는 장점이 있다. 그러나, 적용 범위가 점대점 통신에 국한되고 단순히 총 소요시간(elapsed time)만을 모델링하였으므로 통신 - 계산 중첩에 대한 시간은 예측할 수 없다는 한계를 갖고 있다.

2.2 Xu와 Hwang의 모델

Zhiwei Xu와 Kai Hwang은 IBM SP2 상에서 통신 오버헤드를 모델링하였다[9]. 통신 오버헤드는 소프트웨어(프로세서) 오버헤드, 하드웨어 지연, 메시지 지연을 모두 합한 시간으로 이는 Hockney 모델의 통신 소요시간의 의미와 동일하다. Hockney의 모델이 점대점 통신에 국한되었던 것과 달리 이 모델은 군집 통신까지 설명할 수 있도록 일반화하였다.

$$t = t_0(n) + \frac{m}{r_\infty(n)} \tag{3}$$

Hockney 모델의 경우처럼 통신 오버헤드 t는 메시지 길이 m에 대한 선형 함수로 표현된다. 다만, 지연시간 t_0와

점근 대역폭 r_∞가 Hockney 모델에서는 상수로 주어지는데 비해 여기서는 통신에 참여하는 노드의 개수 n에 대한 함수로 나타낸다. 단, 반드시 선형을 의미하는 것은 아니다. 이 값은 노드의 전송 대역폭으로 해석하는 것이 아니라, 특정 통신 프리미티브에 대한 값으로 해석해야 한다. 가령, 브로드캐스트에 대해 $r_\infty(10) = 30$ MB/s이라면, 이는 10개의 노드가 참여할 경우 초당 30 MB를 브로드캐스트 할 수 있음을 의미하는 것이다. 이 경우 특별한 하드웨어 지원이 없다면 루트 노드는 초당 300 MB를 전송해야 한다. Hockney 모델의 경우에서처럼 최대 점근 대역폭의 절반에 이르기 위한 메시지 길이는 $m_{½}(n)$으로 나타내며, 역시 n에 대한 함수이다.

Xu와 Hwang의 모델은 Hockney의 모델을 군집 통신까지 확장했다는데 의미를 갖는다. 이 모델을 통해서 군집 통신을 사용하는 병렬 프로그램들의 성능을 예측할 수 있다. 그러나, 계산과 통신을 중첩한 최적화는 다룰 수 없고 계산 단계와 통신 단계가 구분된 병렬 프로그램에만 적용할 수 있다는 단점이 있다.

2.3 LogP 모델

David E. Culler 등은 구조 독립적이면서 현실적인 특성을 반영하여 병렬 계산 모델을 제안하였다[5].

- L : *1*-워드 메시지(작은 메시지)를 전송하는데 소요되는 지연시간의 상한(latency).
- o : 메시지를 전송할 때 프로세서가 개입하는 시간(overhead).
- g : 연속적인 메시지 송신 또는 수신 시에 요구되는 최소 간격(gap).
- P : 프로세서-메모리 모듈의 개수. 즉, 노드의 개수.

Hockney 및 Xu의 모델에서 지연시간 t_0는 *0*-바이트 (또는 최소 메시지)를 전송할 때 관측되는 총 소요시간을 의미하지만 LogP 모델의 L은 물리적인 전송매체를 통과할 때의 지연시간만을 의미한다. 즉, 이들 모델에서는 프로세서의 개입이 필요한 시간을 별도로 산정하지 않지만 LogP 모델에서는 프로세서 사이클이 소비되는 시간은 o, 그리고 프로세서 사이클이 소비되지 않는 별도의 하드웨어 또는 물리적 전송매체의 지연시간은 L로 구분하여 나타낸다.

g는 전송매체 또는 네트워크 하드웨어 특성상 요구되는 메시지간의 최소 간격으로 네트워크의 대역폭을 반영한다. 즉, $1/g$는 노드 당 통신 대역폭을 의미하며 $\lceil L/g \rceil$는 어느 순간에 네트워크가 수용할 수 있는 메시지의 최대 개수를 의미한다.

PRAM[12]과 같이 이상화된 모델이 실제 병렬 컴퓨팅 환경을 반영할 수 없는데 반해, LogP 모델은 네트워크 구조에 독립적이면서도 병렬 프로그램의 최적화에 유용한 현실적인 성능 매개변수를 제시하였다는데 의의를 갖는다. 그러나 LogP 모델은 작은 길이의 메시지와 점대점 통신만을 고려 대상으로 삼았다는 한계가 있다. 군집 통신의 경우 점대점 통신의 집합으로 수행될 수 있으나 이때의 대역폭 $1/g$와, 지연시간 L은 고정되어 있기 때문에 네트워크 스위치 상의 충돌 또는 용량 한계 등으로 인한 변수를 효과적으로 반영하지 못한다.

2.4 LogGP 모델

Albert Alexandrov 등은 LogP 모델로부터 긴 메시지를 고려하여 확장한 LogGP 모델을 제안하였다[13].

- G : 긴 메시지를 전송할 때 각 바이트의 최소 시간 간격 (Gap).

$1/g$가 짧은 메시지를 전송할 때의 통신 대역폭이라면 $1/G$는 긴 메시지를 전송할 때의 통신 대역폭이라고 할 수 있다. IBM SP-2, Paragon, Meiko CS-2, Ncube/2와 같은 많은 병렬 컴퓨터들은 긴 메시지를 보다 높은 대역폭으로 전송할 수 있는 특별한 하드웨어 또는 프로토콜을 사용하고 있다. g 값은 짧은 메시지를 연속적으로 전송함으로써, 그리고 G 값은 긴 메시지 (이론상 무한대의 길이)를 전송함으로써 측정할 수 있다.

LogGP 모델은 짧은 메시지와 긴 메시지의 전송 대역폭이 일반적으로 다르다는 것을 반영하였다는 점에서 의미가 있다. LogP 모델 하에서 긴 메시지의 전송은 몇 개의 짧은 메시지 전송의 집합으로 설명할 수 있지만 실제 성능을 묘사하기에는 충분하지 않다. 한편, LogGP 모델 역시 LogP 모델과 마찬가지로 점대점 통신을 고려 대상으로 삼았기 때문에 군집 통신에 참여하는 노드의 수가 늘어남에 따라 변화될 지연시간이나 대역폭을 고려하지 못하고 있다.

