産業技術硏究(江原大學校 産業技術硏究所 論文集) 第 15輯 (1995)
Research Report,The Institute of Industrial Technology, Kangwon Nat'l Univ., Korea, Vol. 15, 1995

균일한 길이 데이터 집합의 분할분배방식

김 희 원 · 김 형 중* · 이 정 문**

Multiple Installment Load Distribution Algorithm for Data Sets

Hee-Won Kim · Hyoung Joong Kim* · Jung-Moon Lee**

ABSTRACT

This paper presents a new method for load distribution in a single-level tree network equipped with front-end processors. This method focuses on effective data distribution over a number of processors minimize job processing time. Optimal multiple installment load distribution algorithm is presented. Minimum number of processors that maximizes efficiency is decided theoretically.

1. 서 론

데이터 분배는 병렬 계산 연구의 중요한 과제 가운데 하나이다. 프로세서들 사이의 효과적인 데이터 분배는 병렬 및 분산 처리 시스템의 효율을 높이는데 매우 중요한 역할을 담당한다. 그러나 효율적인 데이터 분배 방법은 매우 적다.

그래서 대부분의 최적 분배 방식 연구는 임의의 크기로 나눌 수 있는 데이터에 초점을 맞추고 있다[1,2,3,5].

임의의 길이로 나눌 수 있는 데이터 집합들의 최적의 해는 어떤 프로세서도 초기화된 후 계산이 끝날 때까지 쉬지 않는다는 가정하에서 쉽게 얻어질 수 있다. 이 가정은 프로세서에 배정되는 데이터 길이가 제각각으로 임의의 길이에서 잘라야 하지만 대신 같은 시각에 끝난다. 하지만 균등한 크기로 나누어진 데이터에서 어떤 프로세서는 여전히 처리중인 반면에 다른 프로세서들이 일찍 끝마칠 수 있다. 이것은 임의의 길이로 분배되는 최적의 해가 일정한 길이의 데이터 분배에는 적합하지 않다는 것을 의미한다.

여기서 우리는 두가지 방법을 제시하겠는데 하나는 kim, Jeon과 Lee[4]가 제시한 방법이고 다른 하나는 이 논문에서 처음 제시하는 새로운 방법이다. 두번째 방법에서는 주어진 데이터의 양, 통신시간, 계산시간만으로도 이 데이터를 가장 빠르게 처리하기 위한 프로세서의 갯수를 정할 수 있고 또한 각 프로세서에 분배할 최적의 데이터 양을 구할 수 있다.

이 연구에서는 균등한 길이의 데이터 문제를 중점적으로 다루었다. 먼저 2절에서 균등하게 나누어질 수 있는 데이터 문제의 개념을 설명한다. 이전의 연구[2,4]는 해당 챠일드 프로세서에 배정된 데이터를 한꺼

강원대학교 제어계측공학과 석사과정
* 강원대학교 제어계측공학과 부교수
** 강원대학교 제어계측공학과 부교수

번에 전송하는 경우의 최적의 해를 구했다. 그러나 Bharadwaj, Ghose와 Mani[1]는 차일드 프로세서에 전송할 데이터는 여러번에 나누어 전송하는 방법을 채택함으로써 성능이 개선될 수 있음을 보였다. 3절에서 균등한 길이의 데이터를 최적으로 분배하는 방법을 제안한다. 마지막으로 4절에서 결론을 제시한다.

2. 데이터 분배 문제

하나의 루트 프로세서(또는 호스트 프로세서)와 여러개의 차일드 프로세서로 이루어진 트리나 스타형 구조를 갖는 네트워크에서 호스트 프로세서는 차일드 프로세서들에게 적절한 양의 데이터를 분배하는데 이러한 경우에 호스트와 차일드 프로세서 사이에 통신이 발생한다. 프론트-엔드 시스템이 장착되어 있을때는 호스트에서는 계산과 통신이 동시에 이루어질 수 있다. 이때 호스트 프로세서는 한번에 하나의 차일드 프로세서와 통신할 수 있다고 가정한다. 또한 프로세서의 처리속도와 각 프로세서간의 전송속도는 모두 같다고 가정한다.

