병렬처리를 위한 효율적인 사상 기법

김 석 수[†] · 전 문 석[††]

요 약

본 논문은 통신 오버헤드의 정확한 특성을 사용하여 병렬처리를 위한 사상 기법을 표현했다. 목적 함수의 집합은 시스템 그래프에 문제 그래프를 사상하는 것의 최적화를 평가하기 위해서 정형화했다. 이것 중에 하나는 병렬처리의 실시간 응용에 특히 적절하다. 목적함수는 문제 그래프에서 연결선이 가중치를 갖고 있고, 시스템 그래프에서 연결선을 위해 명목거리보다 실제거리를 고용했다는 점에서 기존의 것과 차이가 있다. 이것은 통신 오버헤드를 더 정확하게 측량할 수 있다. 효율적인 사상 기법이 목적함수를 위해서 개발되었고, 초기할당과 쌍 교환의 두 단계에서 최적화 과정이 이루어진다. 이 사상 기법은 시스템 그래프로서 하이퍼큐브를 사용하여 테스트했다.

Efficient Mapping Scheme for Parallel Processing

Seog Soo Kim[†] · Moon Seog Jun[††]

ABSTRACT

This paper presents a mapping scheme for parallel processing using an accurate characterization of the communication overhead. A set of objective functions is formulated to evaluate the optimality of mapping a problem graph into a system graph. One of them is especially suitable for real-time applications of parallel processing. These objective functions are different from the conventional objective functions in that the edges in the problem graph are weighted and the actual distance rather than the nominal distance for the edges in the system graph is employed. This facilitates a more accurate quantification of the communication overhead. An efficient mapping scheme has been developed for the objective functions, where two levels of assignment optimization procedures are employed: initial assignment and pairwise exchange. The mapping scheme has been tested using the hypercube as a system graph.

1. 서 론

일반적으로 스케쥴링 문제는 부타스크(subtask)들의 관계를 기술한 문제 그래프를 병렬구조에서 처리요소의 상호연결을 기술한 시스템 그래프에 사상시키는 것으로 줄일 수 있다.

어떤 제한에 의해서 만들어지는 사상은 응용내용에 따라서 다를 것이다. 다양한 응용에서 공통적인 제한으로 병렬처리에서 특히 중요한 것은 통신 오버헤드를 최소화하는 것이다. 여기에서는 이 제한을 목적함수(objective function)의 형태로 형식적으로 표현할 것이다. 사상 기법은 목적함수에 의존해서 매우 다를 수 있다. 목적 함수는 타스크 하나의 전체 경과시간, 평균 완료시간, 자원의 이용 등을 포함하는 관

† 정 회 원:경원전문대학 전자계산학과 부교수
†† 정 회 원:숭실대학교 컴퓨터학부 부교수
논문접수:1995년 12월 8일, 심사완료:1996년 4월 9일

련된 분야에서 사용된다[4, 14].

보카리(Bokhari)는 시스템 그래프에서 링크에 위치하는 문제 그래프의 연결선의 수를 나타내는 표준화(cardinality)의 목적함수로 사상 기법을 기술했다[5]. 이 표준화는 위치한(matched) 문제 연결선만 고려한 것이다. 그렇지만, 위치하지 않은 연결선들도 시스템 수행능력에 상당한 영향을 갖고 있다는 것을 알 수 있다. 어떤 경우에는 그것들이 시스템의 수행 능력을 결정한다. 그의 기법에서는 모든 문제 연결선이 동등하게 즉, 같은 가중치를 갖는 것으로 생각했다. 그러나 더 일반적인 문제 그래프에서 다른 연결선은 다른 교통밀도(traffic intensity)를 갖기 때문에 각 연결선에 가중치를 가져야만 한다. 목적함수의 다른 예는 제곱할당문제(quadratic assignment problem)에서 발견할 수 있다[8].

위의 간단한 고려로부터 새로운 목적함수를 위한 필요는 특히 병렬처리에서 통신 오버헤드를 정확히 측정할 수 있어야 한다. 그리고 모든 문제 연결선이 고려되고 실제거리들이 명백히 받아들여져야 하며, 주어진 목적함수를 최적화 하는 효율적인 사상 기법이 필요하다.

본 논문에서는 병렬처리를 위한 사상 기법을 제안한다. 모든 문제 연결선에서 임의의 가중치를 고려한 목적함수를 정형화하고, 제안된 목적함수는 매우 일반적이어서 병렬처리의 대부분의 형태에 적용될 것이다. 이 목적함수와 함께 문제 연결선의 통신 오버헤드를 계산하는 기법(scheme)을 기술한다. 그리고, 처리요소 위에 부타스크의 집합을 할당하는 문제를 고려한다. 효율적인 사상 알고리즘을 개발하고, 이것에서 효율적인 해결을 발견하기 위해서 초기할당 기법(initial assignment scheme)과 쌍교환(pairwise exchange)을 기술한다.

본 논문은 다음과 같이 구성된다. 2장에서는 통신 오버헤드와 다른 응용을 위해 목적함수의 세 가지 형태를 정의한다. 3장에서는 병렬 영상처리 모델을 간단히 소개하고, 4장에서는 문제와 시스템 그래프 의 행렬 표현을 기술한다. 5장에서는 목적함수를 계산하는 알고리즘(algorithm)의 내용을 기술하고, 6장에서는 사상의 최적화 기법을 기술한다. 7장에서는 제안된 사상 기법을 하이퍼큐브에 적용한 예와 결과를 보여주고, 8장에서 결론과 앞으로 연구 과제를 기술한다.

2. 목적함수(objective function)

사상 문제는 두 부분으로 구성되어 있는 것을 관찰할 수 있다. 첫째로, 좋은 사상 알고리즘을 개발하기 위해서 최적화를 정확하게 측정할 수 있는 목적함수를 형식화할 필요가 있다. 그리고 목적함수를 효율적으로 최적화하기 위한 사상 기법을 개발하여야 한다. 이 장에서는 통신 오버헤드를 고려해서 적절한 목적함수를 형식화하는 표현 기법을 정의한다.

2.1 통신 오버헤드(Communication Overhead)

문제 그래프 G의 형식적인 기술은 다음과 같다[5].

$$G_p = \langle V_p, E_p \rangle$$
$$V_p = \{ V_{pk} \mid V_{pk} \in G_p \}$$
$$E_p = \{ e_{pij} \mid e_{pij} \in G_p \}$$

V_p는 문제 노드의 집합이고 Ep는 문제 연결선의 집합이다. e_{pij}는 V_{pi}와 V_{pj}사이의 연결선이고, K = 0, 1, ..., N − 1이며 $0 \le i, j < N$이다. 시스템 그래프에 정의된 G_s, V_s, E_s, V_{sk}, e_{sij}는 G_p, V_p, E_p, V_{pk}, e_{pij}에 각각 관련된다. 명확한 기술을 위해서 다음 항목들을 정의한다. 스테이지(stage)는 부타스크를 위한 계산이 실행되는 동안의 시간간격이고, 페이스(phase)는 문제 연결선을 위한 통신이 실행되는 동안 시간간격이며, 스텝(step)은 링크를 통하여 통신되는 시간간격이다. 일반적으로 페이스는 문제 연결선을 실현하기 위하여 몇 개의 링크를 통해서 가야하기 때문에 스텝의 집합으로 구성된다[12]. 일반적으로 몇개의 다른 것들이 선행관계(precedence relationship)때문에 동시에 실행될 수 없는 반면에 어떤 계산(타스크)은 같은 시간에 실행할 수 있다. 따라서 어떤 통신은 같은 페이스를 요구하고, 다른 것은 다른 페이스를 요구한다. 그리고, 문제 연결선 E_p의 집합은 페이스 i에 따라서 부분집합 E_{pi}로 정렬된다. $E_{pi} = \{ \tilde{e}_{pij} \}$ 여기에서 i = 1, 2, ..., Ne, j = 1, 2, ..., L_i이다. 그리고, N_e는 부분집합의 수 즉, 다른 페이스이고, L_i는 부분집합 E_{pi}에서 문제 연결선의 수이다. e_{pij}는 연결선이 요구될 때 페이스에 따라서 정렬된 후에 i번째 부분집합의 j번째 연결선으로 해석된다. e_{pij}에서 j의 값은 특별한 의미를 갖지 않는다. 첨자 j는 i번째 부분집합에서 문제 연결선들을

