論 文

論文80-17-5-6

마이크로프로세서複合에 의한 並列處理에 関한 研究

(A Study on Parallel Processing by Multi-Microprocessors)

鄭 然 澤*, 宋 榮 宰**

(Chung, Yon Tack and Song, Young Jae)

要 約

本 研究에서는, 8085 CPU 4臺를 사용하여서, 主마이크로프로세서의 버스에 DMA장치를 통하여 從마이크로프로세서에 접속되는 複合마이크로프로세서 시스템을 設計하여서 並列處理 시킴으로서 處理効率을 높일 수 있었다. 마이크로프로세서의 動作臺數와 시스템 Throughput의 관계를 측정한 결과, 理想値의 70~80 퍼센트 정도에 머무는 결과를 얻었다.

資源의 割當이나 日程計劃은 主마이크로프로세서가 이루도록 하였고, 마이크로프로세서 사이의 通信 및 共有데이터의 格納은 共有메모리를 사용하도록 하였다. 또한 直列로 作成된 原始프로그램으로 부터 並列性을 檢出하는 方法을 提示하였다.

Abstract

In this study, multi-microprocessors system in which slave microprocessor is connected with master microprocessor bus through the DMA controller is designed by the use of four 8085 CPU. A high degree of processing efficiency could be obtained by making this system work parallel processing. The result of measuring relations between working microprocessor and system throughput was 70-80 percents lower than ideal value.

Master microprocessor takes charge of resource allocation and scheduling, common memory assigns communication between microprocessors and a store of common data. The method of detecting parallelism from source program composed by series is also suggested.

1. 諸 論

반도체기술의 발달로 인하여 마이크로프로세서가 高性能化되고, 염가로 구입할 수 있어도 메모리나 주변장치는 그렇게 싸지는 않다. 따라서, 마이크로프로세서의 價格構成을 보면 극히 평형을 이루지 못하게 된다. 지금 까지의 컴퓨터시스템은 Grosch의 法則[1]이 성립하였으나 최근의 LSI기술은 하드웨어의 價格에 급격한 변화를 가져왔다. 이로 인하여 Grosch의 법칙이 小型計算機 以下의 領域에서는 성립하지 않게 되었다. 즉, 性能은 價格의 1乘以下로 比例하게끔 되었다. 그러므로 單一프로세서(處理裝置)를 사용하는 대신에 조그마한 프로세서를 여러개 結合하여서 그것과 同等以上의 處理能力을 갖도록 하자는 생각이 생긴 것은 당연한 일이다. 따라서, 조그마한 프로세서를 여러개 접속한 複合프로세서 시스템은 並列處理에 의하여, 處理速度의 向上, 信賴性의 向上, 시스템의 擴張性增加, 價格 性能比의 低下등의 利點때문에 많은 연구가 진행되고 있다[2~4].

複合마이크로프로세서 시스템에서는 본래 한개이

* 正會員, 明知大學 電氣工學科 (Dept. of Electrical Engg., Myong Ji Univ.)

** 正會員, 慶熙大學校 電子工學科 (Dept. of Electronics Engg., Kyung Hee Univ.)

接受日字: 1980年 6月 16日

* 이 研究는 문교부 학술연구조성비에 의하여 이루어진 것임.

던 計算 job 를 並列處理 可能한 多數의 타스크(task)로 分割하여서 각 分割타스크를 한臺의 마이크로프로세서에 分擔處理시킨다. 그러나, 본래 하나의 job 를 分割하였으므로 각 프로세서는 다른 것으로부터 완전히 독립하여 타스크處理를 진행하는 것이 不可能하고, 相互 데이터交換을 진행하면서 타스크處理를 진행하여야 한다[5]. 그러므로 複合마이크로프로세서 시스템에서는 시스템의 構成要素인 프로세서 사이에 어떠한 通信用버스를 준비하여서 여러개의 마이크로프로세서로 부터 메모리 액세스 要求의 競合(conflict)을 어떻게 解決할까가 문제가 된다[6]. 또한 그것을 사용하여 어떻게 데이터交換을 할까가 문제가 된다. 따라서 이와같은 문제점을 해결하기 위하여, 8085 CPU 4臺를 結合하여서, 主마이크로프로세서의 버스에 DMA 장치를 통하여 從마이크로프로세서에 접속되는 複合마이크로프로세서 시스템을 設計하였고, 이를 並列處理시킴으로서 CPU의 動作臺數와 시스템 Throughput 와의 관계 그리고 原始프로그램으로부터 並列性을 檢出하는 方法을 硏究報告한다.