우리는 그림 1과 같은 호스트 프로세서 밑에 차일드 프로세서들로 구성된 레벨 1

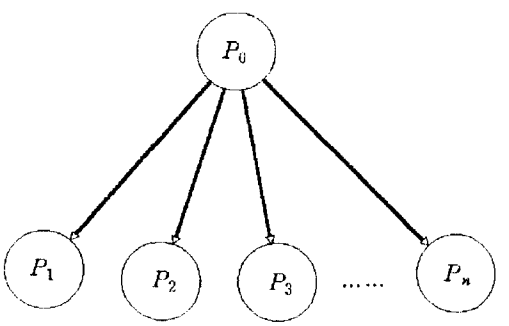

Fig. 1. Single-level tree architecture.

트리 네트워크를 고려한다. 이때 호스트 프로세서를 포함해서 n+1개의 프로세서가 있고 호스트 프로세서에 총 L개의 데이터가 주어졌다고 가정한다.호스트 프로세서는 P_0이라 표시되고 나머지 n개의 차일드 프로세서들 역시 P_i(i=1,2...n)라고 표현한다. 여기서 x_i는 $\alpha_{i,k}$(k=1,2,...m)의 합이고 프로세서 P_i에 할당된 데이터의 갯수이다. 여기서 $\alpha_{i,k}$는 한번에 전송될 수 있는 데이터의 최소단위로써 프로세서 P_i에 할당된 k번째 전송량을 나타낸다.

본 연구에서는 $\alpha_{i,k}=\alpha_{i,l}$ $(1\le k=l\le m)$이고 x_i의 합은 L과 같다. 즉 다음과 같은 관계식이 성립한다.

$$x_i=\sum_{k=1}^{m}\alpha_{i,k} \qquad (1)$$

$$\sum_{i=0}^{n}x_i = L \qquad (2)$$

이 연구에서 w와 v는 각각 프로세서의 계산속도와 전송속도의 역비율 상수를 표현한 것이다. 각각 w=1, v=1일 때 t_p와 t_c는 프로세서에서의 단위계산시간과 단위통신시간을 나타낸다.

이제 많은 양의 데이터가 호스트 프로세서에 주어졌다고 생각해보자. 호스트 프로세서는 먼저 수신된 데이터의 일정한 부분을 자신이 처리하기 위해서 가지고 있고 나머지를 차일드 프로세서들에게 분배하는데 각 차일드 프로세서에 $P_1, P_2, \cdots$ 의 순서대로 분배한다.

우리의 목적은 데이터 처리 시간을 최소화하기 위해서 호스트를 포함하는 프로세서들에게 할당할 데이터의 양을 최적으로 결정하는 것이다. 가장 늦게 처리를 끝마치는 시간, 즉 프로세서들 가운데 가장 늦게 처리를 끝마치는 프로세서의 처리 시간이 전체 처리시간을 의미한다.