구분하기 위해서 사용된다. 다른 부분집합으로부터 두 문제 연결선은 통신 오버헤드 증가 없이 시스템에서 링크를 공유한다. 이것은 그것들이 같은 시간에 링크를 필요로 하지 않기 때문이다. 그래서 이것은 시스템 연결선에 사상될 때 독립적으로 취급된다. 목적함수를 정의하기 전에 문제 연결선의 통신 오버헤드를 조사해본다. 첫째, 각 문제 연결선은 어떤 요소에 의해서 가중치를 갖는다. 예를 들면 다른 것보다 많은 빈도로 사용되는 연결선은 더 큰 가중치를 갖는다. 여기에서 가중치는 연결선을 통하는 교통밀도를 나타낸다. 둘째, 시스템 연결선의 명목거리(nominal distance)는 직접 사용할 수 없고, 이것은 D_{kl}로 표시하며, 시스템 노드 V_{sk}와 V_{sl}사이에 시스템 연결선 e_{skl}의 최단 경로이다. 두 시스템 노드 사이에 링크가 있다면 명목거리는 1이고, 없으면 최단 경로에서 링크의 수가된다. 시스템 연결선이 같은 시간에 하나이상의 문제 연결선을 요구하는 상황을 생각해보면, 어떤 문제 연결선의 통신은 시스템 연결선을 위한 다수의 링크가 없다면 지연된다. 그래서 e_{pij}가 e_{skl}에 사상되는 곳에서 e_{pij}의 통신 오버헤드는 명목거리 D_{kl}과 다르게 된다. 주어진 문제와 시스템 그래프에서는 통신 오버헤드가 특별한 사상에 의존되고, 일반적으로 시스템의 통신 제어규칙에 의존된다. 그러므로 통신 오버헤드를 계산하는 정교한 기법이 필요하다. 또한 각 문제 연결선의 실제 통신 오버헤드를 전반적인 통신 오버헤드의 정확한 측정을 얻기 위해서 사용된다.

2.2 목적 함수

사상의 최적화는 다른 응용에서 다른 기법으로 측정된다. 다시 말하면, 적절한 목적함수는 통신 오버헤드의 더 정확한 측정을 얻기 위해서 각 응용에 적용된다. 여기에서는 세 가지의 목적함수를 정의한다[12]. 첫째, 두개의 문제연결선이 같은 페이스에서 필요하지 않는 경우를 가정한다. 즉, 모든 i에 대하여 $L_i=1$이다. 사실 이것은 병렬 처리의 특별한 경우로서 고려될 수 있는 순차처리의 일종이다.

이경우의 목적함수는 아래와 같이 정의된다.

$$OF_1 \triangleq \sum_{ij} \tilde{c}_{ij} = \sum_{i} \tilde{c}_{i1} \qquad (1)$$

$\tilde{c}_{ij}$는 부분집합 E_{pi}에 속하는 정렬된 연결선 $\tilde{e}_{pij}$의 통신 오버헤드이다. 즉, 목적함수 OF_1은 모든 문제 연결선의 통신 오버헤드의 합이다. 이것은 위치나 할당문제(placement or assignment problem)에서 하나의 측정으로서 받아들여진다. 그러나, 이것은 병렬처리를 위해서는 적절하지 않다.

알고리즘1)

(MARRER : 쌍교환 알고리즘)

입력 : 문제 그래프와 시스템 그래프의 인접행렬

출력 : 쌍교환으로 얻어진 가장 좋은 표준화 값

Begin

문제 그래프와 시스템 그래프의 인접행렬을 입력

repeat

For 각노드 **do**

1) 모든 노드와 임의로 쌍교환을 한 후 표준화를 계산

2) 사상의 표준화에서 가장 크게 얻을 수 있는 것을 선택

3) **if** 가장 큰값 > 기존의 값 **then** 해당되는 쌍을 교환한다.

enddo

until 가장 큰 표준화 값을 얻을 때

end

둘째, 모든 문제 연결선이 동시에 요구되면 즉, 같은 페이스이면 가장 큰 $\tilde{c}_{i1}$가 시스템 수행능력을 결정한다. 이것은 $N_e=1$인 경우와 관계된다. 목적함수 OF의 다른 형태는 다음과 같다.

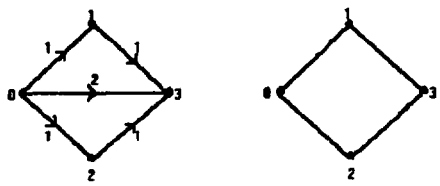
$$OF_2 \triangleq \max_j(\tilde{c}_{ij}) = \max_j(\tilde{c}_{i1}) \qquad (2)$$

어떤 경우에서는 OF_1이 증가되는 반면 OF_2는 감소한다. 그래서, 각 응용을 위해서 적절한 OF를 사용할 필요가 있다. 셋째, OF의 일반적인 형태는 위의 두개를 조합하는 것이다. 즉, 같은 부분집합(E_{pi})에서 연결선은 동시에 요구되고 그 부분집합은 순차적으로 요구되는 경우이다. 다시 말하면, 임의의 i에 대하여 $N_e \geq 1$이고, $L_i \geq 1$인 경우이다. 각 부분집합의 통신 오버헤드는 부분집합에서 가장 큰 $\tilde{c}_{ij}$에 의해서 표현되고, 전반적인 통신 오버헤드는 모든 부분집합의 통신 오버헤드의 합이 된다. 이것은 모든 병렬처리가 같은 시

간에 출발되는 것을 전제로 한다. 이것은 다음과 같다.

$$OF_3 \triangleq \sum_i (\max_j(\tilde{c}_{ij})) \quad (3)$$

관점을 명확히 하기 위해서 그림 1에 있는 문제 그래프와 시스템 그래프를 고려한다.



(그림 1) (a) 문제그래프 (b) 시스템 그래프
(Fig. 1) (a) Problem graph (b) System graph

문제 그래프에서 각 문제 연결선의 가중치는 이동되는 메시지의 수를 표시한다. 모든 통신 단계는 동기적이고, 문제 연결선을 위한 모든 메시지는 같은 물리적 경로를 지나가고, 모든 문제 연결선은 같은 시간에 요구된다. 문제 연결선의 통신 오버헤드는 문제 연결선의 가중치에 의해서 표시된 수의 메시지를 옮기는데 요구되는 스텝의 수이다. 예를들면, 문제 연결선의 가중치가 다른 문제 연결선에 의해서 공유되지 않은 링크에 사상되면 1이고, 문제 연결선의 통신 오버헤드도 1이다. 가중치가 2이면, 문제연결선이 두 링크를 가로질러서 사용된다. 그때 통신 오버헤드는 2가된다.

예 1.1:

$(n_1, n_2, ..., n_N)$은 문제 노드 0, 1, ..., N−1이 시스템 노드 $n_1, n_2, ..., n_N$에 각각 할당되는 사상을 표현한다($0 \le n_i < N$). 여기서 표준화를 받아들인다면, 사상(0 1 2 3)은 문제 연결선이 4개 연결되었기 때문에 최대 표준화는 4가 된다. OF_2의 최소 얻을 수있는 값은 e_{p03}의 통신 오버헤드가 좋을 때 3이기 때문에 3이 된다. 이것은 모든 시스템 연결선의 명목거리가 2보다 크지 않기 때문이다. 여기서 (0 1 2 3)은 사상하면 표준화는 3이다(e_{p01}, e_{p03}, e_{p23}이 연결된다). 그러나 OF_2는 2가 되는 것을 쉽게 알 수 있다. 통신 오버헤드를 더 자세히 특성화하기 위해서 모든 문제 연결선을 고려해야하고, 명목거리보다 실제거리를 고려할 필요가 있다.

3. 병렬 영상처리 모델과 목적함수

이 장에서는 병렬 영상처리 모델을 소개한다. 그리고 이 모델에 기초한 목적함수를 정의한다.

3.1 병렬 영상처리 모델

병렬 영상처리 모델을 정규화하기 위해서 영상처리 타스크의 간단한 표현을 다음과 같이하자.

$$O = f(I) \quad (4)$$

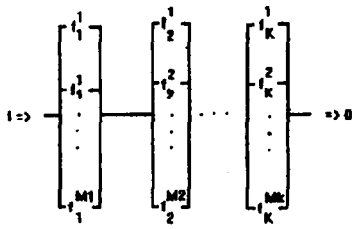
f는 영상처리 함수, I는 입력자료, O는 출력 결과이다. 예를 들면, f는 히스토그램 계산, 로우패스 필터링(low-pass filtering), 연결선 축출(edge detection), 객체 인지(object recognition)등이고, 이때 O는 히스토그램, 좋은 영상(smoothed image), 연결선 지도(edge map), 연결된 모델(matched model)등이다. 디지탈 영상처리는 병렬처리를 위해서 이용될 수 있는 몇 가지 특성을 갖고 있다[15]. 첫째로, 동등한 연산이 영상을 통해서 반복된다. 히스토그램 계산에서 픽셀과 픽셀로, 메디안 필터링(median filtering)에서는 지역과 지역에서 반복적으로 연산이 된다. 다른 말로 하면, 픽셀이나 부분지역을 위한 결과를 얻기 위해서는 단지 지역자료(local data)만 필요하다. 이 공간적인 특성(spatial characteristic)은 영상을 처리요소들의 집합에 의해서 병행적으로 처리할 수 있는 부분 영상의 집합으로 나눌 수 있는 것이다[9]. 둘째로, 몇 개의 처리 함수들이 영상을 순차적으로 적용할 수 있다. 예를 들면, 객체를 인지하기 위한 입력영상의 확장(enhancement), 객체 축출을 위한 세그먼트화, 정규화(normalized), 그리고 모델과 연결되기 위해서 전처리 된다. 연속되는 영상은 동적 풍경 분석(dynamic scene analysis)과 같은 실시간 응용에서 특별히 처리되어야 한다[7]. 임시적인 특성(temporal characteristics)은 처리율을 증가시키기 위해서 파이프라인 환경의 사용을 제안한다. 이 기법에서 전반적인 처리 함수는 파이프라인으로 구성되는 처리 부분함수로 재구성된다. 전자는 공간