2. 並列處理의 모델

並列處理의 모델은 그래프가 사용되는 경우가 많다. 플로우차아트등의 그래프적인 수단으로서 제산과정을 記述하는 일은 順序的인 일반 프로그래밍에 있어서도 이루어 지고 있다. 有向性그래프의 마디가 處理(프로세스)에 대응하여 마디 사이의 연결가지가 데이터 또는 制御情報의 흐름關係를 표시한다. 각 프로세스(process)가 實行되고 있는지 어떤지는 해당 프로세스의 制御情報의 有無에 따라서 판정된다. 프로세스는 實行이 끝나면 링크에 의하여 연결되어 있는 모든 프로세스에 제어정보를 보낸다. 프로세스는 모든 入力링크上에 제어정보를 얻으면 實行을 개시할 수 있다. 한꺼번에 몇개의 프로세스가 實行되어도 좋고, 각 프로세스의 실행시간에 제한은 없다. 프로세스의 실행결과 몇개의 변수 값이 바뀐다. 여기에서 並列處理의 모델로서 프로그램 스키이머를 정의 하면 다음과 같다.

定義 1 : 프로그램 스키이머 S 는 變數, $X_1, X_2, \cdots\cdots, X_n$의 集合 X, 프로세스 $P_1, P_2, \cdots\cdots, P_n$의 集合 P와, 콘트롤 C의 組合으로서

$$S=(X, P, C)$$

로 定義한다. C는 프로세스 사이의 實行順序 관계를 規定한다. 프로세스 P_i에는 入力의 集合 $I(P_i)$와 出力의 集合 $O(P_i)$가 있다.

定義 2 : 스키이머의 解析을 다음과 같이 定義한다.

1) 각 變數 X_i는 狀態의 범위, 즉 0, 1, −1, 2를 가진다.

2) 각 프로세스 P_i는 다음 2개의 함수를 가진다.

(a) 處理函數 F_{Pi} : $F_{pi}(I(P_i))=O(P_i)$.

(b) 制御函數 G_{pi} : 이 함수를 평가한 결과 다음에 실행해야 될 프로세스가 결정된다. 이 情報가 콘트롤 C로 사용된다.

定義 3 : 變數의 履歷 h_i는 실행과정에 있어서 變數 X_i가 취하는 값의 時系列이다. 즉, 변수 X_1이 1 → 0 상태로 변화 하였을 경우의 이력 h_i은 10이다.

變數가 메모리의 語(word)에 대응한다고 생각하면, 스키이머의 이력은 프로그램 실행중의 전 메모리의 語의 내용이 그 과정이 된다.

프로그램 스키이머의 例로서

$$S=(X, P, C),$$

단,

$X=\{a, b, c, d, e, f, g, h\}$,

$P=\{p_1, p_2, p_3, p_4\}$,

$C=\{1\rightarrow 2,\ 1\rightarrow 3,\ 2\rightarrow 4,\ 3\rightarrow 4\}$,

라고 하자. 이 해석의 例로서

각 변수의 값의 범위 :

Fp_1 : a = b + c

Fp_2 : d = a + e

Fp_3 : f = a − g

Fp_4 : h = d + f

로 한 경우를 그림 1에 표시한다.