각 프로세서에 할당될 최적의 양은 전송

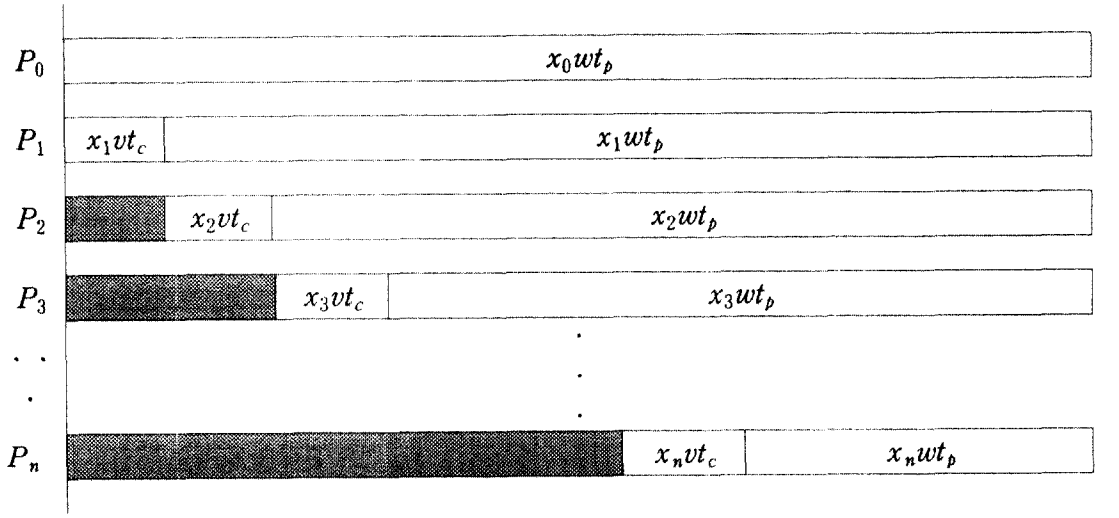

Fig. 2. Timing diagram

능력과 프로세서의 계산 성능에 달렸다. 일단 적당한 양이 결정되면 호스트 프로세서는 자신에게 할당된 데이터의 계산을 시작하고 동시에 통신 채널을 통해서 차례 차례로 주변의 차일드 프로세서들에게 나머지 데이터를 분배한다. 호스트 프로세서로부터 자신의 할당량을 받으면 각 차일드 프로세서는 할당된 양의 계산을 시작한다.

할당될 데이터를 x_i씩 호스트 프로세서가 각 차일드 프로세서에 차례로 보내는 경우와 할당될 데이터 x_i 가운데 $\alpha_{i,k}$씩 즉, 한번에 보낼 수 있는 최소 단위씩 보내는 경우 두 가지로 나누어서 고려해 보았다.

2.1 데이터를 모두 분배하는 경우

그림 2에서 알 수 있듯이 일단 프로세서들에게 분배할 데이터의 양이 결정되면, 호스트 프로세서는 자신의 몫인 x_0만큼의 데이터를 처리하기 시작한다. 동시에 호스트는 x_1만큼의 데이터를 첫번째 차일드 프로세서(P_1)에게 전송하는데 이때 x_1vt_c만큼의 시간이 걸린다. 이 전송시간동안 모든 다른 차일드 프로세서들은 아무런 일도 수행할 수 없는 아이들 상태를 유지한다. x_1vt_c 시간이 지나면 호스트 프로세서는 두 번째 차일드 프로세서(P_2)에게 x_2의 양만큼을 분배한다. 동시에 첫번째 차일드 프로세서는 자신에게 할당된 x_1만큼의 일을 처리하기 시작한다. 같은 방식으로 순서에 따라 모든 데이터가 분배되고 처리될 때까지 차례차례로 반복된다.

데이터가 할당되고 각 프로세서에 분배되었다고 가정할때 처리 완성 시간(T_i, $i=0,1,2,\cdots,n$)은 다음에 나오는 관계에 의해 알 수 있다.

$$\begin{aligned} T_0 &= x_0wt_p \\ T_1 &= x_1(vt_c+wt_p) \\ T_2 &= x_1vt_c+x_2(vt_c+wt_p) \\ &\cdots \\ T_n &= \sum_{i=1}^{n-1} x_ivt_c+x_n(vt_c+wt_p) \end{aligned} \qquad (3)$$

