GEN_BLOCK간 재분산을 위한 통신 스케줄
(Communication Schedule for GEN_BLOCK Redistribution)

육 현 규 † 박 명 순 ††
(Hyun-Gyoo Yook) (Myong-Soon Park)

요 약 배열 재분산은 분산 메모리 컴퓨팅 환경에서 응용 프로그램의 수행 속도를 빠르게 하기 위해 많이 사용되고 있다. 특히 GEN_BLOCK간 재분산은 동적으로 부하가 변화하는 경우 최적화된 성능을 보이기 위해 필요하다. 배열 재분산에 관한 기존 연구들은 대부분 CYCLIC(N)등과 같은 정규 분산 패턴간 재분산에 대해서만 이루어져 왔다. 그러나 GEN_BLOCK등과 같은 비정규 분산 패턴간 재분산에서 발생하는 메시지패싱들은 정규 분산 패턴간 재분산과는 다른 특징을 보이기 때문에 이에 대한 새로운 연구가 필요하다.

본 논문은 GEN_BLOCK간 재분산에서 발생하는 메시지패싱들에 정규 분산 패턴간 재분산에서 발견되는 규칙성은 없는 반면 공간 지역성(spacial locality)이 존재함을 보이고, 이를 기반으로 최소 스텝 정리와 최소 크기 정리가 재분산의 성능을 향상시키는데 중요함을 증명하였으며, 기존의 리스트 스케줄링 방식에 재구성 단계(relocation phase)를 추가함으로써 최적 스케줄을 생성하는 알고리즘을 제시하였다.

마지막으로 제안한 알고리즘의 성능을 평가하기 위해, CRAY T3E와 IBM SP2에서 성능 평가를 수행하였으며, 그 결과 분산 메모리 병렬 머신에서 최소 스텝 정리와 최소 크기 정리를 만족하는 스케줄이 GEN_BLOCK간 재분산의 성능 향상에 중요함을 보였다.

Abstract Array redistribution is usually required to enhance algorithm performance in many parallel programs on distributed memory multicomputers. GEN_BLOCK redistribution, which is redistribution between different GEN_BLOCKs, is essential for load balancing. However, prior research on redistribution has been focused on regular redistribution, such as redistribution between different CYCLIC(N)s. GEN_BLOCK redistribution is very different from regular redistribution. Message passing in regular redistribution involves repetitions of basic message passing patterns, while message passing for GEN_BLOCK redistribution shows locality.

This paper proves that two optimal condition, reducing the number of communication steps and minimizing redistribution size, are essential in GEN_BLOCK redistribution. Additionally, by adding a relocation phase to list scheduling, we make an optimal scheduling algorithm for GEN_BLOCK redistribution.

To evaluate the performance of the algorithm, we have performed experiments on a CRAY T3E. According to the experiments, it was proven that the scheduling algorithm shows better performance and that the conditions are critical in enhancing the communication speed of GEN_BLOCK redistribution.

1. 서 론

데이타 병렬 프로그래밍은 분산 메모리 병렬 컴퓨팅 환경의 프로그래밍 파라다임으로 널리 인정되고 있다. 데이타 병렬 프로그래밍 파라다임에서 연산의 병렬 수행은 분산 저장되는 데이타에 따라 결정되며, 따라서 데이타의 분산 저장 형태가 프로그램의 수행 형태와 성능에 많은 영향을 미치게 된다. 데이타 병렬 프로그래밍 환경에서 분산 저장 형태(이하 분산 패턴과 혼용)는 크게 CYCLIC(N)의 형태로 표현 가능한 정규 분산 패턴

· 본 논문은 과학재단 핵심전문 연구(과제번호 981-0925-130-2)의 지원에 의해 연구되었음
† 학생회원 : 고려대학교 전산과학과
hyun@iLab.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
myongsp@iLab.korea.ac.kr
논문접수 : 1999년 5월 19일
심사완료 : 2000년 3월 8일

과 그렇지 않은 비정규 분산 패턴으로 구분된다.

데이타의 분산 패턴 중 High Performance Fortran (이하 HPF로 표기) 버전 2에서 새로이 채용한 분산 패턴인 GEN_BLOCK은 기존의 분산 패턴들이 연산에 참여하는 프로세서의 성능이나 부하에 관계없이 동일한 수의 배열 원소를 할당함으로써 발생하는 부하 불균형 문제 등을 해결하기 위해 제안되었다[1]. GEN_BLOCK 분산 패턴은 참여 프로세서에게 각기 다른 크기의 데이타 블록을 할당할 수 있도록 함으로써 부하 불균형을 일으키는 프로그램이나, 컴퓨팅 환경에서 최적의 성능을 보이도록 한다.

프로그램 수행 시 프로그램의 특성이나 수행 환경의 변화로 인하여 데이타의 접근 패턴이 바뀌거나, 프로세서 간 부하 불균형이 발생하게 되는 경우가 있다. 이러한 상황을 타개하기 위해서는 새로운 환경에 맞도록 데이타의 분산 형태를 변경해야 한다[2]. 특히 동적 부하 불균형이 발생할 경우 이를 해소하기 위해서는 각 프로세서가 자신의 부하에 맞는 양만큼의 데이타를 처리하도록 데이타의 분산 형태를 변경해야 한다. 따라서 동적 부하 불균형이 발생하는 시스템이나 프로그램의 경우 이를 해소하기 위해서는 GEN_BLOCK간의 재분산이 필수적이다.

배열 재분산은 크게 메시지 생성 단계와 통신 스케줄 단계로 구성된다[3][4][5]. 메시지 생성 단계에서는 재분산 전후의 배열 분산패턴을 효과적으로 검색하여 프로세서간에 교환해야할 데이타 부분과 그렇지 않은 데이타 부분을 결정하고 이를 통해 메시지들을 생성한다. 이때 한 프로세서는 다수의 다른 프로세서에게 메시지를 전달해야 하는 경우가 일반적으로 발생한다. 통신 스케줄 단계에서는 이와 같은 집합적 메시지 전달을 효과적으로 수행하도록 메시지간 전달 순서를 조정하는 일을 담당한다. 집합적 메시지 전달에 가장 효과적인 방법은 비대기 통신 라이브러리를 사용하는 것이다. 비대기 통신 라이브러리를 사용하는 경우 프로세서간 동기화 시간을 없앨 수 있고, 통신과 연산을 동시에 수행할 수 있기 때문에 병렬성이 증대되고 전체적인 통신 시간이 짧아지는 장점이 있다[4][6][7]. 그러나 재분산할 배열의 크기에 해당하는 통신 버퍼가 추가로 요구된다는 문제점 때문에 그 사용에 한계를 갖는다[4][7][8]. 이 때문에 재분산과 관련된 연구의 대부분은 대기 통신 라이브러리를 사용한 효과적인 통신 스케줄에 집중되어 왔다.

재분산 연구중 현재 많은 연구가 이루어지고 있는 CYCLIC(N)과 CYCLIC(M)간의 재분산의 경우 N과 M의 관계에 따라 특정 메시지패싱 패턴이 반복적으로 발생하고, 데이타의 이동이 전체 프로세서간에 이루어지는 특성을 갖는다. 따라서 N과 M의 관계 분석을 통해 반복되는 기본 메시지패싱 패턴을 찾아내고 이들의 전송 시 병렬성을 극대화함으로써 최소의 비용으로 재분산을 마치는데 연구의 초점이 맞추어져 있다. 그러나, GEN_BLOCK간 재분산의 경우 메시지패싱에 반복성이 보이지 않기 때문에 정규 분산 패턴간 재분산 연구 결과를 적용하기가 용이하지 않다.

GEN_BLOCK간 재분산에서 배열 블록은 프로세서 수에 제한된다. 따라서 재분산의 두 단계중 메시지 생성 단계의 오버헤드는 극히 미미하며, 본 연구에서는 메시지 패싱 스케줄링 단계에 중점을 두고 있다. GEN_BLOCK간 재분산에서는 메시지 패싱의 반복성은 없는 반면 한 프로세서에서 송신된 메시지들은 연속된 프로세서들이 수신을 하게 되고, 마찬가지로 한 프로세서가 수신한 메시지들은 연속된 다수의 프로세서로부터 송신되는 특성을 보인다. 본 논문에서는 이를 공간 지역성(Spacial Locality)이라 정의하고, 이를 기반으로 높은 성능을 보이는 스케줄 생성을 위해 최소 크기 정리와 최소 스텝 정리를 제안하고, 이들을 만족하는 최적 스케줄이 반드시 하나 이상 존재함을 증명한다. 또한 이 정리들을 만족하는 스케줄을 생성하는 스케줄링 알고리즘을 제안하여 GEN_BLOCK간 재분산의 성능을 높인다. 마지막으로 대표적인 분산 메모리 병렬 컴퓨터인 CRAY T3E와 IBM SP2에서 성능 평가를 통해 최소 크기 정리와 최소 스텝 정리가 재분산 성능을 높이는데 유효함을 증명한다.