병렬성(spatial parallelism)의 형성이라고 불리고, 후자는 임시적 병렬성(temporal parallelism)의 형성이라 한다. 병렬 영상처리의 기본 생각은 병렬성을 이용할 수 있도록 f와 I를 분할(partition)하여 부타스크를 병렬로 실행하는 것이다. 즉, 타스크는 공간적인 것과 임시적인 차원으로 분할을 한다. 영상처리 함수는 전체 영상 도메인에 순차적으로 적용되는 몇 개의 부함수로 재구성할 수 있고(파이프라인), 각 스테이지(부함수)를 위한 입력자료는 영상 병렬성이 다중처리에서 이용될 수 있을 때 병렬 실행을 위해서 분할된다. 영상처리 타스크는 다음과 같이 표현할 수 있다.

$$\begin{aligned} O &= f(I) \\ &= C_i f_i(I) \\ &= C_i(U f_i^j(I_{ij})) \end{aligned} \tag{5}$$

f_i^j와 I_{ij}는 처리 함수와 파이프라인의 i번째 단계의 다중처리의 j번째 분할의 입력자료이다.

$(1 \le i \le k,\ 1 \le j \le M_i)$, C와 U는 구성(composition)(파이프라인)과 유니온(union)(다중처리)이다. (5)에 기초한 병렬 영상처리를 위한 가능한 모델은 K 파이프라인 단계를 구성하고 각각은 다중처리 된다(그림 2를 참조). M_i는 i단계에서 입력자료의 분할수(또는 사용된 처리요소의 수)이다.



(그림 2) 병렬 영상 처리모델
(Fig. 2) Parallel image processing model

3.2 목적함수

병렬 영상처리 모델에서 통신 오버헤드를 고려해보자. 한 단계에서 부함수(f_{ij})의 계산은 병행적으로 (concurruntly)실행되는 것을 가정한다. 우선, 한 단계에서 다음 단계로 결과를 이동하는 단계들 사이에는 통신이 필요하다. 또한 각 단계에서 처리요소들은 계산하는 동안 서로서로 자료를 교환해야만 할 것이다. 문제 그래프로 이 모델이 표현되고, 연속적인 단계의 각 쌍과 각 단계에서 하나 사이에는 연결선의 집합이 있다. 단지 하나의 영상 프레임만 처리한다면 그림 2에서 보여지는 것처럼 선행관계로 인해서 동시적으로 요구되는 연결선의 집합은 없다. 각 집합 안에 연결선들의 통신이 같은 시간에 실행할 수 있으면, 그것은 OF_3에서 결과를 만들 수 있다. 그러므로 많은 영상 프레임을 연속적으로 처리하는 경우에는 연속적인 단계의 모든 쌍 사이에 연결선의 set(X)와 모든 단계 안에 연결선의 set(Y)는 파이프라인이 실행되면 번갈아서 작동된다. 즉, X, Y, X, Y, ...와 같이된다. 병렬 처리모델에서 대부분의 경우 $N_e = 2$이다. 처리율 즉, 시스템에서 연속된 영상을 처리할 수 있는 속도는 영상처리 시스템의 중요한 수행능력 측정치이다. 병렬처리 모델의 목적함수 OF를 다음과 같이 정의한다.

$$OF_4 \triangleq \max_i(\max_j(\bar{c}_{ij})) \tag{6}$$

이 목적함수는 OF_2에서 유추하였을지라도 OF_2와는 구별된다. 모든 문제 연결선이 OF_2의 경우에는 같은 시간에 요구되나 OF_4의 경우에는 그렇지 않다. OF_4는 많은 자료 집합이 순차적으로 처리되는 파이프라인 다중모듈(multimodule) 시스템에 고용된다. 이 형태의 병렬처리는 실시간 처리 타스크를 처리한다.

예 1.2:

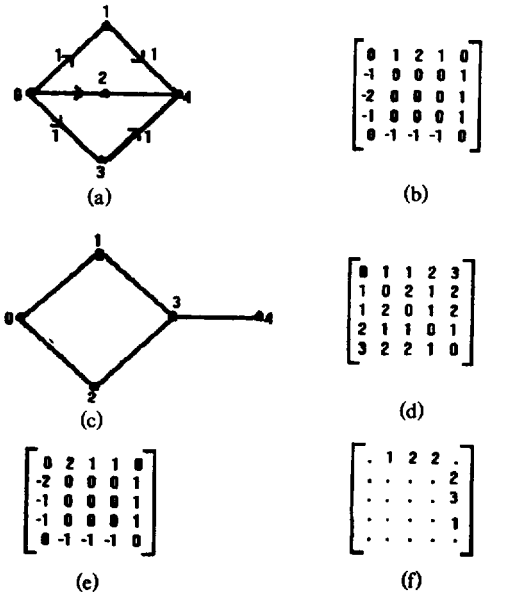
그림 1에서 보여준 예로 돌아가서, $E_{P1} = \{e_{P01}\ e_{P02}\ e_{P13}\ e_{P23}\}$과 $E_{P2} = \{e_{P03}\}$를 가정하자. 문제 연결선 e_{P03}이 E_{P1}에서 어떤 연결선에 대해서 페이스와이즈(phase-wise) 독립이기 때문에 사상(0 1 2 3)과 (0 1 3 2)는 OF_4에서 2가 된다. 2.2절에서 사상(0 1 2 3)은 OF_2에서 3이고, (0 1 3 2)는 2가 되었다.

따라서, 문제 연결선의 페이스는 통신 오버헤드를 계산하는데 고려되어져야 함을 알 수 있다.

4. 보조적인 행렬(Auxiliary Matrices)

사상 문제에 들어가기 전에 다음 장에서 사용될 네

개의 행렬을 정의한다. 그림3에 예시처럼 모든 문제 연결선은 동시에 활성화되는 것을 가정한다. 문제 행렬 P는 문제 그래프를 표시하는 정방행렬이다. 요소 (i, j)는 α_{ij}에 의해서 표시된다. α_{ij}의 크기는 문제 연결선 e_{pij}의 가중치이고 부호는 통신의 방향을 표시한다. α_{ij}가 양수면 v_{pi}에서 v_{pj}로 통신이 흐르고, 음수면 v_{pj}에서 v_{pi}로 흐른다. v_{pi}와 v_{pj}사이에 어떤 방향에서도 통신이 없다면 α_{ij}는 영이 된다. 문제 그래프와 그것의 문제 행렬은 그림 3(a)와 (b)에 예시되었다. 명목거리 행렬 D는 시스템 그래프를 표현하는 정방행렬이다. 요소(i, j)는 문제 연결선 e_{sij} 즉 시스템노드 v_{si}와 v_{sj}사이에 명목거리 D_{ij} 이다. 이것은 그림 3(c)와 (d)에 예시되었다. 할당행렬 A는 특별한 사상을 기술하는 보조행렬이다. 이 할당 행렬은 사상에 따라서 문제 행렬의 열과 행을 교환함으로서 얻을 수 있다. 예를 들면, (0 2 1 3 4)를 사상한다면, 두 번째와 세 번째 열을 교환하고, 두 번째와 세 번째 행을 교환한다. 그림 3 (e)가 이것을 보여준다. 통신 오버헤드 행렬 C는 명목거리와 할당 행렬로부터 얻는다. 요소(i, j)는 특별한



(그림 3) (a) 문제 그래프 (b) 문제 행렬 (c) 시스템 그래프 (d) 명목거리 행렬 (e) 할당 행렬 (f) 통신 오버헤드 행렬

(Fig. 3) (a) Program graph (b) Program matrix (c) System graph (d) Nominal distance matrix (e) Assignment matrix (f) Communication overhead matrix

사상을 위해서 v_{pi}로부터 v_{pj}까지 방향으로 문제 연결선 e_{pij}의 통신 오버헤드를 나타낸다. 음수의 α_{ij}의 문제 연결선 e_{pij}는 그것이 중복되기 때문에 c_{ij}를 정의하지 않는다. 사상 (0 2 1 3 4)를 위한 통신 오버헤드 행렬은 그림 3(f)에 예시되었고, 점(dot)은 관련된 문제 연결선이 사용되지않은것을 나타낸다. c_{ij}를 계산하는 상세한 절차는 다음 장에서 기술한다.