여기서 x_i는 모두 데이터 최소단위의 합

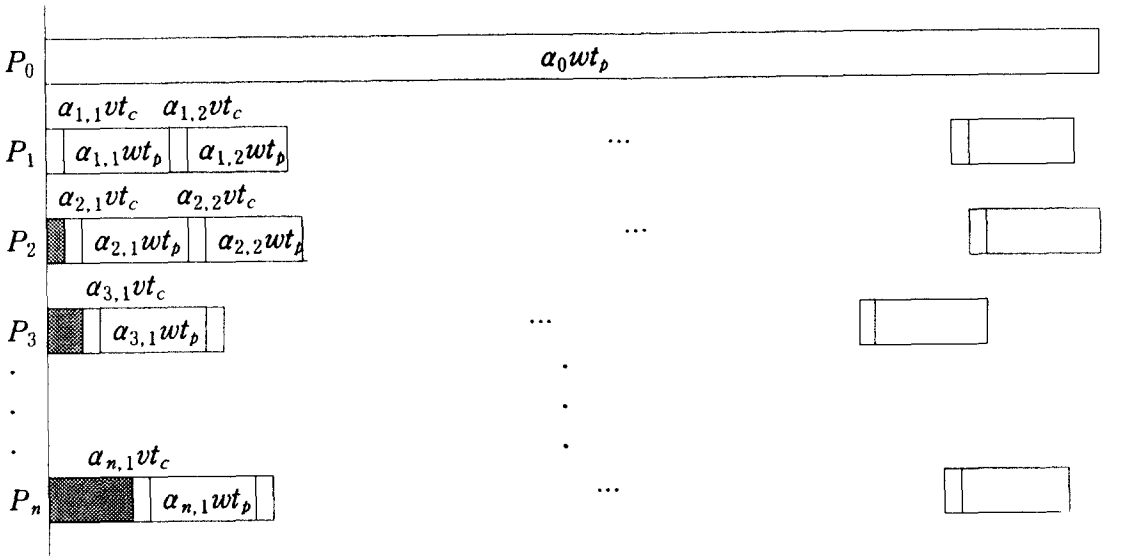

Fig. 3. Timing Diagram

$$\sum_{k=1}^{m}\alpha_{i,k}(vt_c+wt_p) > \sum_{k=1}^{m-1}\alpha_{j,k}(vt_c+wt_p)+\alpha_{j,m}vt_c+\sum_{i=1}^{j-1}\alpha_{i,1}vt_c \ , \qquad (4)$$
$$(i < j \ , i=1,2,\cdots,j=2,3,\cdots)$$

이다. 프로세서들 사이에 할당될 수 있는 부하의 총합은 식(2)에서처럼 L이 되어야 한다.

x_i의 최적 값은 T_i(i=1,2,...,n)의 값을 최소화시키는 값들이다.

2.2 데이터를 분할 분배하는 경우

먼저 위 조건 (4)를 만족시키는 경우에만 호스트 프로세서는 챠일드 프로세서에게 데이터를 분배한다.

식 (4)가 의미하는 것은 P_j로 전송하는 시간이 P_i로 데이터를 전송하고 계산하는 시간보다 더 오래 걸린다면 호스트 프로세서는 P_j로 전송을 하지 않는 것이 더 논리적임을 나타낸다. 이 관계로부터 다음 관계를 알 수 있다.

$$n \le \lceil \frac{t_p}{t_c} \rceil + 2 \qquad (5)$$

여기서 n은 프로세서의 수를 의미하고 $\lceil \cdot \rceil$은 이 값을 초과하지 않는 최대의 정수를 뜻한다.

주어진 통신시간, 계산시간 그리고 데이터의 양에 대하여 가장 효율적이고 최적의 처리시간을 갖는 프로세서의 갯수가 n이다. 실제로 데이터를 일시에 분배하는 방식으로는 프로세서의 갯수를 무한히 증가시켜도 분할분배하는 방식에서 식 (5)의 n개의 프로세서로 구한 처리시간과 비슷한 시간을 구할 수 있을 뿐이다.

이제 프로세서들에게 분배될 데이터의 양이 결정되면 호스트 프로세서는 총 데이터 중 x_0만큼의 일을 처리하면서 동시에 첫 번째 챠일드 프로세서인 P_1에 x_1 가운

데 나눌 수 있는 최소 단위인 $\alpha_{1,1}$만을 전송한다. 이때 걸리는 시간은 $\alpha_{1,1}vt_c$이다. 다음에 P_2에 x_2 가운데 $\alpha_{2,1}$만큼을 전송하고 동시에 P_1은 전송받은 $\alpha_{1,1}$만큼의 데이타를 처리한다. 이 때의 처리시간 즉, 계산시간은 $\alpha_{1,1}wt_p$가 된다.

여기서 한가지 고려할 점은 위에 설명한 것처럼 분할해서 여러차례 데이터를 분배할 때 프로세서 사이의 전송시간이 겹치는 부분이 없어야 한다는 것이다. 식 (4)를 만족하는 경우라면 전송이 겹치는 부분은 없고 식 (5)를 만족하는 갯수의 프로세서만을 사용한다면 최적의 시간과 최적의 프로세서를 사용할 수 있음은 명백하다.

그림 3은 이와같은 상태를 나타낸다.

이 방식을 사용하면 첫 번째 챠일드 프로세서에게 x_1만큼을 보내는 동안 두 번째 이하 챠일드 프로세서(P_2 이하)에게 생겼던 아이들 시간(그림 2의 색칠된 부분)을 거의 줄일 수 있다. 예를 들면 첫번째 방법에서 세번째 챠일드 프로세서인 P_3에 생겼던 아이들 시간이 $(x_1+x_2)vt_c$인 반면에 이 방법에서 세번째 챠일드 프로세서인 P_3의 아이들 시간은 $(\alpha_{1,1}+\alpha_{2,1})vt_c$가 된다. 그러므로 총 데이터 처리시간은 많이 줄게 된다..