본 논문의 2장에서는 본 논문에서 가정하고 있는 분산 메모리 병렬 컴퓨팅 환경의 전형인 GDM(General Purpose Distributed Memory) 모델을 소개하고, GDM에서 통신 시간을 모델링한다. 3장에서는 GEN_BLOCK간 재분산을 모델화하고 그 특성을 기술하며, 4장에서는 최소 스텝 정리와 최소 크기 정리를 정의하고 증명하였으며, 이를 기반으로 재분산 스케줄링 알고리즘을 제안한다. 5장은 CRAY T3E와 IBM SP2등과 같은 실제 분산 메모리 병렬 컴퓨팅 환경에서 최소 크기 정리와 최소 스텝 정리가 재분산 속도에 미치는 영향을 분석하였다. 6장에서는 연구 결과의 미비점과 향후 연구 방향을 기술하고, 7장에서는 재분산과 관련된 그 동안의 연구를 소개한다. 마지막으로 8장에서 결론을 맺는다.

2. 통신 시간 모델

분산 메모리 병렬 컴퓨팅 환경에서 프로세서간 정보

의 교환은 통신을 통해 이루어지며, 프로세서간 통신(interprocessor communication)은 프로세서 내부의 연산이나 데이타 이동보다 많은 시간이 소요된다. GDM (General purpose Distributed Memory) 모델은 분산 메모리 병렬 컴퓨팅 환경에서 사용된 대표적인 분석 모델로써, 프로세서간 통신을 시작 시간 (t_s)과 단위 데이타 전송 시간 (t_m)으로 표현한다[2][3][4][7][9]. 시작 시간은 매 통신마다 고정적으로 발생하는 시간 요소로, 메시지의 크기와는 관련이 없으며, 메시지 전송 요청 및 응답 시간, 문맥 교환 시간, 메시지 헤더 초기화 시간 등이 포함된다. 단위 데이타 전송 시간은 메시지 크기에 비례하여 소요되는 시간들의 합을 메시지 크기로 나눈 값이다. GDM 모델에서 크기 m인 메시지를 전달하는데 소요되는 시간 혹은 최대 크기가 m인 프로세서간 메시지 교환(shuffling) 시간 T_{STEP}은 식 1과 같이 표시할 수 있다.

$$T_{STEP} = t_s + m \times t_m \quad (1)$$

여러 단계의 메시지 교환 시간 T_{TOTAL}의 경우 각 단계의 통신 시간의 합이며, 식 2와 같다.

$$T_{TOTAL} = \sum T_{STEP} \quad (2)$$

이 모델에서는 메시지의 송신 노드와 수신 노드에서의 메시지 경합(node contention)을 허용하지 않으며, 한 통신 스텝에서 한 노드는 하나의 메시지만을 송신할 수 있으며, 마찬가지로 하나의 메시지만을 수신할 수 있다. 또한 상호연결망(interconnection network)상의 스위치 혹은 중간 노드에서의 메시지 경합(link contention) 역시 배제되며, 완전 연결 그래프(fully connected graph) 구조를 가정한다[3][9]. 이러한 이상적인 네트워크는 virtual channel과 cut through routing등의 기술로 최근 개발되는 분산 메모리 병렬 머신의 경우 어느 정도 현실화되고 있다[3].

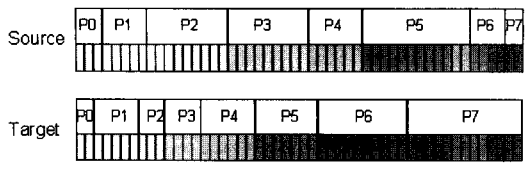
3. GEN_BLOCK간 재분산

HPF 버전 2.0에서 부하 불균형 문제 해소를 위해 제안된 분산패턴인 GEN_BLOCK은 BLOCK이나 CYCLIC등 HPF에서 제시한 기존의 분산 패턴이 모든 프로세서에게 동일한 크기의 데이타를 할당하는데 반하여, 그림 1의 (a)에서와 같이 서로 다른 크기의 블록으로 나뉘어 프로세서들에게 분산 저장된다. 따라서 부하가 많이 발생할 것으로 예상되는 프로세서에게는 작은 크기의 블록을 처리하도록 할당함으로써 부하 불균형 프로그램이나 부하 불균형 시스템에 적합한 분산 패턴이다. 그림 1의 (a)에서와 같이 부하 상황의 변화로 인하여 현재의 분산 패턴이 부적절할 경우 새로운 분산 패턴으로 바꿀 필요가 발생한다.

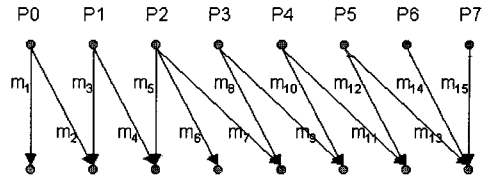
분산 패턴의 변화는 대규모 데이타 이동이 필요하게 되며, 이는 프로세서간 대규모 통신을 야기하게 된다. 그림 1의 (b)는 이를 도식적으로 표현한 것이다. 그림 1의 (b)에서 GEN_BLOCK간 재분산을 위한 메시지는 연속된 프로세서로 전달되는 것을 알 수 있다. 예를들어 임의의 프로세서 P_i와 P_{n-1}, P_n, P_{n+1}이 존재한다고 하자. 만일 P_i가 P_{n-1}과 P_{n+1}에 전달할 메시지가 있다면, P_n에 할당되는 데이타 블록의 크기가 0이 아닌 이상 P_i로부터 P_n으로 전달되는 메시지가 존재하며, 이는 P_n이 수신하는 유일한 메시지이다. 또한 역으로 만일 P_i가 P_{n-1}과 P_{n+1}으로부터 받아야 하는 메시지가 존재한다면, 원래 P_n이 가지고 있던 데이타 블록의 크기가 0이 아닌 이상 P_i는 P_n으로부터 받을 메시지가 존재하고, 이 메시지는 P_n이 송신하는 유일한 메시지이다. 본 논문에서는 이를 메시지의 "공간 지역성 (spacial locality)"이라고 정의한다.

본 논문에서 가정하는 GDM 모델에서는 송수신 프로세서가 서로 다른 메시지의 경우 동시에 전달할 수 있으나, 한 프로세서가 송신(혹은 수신)하는 메시지들의 경우 한번에 전달될 수 없으며 여러 단계에 걸친 전달이 필요한 특징을 갖는다. 본 논문에서는 이처럼 송신(혹은 수신) 프로세서가 같은 메시지들의 집합을 "인접 메시지 집합(neighbor message set)"이라 정의한다. 인접 메시지 집합내의 메시지들은 한번의 통신 스텝에 전달될 수 없으며, 특히 그림 1 (c)에서와 같이 공간 지역성을 갖는 메시지들로 구성된 인접 메시지 집합들은 포함하고 있는 메시지들에 따라 순서화 될 수 있으며, 이웃한 인접 메시지 그룹 사이의 교집합의 크기는 최대 1이며, 이웃하지 않은 그룹 사이의 교집합은 공집합인 성질을 갖는다.

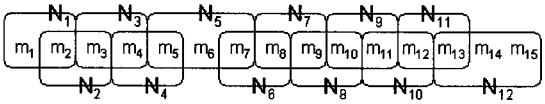
그림 1의 (d)는 재분산에 필요한 메시지들의 전송 순서를 결정한 스케줄이다. 스케줄은 동시에 전송될 수 있는 메시지들의 모임인 스텝으로 이루어져 있다. 인접 메시지 집합의 정의에 따라 같은 인접 메시지 집합내의 메시지들은 서로 다른 스텝에 포함되어야 한다. 그림 1 (a)의 재분산은 3개의 통신 스텝으로 완료된다. 2절에서 기술한 것처럼 한 송신 스텝의 전송 시간은 전달되는 메시지중 가장 큰 메시지의 크기에 따라 결정된다. 따라서 재분산을 완료하는데 소요되는 시간은 스텝의 수와 각 스텝의 최대 크기 메시지들의 합에 의해 결정되며, 그림 1 (d) 스케줄에 따라 재분산을 수행할 경우 재분



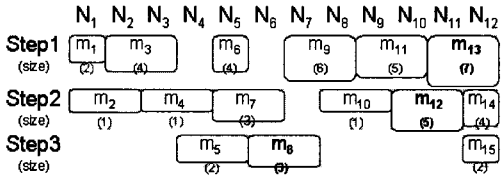
(a) Source and Target Distribution Patterns



(b) Messages and Spatial Locality



(c) Msg. Ordering and Neighbor Msg. Sets



(d) Schedule

Step1 - Step2 ({m_1, m_3},{m_2, m_4}), ({m_6},{m_7}), ({m_9, m_{11}, m_{13}},{m_8, m_{10}, m_{12}, m_{14}})

Step1 - Step3 ({m_6},{m_5}), ({m_9},{m_8}), ({m_{13}},{m_{15}})

Step2 - Step3 ({m_4, m_7},{m_5, m_8}), ({m_{14}},{m_{15}})

(e) Conflict Tuples

그림 1 GEN_BLOCK간 재분산

산의 크기는 식 1과 식 2에 따라 $3t_s+15t_m$가 된다.