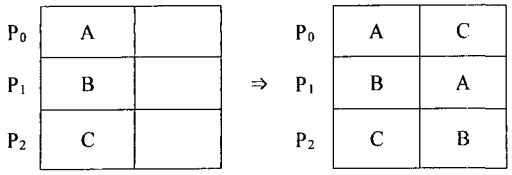
3. 실험 및 고찰

이 절에서는 이더넷 스위치를 통해 연결된 리눅스 클러스터 시스템에서 수행한 원형 쉬프트 통신에 대한 실험 방법을 설명하고 그 결과를 분석한다.

3.1 실험 방법 및 환경

3.1.1 원형 쉬프트 통신

원형 쉬프트 통신에서 각 노드는 다음 노드로의 데이타 송신과 동시에 이전 노드로부터 데이타를 수신한다. 원형 쉬프트 통신은 데이타 병렬 프로그램에서 일반적으로 사용할 수 있는 통신 방법으로 다음 그림과 같이 각 노드가 모두 송수신을 통해 데이타를 교환해야하는 전형적인 형태를 갖는다.



실험은 통신과 계산 작업의 중첩 측면에서 분석하는 것을 목표로 하였다. (그림 1)은 원형 쉬프트 통신과 프로세서의 계산 작업이 단계별로 구분되어 수행되는 경우와 중첩 수행되는 경우를 나타낸다. (a)의 경우 *send-recv*를 통해 원형 쉬프트가 완료된 후 다른 작업에 진입하지만 (b)의 경우 *send-recv* 사이에 다른 독립적인 작업을 수행함으로써 전송 지연시간을 감추고 있다.

```
send(right, msg) ;
recv(left, msg) ;
use a received msg ;
do some work ;
```

(a) 통신 - 계산 작업의 구분 수행

```
send(right, msg) ;
do some work ;
recv(left, msg) ;
use a received msg ;
```

(b) 통신 - 계산 작업의 중첩 수행

(그림 1) 원형 쉬프트 통신과 계산작업의 중첩

메시지 길이가 작다고 가정할 때 단순히 LogP 모델에 따라 분석을 시도해보면 다음과 같다. (a)의 경우 총 소요시간은 $o+L+o+C_1+C_2$가 소요되는데 여기서 C_1은 수신된 메시지를 사용하여 계산하는 작업 시간 그리고 C_2는 메시지와 무관한 독립적인 작업 시간을 나타낸다. (b)의 경우 $C_2 >> L$이라고 할 때 총 소요시간은 $o+C_2+o-C_1$이 소요된다. 즉, L과 g는 프로세서의 개입이 필요하지 않는 시간이므로 다른 작업과 중첩 가능하다. 한편, 이 분석은 메시지의 길이가 긴 경우는 설명할 수 없고, 각 노드가 동시에 통신을 함에 따른 대역폭 제한은 고려하지 않았다.

3.1.2 실험 방법

실험은 (그림 1)의 (b)에서 중첩 수행이 가능한 시간을 추출하는데 초점을 두었다. 전혀 중첩을 하지 않았을 때, 즉 (a) 상태에서 출발하여 점진적으로 중첩 시간 (C_2)을 증가시키면서 실험을 반복하였다. 원형 쉬프트 통신을 제공하는 프리미티브로는 IBM의 MPL (Message Passing Library)의 경우 *mpc_shift*()가 있고, MPI의 경우 *MPI_Sendrecv*()가 있다. 본 실험은 리눅스 클러스터에서 시행하였으므로 MPI 라이브러리를 사용하였다. 다만, *MPI_Sendrecv*()와 같이 단일 프리미티브를 사용할 경우 중첩 수행의 효과를 측정하기 어렵기 때문에 (non-blocking을 사용하면 가능하긴 하지만), *MPI_Bsend*()와 *MPI_Recv*()의 두 프리미티브를 사용하였다.

실험에 사용된 코드의 개략적인 알고리즘은 (그림 2)와 같다. 이 코드는 원형 쉬프트 통신과 일정량의 프로세서 작업을 수행하고 그 경과 시간을 측정한다. 프로세서 작업은 *do_work*()를 통해 이루어지는데 반복 회수는 루프에 의해 조절된다. 수행되는 루프의 총 반복 횟수는 *upper_bd*로써 이 중 원형 쉬프트 통신과 중첩 수행될 부분의 반복 회수는 *overlapped_comp*이다. 통신과 중첩 수행될 이 루프는 *MPI_Bsend*()와 *MPI_Recv*() 사이에 위치하고 나머지 작업량(*upper_bd* - *overlapped_comp*)은 *MPI_Recv*() 이후에 수행된다. 최외곽 루프에서는 *overlapped_comp*를 점진적으로 늘이면서 실험을 반복한다. 전체 프로세서 작업량 중 중첩 수행되는 프로세서 작업이 늘어남에 따라 총 경과 시간은 줄어들 것으로 예측되며 일정 시점이 되면 중첩 작업량을 늘여도 더 이상 시간 단축은 일어나지 않게 된다.

```
for (overlapped_comp = 0 ; overlapped_comp<upper_bd ;
        overlapped_comp += step) {
    min_t = MAX_DOUBLE ;
    for (i = 0 ; i < NTRIALS ; i++) {
        s_time = wtime() ;
        MPI_Bsend(send_msg, msg_size, MPI_BYTE, dest,
                tag, MPI_COMM_WORLD) ;
        for (work = 0 ; work < overlapped_comp ; work++)
            do_work(work) ;    /* do some overlapped work */
        MPI_Recv(recv_msg, msg_size, MPI_BYTE, src,
                tag, MPI_COMM_WORLD) ;
        for (work = overlapped_comp ; work < upper_bd ; work++)
            do_work(work) ;      /* do the rest of the work */
        e_time = wtime() ;
        local_t = e_time - s_time ;
        MPI_Reduce(&local_t, &global_t, 1, MPI_DOUBLE,
            MPI_MAX, root, MPI_COMM_WORLD) ;
        min_t = (min_t > global_t)?global_t:min_t ;
    }
    print_result(msg_size,overlapped_comp, min_t) ;
}
```