이때의 수행시간은 다음과 같다.

$$\begin{aligned} T_0 &= x_0 wt_p \\ T_1 &= x_1(vt_c + wt_p) \\ T_2 &= \alpha_{1,1}vt_c + x_2(vt_c + wt_p) \qquad (6) \\ &\cdots \\ T_n &= \sum_{i=1}^{n-1} \alpha_{i,1}vt_c + x_n(vt_c + wt_p) \end{aligned}$$

여기서도 물론 식(2)가 성립해야 한다.

3. 할당 알고리즘의 개발

[1]에서와 같이 무한히 작은 양으로 데이터를 나눌 수 있다고 가정했을 때의 최종 처리시간은 모든 프로세서에서 똑같다. 이때의 가정은 데이터를 무한히 작은 값으로 쪼갤 수 있으므로 아이들 시간이 없게 할 수 있어 최적의 처리시간을 구할 수 있다. 그러므로 이 최적의 처리시간(여기서는 T^*라 부르기로 함)을 초기값으로 사용하여 각 할당량을 구하면 총 합이 당연히 L보다 작거나 같다. 즉 남은 양을 재분배해야 하는 상황이 발생할 수 있다.

T^*에 근거하여 데이터가 프로세서들에게 할당되었지만 아직 할당될 양이 더 남았다고 가정해 보자. 이 양을 할당하기 위해서 더해지는 데이터의 양이 챠일드 프로세서 P_j에 할당되었다고 가정하자. 그러면 호스트 프로세서의 처리완료시간 T_0와 P_j 앞에 자리잡은 챠일드 프로세서($P_i, i<j$)들의 처리완성시간은 변하지 않을 것이다. 그러나 프로세서 P_j의 처리완성시간(T_j)은 데이터의 전송시간 vt_c과 계산시간인 wt_p의 합의 정수배만큼 증가할 것이다. 증가된 양은 모든 뒤에 있는 프로세서들에게 영향을 미칠 것이다. 그러나 호스트 프로세서에 하나의 데이터가 더해졌을 때는 단지 T_0가 계산시간인 wt_p의 양만큼 증가하지만 다른 어떤 시간증가도 챠일드 프로세서에 발생하지 않는다. 그러므로 알고리즘에서 사용될 참조 시간(T^*)은 다음과 같다.

$$T^* = \begin{cases} T_{\min} + wt_p, & \text{if } T_{\min} = T_0 \\ T_{\min} + vt_c + wt_p, & otherwise \end{cases} \qquad (7)$$

여기서 $T_{\min}$은 현재의 결과인 T_i의 최소값 즉, $T_{\min} = \min\{T_0, T_1, \cdots, T_n\}$이다.

3.1 데이터를 모두 분배하는 경우

먼저 앞에서 설명한 임의의 길이로 나눌 수 있는 경우의 수행시간은 일정한 길이의 데이터 분배에서 최적의 해를 구하기 위한 좋은 시작점이 된다. T^*의 결과를 기초로 각 프로세서의 새로운 할당량은 다음 식과 같이 얻어진다.

$$x_i=\begin{cases}\dfrac{T^*}{wt_p}, & (i=0)\\ \dfrac{T^*}{vt_c+wt_p}, & (i=1)\\ \dfrac{T^*-(x_1+\ldots+x_{i-1})vt_c}{vt_c+wt_p}, & (i=2,3,\ldots,n)\end{cases} \tag{8}$$

또한 이 값을 기초로 호스트와 챠일드 프로세서들의 새로운 작업완성시간은 식 (3)에서 주어진 관계에 의해서 쉽게 얻어진다.

각 연산의 끝에 데이터의 재할당을 끝마쳤을 때 총 할당량이 L과 같은지 아닌지를 다시 체크해야 한다.

식 (8)에서 나온 값들로 (3)과 (7)식을 이용하여 T^*의 값을 갱신한다. 이러한 절차를 식 (2)가 만족할 때까지 반복한다. 식 (2)가 만족되었을 때의 각 데이터 할당량이 최적의 할당량이다.