프로그램 스키이머의 콘트롤 C의 정의 방법에 따라서 모델의 자세한 것이 결정된다. 여기에서는 각 변수에 대하여 상태가 정의된다. 狀態는 idle(0), ready(1), disabled(−1) 또는 blocked(2) 가운데

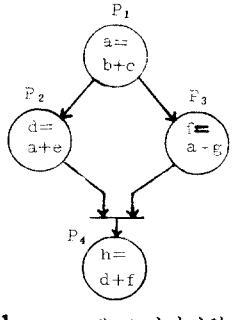

그림 1. 프로그램 스키이머의 例
Fig. 1. Example of program schema.

에서 하나의 값을 취한다. 프로세스는 入力 集合 가운데의 각 변수의 상태를 出力 集合 가운데의 각 변수의 상태로 변환한다.

그림 2는, 프로세스 a 는 입력변수 X_1 과 X_2 가 ready가 되면 실행을 개시하고, 실행이 끝나게 되면 출력변수 X_3 와 X_4를 ready로 하여서 입력변수를 idle로 한다. 프로세스 a 의 제어함수 Ga 는

$$Ga : \underline{X_1\ X_2\ X_3\ X_4} \quad \underline{X_1\ X_2\ X_3\ X_4}$$
$$1\ \ 1\ \ 0\ \ 0 \rightarrow 0\ \ 0\ \ 1\ \ 1$$

로 쓸 수 있다. 즉, 프로세스가 실행을 개시할 수 있는 것은 프로세스의 모든 입력변수가 ready가 되어야만 한다.

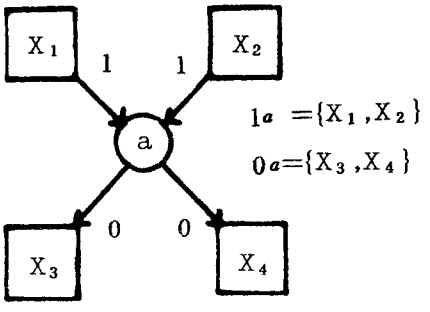

그림 2. 스키이머의 모델
Fig. 2. Schema model.

3. 並列性檢出앨고리즘

프로그래머가 並列處理가 필요한 프로그램을 작성하는 경우에 독립적으로 동작할 수 있는 비동기적인 타스크를 檢證하여서 그 並列性을 프로그램에 집어넣어야 한다. 그러나 타스크를 分配할 때에 그들 타스크가 동시에 똑같은 데이터를 액세스하는 일이 일어 날 수 있다. 이 때문에 병렬로 실행되는 프로세스의 하나가 共有데이터를 變更하고자 할때에 다른 프로세스로부터 그 데이터의 변경을 금지할 필요가 있다. 그러므로 直列로 作成된 프로그램으로부터 그 프로그램이 가지고 있는 並列性을 검출하는 수단이 요구된다. 여기에는 연산문레벨, 文레벨, 타스크레벨의 병렬성이 존재한다.

i) 演算文의 並列性檢出

論理的으로 표현하기 위하여 다음과 같은 연산을 例로 들어 論하고저 한다.

$$A + B + C + D * E * F + G + H$$

인 경우의 앨고리즘은 다음과 같은 형식으로 중간결과를 표시함으로서 실현한다.

R_i (演算數 1, 演算子, 演算數 2, 시작 레벨 = 끝 레벨)

똑같은 시작 레벨을 가진 모든 중간 결과는 並列로 計算할 수 있다. 처음에는 모든 함수 R_i의 시작 레벨, 끝 레벨은 제로에 세트시킨다.