보다 작은 크기의 재분산 스케줄을 생성하기 위해서는 스텝간의 메시지 위치 조정이 필요하다. 그러나 GDM 모델에서는 송신(혹은 수신) 프로세서가 같은 메시지를 동시에 전달하지 못하기 때문에 메시지의 위치 조정에는 제약이 따른다. 예를 들어 스텝 1과 스텝 2에 존재하는 메시지 m_1, m_3와 m_2는 m_1과 m_2, m_3와 m_2가 각각 인접 메시지 집합 N_1과 N_2에 속해있기 때문에 하나의 스텝에 존재할 수 없다. 또한 m_8은 같은 인접 메시지 집합 N_7에 속한 m_9 뿐만 아니라 m_{11}, m_{13}과도 동시 전달할 수 없다. 그 이유는 메시지 m_8과 m_{11}이 같은 스텝에 있을 경우 m_8, m_{11}과 각각 배타 관계인 m_9, m_{10}이 같은 스텝에 존재해야 한다. 그런데 m_9와 m_{10}은 같은 인접 메시지 집합에 속해있어서 동시에 전달될 수 없다. 같은 방식으로 메시지 집합 {m_8, m_{10}, m_{12}, m_{14}}와 메시지 집합 {m_9, m_{11}, m_{13}}은 같은 스텝에서 전달될 수 없다. 따라서 만일 m_8을 스텝 1으로 옮기고자 할 경우 {m_8, m_{10}, m_{12}, m_{14}}내의 모든 메시지를 스텝 1으로, {m_9, m_{11}, m_{13}}은 스텝 2로 옮겨야 한다. 본 논문에서는 이처럼 주어진 두 스텝 내에 존재하는 메시지들 중에 동시에 전달될 수 없는 메시지 집합의 쌍을 "배타 튜플(conflict tuple)"이라 정의한다. 그림 1 (e)는 (d)의 스케줄내의 모든 조합의 두 스텝간에 존재하는 배타 튜플들을 보여준다. GEN_BLOCK간 재분산에 대한 수학적 모델 및 정의는 Appendix I에 있다.

4. 최적화 재분산 스케줄링

이와 같은 모델에서 GEN_BLOCK간 재분산 시간을 단축하기 위해서는 가능한 작은 수의 스텝으로 재분산을 완료해야 하며, 각 스텝의 최대 메시지들의 합이 최소가 되도록 메시지들을 스케줄해야 한다.

4.1 최소 스텝 정리

GDM 모델에 따르면 한 스텝 안에서 병렬로 교환(shuffling)되는 메시지들의 밀집도나 형태로 인한 성능

하락은 없으나, 통신 스텝의 증가는 t_s에 해당하는 시간이 추가로 소요되기 때문에, 큰 폭의 성능하락을 야기한다. 따라서 메시지들간의 병렬성을 최대한 살려서 최소한의 스텝으로 재분산을 수행하는 것이 빠른 시간 안에 재분산을 완료하는데 중요하다.

그런데 GEN_BLOCK간 재분산에 있어서 인접 메시지 집합 내의 메시지들은 같은 스텝에서 수행하지 못하는 성질을 갖는다. 따라서 통신 스텝은 최소한 최대 배치 그룹의 크기보다는 커야 한다. 반면에 만일 통신 스텝 보다 작은 크기의 그룹이 있을 경우 그림 1 (d)에서 보는 바와 같이 그 그룹에 해당하는 열에는 통신 스텝과 그룹 크기의 차에 해당하는 공란이 존재하게 된다. 예를 들어 크기 2인 그룹 N_4의 경우 Step1에 공란이 있게 된다. 이럴 경우 m_1, m_3와 m_2와의 위치를 교환할 경우 m_4가 Step1에 위치할 수 있게 되고 2개의 스텝으로 전송이 완료된다. 이처럼 각 그룹내의 메시지들은 그룹의 크기에 해당하는 스텝 안에 전송이 가능하다. 그러므로 GEN_BLOCK간 재분산을 최적의 속도로 완료하기 위해서는 다음의 최소 스텝 정리를 만족해야 한다. 최소 스텝 정리에 대한 증명은 Appendix II에 있다.

정리1. 최소 스텝 정리 GEN_BLOCK간 재분산을 완료하는데 필요한 최소 스텝 수는 인접 메시지 집합들이 포함한 메시지 수중 최대 값과 같다.

4.2 최소 크기 정리

식 1에서 보는 것처럼 통신 스텝의 수행 시간은 전달되는 메시지들 중 최대 크기 메시지에 따라 결정된다. 따라서 재분산을 수행하는데 있어 각 스텝내의 최대 크기 메시지의 크기를 줄이는 것이 성능향상에 중요하다. 따라서 전체 재분산 크기를 줄이기 위해서는 최대 크기 메시지를 보낼 때, 작은 크기의 메시지보다는 큰 크기의 메시지를 함께 보내는 것이 유리하다. 다시 말해 최대 크기 메시지를 첫 번째 스텝에 배치한다면, 이 메시지와 동시에 전송할 수 있는 메시지 중 가장 큰 메시지를 첫 번째 스텝에 배치해야 다음 스텝에 할당되는 메시지들의 최대 크기를 줄일 수 있으며, 만일 다른 스텝의 최대 메시지 보다 작으면서도 교환 가능한 메시지가 첫 번째 스텝에 존재한다면 이 메시지들의 위치를 바꿈으로써 보다 작은 크기로 재분산을 완료할 수 있다. 정리 2는 이를 일반화한 정리이다. 최소 크기 정리에 대한 증명은 Appendix III에 있다.

정리 2. 최소 크기 정리 최소 크기를 갖는 GEN_BLOCK간 재분산 스케줄은 스케줄 내의 모든 스텝간에 존재하는 모든 배타 튜플에 대하여 최대 메시지의 크기가 큰 메시지 집합이 상위 스텝에 존재한다.

4.3 재배치 스케줄링 알고리즘

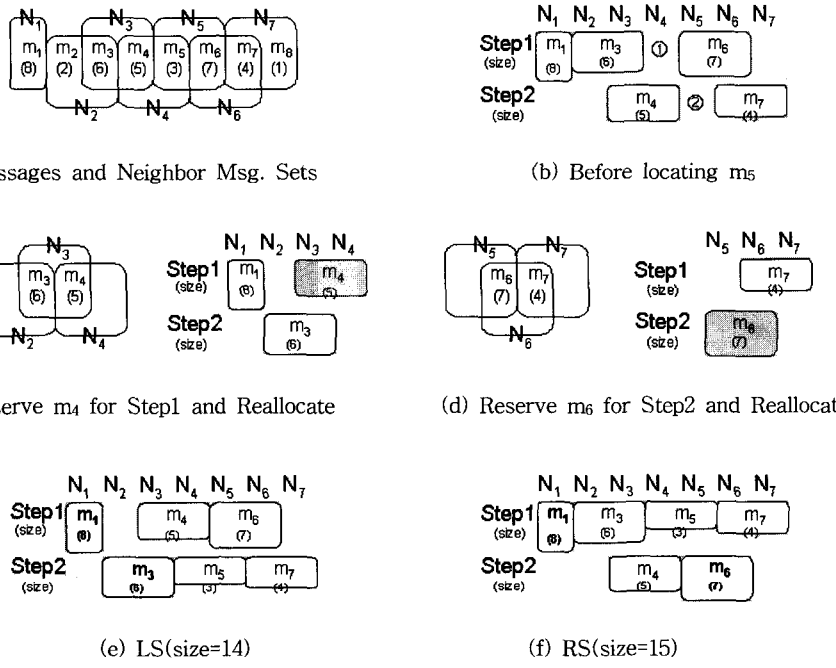
알고리즘 1은 앞에서 기술한 최소 스텝 정리와 최소 크기 정리를 만족하는 최적화된 스케줄을 생성하는 알고리즘이다. 본 알고리즘에 따르면 크기순으로 정렬된 재분산 메시지 집합을 입력으로, 최상위 스텝부터 배치 가능한 순서로 스케줄한다(줄 6-12). 만일 어떠한 스텝에도 배치될 수 없는 메시지가 발생할 경우, 그 메시지를 삽입할 수 있도록 이미 배치된 메시지들을 재배치함으로써(줄 13-16) 스케줄이 최소 스텝 정리를 위배하지 않도록 한다.

알고리즘 1 재배치 스케줄링 알고리즘

```
   Algorithm: locate
       input M : Message Set sorted by size and reservation
             rn : reserved neighbor set
       output S : Schedule
   {
1      Sort(M);
2      SNT(1, rn) = 1
3      for (i=0; i<card(M); i++) {
4          m = M(i);
5          for (j=0; j<card(S); j++) {
6              s = S(j)
7              If ((SNT(s,nb(m,1))==0) && (SNT(s,nb(m,2))==0)) {
8                  SNT(s,nb(m,1)) = 1;
9                  SNT(s,nb(m,2)) = 1;
10                 Insert(m, s);
11                 Go to next;
12         }   }
13         LS = replace(S, locate(LM, nb(m,1)));
14         RS = replace(S, locate(RM, nb(m,2)));
15         if (sz(LS) < sz(RS)) S = LS;
16         else S = RS;
17 next:
18     }
19     return S;
   }

sort(M) sorts the messages in M by size.
SNT(s,n) is flag to indicate "there is n-th neighbor set message in step s"
card(S) is a cardinality of set S
nb(m,i) is i-th neighbor set of message m.
Insert(m,s) inserts message m into step s.
replace(S,S') returns a new schedule in which the position of m in S are
              replaced as that in S'.
sz(S) is a size of schedule S.
```

그림 2 (d)는 이러한 과정을 거쳐 생성된 스케줄로 그림 2의 재분산의 경우는 재배치 과정 없이 스케줄이 생성될 수 있었다. 이처럼 재배치 과정 없이도 모든 메시지가 스케줄 가능한 경우에는 크기가 큰 메시지가 우선적으로 상위 스텝에 위치되기 때문에 최소 크기 정리를 만족시킨다.