3.2 데이터를 분할 분배하는 경우

이 방법이 첫 번째 방법과 다른 점은 3번째인 P_2이하 프로세서의 아이들 시간이 매우 많이 단축되었다는 것이다.(그림 2와 3 참조) 그러므로 3번째 프로세서의 아이들 시간은 2번째 프로세서가 $\alpha_{1,1}$만큼 전송하는 시간 동안이므로 $\alpha_{1,1}vt_c$이고 4번째 프로세서 P_3의 아이들 시간은 2번째와 세 번째 프로세서의 첫 분할 데이터 전송시간인 $(\alpha_{1,1}+\alpha_{2,1})vt_c$가 된다. 이것을 기반으로 하고 T^*의 결과를 기초로 각 프로세서의 새로운 할당량은 다음 식과 같이 구할 수 있다.

$$x_i=\begin{cases}\dfrac{T^*}{wt_p}, & (i=0)\\ \dfrac{T^*}{vt_c+wt_p}, & (i=1)\\ \dfrac{T^*-\sum_{k=1}^{i-1}\alpha_{k,1}vt_c}{vt_c+wt_p}, & (i=2,3,\cdots,n-1)\end{cases} \tag{9}$$

호스트와 챠일드 프로세서들에 의한 데이터 처리 완결시간은 식 (6)에서 주어진 관계에 의해 얻을 수 있다.

역시 초기치 T^*에 의한 분배 이후 남은 데이터의 재할당을 끝마쳤을 때 총 할당량이 L과 같은지 아닌지를 즉, 식 (2)를 만족시키는지를 다시 체크해야 한다.

한가지 고려해야 할 것은 프로세서의 갯수를 식 (5)에서 만족하도록 정하여야 한다는 것이다. 만일 식 (5)를 위반하면 효율이 증가하지 않는다는 것은 명백하다. 왜냐하면 t_p/t_c+3번째 프로세서는 2번째 프로세서와 전송시간이 겹치기 때문이다.

3.3 효율 분석

두 방식의 효율을 비교하여 보자. 데이터를 x_i씩 분배하는 방식을 이용한 결과와 데이터를 $\alpha_{i,k}$, 즉 나눌 수 있는 최소단위씩 분배하는 방식을 이용한 결과를 전송시간(t_c)과 수행시간(t_p), 그리고 프로세서의 갯수 n을 변화시키면서 데이터 처리완결시간을 구해보았다. $\alpha_{i,k}$의 값은 정해져 있는 값이므로 임으로 1로 고정시켰고 L값은 100으로 하였다. 물론 n, t_c, t_p, L의 값은

TABLE 1. Result of Two Algorithms

	x_i씩 분배		$\alpha_{i,k}$씩 분배	
n	5	6	5	6
$t_c:t_p$	1:10	1:10	1:10	1:10
T_0	$24wt_p$	$21wt_p$	$22wt_p$	$18wt_p$
$t_c:t_p$	1:3	1:3	1:3	1:3
T_0	$33wt_p$	$30wt_p$	$26wt_p$	$26wt_p$

얼마든지 변화시킬 수 있다. 시뮬레이션 결과는 표 1에 나타내었다.

위 표를 보면 x_i씩 분배하는 경우 프로세서의 갯수에 비례하여 작업 완결시간이 감소하고, 계산시간에 비해 통신시간이 짧을수록 감소한다. $\alpha_{i,k}$씩 분배하는 경우도 마찬가지이다.

이제 두 방법을 비교하여 보면 같은 조건 (n의 갯수, t_c, t_p, L의 양)에서 $\alpha_{i,k}$씩 분배하는 경우가 x_i씩 분배하는 경우보다 작업 완결 시간이 훨씬 더 감소함을 알 수 있다. 예를 들어 프로세서가 5개이고 $t_c:t_p$ = 1:10인 경우는 약 9%정도의 효율이 증가하였는데 이것은 x_i씩 분배하는 경우에서 프로세서를 하나 더 추가해서 구한 작업 완결시간과 거의 같다. 또 프로세서가 5개이고 $t_c:t_p$=1:3인 경우를 보면 약 21%정도의 효율이 증가하였는데 이것을 x_i씩 분배하는 경우에서 프로세서를 무수히 많이 증가하여야만 얻을 수 있는 아주 좋은 결과로 이때의 프로세서의 갯수는 식 (5)를 만족하는 최대의 갯수로 최적의 수행시간을 구할 수 있는 값이다.