스캔(scan)은 入力列의 右側의 연산자로 부터 시작하여서, 앞서 스캔된 연산자의 우선도 보다 낮은 우선도를 가진 연산자가 검출될 때 까지 右로 부터 左로 계속한다. 例의 연산에서는 이 스캔에 의하여 다음의 補助列을 얻는다.

$$D * E * F + G + H$$

다음에, 이 보조열에 대하여 스캔을 左로 부터 右로 실행하여서 左側의 연산자 보다 낮은 우선도가 검출될 때 까지 계속한다. 그리고, D＊E＊F를 얻는다. 이 시점에서 중간결과 R_1이 얻어진다.

$$R_1 (D, *, E, 0, 1)$$

중간결과 R_1은 演算數의 하나로서 바뀌어서 다음과 같은 새로운 補助列이 얻어진다.

$$R_1 * F + G + H$$

左로 부터 右에의 스캔이 중간결과가 만들어지지

입力 : A + B + C + D＊E ＊F + G + H

右 → 左스캔	左 → 右스캔
D＊E ＊F + G + H	R_1＊F + G + H
	R_2 + G + H
A+B+C+R_2+G+H	R_3 + C + R_2 + G + H
	R_4 + R_5 + R_2
	R_6 + R_2
	R_7

中間結果	演算數 1	演算子	演算數 2	시작	끝
R_1	D	*	E	0	1
R_2	F	*	R_1	1	2
R_3	A	+	B	0	1
R_4	C	+	G	0	1
R_5	H	+	R_3	1	2
R_6	R_4	+	R_5	2	3
R_7	R_2	+	R_6	3	4

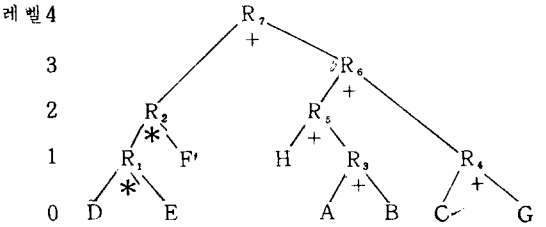

그림 3. (A+B+C+D＊E＊F+G+H) 의 앨고리즘
Fig. 3. (A+B+C+D＊E＊F+G+H) algorithm.

않을 때 까지 계속되고, 다음에 右로부터 左에의 스캔이 다시 이루어 진다. 이 처리의 결과를 그림3에 표시한다. 이 tree에서 同一레벨에 있는 演算이 並列處理가 可能하고, tree의 높이가 병렬처리에 요하는 스텝數이다.

ii) 文사이의 並列性檢出

이것은 프로그램文을 프로세스라고 생각하여 서로 依存關係가 없는 文을 병렬처리 시킨다.

프로그램 P를 代入文 S(i)의 順列, {S(1), S(2), ……, S(n)}, 이라고 하자. 단, 文番號 i는 해당 프로그램이 順序的으로 실행되는 경우의 實行順序에 대응한다. 즉, j < k 이면 S(j)는 S(k)보다 먼저 실행된다. 이 과정을 그림4에 표시한다.

프로그램 가운데의 變數名을 變更함에 의하여 並列性을 얻도록 할 수 있다.

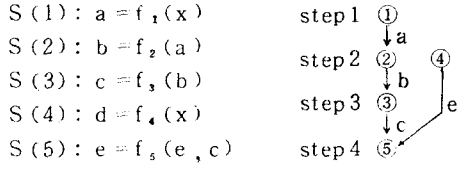

그림 4. 정당한 實行順序의 例
Fig. 4. Example of general execution procedure.

예를들면, 프로그램

S(1) : k = f_1(a)
S(2) : b = f_2(k)
S(3) : k = f_3(c)
S(4) : d = f_4(k)
S(5) : k = f_5(e)
S(6) : h = f_6(k)

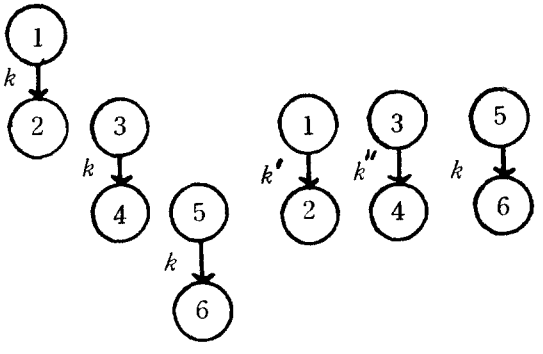

그림 5. 變數名 變更에 의한 並列性增大의 例
Fig. 5. Example of parallelism increase by changed variable name.