(a) Messages and Neighbor Msg. Sets

(b) Before locating m_5

(c) Reserve m_4 for Step1 and Reallocate

(d) Reserve m_6 for Step2 and Reallocate

(e) LS(size=14)

(f) RS(size=15)

그림 2 메시지 재배치 과정

그림 2의 예와 같이 크기에 따른 배치 과정에서 최소 스텝 정리를 위배하는 스케줄이 생성될 경우 이미 배치된 메시지들의 위치를 바꿀 필요가 있다. 그림 2의 재분산은 최대 인접 메시지 집합의 크기가 2이므로 2개의 스텝으로 스케줄될 수 있다. 본 예는 (b)에서 보이는 것처럼 크기순으로 배치해 나가면 메시지 m_5에서 인접 메시지 집합내의 메시지인 m_3와 m_6가 각각 스텝 2와 스텝1에 이미 배치되어 더 이상 배치될 수가 없는 상황이 발생한다. 이 상황에서 m_5가 배치될 수 있는 위치는 ①과 ②이다. 재배치 스케줄링 알고리즘은 기존에 배치된 메시지들을 재배치함으로써 ①과 ② 각각의 경우에 m_5가 위치되는 스케줄을 작성해 보고 이중 최소 크기 정리를 만족하는 스케줄을 선택한다. 예를 들어 m_5를 ②에 위치시키는 스케줄을 만들 경우 (c)에서 보는 것처럼 m_5가 위치할 인접 집합인 N_4와 N_5중 N_4 이전에 위치된 메시지들을 가지고 부분 스케줄을 작성한다. 이때 m_5와 충돌을 일으켰던 m_4는 스텝 1에 고정시킴으로써 m_5가 ②위치에 놓일 수 있도록 한다. 반대로 ①을 선택했을 경우는 (d)에서 보이는 것과 같이 N_5 이후에 위치된 메시지들로 재귀 호출을 수행한다. 이때 m_6는 스텝 2에 고정함으로써 m_5가 ①에 위치할 수 있도록 한다. 부분 스케줄 작성이 완료되었으면, (e)와 (f)에서와 같이 전체 스케줄 S에서 부분 스케줄 부분을 교체한 LS와 RS를 생성한다. 마지막으로 LS와 RS중 크기가 작은 LS를 S로 선택함으로써 충돌을 해결하고 최소 스텝 정리를 만족시키는 스케줄을 생성할 수 있다. 또한 (a)의 경우 ①와 ②에 스케줄된 메시지가 없기 N_4와 N_5를 중심으로 때문에 배치 튜플이 ({m_3}, {m_4})와 ({m_6}, {m_7})로 양분된다. 반면 (e)의 경우 ({m_4, m_5}, {m_3, m_5, m_7})과 같이 이들이 하나로 합쳐지면서 큰 메시지가 스텝 1에 놓임으로써 최소 크기 정리를 만족시키게 된다.

5. 성능 평가

본 장에서는 대기 통신 라이브러리를 이용한 재분산에서 최소 스텝 정리와 최소 크기 정리의 영향을 측정하는 다양한 실험을 수행하였다. 본 실험은 수퍼 컴퓨팅 센터가 보유하고 있는 Cray T3E와 서울대의 IBM SP2에서 수행되었으며, 사용한 통신 라이브러리는 MPI이고, 사용한 언어는 Fortran 77이다.

성능 평가를 위해 본 실험에서는 다음의 5가지 스케

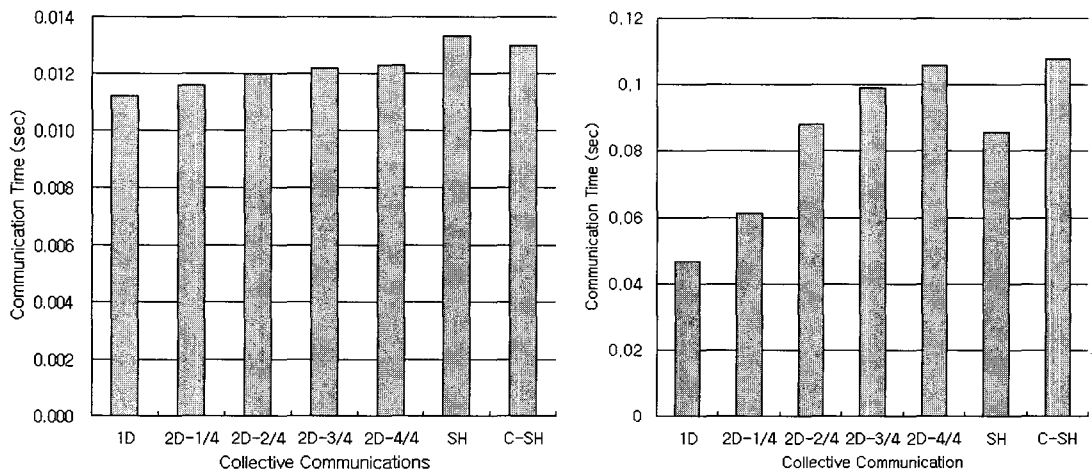
줄링 알고리즘을 사용하였다. NBLK는 비대기 통신 라이브러리를 사용한 재분산으로 MPI에서 대표적인 일대일 비대기 통신 라이브러리인 MPI_ISEND를 사용하였다. 나머지 스케줄들은 대기 통신 라이브러리를 사용한 재분산으로써, 스텝 단위의 메시지 전송이 손쉬우며 한 스텝에서 한 프로세서는 하나의 메시지를 송신하고 수신할 수 있는 등 GDM 모델의 가정을 잘 만족하는 대기 통신 라이브러리인 MPI_SENDRECV를 사용하였다.

- NBLK는 비대기 통신 라이브러리를 사용하여 무작위 순으로 메시지를 전송하였다. 전송 시간은 첫 번째 메시지 전송 순간부터 전송된 모든 메시지가 송수신 완료되는 순간까지로 하였다.
- OPT은 재배치 스케줄링 알고리즘으로 생성한 스케줄로 최소 스텝 정리와 최소 크기 정리 모두가 만족됨을 보장한다.
- MIN_STEP은 무작위로 리스트 스케줄링으로 생성하였으며, 최소 스텝 정리를 위배할 경우 메시지 재배치를 통해 최소 스텝 정리를 만족시키도록 한 스케줄이다.
- MIN_SIZE는 메시지의 크기에 우선 순위를 둔 리스트 스케줄링으로 생성하였으며, 최소 크기 정리가 만족됨을 보장한다.
- RANDOM은 무작위 리스트 스케줄링 알고리즘을 사용한 스케줄이다. 최소 스텝 및 크기 정리 모두가 만족됨을 보장하지 못한다.

본 실험에서는 제안하는 최소 스텝 정리와 최소 크기 정리가 각기 재분산 성능에 어떠한 영향을 미치는가를 관찰하기 위해 단순히 제안하는 스케줄인 OPT와 이상적인 스케줄로 간주할 수 있는 비대기 통신 라이브러리를 사용한 NBLK 사이의 비교뿐만 아니라 최소 스텝 정리만을 만족하는 MIN_STEP, 최소 크기 정리만을 만족하는 MIN_SIZE, 모든 정리를 만족시키지 않는 RANDOM의 성능을 비교 분석하였다. 또한 서로 다른 상호연결망 구조를 갖는 CRAY T3E와 IBM SP2에서 실험을 수행함으로써 상이한 환경에서 두 최적화 정리의 영향을 측정하였다.

본 실험에서 사용한 CRAY T3E와 IBM SP2는 상이한 상호연결망(interconnection network)을 가진 머신이다. CRAY T3E는 3D Torus 구조의 상호연결망을 채택하고 있다. 따라서 짧은 라우팅 거리를 확보할 수 있으며, virtual channel, adaptive routing 기법 등을 채택하여 메시지 전달 속도를 향상시키고 상호 연결망 상에서의 경합(contention)을 최소화하였다. 반면 IBM SP2는 다단계 상호연결망(Multistage Interconnection Network)을 채택하였다. 따라서 전체 노드간 메시지 교환시 경합 발생 확률이 높고, 경합 발생시 일부 메시지의 전달이 늦어지는 문제를 갖는다[5][10].