(그림 2) 실험에 사용된 코드의 알고리즘

do_work()은 단순한 반복문으로써, 캐쉬 영향을 최소화하고 루프 불변 코드에 따른 컴파일러 최적화를 배제하였다. 프로세서들은 록스텝 (lock-step)으로 동기화되어 있는 것이 아니므로 경과 시간에 차이가 있다. 배분된 작업이 모두 끝난다는 의미에서 가장 긴 경과 시간을 실험 결과로 채택하였다. *MPI_Reduce*()는 각 프로세서로부터 경과 시간을 모아 그 중 가장 큰 값을 얻어내기 위한 것이며 실험 오차를 줄이기 위해 실험 매개변수별로 NTRIALS 회수만큼 반복하고 이 중 최소값을 택하였다.

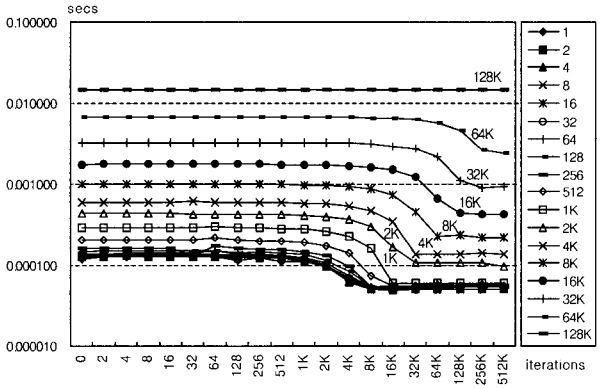
MPI의 표준 송신인 *MPI_Send*()는 대응되는 *MPI_Recv*()가 호출되기 전에 제어가 넘어온다는 어떠한 보장도 하지 않는다. 따라서 원형 쉬프트에서 *MPI_Send*()와 *MPI_Recv*()를 사용한다면 교착 상태가 발생하지 않는다는 것을 보장할 수 없다.[1] 따라서 본 실험에서는 표준 송신 대신 버퍼형 송신(buffered send)인 *MPI_Bsend*()를 사용하였다.

3.1.3 실험 환경

실험에 사용된 시스템의 각 노드는 인텔 Pentium III 800 MHz 프로세서를 탑재하고 있으며 512MB의 메인 메모리를 장착하고 있다. L1 캐쉬는 명령어와 데이타 각 16 KB이고, L2 캐쉬는 통합 256 KB으로 되어있다. 노드들은 고속 이더넷 스위치를 통해 연결되어 있다.[2] 시스템에 탑재된 운영체제는 Red Hat Linux release 6.2이고 실험에 사용된 메시지 전송 라이브러리는 LAM 6.5.1 / MPI이다.

3.2 실험 결과

앞서 (그림 2)에서 제시한 실험 코드를 통해 원형 쉬프트 통신과 계산 작업의 중첩 효과를 관찰해보았다. (그림 3)은 2개 노드 상에서 중첩 효과를 보여주는 실험 결과이다. 그림에서 *x*축은 원형 쉬프트 통신과 중첩되어 수행되는 루프 반복의 수를 나타내고 *y*축은 로그 스케일의 소요시간(초)을 나타낸다. 중첩되는 루프 반복의 수를 늘임에 따라 소요 시간은 점차적으로 감소하지만 어느 지점 이후부터는 더 이상 감소하지 않는다. 이는 원형 쉬프트 통신에서 계산 작업과의 중첩 효과를 기대할 수 있는 시간의 한계를 보여준다. 메시지의 크기가 증가하면 중첩할 수 있는 시간도 늘어난다. 그림에서 메시지 크기가 클수록 한계점에 도달하기까지 더 많은 루프 반복이 중첩됨을 알 수 있다.



(그림 3) 원형 쉬프트 통신과 계산 작업의 중첩에 따른 소요시간 변화

일반적으로 메시지 크기가 크면 원형 쉬프트 통신 시간이 증가하지만 중첩할 수 있는 시간 또한 증가하여 시간 단축 효과도 커진다. 그런데 (그림 3)에서 메시지 크기가 64 KB까지는 중첩에 의해 시간 단축이 선형적으로 이루어진데 비해 128 KB에 이르면 전혀 시간이 단축되지 않는다. 128 KB에서 중첩 효과가 없는 이유는 LAM/MPI의 short 프로토콜과 long 프로토콜의 차이에 기인한 것으로 분석된다. LAM/MPI에서는 short 메시지를 헤더와 함께 바로 목적지에 전송하지만 long 메시지의 경우 먼저 헤더(일부 데이타 포함)를 전송한 후 수신측으로부터 ack를 기다리고 ack가 오면 메시지 데이타의 나머지 부분을 보낸다[14]. 이때 ack는 수신측에서 대응되는 *MPI_Recv*() 호출되었을 때 전송된다. short와 long의 경계는 LAM/MPI의 설치 단계에서 임의로 설정할 수 있지만 디폴트는 64 KB이다. 따라서 64 KB를 초과하는 메시지는 데이타의 대부분이 *MPI_Recv*()가 호출된 후 전송되기 때문에 *MPI_Bsend*()와 *MPI_Recv*() 사이에 아무리 많은 프로세서 작업을 배치하더라도 지연시간 감춤 효과가 나타나지 않는 다. 따라서 중첩 효과라는 측면에서는 64 KB 이하의 메시지만 관심의 대상이다. 만일 64 KB를 초과하는 메시지를 사용하는 문제를 다루고 있다면 루프 타일링 (loop tiling)[15] 등을 이용하는 것이 바람직할 것이다.

<표 1>은 노드 개수 및 메시지 크기 별로 중첩을 통해 얻을 수 있는 최대 단축 시간을 나타낸 것이다. 이것은 전혀 중첩을 하지 않았을 때의 소요시간에서 중첩을 하였을 때 얻을 수 있는 가장 적은 소요시간을 뺀 것이다. 이 중첩 가능 시간은 물리적 지연 시간(L)과 메시지 패킷 간의 최소 간격(g)의 합으로써 프로세서의 개입이 필요 없는 시간을 의미한다. 한편, <표 2>는 프로세서의 개입이 필요하여 중첩할 수 없는 시간(o)을 나타낸다. 가령, 프로세서가 사용자 공간에 있는 메시지를 시스템 버퍼 또는 네트워크 인터페이스로 복사하는데 소요되는 시간이다.