표 1에서 보면 $t_c:t_p$ = 1:3인 경우 효율적인 최대 프로세서의 갯수는 식 (5)에 의하면 5이고 실제로 이 경우 프로세서를 6개로 늘려보았으나 처리시간이 감소하지 않음을 알 수 있다. 하지만 이 최대 프로세서 처리시간은 일시에 분배하는 경우에 프로세서를 무수히 많이 늘려야만 얻을 수 있는 효율이고 아무리 많이 늘려도 이 이상 효율이 거의 증가하지 않으므로 매우 비경제적임을 알 수 있다.

참고로 이 표에서 구한 수행시간 T_0는 호스트 프로세서의 수행시간으로 작업 완결시간이라 할 수는 없지만 (어느 프로세서가 가장 늦게 수행을 끝내는지 알 수 없으므로) 거의 비슷한 시간대에 수행을 끝내므로 대표값으로 사용하여도 문제가 없다.

이 알고리즘에서 고려할 점이 있는데 이렇게 구한 x_i의 값 중 두 개 이상이 증가하여 총 합인 L보다 클 경우가 생긴다. 이때는 그 증가된 값들을 갖는 프로세서의 수행 시간을 계산하여 더 큰 수행 시간을 갖는 프로세서의 값에서 $\alpha_{i,k}$를 빼서 최적의 값을 구한다. 예를 들자면 $\alpha_{i,k}$의 값이 1이고 L이 100이라고 하자. 이때 x_3의 값이 24.9이고 x_4의 값이 17.8이라고 하면 실제로 전송하는 데이터는 x_3가 27개이고 x_4가 17개가 된다.(나눌 수 있는 최소단위가 1이라면) 이때의 합이 99이 될때 합이 L보다 작으므로 다시 계산을 하였을 때 x_3와 x_4의 값이 모두 증가하여 x_3가 25.2로 25개 x_4가 18.1로 18개가 될 수가 있다. 이때의 합은 2가 증가함으로 L보다 크다. 이런 경우에는 두 프로세서의 수행 시간을 계산하여 더 오래 걸리는 프로세서에서 $\alpha_{i,k}$만큼 즉 1을 빼주었을 때의 x_i값이 최적이 된다.

4. 결 론

이 연구에서는 단일 레벨 트리 네트워크에서 처리시간을 최소화하는 부하분배의 새로운 방법을 제시한다. 데이터가 균등하게 나누어지는 경우와 프론트-엔드 프로세

서가 있는 경우만을 다루었다. 만일 일정한 데이터의 양이 정해지고 각 프로세서의 처리속도와 각 프로세서간의 통신속도가 결정되면 이 알고리즘을 이용하여 최적으로 처리하기 위한 프로세서의 갯수와 각 프로세서에 할당될 데이터의 양을 구할 수 있다. 그러므로 매우 큰 양의 데이터를 처리할 때 최적의 처리시간을 구하기 위한 조건을 쉽게 갖출 수 있으므로 매우 유용하게 쓰일 수 있다.

후 기

본 연구는 한국통신 연구개발단의 지원(95-09-006)에 의해 이루어졌다.

참 고 문 헌

[1] V. Bharadwaj, D. Ghose and V. Mani, "Multi-Installment Load Distribution in Tree Networks with Delays," *IEEE Trans. Aero. and Electr. Syst.*, Vol. 31, No. 2, pp. 555-567, 1995.

[2] S. Bataineh and T. G. Robertazzi, "Ultimate Performance Limits for Networks of Load Sharing Processors," *CEAS Technical Report 623*, State University of New York at Stony Brook, 1992.

[3] V. Mani and D. Ghose, "Distributed Computation in Linear Networks: Closed-Form Solutions," *IEEE Trans. Aero. Electr. Syst.*, Vol. 30, No.2, pp. 471-483, 1994.

[4] H. J. Kim, T. B. Jeon and J. G. Lee, "An Optimal Data Distribution Algorithm for Elliptic solver," *Advanced Computational and Design Techniquesin Applied Electromagnetic Systems*, Elsevier Science B.V., pp. 215-218, 1995.

[5] S. Bataineh and T. G. Robertazzi, "Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job, "*CEAS Tech. Report 627,* State University of New York at Stony Brook, 1992.