그림 3은 이를 실험적으로 확인하기 위해 하나의 통신 스텝에 전달이 가능한 여러 메시지 집합들을 T3E와



(a) CRAY T3E (b) IBM SP2

그림 3 CRAY T3E와 IBM SP2에서 메시지 교환 시간

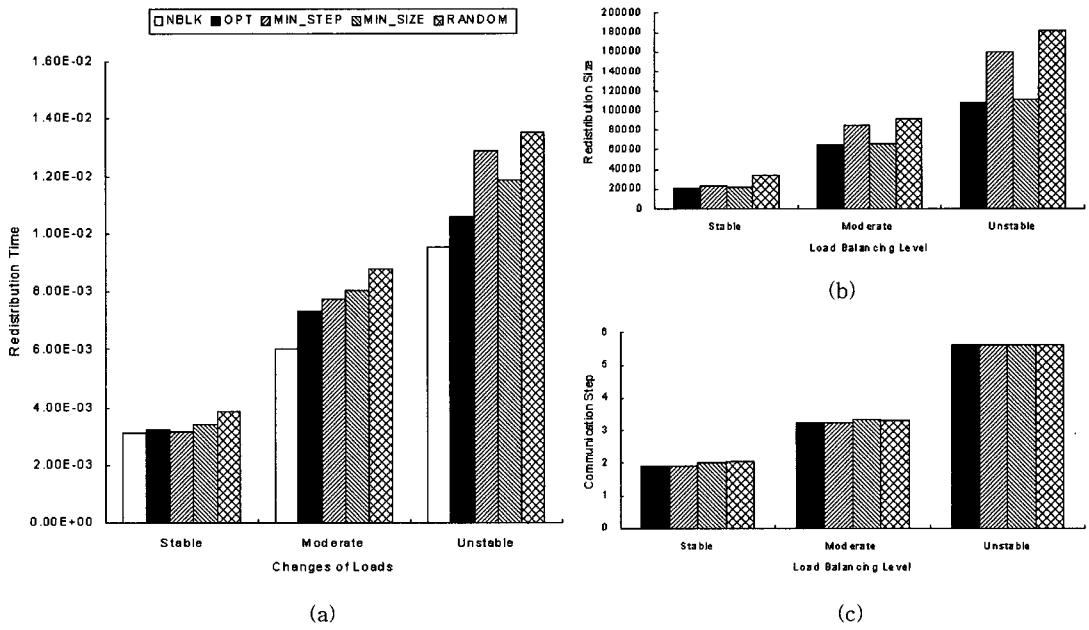


그림 4 CRAY T3E에서 부하 변화에 따른 재분산 시간

SP2에서 전달했을 때 소요되는 시간을 측정한 그래프이다. 이 실험에서 사용한 메시지 집합은 다음과 같다.

- 1D : 두 노드사이에 하나의 메시지를 전달하는 통신이다.
- 2D : 두 노드 사이에 두 개의 메시지를 서로 교환하는 통신이다.
- 2D-n/4 : 2D 형태의 통신으로 이때 한 메시지의 크기는 기준이 되는 메시지의 n/4이다.
- SH : 네 노드가 쉬프트 형태로 메시지들을 전달한다.
- C-SH : 네 노드가 순환 쉬프트 형태로 메시지들을 전달한다.

이 실험 결과 T3E의 경우는 메시지의 수와 메시지의 크기에 관계없이 상대적으로 일정한 전달시간을 요구하는데 비해, SP2는 메시지 수와 크기가 증가함에 따라 통신 완료 시간이 현저히 느려지는 것을 관찰할 수 있다.

이와 같은 상호연결망의 특징을 고려해 볼 때 T3E의 경우는 본 논문에서 가정한 GDM 모델에 좀 더 근접하며, SP2의 경우는 GDM과는 거리가 있다. 본 논문에서는 이처럼 상이한 환경에서 실험을 수행함으로써 최소 크기 정리와 최소 스텝 정리가 GEN_BLOCK 재분산에 어떤 영향을 미치는지를 관찰하고자 한다.

5.1 CRAY T3E에서의 실험 결과

그림 4는 시스템의 부하 불균형 정도에 따른 재분산 시간을 측정한 결과이다. 본 실험에서 가정한 부하 불균형 정도는 'stable', 'moderate', 'unstable'의 3단계로 나누었다. stable 상태는 부하의 변화가 크기 않은 상태를 가정한 것으로 각 프로세서의 블록 크기의 표준 편차가 평균의 10 % 이내에 머무르는 분산 패턴들을 사용하였으며, 실험 결과는 서로 다른 20개의 분산 패턴간에 재분산 시간을 측정, 이의 평균을 최종 결과값으로 삼았다. moderate 상태는 표준 편차가 평균의 45 % 에서 55 % 사이에 있는 분산 패턴들을 사용하였으며, unstable은 표준 편차가 평균의 90 % 에서 100 %에 이르는 분산 패턴들을 사용하였다.

이 실험 결과 모든 경우에서 비대기 통신 라이브러리를 사용한 NBLK가 가장 우수한 성능을 나타냈으며, 최소 크기 정리와 최소 스텝 정리를 만족시킨 스케줄을 사용한 OPT가 대기 통신 라이브러리를 사용한 다른 스

케줄보다는 우수한 성능을 보여주고 있고, 스케줄을 전혀하지 않은 RANDOM의 경우 모든 상황에서 최악의 성능을 보인다.

그림 4 의 (b)와 (c)는 최소 스텝 정리와 최소 크기 정리의 영향을 살펴보기 위한 그래프이다. 그림 4 (b)에 따르면 최소 크기 정리를 만족한 스케줄인 OPT와 MIN_SIZE가 가장 작은 재분산 크기를 보임을 알 수 있다. 특히 이러한 경향은 unstable 상황에서 극명하게 나타나며, 그림 4 (a)의 unstable 상황에서 MIN_STEP의 성능이 악화되는 주원인으로 생각된다. (c) 그래프는 각 스케줄의 평균 통신 스텝 수를 나타낸 것으로 최소 스텝 정리를 만족하는 스케줄과 그렇지 못한 스케줄간의 차이가 크지 않음을 알 수 있다. 이는 GEN_BLOCK 간 재분산이 상당히 sparse한 집합적 통신으로 임의의 방법으로 스케줄을 하더라도 최소 스텝 수에 근접하는 스텝 안에 재분산을 마칠 수 있기 때문이다. 그러나 통신 스텝의 증가는 단순한 메시지 크기와는 달리 식 1과 2에서도 알 수 있듯이 t_s값이 추가되기 때문에 큰 폭의 성능 하락을 야기한다. 이 때문에 재분산 크기가 거의 같은 MIN_SIZE와 OPT가 약간의 통신 스텝 차이로 많은 성능 차이를 나타내게 된다.

이처럼 최소 스텝 정리와 최소 크기 정리는 어느 것이 절대적인 우위를 보이지는 않으며, 재분산 전후의 배열 분산 패턴과 통신 환경에 따라 성능에 미치는 영향의 크기가 달라진다. 재분산 전후의 배열 분산 패턴은 재분산에 필요한 메시지들의 수와 송수신 프로세서들, 크기 등을 결정하며, 이에 따라 두 정리의 영향이 달라진다. 또한 통신 환경 특히 식 1에서 t_s와 t_m의 비는 최소 스텝 정리가 미치는 영향의 크기를 결정한다. 다시 말해 t_s가 상대적으로 큰 환경에서 스텝의 증가는 큰 폭의 성능 하락으로 이어지며, t_s가 0인 환경에서는 최소 스텝 정리가 무의미해진다.

그림 5는 moderate 상황에서의 재분산을 배열 크기와 프로세서 수를 변경시키면서 측정한 결과이다. 이러한 결과들에서도 NBLK는 항상 우수한 결과를 나타냈고, OPT는 NBLK에는 미치지 못하지만 많은 경우 대기 통신 라이브러리를 사용한 다른 스케줄보다는 우수한 결과를 보인다.