를

S(1) : k' = f_1(a)
S(2) : b = f_2(k')
S(3) : k'' = f_3(c)
S(4) : d = f_4(k'')
S(5) : k = f_5(e)
S(6) : h = f_6(k)

와 같이 수정하면 그림5와 같이 並列性이 증가한다.

iii) 타스크레벨의 並列性檢出

프로그램 가운데의 서브루우틴을 타스크로 간주하여서 서브루우틴 사이의 依存關係를 文레벨과 같은 方法으로 찾으면 타스크레벨의 병렬처리가 가능하다. 또, 서브루우틴內의 文레벨의 並列性으로 부터 적당한 文의 集合을 만들어서 이것을 task로 간주할 수도 있다. 그림6에 例를 표시한다.

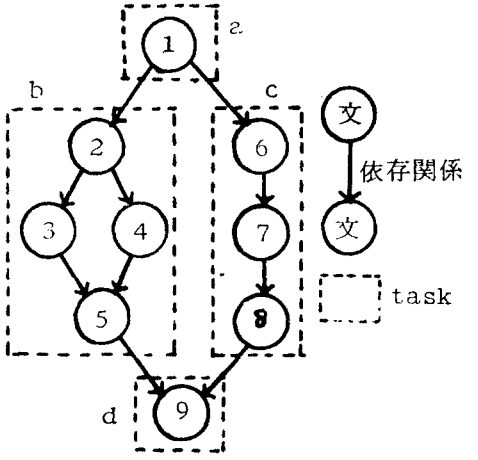

그림 6. task의 合成例
Fig. 6. Example of task composition.

4. 시스템의 構成要素 및 動作方式

複合마이크로프로세서 시스템을 구성하는 경우에 그 要素프로세서는 하드웨어적으로 똑같은 것이 價格性能比가 우수하다. 그러므로, 구입하기가 제일 쉽고, 應用에 적합하다고 생각되어 8085를 선택하여서 複合시스템을 設計하였다.

그림7에 블록도를 표시한다.

구성요소를 크게 나누면 다음과 같다.

a) Intel 8085 CPU 4臺

b) 메모리

(i) 각 CPU 固有의 局所메모리(Local Memory) 4臺(512 바이트)

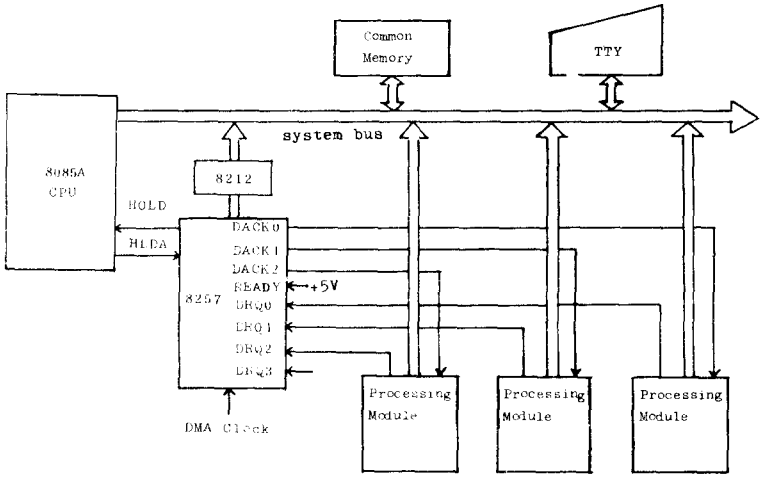

그림 7. 複合시스템의 블록도
Fig. 7. Block diagram of multiple system.