5. 목적함수의 평가

5.1 동기식 통신(Synchronous Communication)

문제 그래프와 시스템 그래프가 주어졌을 때 특별한 사상을 위한 목적함수 OF를 계산하는 과정을 기술한다. 통신의 동기화모드가 사용된 것 즉, 모든 링크를 위한 스텝은 동기화(synchronized)된 것을 가정한다. 다음과정에서 문제 연결선 $\tilde{e}_{pij}$는 시스템 연결선 e_{skl}에 사상된 것을 가정한다.

알고리즘 2)

(동기식통신에서 목적함수 계산을 위한 알고리즘)

입력 : 연결되고,방향이 있는 문제 그래프 P의 인접행렬, 연결되고,비방향인 시스템 그래프 S의 명목거리 행렬

출력 : 문제 그래프 P에서 얻은 할당 행렬 A와 명목거리 행렬 D를 이용한 통신 오버헤드 행렬 C_{ij}

OF_cal_sync(P : 문제그래프; M : 사상)

begin

1) 문제 그래프의 인접행렬로 부터 할당 행렬 A를 만든다.

2) 문제 연결선이 활성화될 때 페이스(phase)에 따라서 부분집합 E_{pi}에 E_p(양수 α_{ij}를 갖는 문제 연결선만)를 정렬한다.

3) **For** 각 E_{pi} **do**

3.1) **For** $\tilde{e}_{pij}$ **do**

D에 A를 얹어놓아서 관련된 시스템 연결선 (eskl)의 명목거리를 찾는다.

3.2) **If** e_{skl}에 관련된 $\tilde{e}_{pij} > 1$ **then**

$\tilde{e}_{pij}$를 위한 경로 e_{skl}을 연속된 링크(동등한 노드)로 재구성한다.

(재구성 기법은 병렬처리 구조의 위상과 라우팅 규칙에 의존된다.)

enddo

3.3) **For** 각 $\tilde{e}_{pij}$ **do**

3.3.1) v_{sk}에 패킷을 할당한다.(패킷의수는 $\tilde{e}_{pij}$의 가중치인 α_{ij}와 같다.)

3.3.2) 각 패킷에 출발지(SO), 목적지(DT), 스텝(ST)등을 결합한다. v_{sk}와 v_{sl}은 출발지와 목적지이고, 스텝(초기에는 0)은 패킷이 현 시스템노드에 도착하는 스텝을 표시한다.

enddo

3.4) **Repeat**

3.4.1) ST = ST + 1

3.4.2) 패킷을 다음 노드로 옮긴다.

(패킷의 수 또는 옮길 수 있는 패킷은 특별한 시스템의 구현 내용에 의해서 결정된다)

3.4.3) **If** 패킷이 목적지에 도달

then 패킷을 제거한다.

If 출발지로부터 마지막패킷

then $c_{ij} = v_{sk}$ 부터 v_{sl} 까지 마지막 패킷의 스텝수

else 현노드의 큐에 패킷을 입력한다.

until 모든 패킷이 그들의 목적지에 도달

enddo

4) OF를 계산한다.

end

위 알고리즘 2에서 패킷은 각 문제 연결선의 통신 오버헤드를 계산하기 위한 보조적인 자료구조이다. 대부분 경우에서 문제 연결선의 재구성을 하는 3.2)는 주어진 시스템의 라우팅 규칙에 의해서 프라이어리(priori)를 결정한다. 라우팅 규칙을 사용자에 의해서 제어할 수 있다면 더적은 OF를 얻기 위해서 재구성을 최적화하기 위한 여지가 있다. 3.4.2)를 위한 패킷의 선택은 특별한 시스템의 라우팅 기법에 의존된다. 합리적인 선택은 한번에 하나의 패킷이고, 그 패킷은 가장 먼 목적지에 배당된다. 패킷의수 3.3.1)와 패킷의증가인 3.4.1)은 시스템에 따라서 조정될 것이다.

정 리 1:

위 알고리즘에 의해서 걸리는 시간의 대부분은 3.4)의 반복에 의해서 결정된다. 반복되는 횟수는 특별한 할당과 라우팅 규칙에 의해서 아주 유동적이다. 그렇지만 그것의 최대치는 $ND_M\alpha_M$이다. N은 문제 노드의 수이고 D_M은 시스템 그래프의 직경(diameter) (즉, 최대명목 거리), 이며 α_M은 문제 그래프에서 가중치의 최대 값이다. 대부분의 경우 반복 횟수는 최대치(upper bound)보다 매우 적다.

증 명:

N = 1 일 때는 문제 그래프의 가중치 α_M가 0 이기 때문에 전체 값이 0 이 된다. N > 1 일 때 귀납법에 의해서 증명한다. 반복횟수가 최대치보다 크다고 가정하자 문제 노드의 수가 1 보다 크고, 가중치가 모두 1 이면서 시스템 그래프의 최대명목 거리가 1 이라고 하면 모든 노드가 시스템 노드에 할당되어 연결선을 통해서 통신될 때 최악의 경우로 한노드에서 모두 대기상태가된다. 이때 반복횟수는 N − 1 이된다. 따라서 N > N − 1 이기 때문에 반복횟수는 최대치보다 클 수가 없다. 가중치가 1 보다 크고, 시스템의 최대명목 거리가 1 이상일 때 위내용으로 부터 반복 횟수가 최대치 $ND_M\alpha_M$ 보다 클 수가 없다.

5.2 비동기 통신(Asynchronous Communication)

문제 연결선을 위한 통신이 임의의 시간에 발생된다면, 즉, 통신의 비동기 형태인 경우 위의 내용은 OF의 정확한 평가를 할 수 없다. 이것은 모든 부타스크의 계산시간이 같지 않을 때와 각 처리요소에서 결과가 이용되는 경우에 그 결과를 이동시킬 때 발생한다. 비동기 통신 형태에는 두 가지 고려사항이 있다. 첫째는 부타스크의 계산 시간이 일정하나(자료집합에 독립) 다른 부타스크에서는 다르다. 부타스크의 우선순위관계나 유한 버퍼 때문에 스테이지의 초기에 동기화는 합리적인 구현이다. (반면에 통신(페이스와 스텝)은 비동기이다) 이때, 스테이지와 페이스로 구성되는 모든 시간 간격에서 동일한 통신 형태가 반복된다. 어떤 페이스에서는 동기화되고, 다른 곳에서는 그렇지 않다. 동기화된 페이스의 수가 비동기화된 페이스의 수보다 매우 더 크면 앞에서 기술한 평가절차를 사용한다. 대부분의 페이스와 스텝이 다른 경우(instance)에서 출발하면, 다른 문제 연결선의 통

신 사이에 충돌을 무시할 수 있는 합리적인 근접이 일어나야 한다. 이런 경우 시스템 연결선의 명목거리를 사용하여 처리하는 과정이 아래에 기술되었다. 이것이 OF의 정확한 값을 계산할 수 없을지라도 응용을 위해서 충분히 정확한 평가를 할 수 있을 것이다. 비동기 통신 형태에서 고려되어야할 다른 경우는 모든 부타스크를 위해서 그것의 평균값이 동일하게 조정될지라도 자료 집합에서 자료 집합까지 각 부타스크의 계산 시간이 변화하는 경우이다. 즉, 각 부타스크의 계산시간은 임의의 변수(random variable)이고, 페이스도 임의의 것이 된다.

알고리즘 3)

(비동기식 통신에서 목적함수 계산을 위한 알고리즘)

입력: 연결되고, 방향이 있는 문제 그래프 **P**의 인접행렬, 연결되고, 비방향인 시스템 그래프 **S**의 명목거리 행렬

출력: 문제 그래프 **P**에서 얻은 할당 행렬 **A**와 명목거리 행렬 **D**를 이용한 통신 오버헤드 행렬 C_{ij}

OF_cal_async(P: 문제그래프; **M:** 사상)

begin

1) 문제 그래프의 인접행렬로 부터 할당 행렬 A를 만든다.

2) 문제 연결선이 활성화될 때 페이스에 따라서 부분집합 E_{pi}에 E_p를 정렬한다.

3) **For** 각 E_{pi} **do**

3.1) **For** $\tilde{e}_{pij}$ **do**

3.1.1) D에 A를 얹어놓고 e_{skl}의 명목거리(D_{kl})을 찾는다.

3.1.2) 통신 오버헤드 $\tilde{c}_{ij}$는 $\tilde{\alpha}_{ij} D_{kl}$이다.

enddo

enddo

4) OF 계산한다.

end

정 리 2:

위알고리즘 3에서 정렬은 어떤 문제연결선이 다른 것과 구별할 수 있을 때만 실행한다. 그래서 그것을 다른 집합으로 그룹화 한다. 3.1.1)과 3.1.2)는 N 시간에 실행된다.