5.2 IBM SP2에서의 실험 결과

본 실험에서는 앞 절에서 소개한 실험과 동일한 실험을 IBM SP2를 대상으로 수행하였다. 그림 6은 그 결과 그래프들이다. 이 그래프들에 따르면 최소 스텝 정리를 만족하는 스케줄을 사용한 OPT와 MIN_STEP의 성능이 상대적으로 낮아졌음을 알 수 있다. 즉, OPT의 경우는 최소 크기 정리만을 만족하는 MIN_SIZE와 거의 같은 성능을 나타내며, 심지어 MIN_STEP은 무작위 스케

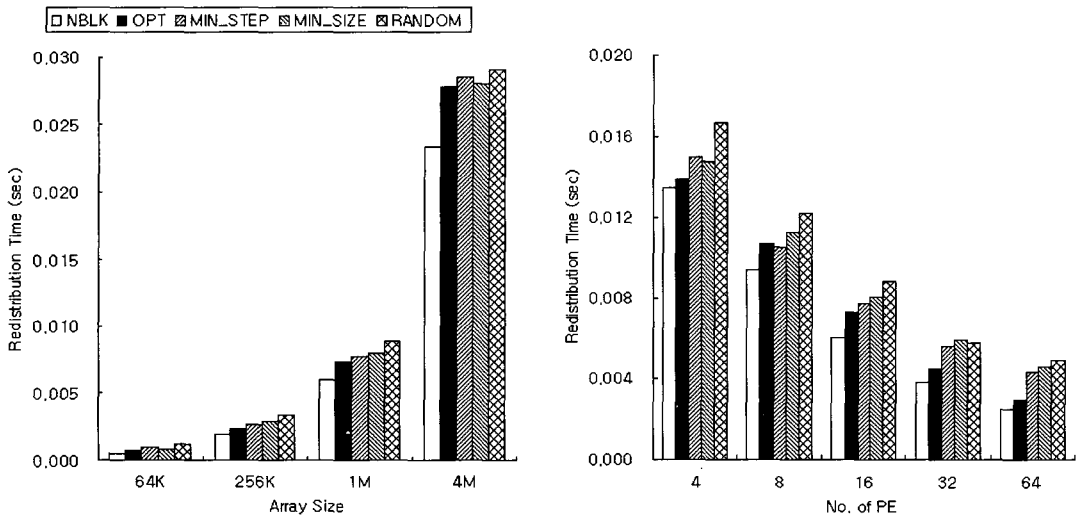


그림 5 CRAY T3E에서 배열 크기와 프로세서 수의 변화에 따른 재분산 시간

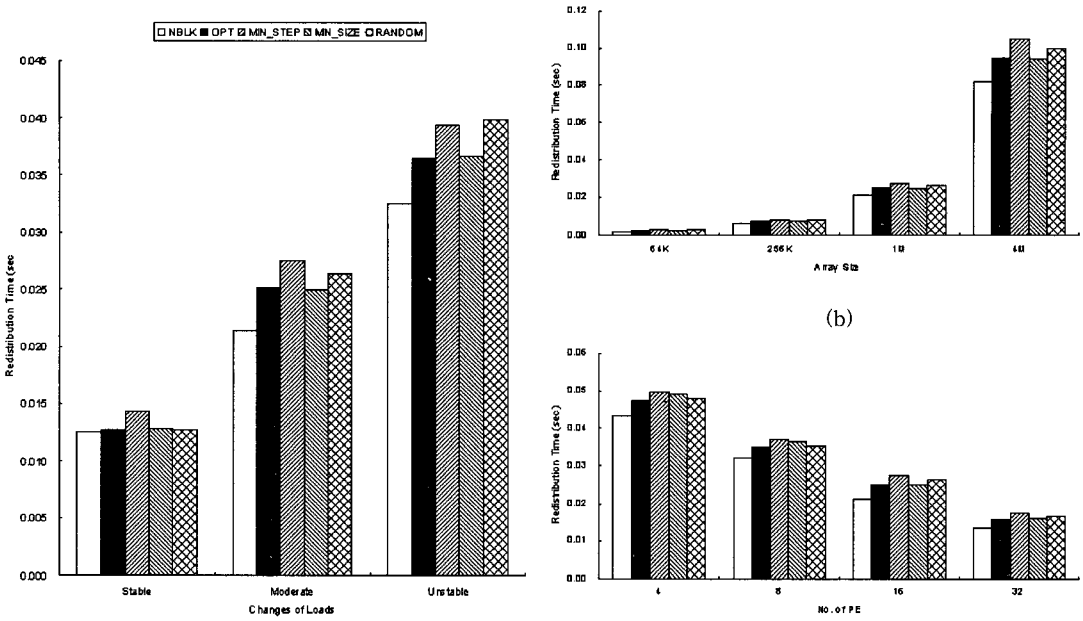


그림 6 IBM SP2에서 부하 변화에 따른 재분산 시간

줄을 사용한 RANDOM보다도 나쁜 성능을 보인다.

이와 같이 예상과는 다른 결과를 보이는 이유는 앞에서도 언급했듯이 SP2가 본 연구에서 가정한 GDM 모델과는 거리가 있는 컴퓨팅 환경이고, 특히 집합적 메시지 전달에 있어서 상호연결망상의 스위치에서 경합 발생 확률이 높기 때문이다. 최소 스텝 정리는 메시지들의 병렬성을 극대화함으로써 최소한의 통신 스텝에 재분산 메시지의 전달을 완료하고 이를 통해 성능 향상을 꾀하는 것을 목적으로 하고 있다. 그러나 SP2의 경우는 그림 3에서 보듯이 병렬성이 증가될수록 더 오랜 시간이 소요되는 특징을 보인다.

6. 향후 연구 방향

그림 4, 5, 6등에서 보이는 바와 같이 최소 크기 및 최소 스텝 정리를 만족하는 경우에도 여전히 비동기 라이브러리를 사용하는 경우와는 큰 성능 차이를 나타내고 있다. 이러한 성능 차이의 주원인은 이미 지적했듯이 비동기 라이브러리를 사용하는 경우 통신과 연산을 병렬 수행할 수 있고, 프로세서간 동기화 오버헤드가 없기 때문이다.

이중 특히 본 연구에서 제안하는 방식은 스텝 단위로 메시지를 전송하기 때문에 프로세서간 동기화 오버헤드가 높다. 즉 메시지 전송을 마친 프로세서의 경우에도 다른 모든 프로세서가 해당 스텝을 마칠 때까지 다음 메시지 전송을 시작할 수 없는 문제를 가진다. 이를 극복하기 위해서는 프로세간 동기화가 없이 메시지의 크기와 통신 속도에 따른 재분산에 대한 모델링 및 스케줄링 방법을 연구해야 한다.

그리고 현재 연구는 GEN_BLOCK 형태로 분산되어 있는 1차원 배열간의 재분산에 대한 것이었다. 앞으로는 이를 다차원으로 확대하고, 기존의 정규 분산 패턴과 GEN_BLOCK간의 재분산에 대해서도 계속 연구를 수행해야 한다.

7. 관련 연구

Stichnoth 등은 일반적인 CYCLIC(n) 분산 패턴에서 전역 주소와 지역 주소 사이의 변환 공식과 CYLIC(n)-to-CYCLIC(m)의 재분산시 각 프로세서가 다른 프로세서에게 송신할 데이타와 모든 다른 프로세서로부터 수신할 데이타들을 (시작, 끝, 간격)의 트리플인 closed-form을 구하는 방법을 제시하였다. 그리고 재분산과 같은 all-to-all 통신시 수신측이 수신되는 데

이타를 빠르게 처리하는 것이 속도향상에 중요한 요소임을 실험을 통해 입증하였다. 또한 논문에서는 전역-지역 주소 변환과정이 송수신측 모두 필요함을 이용하여 송신측에서 수신측이 필요한 변환된 지역 주소를 데이타와 함께 전송함으로써 수신측의 부담을 줄여 통신 속도를 향상시키는 deposit model을 제안하였다[11].

Kalns 등은 BLOCK-to-CYCLIC(c)의 재분산에서 데이타를 할당받는 프로세서의 순서를 바꾸어 줌으로써 실제 발생하는 통신의 양을 줄이는 방법을 제안하였다. 저자들은 HPF 표준안에 CYCLIC 분산이 꼭 0번부터 순서대로 되어야 한다는 규정이 없으며, 단지 한 프로세서에 속한 데이타의 간격이 프로세서의 수이면 된다는 것을 발견하고, 데이타를 할당받는 프로세서의 순서를 변화시킴으로써 최소의 통신만이 발생하도록 하는 방법을 제안하였다[12].

Gupta 등은 BLOCK-to-BLOCK, CYCLIC-to-BLOCK, BLOCK-to-CYCLIC, CYCLIC-to- CYCLIC 재분산에 필요한 송수신 프로세서 집합(PSend, PRecv)과 송수신 데이타 집합(DSend, DRecv)을 closed-form으로 계산해내는 방법을 제시하고, closed-form으로 표현이 안돼는 CYCLIC(n)-to-CYCLIC(m)의 재분산은 가상 프로세서의 개념을 도입하여 위의 네 가지 경우 중 하나로 변환하여 수행하는 메커니즘을 제시하였다[13].

Thakur 등은 CYCLIC(m)-to-BLOCK, BLOCK-to-CYCLIC(m), CYCLIC(n)-to-CYCLIC(kn), CYCLIC(kn)-to-CYCLIC(n) 재분산방법을 제시하고, CYCLIC(m)-to-CYCLIC(n)의 일반적인 재분산을 CYCLIC(m)-to-CYCLIC(lcm(n,m)) -to-CYCLIC(n)의 두 단계로 바꾸어서 수행하는 방법을 제시하였다[6].

이 논문에서는 특히 비동기 통신과 동기 통신의 장단점을 비교하였으며, 비동기 통신은 비록 버퍼영역이 많이 필요하지만 통신과 연산이 중복 수행되어 통신 오버헤드를 줄일 수 있다고 평가하였다. 이 논문은 초기 재분산 논문들에서 보이는 몇 가지 가정을 처음으로 나타낸 논문이다. 첫 번째 가정은 일반적인 재분산시 최소 공배수를 사용하는 것이다. 두 번째 가정은 동기 통신과 비동기 통신을 구분하였고, 비동기 통신을 보다 높이 평가하였다. 또한 비록 2차원 배열이긴 하지만 간접 통신(indirect communication)의 효과를 처음으로 언급하였다. 이 논문의 영향으로 이후의 연구들에서는 일반적인 재분산 방법에 대해 연구하기 보다 CYCLIC-to-CYCLIC(m) 혹은 CYCLIC(n)-to-CYCLIC(kn)의 최적화 재분산에 초점을 맞추었다. 또한 스케줄링과 간접 통신을 통해 동기 통신의 성능을 비동기 통신과 비슷하게 만드는데 주안점을 두었다.