(ii) 각 CPU에 共通된 共有메모리(RAM 및 ROM) 1臺(2k 바이트),

c) DMA 콘트로울러 1臺

d) Teletypewriter(ASR 33)와 그 인터페이스

e) chip select logic

f) latch

g) wait 發生回路

h) 制御信號發生回路

i) 키이보오드와 그 인터페이스

이며, 이들은 5臺의 基板에 實裝되어 있다. 處理모듈(PM)을 그림 8에 표시한다. 각 PM은 CPU와 局所메모리(LM) 및 I/O로 되어 있고, DMA 콘트로울러(DMAC)를 通하여 통제되도록 하였다.

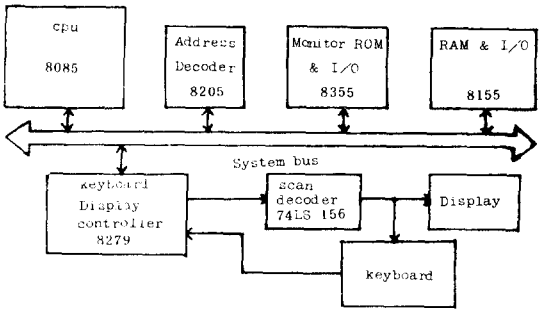

그림 8. 처리 모듈
Fig. 8. Processing module.

8085는 RIM命令과 SIM命令에 의하여 直列데이터를 그 端子로 부터 入出力할 수 있는 장점이 있으므로 TTY(ASR 33)를 사용하여 데이터를 전송하도록 하였다. 데이터轉送時 1 wait 를 얻기 위하여 SN 74574의 $\overline{Q}$를 8085의 READY단자에 접속함으로서 이루어 지도록 하였다.

共有메모리(CM)는 마이크로프로세서 사이의 通信 및 共有데이터의 格納등에 사용한다.

그러나 單一共通버스方式의 複合시스템을 생각하는 경우에는 여러개의 프로세서로 부터의 메모리 액세스 要求를 어떠한 順序로 받아서 버스上의 競合(conflict)을 防止할까가 문제가 된다[7]. 그러므로 본 시스템에서는 이를 解決하기 위하여 專用의 마이크로프로세서를 준비하여서, 이 主마이크로프로세서의 버스에 DMAC를 통하여 從마이크로프로세서에 접속되는 粗結合方式을 채택하여서 모듈 사이의 모든 데이터를 주고 받도록 設計하였다. 여기에서 만약에 主마이크로프로세서가 障害를 일으키면 시스템 전체의 운전이 정지될 염려가 있으므로 독립적으로 동작하여 각각 자신의 運營體制(OS)를 가지도록 하여서 從을 主로 바꾸어서 운전을 계속할 수 있도록 하여서 융통성을 증가 시키도록 하였다. 메모리 액세스 競合을 감소시키기 위하여 가능한 메모리 액세스를 PM에 分散시키고, 타스크를 프로세서에 分配하면 그 위에서 완전히 처리가 끝날 때 까지 走行하도록

하여서 중간에 데이터 交換을 가급적 하지 않도록 하였다.

마이크로프로세서 사이의 通信은 交信바이트數에 따라서 數바이트의 命令과 數十～數百바이트로 나눌 수 있다. 본 시스템에서는 DMAC가 PM로 부터 메모리 액세스를 要求받으면, 主마이크로프로세서에 HOLD를 요구하고, 시스템버스의 점유권을 얻어서 가장 우선도가 높은 PM에 응답신호를 보내고, 그 다음에 목적의 어드레스, 바이트數, R/W의 구별등을 共有메모리의 각 프로세서에 할당된 고정영역에 데이터를 격납하므로서 通信이 이루어 지도록 한다. 마이크로프로세서 사이의 通信과정을 그림 9에 표시한다.

전실험 시스템을 그림 10에 표시한다.