증 명:

할당행열의 크기가 N×N이고, 명목거리 행렬의 크기도 N×N이다.

D를 A에 얹어 놓고 0이 아닌 요소 즉,문제 노드의 갯수에 해당하는 명목거리를 순차적으로 찾는 시간 복잡도는 O(N)이 된다.

6. 사상의 최적화

앞장에서는 특별한 사상을 위한 통신 오버헤드를 계산하기 위한 기법을 기술했다. 이것은 대부분의 병렬처리 구조에 적용될 수 있는 것이다. 이제는 시스템 그래프에 문제 그래프를 사상하는데 있어서 최적화를 논의한다. 앞에서도 기술했듯이, 최적화는 두개의 다른 단계에서 가능하다. 첫째는 나중에 수행되는 최적화 과정에서 많은 시간을 줄일 수 있도록 좋은 초기할당을 성취하는 것이다. 둘째는 사상을 증진시키기 위해서 변화된 쌍교환 기법을 사용하는 것이다. 두 과정을 함께 적용하므로 해서 효율적이고 좋은 사상을 이룰 수 있다. 다른 말로 하면, 초기할당에서 최적이고 만족스러운 해결을 발견하는 것이 긴급한 것이 아니고, 이 단계에서 알고리즘도 매우 복잡한 것은 아니다. 또한 합리적이고 좋은 초기할당을 만들면 소모적인 쌍교환 기법를 하지 않고 쉽게 해를 구할 수 있기 때문에 두 과정은 서로 보수적인(complement) 관계이다.

6.1 초기할당(initial assignment)

이 과정의 목적은 임의의 초기할당에 의존하는 대신에 가능한 작은 OF를 얻을 수 있도록 노력하는 것이다. 이 과정에서 얻어진 사상이 주어진 기준에 최적화 되지 않고 만족스럽지 않다면 다음에 기술할 최적화기법의 단계를 처리한다. 문제 노드 v_{pi}의 통신밀도(communication intensity)는 $H_{pi} = \sum_j |\alpha_{ij}|$로 정의한다. 노드 v의 차수는 d(v)로 나타낸다. 할당되기 위한 문제 노드의 선택은 통신 밀도와 연결성에 기초를 두고, 관련된 시스템 노드의 선택은 OF로 부터 유출된 어떤 측정치에 의해서 결정된다.

알고리즘 4)

(초기할당 알고리즘)

입력:문제 그래프 P의 인접행렬(가중치로 표현)
출력:적절한 OF를 갖는 할당 행렬
Init_assign (P:문제그래프;S:시스템그래프)
begin
1) V_p로부터 가장 큰 통신밀도를 갖는 문제 노드 v_{pi}를 찾는다. 같은 것이 있는 경우는 임의로 하나를 선택한다.
2) 시스템 노드 v_{sj}에 v_{pi}의 할당은 가능한 한 $d(v_{pi})$와 $d(v_{sj})$가 같은 곳에 한다.
3) V_p로부터 V_{pi}를 제거한다.
4) **while** $|V_p|>0$ **do**
4.1) 이미 할당된 문제 노드에 인접된 문제 노드로부터 가장 큰 통신 밀도를 갖는 문제 노드 v_{pk}를 찾는다.
4.2) 최소화(또는 최대화)된 OF로 부터 유추된 어떤 측정치에 의해서 시스템 노드 v_{sl}에 v_{pk}를 할당한다.
4.3) $V_p - v_{pk}$
enddo
end

위 알고리즘 4의 4.2)에서 OF_2 또는 OF_4가 받아들여진다면, 하나의 가능한 측정치는 v_{sl}과 v_{sm}사이에 명목거리의 최대값이 된다. v_{sm}은 문제노드 v_{pn}가 이미 할당된 시스템 노드이고, v_{pk}는 v_{pn}에 인접되어 있다. 여기서 최대값을 최소화 시키는 v_{sl}을 선택한다. OF_1이 사용되면 그 측정되는 최대값 보다도 명목거리의 합이 된다.

정 리 3:
초기할당 과정의 시간 복잡도는$(N-1)$번 반복되는 4)과정에 의해서 결정된다. 4.1)과 4.2)과정은 N개 노드보다 더 적은 경우를 요구한다. 따라서 이과정의 실행은 $O(N^2)$이다.

증 명:
$V_p=1$ 일 때 4.1)과정은 O(1)이다.
$V_p>1$ 때 4.1)과정은 V_p-1 번 수행된다. $N>|V_p|-1$이기 때문에 4)과정은 $O(N^2)$ 이다.

6.2 쌍교환
초기할당이 최적의 해결을 보장하지 않기 때문에 현재 사상에서 두 문제 노드를 교환해서 사상을 증진시키고, 더 좋은 사상을 만들 수 있도록 반복적으로 검토한다. 제안된 쌍교환 기법은 모든 쌍을 검토하는 간단한 쌍교환 기법과 다르다. 어떤 기준에 도달될 때까지 선택된 쌍을 조사한다. 하지만 최악의 경우에는 선택된 쌍이 모든 쌍이 될 수도 있다. 이 기법은 불필요한 반복을 피하도록 하는 것이다. 이것은 매우 간단한 기법이지만 초기할당 기법이 크고 불필요한 탐색공간을 제거했기 때문에 효율적이고 합리적인 좋은 해를 기대한다. 물론 이것은 아직까지 지역적인 최적화이다. 그러므로 해가 만족스럽지 않다면 잠재적인 지역 최적화를 제거하는 다른 초기할당을 시도한다. 쌍교환 기법을 포함하는 사상의 전체과정을 위한 구현은 다음과 같다.

알고리즘 5)
(쌍교환 알고리즘)
입력:문제 그래프 P로 부터 초기할당 알고리즘을 수행해서 얻어진 초기할당 행렬
출력:OF 값을 평가하여 만족스러운 사상이 되는 할당 행렬
Mod_pair_xchange(P:문제그래프;S:시스템그래프)
begin
1) 초기할당을 만든다.
2) OF를 평가한다.
3) **Repeat**
3.1) OF로 부터 유추된 어떤 특정 치에 따라서 교환하기 위한 문제노드 v_{pi} 후보를 찾는다.
3.2) **For** 각 문제노드(v_{pj}) **do**
3.2.1) v_{pi}와 v_{pj}를 임시로 교환한다.
3.2.2) OF를 평가한다.
enddo
3.3) 가장 작은 OF를 갖는 문제 노드(v_{pJ})를 결정한다.
3.4) 새로운 OF가 구 OF보다 더 크지 않다면 v_{pi}와 v_{pJ}를 교환한다.
until 만족된 사상
end

쌍교환 순환 3)은 받아들일 수 있는 사상이 얻어지거나 반복되는 순환이 종료되었을 때 완료된다. 3.1)

의 측정치는 적용한 OF에서 정규화 된다. OF_i이 고용되면, 합리적인 측정치는 문제 노드에 연결된 문제 연결선의 통신 오버헤드의 합이 된다. 그리고 그 후보는 가장 큰 합이 되는 문제 노드이다. OF_2와 OF_4가 고용되면 가장 큰 통신 오버헤드의 문제 연결선이 연결된 문제 노드의 하나가 후보가 될 수 있다. 이유는 OF가 문제 연결선의 통신 오버헤드의 최대값에 의해서만 결정되기 때문이다. 이것은 감소시킬 필요가 있는 것이다. 전체사상 과정의 시간 복잡도는 초기할당이 받아들일 수 있는 해가 되지 않는다면 3)에 의해서 결정된다.

정 리 4:

N_i는 쌍교환 순환의 반복횟수이다. 이때 전체 과정은 $O(N_iN^2)$이 될 것이다. 이것은 3.1)과정이 D_M과 α_M이 N과 독립적인 것을 가정한 O(N)의 OF 평가를 포함해서 O(N)반복을 요구하기 때문이다. N_i는 문제와 시스템 그래프, 그리고 해의 수납성(acceptability)을 위한 기준에 따라서 매우 방대할 것이다.

증 명:

정리 3에 의해서 반복되는 과정은 $O(N^2)$ 이다.

여기서 쌍교환하는 횟수 N는 $O(N_i)$이기 때문에 전체과정은 $O(N_iN^2)$이된다.

7. 하이퍼큐브 구조에 적용

제안된 사상 기법은 하이퍼큐브를 사용하여 예시하였다. 하이퍼큐브의 사용은 시스템 그래프에서 최단경로의 평가가 간단하다. 본 논문에서는 최하위 비트의 보수를 이용하여 최단경로를 구한다. 이번 장에서는 하이퍼큐브의 특성[4, 9]과 시험결과를 제시하였다.

7.1 하이퍼큐브의 특성

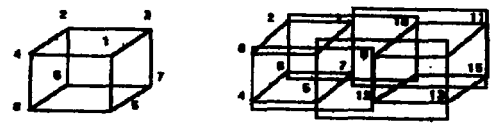
전체 사상 알고리즘은 초기할당과 쌍교환으로 구성된다. 또한 OF의 평가에 대한 구현이 포함된다.