<표 1>은 메시지 크기가 증가함에 중첩 가능한 시간은 일반적으로 증가함을 보여준다. 메시지 크기의 증가는 이더넷 상에서 전송되는 프레임 크기 또는 개수의 증가를 가져오고 따라서 중첩 가능한 시간은 증가하게 된다. 한편, 같은 메시지 크기에 대해서도 노드 개수가 많을수록 중첩 가능 시간은 증가함을 보여주는데, 이는 통신에 참여하는 노드 개수가 많아짐에 따라 지연시간이 증가하기 때문인 것으로 추측된다.

<표 2>는 원형 쉬프트 통신 소요시간에서 <표 1>의 중첩 시간을 뺀 값이다. 프로세서 오버헤드도 메시지 크기가 증가함에 따라 증가하는 추세를 보이는데 이는 버퍼 복사를 해야 할 데이타의 양이 많아지기 때문인 것으로 해석된다. 한편, 근소한 값이지만 노드 개수의 증가에 따라서도 증가하는 추세를 보이고 있다. 이는 재전송 등의 이유로 프로토콜 계층의 오버헤드가 증가하기 때문인 것으로 추측된다.

1) 모든 프로세서가 *MPI_Send*()를 호출한 후에 *MPI_Recv*()를 호출하는 경우에 교착 상태가 발생할 수 있다.

2) Nortel Baystack 24-T

〈표 1〉 원형 쉬프트 통신에서 중첩 가능 시간(단위 : 초)

msg size (bytes)	n = 2	n = 3	n = 4	n = 5	n = 6	n = 7	n = 8
1	0.000069	0.000068	0.000065	0.000083	0.000167	0.000135	0.000216
2	0.000076	0.000068	0.000056	0.000074	0.000137	0.000118	0.000204
4	0.000080	0.000067	0.000067	0.000073	0.000117	0.000129	0.000217
8	0.000080	0.000060	0.000067	0.000060	0.000154	0.000119	0.000208
16	0.000083	0.000061	0.000067	0.000072	0.000159	0.000121	0.000192
32	0.000084	0.000063	0.000071	0.000076	0.000070	0.000134	0.000181
64	0.000068	0.000066	0.000077	0.000075	0.000135	0.000141	0.000202
128	0.000092	0.000088	0.000078	0.000082	0.000134	0.000152	0.000212
256	0.000110	0.000103	0.000101	0.000101	0.000174	0.000171	0.000224
512	0.000149	0.000151	0.000151	0.000162	0.000222	0.000213	0.000245
1K	0.000233	0.000231	0.000234	0.000242	0.000313	0.000294	0.000348
2K	0.000324	0.000330	0.000336	0.000342	0.000396	0.000388	0.000443
4K	0.000465	0.000459	0.000467	0.000472	0.000524	0.000537	0.000580
8K	0.000774	0.000751	0.000745	0.000747	0.000812	0.000812	0.000853
16K	0.001328	0.001328	0.001317	0.001328	0.001350	0.001370	0.001415
32K	0.002432	0.002420	0.002377	0.002426	0.002423	0.002408	0.002512
64K	0.004422	0.004325	0.004279	0.004161	0.004329	0.004290	0.003555
128K	0.000052	0.000071	-0.000007	0.000014	-0.000076	-0.000104	0.000058

〈표 2〉 원형 쉬프트 통신에서 중첩할 수 없는 시간(단위 : 초)

msg size (bytes)	n = 2	n = 3	n = 4	n = 5	n = 6	n = 7	n = 8
1	0.000050	0.000053	0.000054	0.000052	0.000053	0.000060	0.000061
2	0.000050	0.000054	0.000064	0.000058	0.000064	0.000071	0.000071
4	0.000050	0.000056	0.000062	0.000062	0.000062	0.000072	0.000065
8	0.000050	0.000064	0.000063	0.000063	0.000062	0.000071	0.000065
16	0.000050	0.000063	0.000063	0.000063	0.000065	0.000073	0.000065
32	0.000052	0.000064	0.000063	0.000064	0.000063	0.000071	0.000066
64	0.000053	0.000066	0.000062	0.000065	0.000065	0.000071	0.000068
128	0.000054	0.000058	0.000064	0.000064	0.000066	0.000072	0.000072
256	0.000054	0.000064	0.000065	0.000065	0.000067	0.000075	0.000074
512	0.000056	0.000066	0.000062	0.000064	0.000070	0.000072	0.000075
1K	0.000059	0.000070	0.000069	0.000071	0.000072	0.000080	0.000076
2K	0.000107	0.000108	0.000104	0.000108	0.000108	0.000113	0.000114
4K	0.000138	0.000154	0.000149	0.000150	0.000159	0.000155	0.000160
8K	0.000226	0.000249	0.000251	0.000251	0.000250	0.000254	0.000264
16K	0.000433	0.000437	0.000450	0.000462	0.000477	0.000457	0.000460
32K	0.000912	0.000964	0.000963	0.000951	0.000989	0.001005	0.000953
64K	0.002423	0.002592	0.002577	0.002648	0.002596	0.002640	0.003347
128K	0.014631	0.014758	0.014681	0.014785	0.014866	0.014897	0.014906

4. 군집 통신 모델로의 확장

LogP 또는 Xu와 Hwang의 모델은 앞서의 실험 결과를 설명하기에 충분하지 않다. 이 절에서는 원형 쉬프트 통신과 같은 군집 통신에 적용할 수 있으면서 계산 - 통신 중첩을 설명할 수 있는 모델로 확장하는 방안에 대해 논의한다.