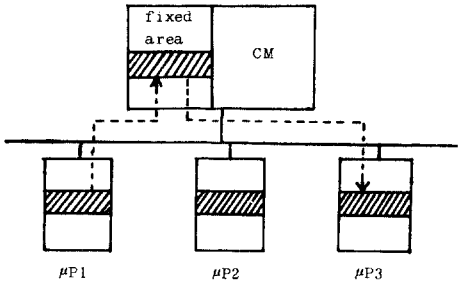

그림 9. 마이크로프로세서 사이의 通信
Fig. 9. Communication between microprocessors.

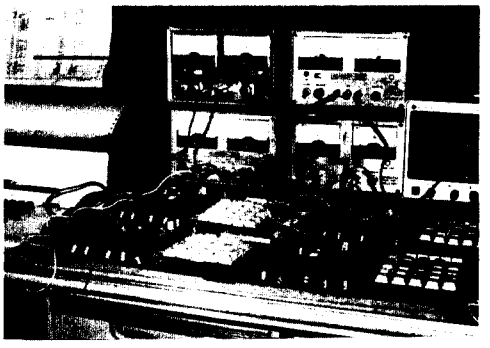

그림 10. 전 실험 시스템
Fig. 10. All experiment system.

5. 응용프로그램

본 試作시스템과 같이 複合마이크로프로세서 시스템을 구성하는 경우에는 하드웨어적인 속도는 병렬처리 시킴으로써 올릴 수 있으나 우선 문제가 되는 것은 연산능력의 저하이다. 본 시스템의 병렬처리 알고리즘과 같은 연산을 포함한 여러가지의 데이터처리에는 매우 복잡한 數値演算이 필요하게 되어서 効率이 좋은 프로그램을 작성하여야 한다. 따라서, 여기에서는 매우 복잡하리라 생각되는 10進8行 정도의 가감승제산을 처리시키는 효율적인 프로그램을 작성하여서 複合시스템에 동작 시켜 보고자 한다. 앞서 3장에서와 같은 방법으로 並列性을 검출하여 보았으나 條件分岐命令이나 轉送命令이 90퍼센트 이상이 되므로 연산문레벨이나 文레벨의 병렬성은 메모리 액세스 경합이 많아지므로 여기에서는 타스크 크기를 서브루우틴레벨로 하여서 가급적 처리노드에 分散시키는 방향으로 프로그램을 작성하였다. 프로그램량은 362스텝이었으며, 프로그램에 사용된 記憶量은 665바이트 이었다.

6. 結果 및 考察

앞서의 應用프로그램을 사용하여서 動作마이크로프로세서臺數를 1臺부터 4臺까지 실행시킨 경과의 處理速度는 다음과 같다. 처리속도는 8155의 타이머로서 각 命令의 실행에 따라서 모니터의 타이머 인터러프트를 통하여 14비트의 카운터/타이머로 측정하였다.

1臺로서는 11,407 ms 이었으나, 2臺로서는 6,713 ms, 3臺로서는 4,875 ms, 4臺로서는 4,321 ms의 계산시간이 소요되었다.

이를 퍼센트로 나타내면 다음과 같다.

	한대당의 Throughput	전체의 Throughput
μP 1대	100 %	100 %
μP 2대	85 %	170 %
μP 3대	78 %	234 %
μP 4대	66 %	264 %

위의 표에서 마이크로프로세서台數와 速度向上率의 관계를 그래프로 나타내면 그림 11과 같다.

複合마이크로프로세서 시스템의 性能은 理想的으로는 n臺의 마이크로프로세서를 사용함에 의하여 1대의 마이크로프로세서에 비하여 n倍의 Throughput가 얻어진다. 그러나 理想値의 70～80 %선에 머무름을 확인할 수 있었다.

이의 原因으로 생각되는 것은,

i) 병렬처리에 수반되는 overhead

ii) 시스템 Deadlock

iii) 병렬처리에 적합하지 않은 부분의 처리를 들 수 있다.