Kaushik 등은 CYCLIC(n)-to-CYCLIC(kn) 재분산을 다단계 간접 통신(multiphase indirect communication)을 통해 통기 통신의 성능을 개선하였다. 이 논문에서는 CYCLIC(n)-to- CYCLIC(kn) 재분산에서 k의 크기가 클 경우 병렬로 수행할 수 없는 통신의 수가 많아져서 여러 단계의 통신이 필요하지만 이를 n->k'n->kn (k' < k)로 중간 단계를 거칠 경우 메시지의 크기는 커지지만 통신의 개수는 줄어들어서 통신 단계가 줄어듦을 발견하고 효과적인 중간단계를 찾는 방법을 제시하여 성능을 향상 시켰다[9].

Walker 등은 CYCLIC(n)-to-CYCLIC(kn) 재분산에서 동기 통신을 사용하는 방법도 스케줄링을 통하여 효과적으로 하면 비동기 통신을 사용하는 방법에 비견할 만큼의 속도를 낼 수 있음을 보여주는 논문이다. 이 논문은 특히 아주 단순한 코드를 통해 MPI를 사용하여 재분산을 효과적으로 할 수 있음을 보여주었다. 이 논문에서 사용한 MPI_SENDRECV 함수는 최대한의 병렬성을 보장하면서도 단계적 스케줄링을 가능하게 하여 스케줄링을 통해 통신의 순서를 조정할 수 있는 기반이 되었다[7].

Lim 등은 [7]에서 제안한 방식에 간접 통신을 도입하여 비동기 통신보다도 높은 성능을 보이는 방법을 제시하였다. 간접 통신은 대단히 많은 통신을 수행해야 할 경우 송신 프로세서가 직접 수신 프로세서에게 메시지를 전달하는 것이 아니라 존재하는 다른 메시지들에 업혀서 전달됨으로써 메시지의 크기는 어느 정도 증가하지만 전체적인 메시지 수와 단계를 줄여 성능향상을 꾀하는 방식이다[3].

Desprez 등은 [7]을 일반적인 재분산으로 확장한 논문이다. 이 논문에서는 CYCLIC(n)-to- CYCLIC(m) 재분산에서 동일한 통신 패턴이 나타나는 수퍼 블록의 크기를 LCM(n,m)으로 하고, 다양한 통신 형태를 보이는 예를 통해서 최적화의 가능성을 증명한다. 그러나 이 논문은 결정적으로 최적화된 통신을 생성하는 스케줄링 방법을 만들어내는 알고리즘을 구체적으로 제시하고 있지는 못하다[4].

8. 결 론

본 논문에서는 동적 부하 불균형 환경에서 수행되는 데이타 병렬 프로그램에 필수적인 GEN_BLOCK간 재분산 스케줄링 알고리즘을 제안하고 성능 평가를 수행하였다.

GEN_BLOCK간 재분산은 기존 BLOCK 혹은 CYCLIC(N)사이의 재분산과는 달리 메시지 전달에 일정 패턴의 반복성이 발견되지 않는 반면, 연속된 메시지들은 인접한 프로세서로 전달되는 공간 지역성이 존재한다. 본 논문에서는 공간 지역성이 존재하는 집합적 통신 속도를 향상시키기 위해서는 최소 스텝 정리와 최소 크기 정리를 만족해야 함을 제시하고, 이를 수학적으로 증명하였으며, 재배치 스케줄링 알고리즘을 제안하여 이 두 정리를 만족하는 메시지 스케줄을 생성하였다.

제안하는 정리들이 각기 재분산 속도에 미치는 성능 향상에 미치는 영향을 측정하기 위해, GDM 모델을 만족하는 분산 메모리 병렬 컴퓨팅 환경인 CRAY T3E와 그렇지 않은 IBM SP2에서 각각 다양한 조건의 실험을 수행하였다. 이 실험 결과 대상 환경이 GDM 모델에 가까워질수록 최소 스텝 정리와 최소 크기 정리가 재분산 성능향상에 중요함이 증명되었다. 또한 IBM SP2등과 같이 GDM 모델을 만족시키지 못하는 환경에서는 상호 연결망에서의 경합으로 말미암아 메시지 전송의 병렬성을 극대화하는 것이 항상 높은 성능을 보장하지 못함을 발견하였다.

본 논문은 GEN_BLOCK간 재분산에 대한 첫 번째 연구로서 실험과 향후 연구방향에서도 기술했듯이 아직도 성능 개선을 위해 많은 연구를 수행해야 한다. 본 논문에서는 보다 높은 성능의 재분산을 위한 다양한 연구의 필요성을 제시하였으며, 현재 이에 대한 연구가 진행중이다.

참 고 문 헌

[1] High Performance Fortran Forum, *High Performance Fortran Language Specification version 2.0*, Rice University, Houston, Texas, October 1996.

[2] Yeh-Ching Chung, Ching-Hsien Hsu, and Sheng-Wen Bai, "A Basic-Cyclic Calculation Technique for Efficient Dynamic Data Redistribution," IEEE Transaction on Parallel and Distributed Systems, Vol.9, No.4, April 1998.

[3] Young Won Lim, Prashanth B. Bhat, and Viktor K. Prasanna, "Efficient Algorithm for Block-Cyclic Redistribution of Arrays," *IEEE Symposium on Parallel and Distributed Process*, October 1996 and will be published in Algorithmica.

[4] Frederic Desprez, Jack Dongarra, Antoine Petitet, Cyril Randriamaro, and Yves Robert, "Scheduling Block-Cyclic Array Redistribution," *CRPC-TR97714-S*, February 1997.

[5] G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Transaction on Computers*, vol. 34, pp. 943-948, Oct. 1985.

[6] Rajeev Thakur, Alok Choudhary, and Geoffrey Fox, "Runtime Array Redistribution in HPF Programs," *Proceedings of SHPCC'94*, pp.309-316, 1994.

[7] David W. Walker, Steve W. Otto, "Redistribution of Block-Cyclic Data Distribution Using MPI," *Concurrency: Practice and Experience*, Vol.8 No.9, pp.707-728, November 1996.

[8] Rajeev Thakur, Alok Choudhary, and J. Ramanujam, "Efficient Algorithms for Array Redistribution," *IEEE Transactions on Parallel and Distributed Systems*, Vol.7 No.6, June 1996.

[9] S.D. Kaushik, C.-H. Huang, J. Ramanujam, and P. Sadayappan, "Multi-Phase Array Redistribution: Modeling and Evaluation," *Proceedings of 9th Internaltional Parallel Processing Symposium*, pp.441-445, April 1995.

[10] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni, *Interconnection Networks*, IEEE Computer Society Press, pp 155, 1997.

[11] James M. Stichnoth, David O'Hallaron, and Thomas R. Gross, "Generating Communication for Array Statements: Design, Implementation, and Evaluation," *Journal of Parallel Distributed Computing*, pp.150-159, April, 1994.

[12] Edger T. Kalns and Lionel M. Ni, "Processor Mapping Techniques Toward Efficient Data Redistribution," *Proceedings of the 8th Internaltional Parallel Processing Symposium*, April 26-29, 1994, Cancun, Maxico

[13] S.K.S Gupta, S.D. Kaushik, C.-H. Huang, and P. Sadayappan, "Compiling Array Expressions for Efficient Execution on Distributed-Memory Machines", *Technical Report OSU-CISRC-4*.

Appendix I GEN_BLOCK간 재분산 모델[1)]

정의: 메시지 (Message)
(송신 프로세서 번호, 수신 프로세서 번호, 크기)로 이루어진 트리플(triple)이다. 그리고 $sx(m)$, $rx(m)$, $sz(m)$은 각각 메시지 m의 송신 프로세서 번호, 수신 프로세서 번호, 크기를 의미한다.