4.1 기존 모델의 문제점

기존의 모델들은 점대점 통신을 가정하거나 (LogP) 또는 전체 경과 시간만을 관심 대상으로 하고 있다(Xu와 Hwang의 모델). 군집 통신에 점대점 통신 모델을 적용하는 것은 사용자 계층의 종합적인 상황을 반영하지 않는 문제점을 갖고 있다. 또한 통신에 소요된 시간만을 설명하는 Xu와 Hwang의 모델은 계산 - 통신 중첩에 사용될 수 없는 한계를 갖는다.

원형 쉬프트 통신에 대한 앞 절의 실험 결과는 중첩 가능한 시간과 중첩 불가능한 시간 모두가 노드 개수 n에 종속적임을 보여준다. 이는 LogP 모델을 적용할 경우 L, o, g 값이 상수가 아니며 노드 개수에 따라 달라질 것이라는 예상을 가능하게 한다. LogP 모델에서 L값은 하드웨어적인 특성을 나타내며 물리적 지연시간의 상한을 의미한다. 가령, 다단계 연결망(multistage interconnection network)을 사용하고 있다면 단계 수에 비례하는 값을 갖게 될 것이다. 그러나, 이는 부하가 거의 없는 경우, 즉 충돌 가능성을 배제한 상태에서 의미를 갖는다. 예를 들면 2개 노드만 참여하는 점대점 통신과 같은 경우이다. 이 모델은 네트워크가 포화 상태에 이르기 전까지는 지연시간이 비교적 민감하게 반응하지 않는다는 전제를 하고 있다. 그러나 앞 절의 실험 결과가 보여주듯 통신에 참여하는 노드 개수 n에 따라 지연시간은 달라질 것으로 예상된다. 한편 $1/g$값은 노드의 통신 대역폭을 나타낸다. 낮은 수준의 메시지 전송을 하는 경우 이 값은 네트워크의 물리적 대역폭에 접근하겠지만, MPI와 같은 메시지 함수를 사용하는 경우 개별적으로 다른 값이 될 것이다. 또한 군집 통신을 하는 경우 역시 통신에 참여하는 노드 개수 n에 의해 다른 값을 갖게 될 것이다.

정리해보면 LogP 모델의 각 매개변수는 하드웨어 특성을 반영하지만, 알고리즘 특성 또는 사용자 계층의 통신 패턴을 반영하지는 않는다. 따라서, 매개변수(L과 g)를 하드웨어 상수 값으로 고정시킨 LogP 모델에서는 원형 쉬프트 통신 하에서의 성능을 정확하게 예측하기 어렵다. 물론 원형 쉬프트 통신을 보다 낮은 수준의 프리미티브들로 분해해 분석을 시도할 수도 있다. 그러나, 이러한 하드웨어 수준의 값들보다는 응용 프로그램 수준에서의 값들을 얻는 것이 유용한 방법이 될 것이다.

Xu와 Hwang의 모델에서는 점근 대역폭이 $r_\infty(n)$으로 나타내진다. 즉, 점근 대역폭은 통신에 참여하는 노드 개수 n의 함수이며, 이는 통신 패턴에 따라 달라진다. 이것은 LogP 모델과 달리 원형 쉬프트 통신에 대한 대역폭을 활용할 수 있음을 의미한다. 그러나, 이 모델에서는 t_0가 소프트웨어 오버헤드와 하드웨어 지연을 합한 값(LogP 모델에서

$2o+L$)을 의미하기 때문에, 계산-통신 중첩에 따른 분석이 불가능하다.

4.2 원형 쉬프트 통신 모델

이 절에서는 LogP 모델에서 L과 g값을 하드웨어 특성에 의해 고정하였던 것을 원형 쉬프트 통신에 적용하기 위해 각각을 n에 대한 함수로 확장하고, 또한 사용자 계층에서의 관측값을 사용하는 것으로 제안한다. nLogP로 명명된 이 모델의 각 매개변수는 다음과 같다.

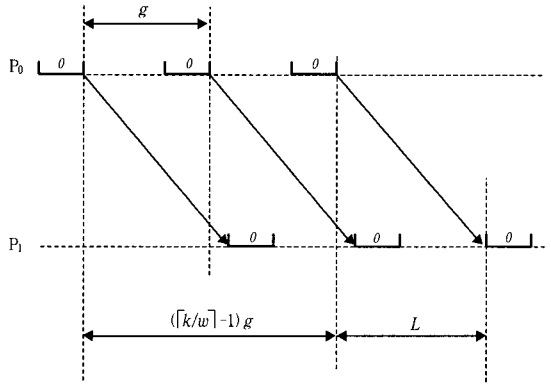
- n : 통신에 참여하는 프로세서(노드)의 개수
- $L(n)$: n개의 프로세서가 통신에 참여할 때 최대 지연 시간
- $o(n)$: 프로세서 오버헤드
- $g(n)$: n개의 프로세서가 통신에 참여할 때 최대 대역폭의 역수
- P : 시스템을 구성하는 프로세서의 개수

P는 현재 작업과 무관하게 시스템에 존재하는 프로세서(노드)의 개수이다. LogP 모델에서 네트워크 위상(topology)에 따라 g값을 얻는데 이 값이 이용된다. g값은 네트워크 위상에 따라 달라지는 평균 거리에 영향을 받기 때문이다. 가령, 하이퍼큐브(hypercube)에서 평균 거리는 $(\log P)/2$가 되며, g값은 여기에 비례한다. 한편, n은 실제로 통신에 참여하는 노드의 개수이다. 앞서 언급했듯이 이 값에 의해 L과 g가 영향을 받기 때문이다. 따라서 $L(n)$과 $g(n)$으로 표현되며, 각각은 통신 패턴에 따라 또는 사용하는 라이브러리에 따라 다른 함수가 될 것이다.