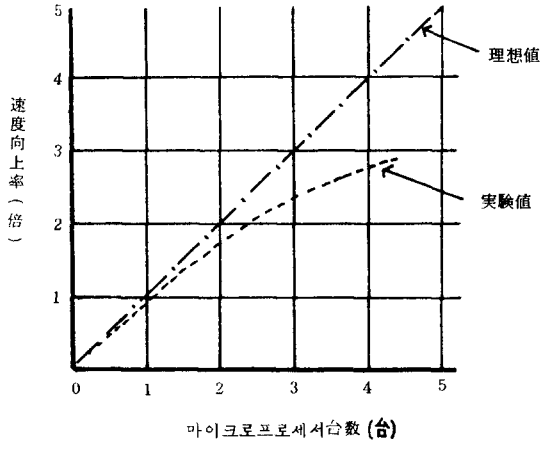

그림 11. 마이크로프로세서台數와 速度向上率
Fig. 11. A number of working microprocessor VS speed elevation ratio.

7. 結 論

마이크로프로세서 4臺를 사용하여서, 主마이크로프로세서의 버스에 DMA 콘트로울러를 통하여 從마이크로프로세서에 접속되는 複合시스템을 設計하여서 병렬처리 시킴으로서 處理速度를 向上시킬 수 있었다. 그러나 理想値를 밑도는 결과이었다. 이의 가장 주된 원인은 資源의 불균형 할당에 의한 시스템 Deadlock 이라고 생각되므로 이를 방지하기 위한 방법을 좀더 모색하여야 겠다.

資源의 分配나 日程計劃(scheduling)은 각 마이크로프로세서가 實行하는 것이 불리하므로, 본 시스템에서는 主마이크로프로세서를 制御마이크로프로세서로 사용하여서 원할히 이루도록 하였다.

마이크로프로세서 사이의 共有데이터의 格納은 共有메모리를 사용하도록 하였다.

또한, 並列處理를 의식하지 않고 FORK-JOIN 命令, Test and Set 命令 그리고 Semaphore와 같은 것이 준비되지 않은 8085와 같은 시스템에서의 병렬처리는 DMA 콘트로울러가 우선순위를 부여하여 통제하는 것이 좋음을 확인 하였다.

參 考 文 獻

1. 相磯秀夫：ユンピュータ・アーキテクチャ(2), エレクトロニクス, pp. 153-154(1973年 9月號).
2. C. G. Bell, et al: C. mmp-A multi miniprocessor, Proc. AFIPS FJCC, Vol. 39, pp. 765-777(1972).
3. K. Ohmori, et al: MICS A multi-microprocessor system Proc. AFIPS FJCC, Vol. 41, pp. 98-102(1974).
4. G. H. Barnes, et al.: The ILLIAC IV Computer, IEEE Trans. Computer, Vol. c-17, No. 8, pp. 746-757(1968).
5. A. J. Bernstein: Analysis of Programs for Parallel Processing, IEEE Trans. Computer, Vol. c-15, No. 10, pp. 757-762(1966).
6. K. T. Fung, et al.: On the Analysis of Memory Conflict and Bus Contentions in a Multiple Microprocessor System, IEEE Trans. Computer, Vol. c-27, No. 1, pp. 28-37(1979).
7. C. K. Wong: Parallel Genaration of Binary Search Trees, IEEE Trans. Computer, Vol. c-23, No. 3, pp. 268 271(1974).
8. J. E. Juliussen, et al.: Multiple Microprocessors with common main and control memories, IEEE Trans. Computer, Vol. c-22, No. 11, pp. 999-1007(1973).
9. P. j. Denning, et al.: Operating System Theory, Prentice-Hall, pp. 83-128(1973).
10. A. N. Haberman: Synchronization of Commucating Processes, CACM, Vol. 15, No. 3, pp. 171-176(1972).
11. J. W. Havender: Avoiding Deadlocks in Multi-Tasking Systems, IBM System Journal, Vol. 2, No. 2, pp. 74-84(1968).
12. B. W. Lampos: A Scheduling Philosophy for Multi-Processing Systems, CACM, Vol. 11, No. 5, pp. 347-360(1968).