일반적으로 그래프에서 두 노드사이에 최단경로를 찾는 가장 잘 알려진 알고리즘은 N이 노드의 수인 경우 $O(N^2)$의 탐색과정이 요구된다. 최단경로의 라우팅 정보는 경로를 따라서 모든 노드에 저장되어야 한다. 그러므로 하이퍼큐브 같은 정규 구조의 그래프로 병렬 구조를 표현할 수 있다면 위 문제는 상당히 간단해진다.

다음은 하이퍼큐브의 특성을 기술한 것이다. 그림 4는 N=8과 N=16인 하이퍼큐브구조를 보여준다. 노드 R의 주소표현은 이진수로 $(r_{n-1}, ..., r_1 r_0)$이고 노드 S는 $(s_{n-1}, ..., s_1 s_0)$이며, $N=2^n$이다.

1) R과 S는 하나의 i에 대하여 $r_i \neq s_i$면 서로 인접되어있다.
2) i의 구별된 값 L에 대하여 $r_i \neq s_i$면 $I(R, S)=L$, 여기서 I(,)는 두 노드 사이에 최단거리를 나타낸다.
3) $I(R, S)=L$이면 R과 S사이에 길이 L의 다른 최단경로는 L! 이다.

R과 S사이에 L 비트가 다르다고 가정하자. R부터 S로(또는 S부터 R로) 가기 위해서, R(또는 S)이 L비트 보수를 가져야만 한다. 그러므로 특성 1)로 인해서 한번에 한 비트만 보수화할 수 있기 때문에 거리는 L이다. 특성 2)도 이 경우이다. 우선 L 비트 가운데 임의의 하나가 보수화 되고, 나머지는 L−1비트 중에 또 보수화 되는 과정을 거친다. 특성 3)도 마찬가지다. 위의 내용으로부터 하이퍼큐브구조 에서 최단 경로를 찾는 것은 탐색 과정이 필요치 않음이 분명하다. 출발지와 목적지가 주어지면 중간 노드(즉, 경로)는 간단한 기법으로 얻는다. 특성 iii)에서는 하나의 메시지를 라우트 할 수 있는 경우가 L! 있음을 기술한다. 가능한 최단 경로의 집합으로 부터 긴 경로를 선택하는 것은 특별한 시스템의 구현 내용에 의존된다. 이 테스트에서 최하위 비트가 우선 보수화 된다. 이것은 또한 메시지가 필요한 링크를 따라서 이동될 수 있는 것을 가정한다. 다른말로 하면 한 노드의 세링크가 사용중(busy)이면, 같은 시간에 세개의 다른 이



(a) (b)

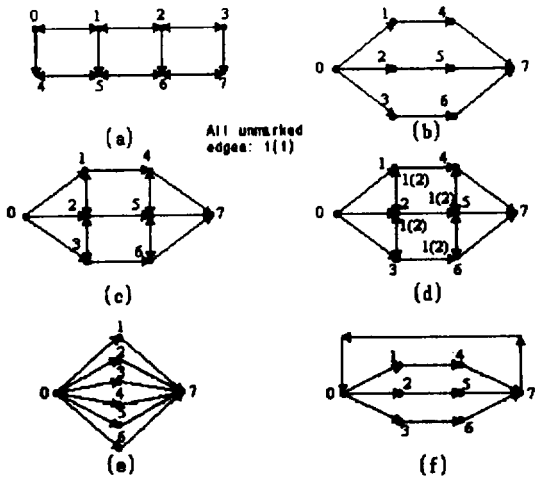
(그림 4) (a) 8개 노드 하이퍼큐브 (b) 16개 노드 하이퍼큐브
(Fig. 4) (a) 8-node hypercube (b) 16-node hypercube

웃노드까지(또는 부터) 세개의 메시지를 보낸다.(또는 받는다)

7.2 결과와 분석

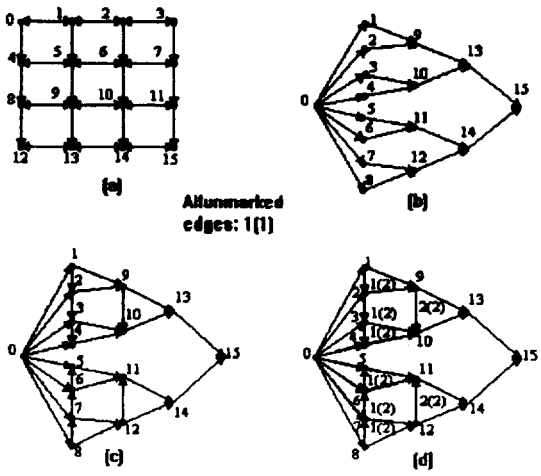
시스템 그래프를 위해 8개노드와 16개노드 하이퍼큐브가 사용되고, 몇 개의 문제 그래프가 적용되었다. 이 시험은 실시간 병렬처리에 특히 적합한 목적함수 OF_2와 OF_4로 실행했다. OF_2는 OF_4의 특별한 경우로서 고려된다. 구현에서는 앞에 기술한 알고리즘을 정확히 수행하였으며, 측정치가 초기할당에서 시스템 노드 선택을 위해 제안되었고 바꾸기 위한 후보 노드가 선택된다. 동기식 통신이 가정된다. 문제 그래프는 그림 5와 그림 6에서 보여진다.

그림 5와 그림 6에서 문제 그래프는 실제 응용에서 나타날 수 있는 경우이다. 예를 들면, G_{p1}과 G_{p6}는 처리되기 위한 자료 집합이 몇 개의 부분집합으로 구성되어 있는것을 보여준다. 이 부분집합은 인접된 처리요소 사이의 통신으로 단일 처리요소들에 의해서 처리된다. 그림 5는 자료분배, 병렬계산, 그리고 결과의 수집을 표현한 문제 그래프이다. 그들중 나머지는 3장에서 소개한 병렬처리 모델로서 고용된 다수의 처리요소의 각각에서 처리의 몇 가지 단계를 구성하는



(그림 5) 8개 노드의 문제 그래프 (a) G_{p1} (b) G_{p2} (c) G_{p3} (d) G_{p4} (e) G_{p5} (f) G_{p6}

(Fig. 5) Problem graphs of 8 nodes (a) G_{p1} (b) G_{p2} (c) G_{p3} (d) G_{p4} (e) G_{p5} (f) G_{p6}



(그림 6) 16개 노드의 문제 그래프 (a) G_{p7} (b) G_{p8} (c) G_{p9} (d) G_{p10}

(Fig. 6) Problem graphs of 16 nodes (a) G_{p7} (b) G_{p8} (c) G_{p9} (d) G_{p10}

타스크들을 기술한 것이다. 이 결과로부터 대부분의 경우 최적의 해를 위한 사상에는 초기할당 알고리즘이 고려되고, 최적의 사상을 초래하는 초기할당 다음에 쌍교환이 몇 번 반복되는 것을 알 수 있다. 8개노드 하이퍼큐브에서 교환횟수는 적어도 8이고, 16개노드에서는 16이 된다.

예 1.3:

문제 그래프 G_{p1}의 모든 처리요소는 2×4 메쉬로 이웃 처리요소 각각으로 부터 보내거나 받거나 할 수 있어서 최소의 OF_4는 2이다. 유사한 인수가 G_{p6}인 4×4 메쉬에도 적용될 수 있다. G_{p3}과 G_{p4}로부터 문제 연결선의 페이스에 대한 고려가 필요함을 알 수 있다. 즉, G_{p3}에서 e_{p12}, e_{p23}, e_{p45}, e_{p56}의 페이스가 G_{p4}에서 나머지 것과 다르다면 더 작은 OF를 만들 수 있다. G_{p5}의 경우 OF의 평가를 위한 과정에서 라우팅 규칙을 변경한다면 최소 가능한 OF_4가 2가 되는 것을 쉽게 볼 수 있다. 그러나, 이것은 다른 목적지를 위한 중간 노드를 결정하기 위해서는 다른 규칙이 적용되는 것을 요구한다.