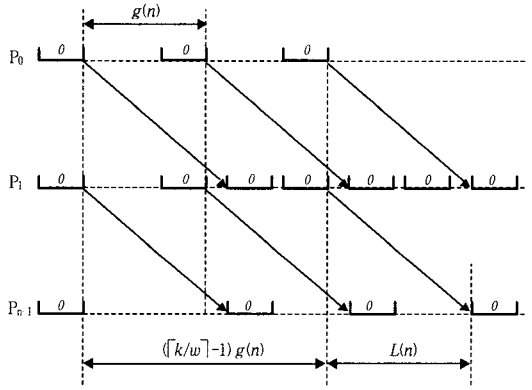
nLogP 모델은 LogP 모델과 마찬가지로 짧은 메시지에 적용된다. 만일 DMA과 같은 하드웨어 지원이 있다면 긴 메시지에 대한 모델인 LogGP를 확장한 nLogGP를 고려해 볼 수 있다. 그러나 본 논문의 실험 환경은 긴 메시지를 위한 특별한 하드웨어를 사용하고 있지 않으므로 nLogP를 한정하여 기술한다. 긴 메시지에 대한 하드웨어 지원 시의 모델에 대해서는 향후 연구 과제로 남긴다.

4.3 적용 예

원형 쉬프트 통신은 각 프로세서에서 송신과 수신이 모두 발생하므로 소요 시간은 $o(n)+L(n)+o(n)$이 된다. 그러나 이것은 짧은 메시지에 대한 것이고, 긴 메시지의 경우는 다르다. LogP 모델 하에서 긴 메시지는 여러 개의 짧은 메시지로 분할되어 전송된다. k 바이트의 메시지는 $\lceil k/w \rceil$ 개의 메시지로 분할되어 전송된다. 여기서 w는 하드웨어적으로 결정된 짧은 메시지 길이이다.[3] 이때의 소요 시간은 $o+(\lceil k/w \rceil - 1)\cdot\max\{g, o\}+L+o$가 된다[13].[4] $g>o$인 경우 이 값은 $o+(\lceil k/w \rceil - 1)\cdot g+L+o$가 된다. (그림 4)의 (a)는 긴 메시지의 전송 과정을, 그리고 (b)는 긴 메시지를 원형 쉬프트로 보낼 때의 과정을 나타낸 것이다.



(a) 점대점 전송(LogP)



(b) 원형 쉬프트(nLogP)

(그림 4) 긴 메시지 전송 과정

원형 쉬프트에서의 소요 시간은 $o(n)+(\lceil k/w \rceil-1)\cdot g(n)+L(n)+o(n)$가 된다. 단, $g(n)>2o(n)$인 경우에 대해서이다. 이 때 $g(n)$ 시간 동안 프로세서는 수신에 필요한 작업(o 시간)과 송신에 필요한 작업(o 시간)을 마칠 수 있어야 한다. 그렇지 않은 경우는 통신 대역폭보다 프로세서 대역폭이 작은 경우로 이 때의 소요 시간은 $o(n)+(\lceil k/w \rceil-1)\cdot 2o(n)+L(n)+o(n)$가 된다.

〈표 3〉 추정된 o, g 및 L 값

	2	3	4	5	6	7	8
o	0.000026	0.000028	0.000028	0.000029	0.000028	0.000029	0.000036
g	0.000146	0.000148	0.000146	0.000145	0.000146	0.000146	0.000145
L	0.000295	0.000303	0.000304	0.000313	0.000385	0.000367	0.000391

3) 실험 환경의 mtu는 1500 bytes. 그러나, 실험 환경에서 MPI 헤더와 프로토콜 계층의 헤더 등을 제외하고, 순수 데이타는 약 1448 bytes로 추정됨.

4) 원문에는 $o+\lceil(k-1)/w\rceil\cdot\max\{g, o\}+L+o$로 나와 있으나 이는 오기(誤記)로 생각됨.

$o(n) + (\lceil k/w \rceil - 1) \cdot g(n) + L(n) + o(n)$의 시간 중에서 프로세서가 개입하는 시간($T_p$)은 모두 $o(n) + (\lceil k/w \rceil - 1) \cdot 2o(n) + o(n)$가 된다. 나머지 $(\lceil k/w \rceil - 1) \cdot (g(n) - 2o(n)) + L(n)$ 시간은 프로세서 사이클을 소비하지 않는 시간(T_c)이다. 이 T_c 시간 동안 프로세서는 통신 작업과 독립적인 유용한 작업을 할 수 있다. 물론 문맥 교환(context switching)에 따른 오버헤드가 있지만, 여기에서는 무시하기로 한다. 3절의 실험한 결과로 얻어진 T_p 및 T_c를 토대로 추정한 L, o, g 값은 <표 3>과 같다.

<표 3>은 노드 개수 n이 증가함에 따라 L 값도 따라서 증가하는 경향을 보이고, o 값은 근소한 증가함을 보여준다. g 값은 예상과 달리 거의 변화가 없는데, 실험 노드 개수가 작기 때문인 것으로 추정된다.

```
int          i, j, t ;
double       A[128][128], B[128][128] ;
for (t = 1 ; t <= time ; t++) {
   for (i = 0 ; i < 128 ; i++)
      for (j = 0 ; j < 128 ; j++)
         A[i][j] = (B[i][j-1]+B[i][j+1]+B[i-1][j]+ B[i+1][j]) * 0.25 ;
   for (i = 0 ; i < 128 ; i++)
      for (j = 0 ; j < 128 ; j++)
        B[i][j] = A[i][j] ;
}
```

(a) 순차 버전

```
int      i, j, t ;
double A[34][128], B[34][128] ;
int      tag = 1, lproc, rproc, myp, nproc ;
MPI_Init(&argc, &argv) ;
MPI_Comm_rank(MPI_COMM_WORLD, &myp) ;
MPI_Comm_size(MPI_COMM_WORLD, &nproc) ;
lproc = (myp == 0)?nproc-1 : myp-1 ;
rproc = (myp+1) % nproc ;
for (t = 1 ; t <= time ; t++) {
A  MPI_Bsend(&B[1][0], 128, MPI_DOUBLE, lproc, tag,
     MPI_COMM_WORLD) ;
   MPI_Bsend(&B[32][0], 128, MPI_DOUBLE, rproc, tag,
     MPI_COMM_WORLD) ;
B  for (i = 2 ; i < 32 ; i++)
       for (j = 0 ; j < 128 ; j++)
         A[i][j] = (B[i][j-1]+B[i][j+1]  +B[i-1][j]+B[i+1][j]) * 0.25 ;
C  MPI_Recv(B[33][0], 128, MPI_DOUBLE, rproc, tag,
     MPI_COMM_WORLD) ;
   MPI_Recv(B[0][0], 128, MPI_DOUBLE, lproc, tag,
     MPI_COMM_WORLD) ;
D  for (i = 1 ; i < 33 ; i+=31)
       for (j = 0 ; j < 128 ; j++)
         A[i][j] = (B[i][j-1]+B[i][j+1]  +B[i-1][j]+B[i+1][j]) * 0.25 ;
   for (i = 1 ; i < 33 ; i++)
        for (j = 0 ; j < 128 ; j++)
           B[i][j] = A[i][j] ;
 }
```