정의: 재분산 (Redistribution)
GEN_BLOCK 분산 패턴간의 재분산을 위한 메시지들의 집합이다. 재분산 R에 속한 서로 다른 모든 메시지 m과 m'에 대하여,

1) Appendix에서 사용한 각종 기호는 집합론에서 사용하는 기호와 동일한 의미로 사용하였으며, 단 프로세서 번호와 관련하여 프로세서 번호는 일련번호이며, $p = q$는 프로세서 p와 프로세서 q가 같음을 $p > q$는 프로세서 p의 번호가 q의 번호보다 큼을, $p \geq q$는 $p = q$ 또는 $p > q$임을 의미한다.

i) $(sx(m)=sx(m'))\wedge(rx(m)=rx(m'))\Rightarrow(m=m')$
ii) $sx(m)>sx(m')\Rightarrow rx(m)\geq rx(m')$
iii) $rx(m)>rx(m')\Rightarrow sx(m)\geq sx(m')$

정의: 인접 메시지 집합 (Neighbor Message Set)
송신자가 같은 메시지들의 집합 혹은 수신자가 같은 메시지들의 집합이다. 즉, 재분산 R에 속한 서로 다른 모든 메시지 m과 m'에 대하여,

$$(m\in G)\wedge(m'\in G)\Rightarrow(sx(m)=sx(m'))\vee(rx(m)=rx(m'))$$

정의: 통신 스텝 (Communication Step)
MPI_SENDRECV 함수를 사용하여 한번에 수행될 수 있는 메시지들의 집합이다. 이 메시지들은 MPI_SENDRECV의 특징에 의하여 다음과 같은 특징을 갖는다. 통신 스텝 S에 속한 서로 다른 모든 메시지 m과 m'에 대하여,

$$(sx(m)\neq sx(m'))\wedge(rx(m)\neq rx(m'))$$

정의: 스케줄 (Schedule)
재분산을 수행하는 메시지들의 스케줄로, 통신 스텝의 집합으로 이루어지며, 다음과 같은 특징을 갖는다.
첫째, 재분산 R내의 메시지는 하나의 스텝에만 포함된다. 수식으로 표현하면, 스케줄 C안에 포함된 서로다른 모든 스케줄 S와 S'에 대하여

$$S\cap S'=\emptyset$$

둘째, 재분산 R내의 모든 메시지는 스케줄을 구성하는 스텝들 중 적어도 하나에 포함되어야 한다. 수식으로 표현하면, R내의 모든 메시지 m에 대하여

$$m\in \bigcup_{S\in C} S$$

정의: 배타 튜플 (Conflict Tuple)
재분산 메시지 집합의 스케줄 C에 스텝 S와 S'가 있다고 하자. 이때 다음과 같은 메시지 집합 쌍 (M,M')를 배타적 메시지 집합 쌍이라 정의한다.
m과 m'은 각각 S와 S'에 포함되어 있는 메시지이며, S가 선행되는 통신 스텝일 경우 M과 M'은 다음과 같은 메시지들로 구성된다.

$$(\exists m'\in M', (sx(m)=sx(m'))\vee(rx(m)=rx(m'))) \Rightarrow m\in M,$$
$$(\exists m\in M, (sx(m)=sx(m'))\vee(rx(m)=rx(m'))) \Rightarrow m'\in M'$$

정의: 전송 크기 (Communication Size)
메시지 혹은 메시지의 집합인 인접 메시지 집합, 통신 스텝 등은 모두 전송 크기를 가지며, 전송 크기는 다음과 같이 결정된다.
i) 메시지 m의 전송 크기 $sz(m)$: $sz(m)=t_s+m\times t_m$
ii) 인접 메시지 집합 N의 전송 크기 $sz(N)$:
$sz(N)=sz(m)$,
단, m은 N에 속한 메시지중 전송 크기가 가장 큰 메시지
iii) 통신 스텝 S의 전송 크기 $sz(S)$:
$sz(S)=sz(m)$,
단, m은 S에 속한 메시지중 전송 크기가 가장 큰 메시지

Appendix II 최소 스텝 정리

보조정리 1. 재분산 R내의 메시지들로 구성된 모든 인접 메시지 집합 N에 대하여, 임의의 통신 스텝 S에는 N에 속한 메시지가 한 개 포함될 수 있다.
증명.
위 보조정리는 다음 두 가지를 증명하면 증명된다.
i) S에는 N에 속한 메시지가 포함될 수 있다.
N이 아닌 모든 인접 메시지 집합 N'에 대하여 N에 속한 메시지 m과 N'에 속한 메시지 m'가 있을 때, 만일 m과 m'가 다른 메시지라면, 인접 메시지 집합의 정의에 따라 m의 송신자와 m'의 송신자는 서로 다르고, m의 수신자와 m'의 수신자가 서로 다르다. 따라서 통신 스텝의 정의에 의해 m과 m'는 모두 S에 포함될 수 있다. 만일 m과 m'가 서로 같은 메시지라면, 통신 스텝의 정의에 의해 m과 송신자 혹은 수신자가 같은 메시지가 통신 스텝 S에 없으므로 m은 통신 스텝 S에 포함될 수 있다. 따라서 임의의 통신 스텝 S에 대하여 N에 속한 메시지는 최소 한 개가 포함될 수 있다.
ii) S에는 N에 속한 메시지가 두 개 이상 포함될 수 없다.
N에 포함되어 있는 m과 m'가 같은 통신 스텝 S에 포함되어있다고 하자. 인접 메시지 집합의 정의에 의해 m의 송신자와 m'의 송신자가 같거나 혹은 m의 수신자와 m'의 수신자가 같다. 그러나 통신 스텝의 정의에 따르면 m의 송신자와 m'의 송신자는 달라야 하고, m의 수신자와 m'의 수신자 역시 달라야 한다. 이는 서로 모순이며, 따라서 같은 통신 스텝 S에는 같은 그룹의 메시지들은 두 개 이상 포함될 수 없다.

정리. 최소 스텝 정리
재분산을 수행하는 스케줄에서 통신 스텝의 최소 수는 그룹의 최대 랭크와 같다.
증명
보조 정리 1에 따르면 모든 그룹은 임의의 통신 스텝에 단 하나의 메시지 전송이 가능하다. 따라서 랭크가 n인 그룹은 n개의 통신 스텝으로 모든 메시지를 전송할 수 있으며, 결국 재분산을 수행하기 위해서는 그룹의 최대 랭크와 같은 수의 통신 스텝이 필요하다.

Appendix III 최소 크기 정리

보조 정리 2. 통신 스텝 $S1$과 $S1'$에 속한 서로 배타적인 통신 집합 M과 M'이 있다고 하자. 그러면 $S1$과 $S1'$내의 메시지들을 포함하는 다음과 같은 통신 스텝 $S2$와 $S2'$이 존재한다.

$$S2=(S1-M)\cup M'$$
$$S2'=(S1'-M')\cup M$$

증명
통신 스텝과 배타 관계 집합의 정의에 따라 $S1-M$내에 존재하는 메시지들과 M'의 메시지들은 송신자와 수신자가 서로 다르다. 따라서 $(S1-M)\cup M'$내의 메시지들은 통신 스텝이 될 수 있다. 마찬가지로 $(S1'-M')\cup M$내의 메시지들 역시 통신 스텝이 될 수 있다.

정리: 최소 크기 정리
최소 크기를 갖는 스케줄 C가 있다고 하자. C안에 있는 모든 스텝간에 존재하는 배타적 통신 집합 쌍 (M, M')에 대하여

$sz(M) \geq sz(M')$이다.

증명

최소 크기를 갖는 스케줄 C에 존재하는 배타 메시지 집합 쌍 중 $sz(M)<sz(M')$인 쌍 (M, M')이 존재한다고 가정한다. 그러면 위 보조 정리에 의해 M과 M'을 서로 교환할 수 있으며, 교환 한 새로운 스케줄 C'를 고려해보자. 이때 S가 M이 속한 통신 스텝이고, S'가 M'가 속한 통신 스텝이라고 하자.

i) $sz(S) > sz(S')$일 때,

경우1) $sz(S)=sz(M)$일 경우, 위 가정 $sz(M) > sz(M')$에 위배된다.

경우2) $sz(S)>sz(M)$이고 $sz(S')=sz(M')$일 경우, $sz(S)>sz(M')$이므로 $sz(C')=sz(S)+l+E$ 이다. 이때 $l < sz(M')$이므로 $sz(C') < sz(C)$이다.

경우3) $sz(S)>sz(M)$이고, $sz(S')>sz(M')$일 경우, $sz(C')=sz(S)+sz(S')+E$이므로 $sz(C)=sz(C')$이다.

ii) $sz(S) < sz(S')$일 때,

S와 S'를 바꾸어 생각하면 $sz(S) > sz(S')$일 때와 같다.

즉, 모든 경우에 대하여 $sz(C) \geq sz(C')$인 C'가 존재하며, 이는 C가 최소라는 가정에 위배된다. 따라서 최소 크기를 갖는 스케줄 C안에 있는 모든 스텝간에 존재하는 배타 튜플 (M, M')에 대하여 $sz(M) \geq sz(M')$이다.

육 현 규

1994년 고려대학교 전산과학과 학사. 1996년 고려대학교 전산과학과 석사. 1998년 ~ 현재 고려대학교 전산과학과 박사과정. 관심분야는 병렬처리, 분산처리

박 명 순

1975년 서울대학교 전자공학과 학사. 1982년 University of Utah 전기공학과 석사. 1985년 University of Iowa 전기전산공학과 박사. 1975년 ~ 1980년 국방과학연구소 연구원. 1985년 ~ 1987년 Marquette 대학교 조교수. 1987년 ~ 1988년 포항공과대학 전자전기공학과 조교수. 1988년 ~ 현재 고려대학교 컴퓨터학과 교수. 관심분야는 병렬처리, IP QoS, IP Multicast, Internet Computing