각 연결선의 가중치가 주어졌고, 괄호 안에 숫자는 요구되는 페이스를 표시한다. 알고리즘 1의 수행 결과를 표 1과 표 2에서 볼 수 있다. 각 문제 그래프의 표준화 크기가 나타나 있다. 표 1에서 볼 수있듯이 표

〈표 1〉 8 개 노드를 갖는 하이퍼큐브의 MAPPER 수행결과
〈Table 1〉 MAPPER results for 8-node hypercube

G_p	초기할당후 사상	표준화	교환횟수	쌍교환후 사상	표준화
G_{p11}	(4 0 1 5 6 2 3 7)	8	0	(4 0 1 5 6 2 3 7)	8
G_{p12}	(1 2 5 6 0 3 4 7)	10	0	(1 2 5 6 0 3 4 7)	10
G_{p21}	(0 1 2 4 3 6 5 7)	5	0	(0 1 2 4 3 6 5 7)	5
G_{p31}	(2 3 0 4 5 1 7 6)	5	1	(2 1 0 4 5 3 7 6)	4
G_{p32}	(2 5 0 1 3 4 7 6)	6	1	(2 1 0 5 3 4 7 6)	4
G_{p41}	(2 3 0 4 5 1 7 6)	6	11	(3 7 5 1 0 4 2 6)	6
G_{p42}	(2 5 0 1 3 4 7 6)	6	8	(5 3 6 0 4 2 7 1)	4
G_{p51}	(0 1 2 5 6 4 7 3)	5	2	(4 1 2 5 0 6 7 3)	4
G_{p52}	(0 1 2 7 5 3 4 6)	5	2	(4 1 2 7 0 3 7 6)	3
G_{p61}	(0 2 7 5 1 3 4 6)	1	2	(1 2 4 5 0 3 7 6)	1

〈표 2〉 16 개 노드를 갖는 하이퍼큐브 의 MAPPER 수행결과
〈Table 2〉 MAPPER results for 16-node hypercube

G_p	초기할당후 사상	표준화	교환횟수	쌍교환후 사상	표준화
G_{p71}	(12 4 5 13 8 0 1 9 10 2 3 11 14 6 7 15)	20	0	(12 4 5 13 8 0 1 9 10 2 3 11 14 6 7 15)	20
G_{p72}	(5 6 9 10 1 2 13 14 4 7 8 11 0 3 12 15)	16	0	(5 6 9 10 1 2 13 14 4 7 8 11 0 3 12 15)	16
G_{p81}	(0 1 5 4 10 8 12 7 13 3 6 9 15 2 11 14)	5	1	(0 1 5 4 10 8 12 2 13 3 6 9 15 7 11 14)	5
G_{p82}	(0 1 13 9 3 2 10 7 5 11 4 14 6 8 15 12)	5	1	(0 1 7 9 3 2 10 13 5 11 4 14 6 8 15 12)	5
G_{p91}	(0 9 1 8 10 12 5 7 13 3 2 6 4 11 14 15)	16	6	(9 1 3 7 13 12 10 8 5 0 2 6 4 11 14 15)	13
G_{p92}	(0 2 10 9 12 6 11 7 3 1 4 13 5 8 14 15)	16	5	(9 1 10 2 12 8 11 3 7 0 6 13 5 4 14 15)	12
G_{p101}	(0 9 1 8 10 12 5 7 13 3 2 6 4 11 14 15)	16	1	(9 0 1 8 10 12 5 7 13 3 2 6 4 11 14 15)	16
G_{p102}	(0 2 10 9 12 6 11 7 3 1 4 13 5 8 14 15)	16	1	(1 2 10 9 12 6 11 7 3 0 4 13 5 8 14 15)	15

〈표 3〉 8 개 노드를 갖는 하이퍼큐브 의 결과
〈Table 3〉 Results for 8-node hypercube

G_p	초기 할당		쌍교환		
	사상(할당 행렬)	OF_4	교환횟수	사상	OF_4
G_{p1}	(4 0 1 5 6 2 3 7) (1 2 5 6 0 3 4 7)	2 2	0 0	(4 0 1 5 6 2 3 7) (1 2 5 6 0 3 4 7)	2 2
G_{p2}	(0 1 2 4 3 6 5 7)	1	0	(0 1 2 4 3 6 5 7)	1
G_{p3}	(2 3 0 4 5 1 7 6) (2 5 0 1 3 4 7 6)	5 5	1 1	(2 1 0 4 5 3 7 6) (2 1 0 5 3 4 7 6)	3 3
G_{p4}	(2 3 0 4 5 1 7 6) (2 5 0 1 3 4 7 6)	4 4	11 8	(3 7 5 1 0 4 2 6) (5 3 6 0 4 2 7 1)	2 2
G_{p5}	(0 1 2 5 6 4 7 3) (0 1 2 7 5 3 4 6)	5 5	2 2	(4 1 2 5 0 6 7 3) (4 1 2 7 0 3 7 6)	3 3
G_{p6}	(0 2 7 5 1 3 4 6)	4	2	(1 2 4 5 0 3 7 6)	3

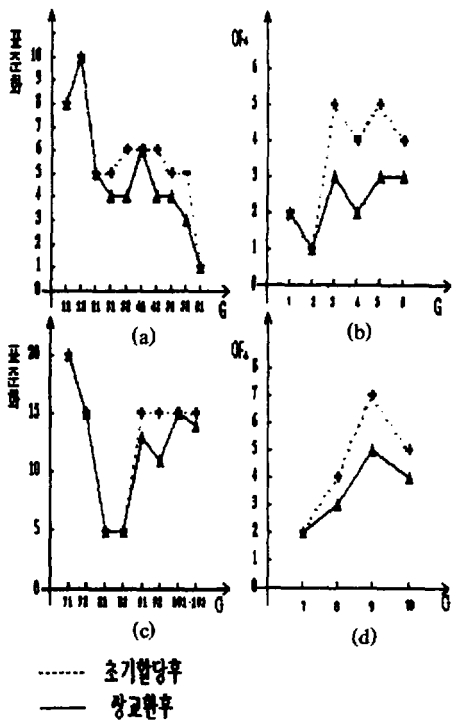
〈표 4〉 16 개 노드를 갖는 하이퍼큐브의 결과
〈Table 4〉 Results for 16-node hypercube

G_p	초기 할당		쌍교환		
	사상(할당 행렬)	OF_4	교환횟수	사상	OF_4
G_{p7}	(12 4 5 13 8 0 1 9 10 2 3 11 14 6 7 15) (5 6 9 10 1 2 13 14 4 7 8 11 0 3 12 15)	2 2	0 0	(12 4 5 13 8 0 1 9 10 2 3 11 14 6 7 15) (5 6 9 10 1 2 13 14 4 7 8 11 0 3 12 15)	2 2
G_{p8}	(0 1 5 4 10 8 12 7 13 3 6 9 15 2 11 14) (0 1 13 9 3 2 10 7 5 11 4 14 6 8 15 12)	4 4	1 1	(0 1 5 4 10 8 12 2 13 3 6 9 15 7 11 14) (0 1 7 9 3 2 10 13 5 11 4 14 6 8 15 12)	3 3
G_{p9}	(0 9 1 8 10 12 5 7 13 3 2 6 4 11 14 15) (0 2 10 9 12 6 11 7 3 1 4 13 5 8 14 15)	7 7	6 5	(9 1 3 7 13 12 10 8 5 0 2 6 4 11 14 15) (9 1 10 2 12 8 11 3 7 0 6 13 5 4 14 15)	5 5
G_{p10}	(0 9 1 8 10 12 5 7 13 3 2 6 4 11 14 15) (0 2 10 9 12 6 11 7 3 1 4 13 5 8 14 15)	5 5	1 1	(9 0 1 8 10 12 5 7 13 3 2 6 4 11 14 15) (1 2 10 9 12 6 11 7 3 0 4 13 5 8 14 15)	4 4

1의 G_{p11}, G_{p12}과 G_{p31}, G_{p32}, 표 2의 G_{p41} G_{p42}의 초기 할당의 사상에서 표준화의 크기가 다른 것을 알 수 있다. 이것은 통신 오버헤드를 측정하는데 연결(matched)되지 않은 연결선도 시스템의 수행 속도에 영향을 줄 수 있다는 것을 보여준다. 표 1과 표 2에서 쌍교환 후의 사상은 전체적으로 표준화의 크기가 줄어들어 시스템 그래프에 연결되는 문제 그래프의 사상에 최적화를 이루지 못함을 보여준다.

본 논문에서 제안한 알고리즘을 통하여 만들어진 통신 오버헤드의 크기는 초기할당에서 쌍교환후에 일률적으로 줄어들었다. 가중치를 고려하여 수행하였음에도 전체적인 통신 오버헤드가 감소되었다. 이 결과는 표 3과 표 4에서 보여진다.