(b) 병렬 버전 (통신-계산 중첩)

(그림 5) 변형 Jacobi 프로그램

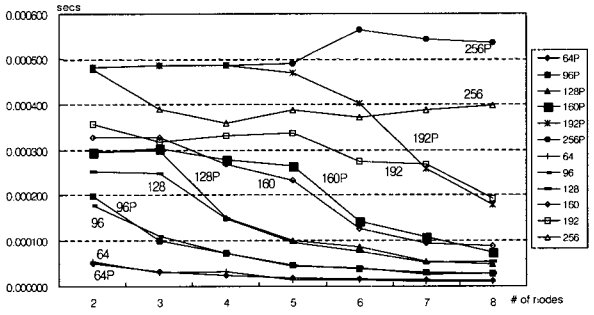
이제 변형된 Jacobi 프로그램을 분석해보기로 한다. 원래 Jacobi 프로그램은 쉬프트 통신을 하지만, 본 논문의 결과를 적용해보기 위해 원형 쉬프트 통신을 사용하도록 변형하였다. 변형된 Jacobi 프로그램은 (그림 5)와 같다. 데이터의 좌우 가장 바깥 부분은 반대편과 연속된다고 가정하여(wrap around), 원형 쉬프트 통신이 사용될 수 있도록 하였다.

(그림 5)의 (a)는 순차 버전이고, (b)는 4개의 프로세서를 사용하는 병렬 버전이다. 병렬 버전은 통신과 계산을 중첩 실행할 수 있도록 만들어졌다. 그림에서 A 부분은 두 개의 원형 쉬프트 송신(좌·우), B 부분은 원형 쉬프트 통신과 무관한 계산 작업, C 부분은 두 개의 원형 쉬프트 수신(좌·우), 그리고 D 부분은 수신 결과를 사용하는 작업이다. 원래 B와 D 부분은 하나의 루프였음에 주목할 필요가 있다. 원래 루프에서 가장 바깥쪽 부분을 제외한 내부는 메시지 수신 이전이라도 계산이 가능하다.

이제 n개의 프로세서를 갖고, 문제 크기가 64 × 64, 96 × 96, 128 × 128, 160 × 160, 192 × 192, 256 × 256인 경우에 대해 각각 분석해보기로 한다. B 부분 수행 시간을 T_B라고 할 때, 이 값이 통신에서 프로세서 개입이 필요하지 않는 시간 T_c 보다 크다면 T_c 시간은 완전히 가려질 것이다. 즉, B 부분 수행 시간만이 고려된다. 반대의 경우에는 T_c 시간이 고려될 것이다. T_p는 통신에서 프로세서 개입이 필요한 시간, T_c는 통신에서 프로세서 개입이 필요하지 않는 시간, T_B 및 T_D는 각각 B 부분 및 D 부분 수행 시간, 그리고 T_r은 그 외 나머지 부분 수행시간을 나타낸다고 할 때 전체 소요 시간은 다음과 같다.

$$T = T_p + \max\{T_B, T_c\} + T_D + T_r$$

또한 중첩으로 얻을 수 있는 이득은 $\min\{T_B, T_c\}$이 될 것이다. 메시지 크기는 문제 크기에 영향을 받기 때문에 T_c는 문제 크기에 따라 달라진다. 앞에서 추정한 L, g 값을 통해 T_c를 구하고, 또한 실험적으로 T_B 값을 얻어 중첩 이득을 예측해보면 (그림 6)과 같다. 그림에서 '64P'는 64×64에서의 중첩 이득 예측 값을, 그리고 '64'는 실험 값을 나타낸다.



(그림 6) Jacobi에서의 중첩 효과

64×64와 96×96의 경우 추정 값과 실험 값이 거의 일치

하는 경향을 보인 반면, 문제 크기가 커질수록 두 값은 차이가 났다. 여기에는 크게 두 가지 원인이 작용하는 것으로 분석된다. 먼저 문제 크기가 작은 경우(64×64와 96×96)에는 T_B 값이 T_c 값보다 작아서 중첩 이득의 한계가 T_B가 된다. 이 경우 중첩 이득은 거의 정확히 T_B 값이 된다. 문제 크기가 좀 더 큰 경우에는 노드 개수에 따라 T_B 또는 T_c가 중첩 이득이 되는데, 128×128, 160×160이 여기에 해당된다. 다음으로 문제 크기가 이 보다 더 커지는 경우(192 × 192, 256 × 256)는 대부분의 시간이 통신보다는 계산에 의존적이고, 또한 캐쉬 메모리 영향이 크다. 실험 값이 추정 값보다 작은 이유는 중첩 효과를 얻기 위해 루프를 분할함으로써((그림 4)의 B 및 D), 지역성에 측면에서 손해를 보기 때문인 것으로 생각된다. 이 경우는 통신 모델 단독보다는 다른 모델의 적용이 필요할 것으로 보인다.

5. 결론 및 향후 연구

통신과 계산 작업의 중첩에 의한 시간 단축은 흔히 접근할 수 있는 최적화 방법 중의 하나이다. LogP 모델은 점대점 메시지 전송으로 표현된 알고리즘에서 유용한 성능 모델이긴 하지만, 원형 쉬프트 통신과 같은 군집 통신을 설명할 수는 없다. 군집 통신의 사용자 측면보다는 오히려 군집 통신 자체의 알고리즘에 유용한 모델일 수 있다. 또한 Xu와 Hwang의 군집 통신 모델은 최종 소요 시간만을 위한 모델로써, 중첩을 통한 최적화에 사용될 수 없다.