이것은 제안된 사상 알고리즘이 최적의 해를 항상 만들지 못할지라도 잘 적용되는 것을 보여준다. 표3의 G_{p3}와 G_{p4}, 표 4의 G_{p9}과 G_{p10}은 페이스를 고려하지 않았을 때에는 동일한 크기로 통신 오버헤드가 형성되는데 페이스를 두었을 때 G_{p3}는 5, G_{p4}는 4로 그리고 G_{p9}는 7, G_{p10}는 5로 각각 통신 오버헤드가 감소하여 페이스를 고려하는 것이 전체 시스템의 수행능력에 영향을 주는 것을 볼 수 있다. 그림 7은 쌍교환의 반복 회수에 대한 알고리즘1의 결과와 본 논문에서 제안한 알고리즘의 결과를 비교한 것이다. 시스템 그래프가 8개 노드의 하이퍼큐브인 경우 (a)에서 보듯이 초기할당의 사상과 비교하여 쌍교환 후의 사상이 G_{p41}에서 변화가 있음을 보여준다. (b)에서는 초기할당의 사상과 쌍교환 후의 사상에서 통신 오버헤드가 일정함을 보여주고 있다. 16개 노드의 하이퍼큐브



(그림 7) 초기할당과 쌍교환후의 통신 오버헤드의 변화
(a) 8개노드 하이퍼큐브의 표준화크기
(b) 8개노드 하이퍼큐브의 OF_4 크기
(c) 16개노드 하이퍼큐브의 표준화크기
(d) 16개노드 하이퍼큐브의 OF_4 크기

(Fig. 7) Change of communication overhead after initial assignment and pairwise exchange
(a) Cardinality of 8-node hypercube
(b) OF_4 of 8-node hypercube
(c) Cardinality of 16-node hypercube
(d) OF_4 of 16-node hypercube

인 경우 (c)에서는 G_{p91}부터 G_{p102}까지가 변화가 있음을 알 수있고 (d)에서는 일정한 모양으로 감소되고 있음을 보여준다.

8. 결 론

병렬처리에서 시스템 그래프에 문제 그래프를 사상하는 최적의 측정치로서 세개의 다른 목적함수를 정형화했다. 이 목적함수는 다른 응용에서 통신 오버헤드를 나타낸다. 그들 중 하나는 앞에서 제안한 병렬 영상처리 모델에 기초되어서 병렬처리의 실시간 응용에 특히 접합하다. 이 목적함수들은 문제 연결선이 가중치가 있고, 시스템 연결선을 위한 명목거리에서 고용된 점에서 기존의 다른 OF와 다르다. 이 목적함수는 실제 통신 오버헤드를 더 정확하게 수량화할 수 있다. 목적함수를 기초로 본 논문에서는 사상 알고리즘을 개발했다. 이 알고리즘에서는 초기할당 기법과 초기할당을 위해서 쌍교환 기법을 반복적으로 적용한다. 사상기법의 중요한 생각은 더 가까이 위치하고 있는 것보다 더 자주 서로 통신하는 부타스크에 있다. 구현결과에서 두 기법이 함께 고용되므로서 최적이고, 합리적이며 좋은 사상을 발견할 수 있었다. 그렇지만 제안된 과정의 다양함을 주장하기보다 그 과정의 내용이 특별한 응용에 따라서 변경될 필요가 있음을 밝혀둔다.

본 논문에서는 다음과 같은 결론을 얻었다.

1) 모든 문제 연결선에 연결선의 통신 양에 따라서 가중치를 주어 기존의 알고리즘에서 모든 연결선에 동일하게 1을 준 것보다 교통 밀도에 따라 통신 오버헤드를 계산할 수 있었다.
2) 모든 문제 연결선의 페이스가 고려되어 기존의 내용에서 줄일 수 없었던 통신 오버헤드를 감소시켰다.
3) 시스템 연결선의 명목거리 보다도 실제거리가 적용되어 실시간 시스템에서 병렬처리를 위한 사상을 가능하게 하였다.
4) 다른 형태(병렬처리 구조의 위상과 라우팅 규칙)의 목적함수와 사상 알고리즘은 다른 형태의 응용에 의해서 받아들여진다.
5) 제안된 사상 알고리즘은 두개의 기법을 고용함으로서 최적이고, 합리적이며 좋은 사상을 발견하게 하였다.

앞으로의 연구는 병렬 알고리즘의 의존 구조(dependency structure)가 병렬 컴퓨터의 프로세서 상호연결 관계와 다를 때와 알고리즘에 의해서 산출된 프로세스(process)의 수가 이용할 수 있는 프로세서(processor)의 수를 초과할 때의 경우를 고려한 일반적인 사상 기법을 위하여 본 논문에서 제시한 알고리즘을 변경 및 보완을 해야한다.

참 고 문 헌

[1] J.K. Aggarwal, L.S. Davis, and W.N. Martin, "Correspondence processes in dynamic scene analysis," proc. IEEE, vol. 69, pp.562-572 May 1981.

[2] F.Berman and L.Snyder, "On mapping parallel algorithms into parallel architectures," in proc. Int. Conf. Parallel processing, Aug. 1984, pp. 307-309

[3] D.Bernstein, M.Rodeh, and I.Gertner, "On the complexity of scheduling problems for parallel/pipelined machines," IEEE Trans. Comput., Vol. c-38, no.9, pp.1308-1313, Sept. 1989.

[4] S.H.Bokhari, "Dual-processor scheduling with dynamic reassignment," IEEE Trans. Software Eng., vol. SE-5, pp.341-349, July, 1979.

[5] S.H.Bokhari, "On the mapping problem," IEEE Trans. Comput., vol C-30, pp.207-214, Mar, 1981.

[6] S.H.Bokhari, "A shortest tree algorithm for optimal assignments accross space and time in a distributed processor system," IEEE Software Eng. vol.SE-7, no.6, pp.335-341, Nov. 1981.

[7] T.L.Casavant and J.G.Kuhl, "A taxonomy of scheduling in qeneral-purpose distributed computing systems," IEEE Trans. Software Eng., Vol.14, no.2, pp.42-45, Feb. 1988.

[8] M.Haran and J.M.Kurtzberg, "A review of the placement on quadratic assignment problems," SIAM Rev., vol.14, pp.324-342, Apr., 1972.

[9] Kai Hwang, "Computer Architecture and Parallel

processing", 1985, McGRAW-HILL Int. Editions.

[10] S.J.Kim and J.C.Browne, "A general approach to mapping of parallel computations upon multiprocessor architectures," in proc. Int. Conf. Parallel processing, Aug. 1988, pp.1-8

[11] S.Y.Kung, "VLSI Array processors", Prentice-Hall International Editions, 1988.

[12] S.Y.Lee and J.K.Aggarwal, "A mapping strategy for parallel processing," IEEE Trans. Comput, Vol.c-36, no.4, pp.433-442, Apr. 1987.

[13] Dan I. Moldovan, "Parallel Processing", Morgan kaufmann publishers San Mateo, California, 1993.

[14] H.S.Stone, "Multiprocessor Scheduling with the aid of network flow algorithms," IEEE Trans. Software Eng., vol SE-3, pp.85-93, Jan, 1977.

[15] Special Issue on Computer Architectures for Image processing, IEEE Computer, vol. 16, Jan., 1983.

[16] F.Thomson Leighton, "Introduction to PARALLEL ALGORITHMS and ARCHITECTURES: Arrays, Trees, Hypercubes", 1992, Morgan Kaufmam publishers, Inc.

[17] J.D.Ullman, "NP-complete scheduling problems," J.Comput. Syst.sci., Vol.10. pp.384-393, 1975.

[18] K.Vairavan and R.A.Demillo, "On the computational complexity of a generalized scheduling problem," IEEE Trans. Comput., Vol.c-25, pp. 1067-1073, Nov. 1976.

[19] M. Lo Virginia and Sanjay Rajopadhye, "Mapping Divide-and-Conquer Algorithms to Parallel Architectures", Int. Conf. on Parallel Processing III", pp.127-135, 1990.

[20] T.Yang, C.Fu, A. Gerasoulis and V.Sarkar, "Mapping Iterative Task Graphs on Distributed Memory Machines", Int. Conf. on Parallel Processing II, pp.151-158, 1995.

[21] 김석수, 전문석, 이철희, "병렬알고리즘 사상을 위한 일반화된 기법의 사상모델", vol.22, No.1 한국정보과학회 봄 학술발표 논문집, 1995년

김 석 수

1982년 숭실대학교 전자계산학과 졸업(학사)

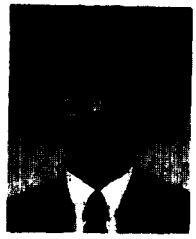
1987년 숭실대학교 대학원 전자계산학과 졸업(석사)

1992년~현재 숭실대학교 대학원 전자계산학과 박사과정

1984년~1987년 한국과학기술원 시스템공학센터 연구원

1989년~현재 경원전문대학 전자계산학과 부교수

관심분야:병렬알고리즘, 병렬컴퓨터 구조, 병렬처리 이론, 폴트톨로런스



전 문 석

1980년 숭실대학교 전자계산학과 졸업(학사)

1986년 Univ. of Maryland 전산과(석사)

1988년 Univ. of Maryland 전산과(박사)

1989년 Morgan State Univ. 전산수학과 조교수

1991년 New Maxico State Univ. 부설 Phsical Science Lab 책임연구원

1991년~현재 숭실대학교 컴퓨터학부 부교수

관심분야:병렬알고리즘, 병렬컴퓨터 구조, 대규모 집적회로, 병렬처리 이론, 폴트톨로런스