본 논문에서는 원형 쉬프트 통신 시에 중첩을 통해 얻을 수 있는 이득을 실험하고, 이를 분석하였다. 실험 데이타를 하위 수준의 점대점 전송 관점에서 분석하는 것은 종합적인 효과를 예측하기 매우 어려울 뿐 아니라, 최종 사용자 프로그램의 최적화 측면에서 유용한 방법은 아니다. 성능 모델은 사용자 코드를 최적화할 수 있는 수준, 가령 MPI 함수를 사용하는 수준에서 설명할 수 있어야 한다. 많은 데이타 병렬 프로그램들은 개별적인 점대점 전송까지 고려하여 최적화를 하지 않으며, 또한 군집 통신을 사용하고 있다. 따라서 점대점 전송의 최적화를 고려하는 LogP 모델은 데이타 병렬 프로그램을 설명하기에 충분하지 않다. 본 논문에서는 사용자 수준의 실험 데이타를 설명할 수 있도록 LogP 모델을 확장하였으며, 또한 확장된 모델 nLogP는 통신에 참여하는 노드 개수 n에 대한 함수로 각 매개 변수를 표현한다. 실험 환경에서 추출된 매개 변수들은 작은 값이긴 하지만, n에 대해 종속적임을 보여준다. 한편, Jacobi 예제에서 살펴본 바에 의하면, 실제 성능 이득을 예측하는데 있어 문제 크기가 커지는 경우 통신 모델만으로는 어렵다는 것을 보여준다. 메모리 작업이 $O(m^2)$인데 비해 통신 작업은 $O(m)$으로, 메모리 작업의 영향이 크기 때문이다. 메모리 성능 모델과의 결합에 대해서는 향후 연구 과제로 남는다.

원형 쉬프트 이외의 다른 군집 통신들, 가령 브로드캐스트의 경우는 매개 변수들이 n에 대해 매우 종속적일 것으로 예상된다. 이러한 매개 변수들을 추출하면 브로드캐스트의 중첩 최적화에 유용하게 사용될 것이다. 다른 군집 통신들에 대해서도 모델을 확장하는 것은 향후 연구 과제로 남는다.

참 고 문 헌

[1] D. E. Culler, A. Arpaci-Dusseau, R. Arpaci-Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa and F. Wong, "Parallel Computing on the Berkeley NOW," In Proc. of 9th Joing Symp. on Parallel Processing, Kobe, Japan, 1997.

[2] T. Sterling et al., "BEOWULF : A Parallel Worskstation for Scientific Computation," In Proc. of Int'l Conf. on Parallel Processing, 1995.

[3] A. Rogers and K. Pingali, "Process Decomposition though Locality of Reference," In Proc. of the SIGPLAN '89 Conf. on Programming Language Design and Implementation, June, 1989.

[4] C. Koelbel and P. Mehrota, "Programming Data Parallel Algorithms on Distributed Memory Machines Using Kali," In Proc. of the 1991 ACM Int'l Conf. on Supercomputing, June, 1991.

[5] D. E. Culler et al., "LogP : Towards a Realistic Model of Parallel Computation," In Proc. of ACM Symp. on Principle and Practice of Parallel Programming, ACM Press, pp.1-12, 1993.

[6] MPI Forum, "MPI : A Message Passing Interface Standard," Tech. Report CS-94-230, Computer Science Dpept., University of Tennessee, April, 1994.

[7] A. Geist et al., "PVM 3 User's Guide and Reference Manual," September, 1994.

[8] "IBM Parallel Environment for AIX : MPL Programming and Subroutine Reference," GC23-3893-00, IBM Corp.

[9] Zhiwei Xu and Kai Hwang, "Modeling Communication Overhead : MPI and MPL Performance on the IBM SP2," IEEE Parallel and Distributed Technology, Vol.4, No.1, pp.9-23, Spring, 1996.

[10] T. Agerwala et al., "SP2 System Architecture," IBM Systems Journal, Vol.34, No.2, 1995.

[11] R. W. Hockney, "The Communication Challenge for MPP : Intel Paragon and Meiko CS-2," Parallel Computing, Vol.20, No.3, pp.389-398, March, 1994.

[12] S. Fortune and J. Wyllie, "Parallelism in Random Access Machines," In Proc. of the 10th Annual Symp. on Theory of Computing, pp.114-118, 1978.

[13] A. Alexandrov et al., "LogGP : Incorporating Long Messages into the LogP Model," In Proc. of the 7th Annual Symp. Parallel Algorithms and Architectures, ACM Press, pp.95-105, 1995.

[14] J. M. Squyres et al., "LAM Installation Guide."

[15] D. F. Bacon, S. L. Graham and O. J. Sharp, "Compiler Transformations for High-Performance Computing," ACM Computing Surveys, Vol.26, No.4, pp.345-420, December, 1994.

김 정 환

email : jhkim@kku.ac.kr

1991년 서울대학교 계산통계학과 (학사)

1993년 서울대학교 대학원 전산과학과(이학석사)

1999년 서울대학교 대학원 전산과학과(이학박사)

1999년~2000년 삼성전자 통신연구소 연구원

2001년~현재 건국대학교 자연과학대학 컴퓨터·응용과학부 조교수

관심분야 : 병렬처리, 분산 시스템, 결함 내성, 그룹 통신

노 정 규

email : jkrho@skuniv.ac.kr

1991년 서울대학교 계산통계학과(학사)

1993년 서울대학교 대학원 전산과학과(이학석사)

1999년 서울대학교 대학원 전산과학과 (이학박사)

1999년~2001년 삼성전자 통신연구소 연구원

2002년~현재 서경대학교 컴퓨터과학과 전임강사.

관심분야 : 소프트웨어공학, 통신 소프트웨어

송 하 윤

email : hayoon@hongik.ac.kr

1991년 서울대학교 계산통계학과(학사)

1993년 서울대학교 대학원 전산과학과(이학석사)

2001년 Univ. of California, Los Angeles (박사)

2001년~현재 Scifin Finantial Service CTO

2001년~현재 홍익대학교 정보·컴퓨터공학부 전임강사

관심분야 : 병렬처리, 시뮬레이션, 위성통신, 성능평가, 데이타 방송