'88 韓國自動制御學術會議論文集 1988. 10. 21~22

로봇 운동 제어의 실시간 연산을 위한 병렬처리구조

○고 경 철　　조 형 석

한국과학기술원 생산공학과

A Proposed Parallel Processing Structure For Robot Motion control

K.C. KOH　　H.S. CHO

Dept. of Production Engineering, KAIST

ABSTRACT

The realization of high quality robot control needs the improvement of computing speed of controller. In this paper, parallel processing method is considered for this purpose. A S/W algorithm for task scheduling is developed first, and then, an appropriate H/W structure is proposed. This scheme is applied to calculate inverse kinematics of PUMA robot. The simulation results show that the computing time when using three 8086/87's is reduced to 4.23 msec compared to 10 msec in case using one 8086/87.

1. 서 론

최근 로봇에 관한 연구방향은 로봇의 고속화와 정밀성을 추구하는 측면과 센서및 지능성을 부여하는 측면으로 나눌 수 있으며, 이에 따라 로봇의 제어기능의 중추적 역할을 담당하는 CPU부의 연산기능의 강화및 I/O의 실시간 처리 능력이 절실히 요구되고있다. 특히 로봇의 속도가 높아질수록 다이나믹 커플링효과가 커지고, 운동궤적의 경로오차가 커지게 된다. 따라서 로봇의 제어알고리즘에는 다이나믹 보상이 이루어 져야 하며, 경로계획에서는 좀 많은 중간경로점을 계산해 주어야 할 필요가 있다. 이는 로봇 제어기의 설계시, 좀더 많은 비트수를 갖는 CPU의 채용이나, 시스템클럭을 높여야하는 것을 의미하나, 실제 이방법은 한계가 있다. 또 다른 방법으로는 실수연산 기능이 강화된 어레이 프로세서의 사용이 있으나, 설계상의 어려움이 따르며, 유연성이 떨어지는 단점이 있다. 그러므로 기존의 CPU 를 여러개 묶어서 사용하는 병렬처리 (Parallel Processing) 방법이 효율적이며 현재 이 방법을 이용한 로봇제어기의 설계에 관한 연구가 활발하다. 이미 여러개의 CPU를 이용한 로봇제어의 구조는 상용화된 로봇에서도 볼 수 있다. 이는 그림.1과 같이 SUPERVISOR용 CPU, 경로계획연산용 CPU, 축제어용 CPU등으로 구성되는 구조로서 하나의 CPU가 하나의 연산블럭을 전용으로 계산하는 것으로 되어있다. 그러나 이 구조는 실제연산은 병렬로 수행되나, 데이터흐름 (Data Stream)은 직렬인 MISD (Multi Instruction Single Data stream) 구조로써, 일명 Pipelined Structure라고 볼 수 있다. 그러나 실제 샘플링주기 (Sampling Rate) 를 줄이기 위해서는, 각 연산블럭 자체의 계산을 빠르게 해야하며, 따라서 하나의 연산 블럭을 여러개의 Sub-Task들로 나누어 이를 병렬처리하는 MIMD(Multi Instruction Multi Data stream)구조가 바람직하다. 로봇의 운동제어에 있어 이러한 MIMD방식의 병렬처리에 관한 연구를 살펴보면, T.Watanabe가 Elbow Robot의 역 Kinematics변환을 7개의 CPU로써 수행함에 있어, 각 CPU간의 동기를 위한 H/W구조및 Task Scheduling 알고리즘을 제안한 바 있고[1], J.S. Luh등의 Branch and Bound Method를 수정한 Task Scheduling 알고리즘이 있다.[2] 그러나 위의 연구들은 주로 CPU의 갯수를 정해 놓은 상태에서의 알고리즘및 하드웨어의 통신방법에 관한 것이고, 하나의 연산블럭을 여러개의 CPU로 병렬로 계산할 때, 가장 효율적이며 계산시간을 줄일 수 있는 CPU갯수를 정하는 문제를 고려하지 않았다. 따라서 본 논문에서는 PUMA ROBOT의 역 Kinematics 변환을 병렬처리할 연산블럭으로 선정하여, 새로이 제안한 Sub-optimal Task Scheduling 알고리즘을 전산시뮬레이션을 통해 그 유용성을 분석하고, 효율성과 계산시간의 단축성을 고려한 CPU갯수를 정하는 방법을 제시하며, Sub-Task들의 수행 우선순위를 정하여 이를 처리하는 H/W구조에 관해 기술하고자 한다.

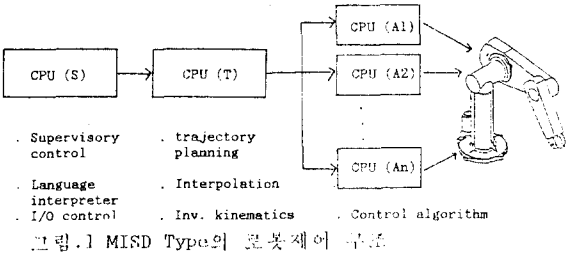

그림.1 MISD Type의 로봇제어 구조

2. 병렬처리에서의 고려해야 할 점

주어진 Task를 여러개의 CPU를 이용하여 병렬계산을 하고자 할 때, 다음과 같은 점들을 고려한다.

1) Maximum Parallel Rate (Rp)max

병렬처리하고자하는 Task의 총 수행시간을 Tt라 하고, 이를 병렬로 계산하였을 때의 수행시간을 Tp라 할 때, Parallel Rate(Rp)는 다음식으로 구해진다.

$$Rp = Tt / Tp \qquad (1)$$

그리고 Critical Path를 따르는 Task들의 수행시간의 합을 Tc라 할 때, 아무리 CPU수를 증가시켜도 Tp는 Tc이하로 줄어들 수 없으므로 주어진 Task의 최대 Rp인 (Rp)max를 다음과 같이 구할 수 있다.

$$(Rp)max = Tt / Tc \qquad (2)$$

이같이 구한 (Rp)max는 병렬처리 하고자 하는 Task의 병렬정도를 나타내는 값으로 이 값이 클수록 많은 CPU를 채택할수있으며, 따라서 병렬계산시간

(Tp) 을 더욱 짧게 할 수 있는 것이다. 이 때 Nc를 Tp가 Tc되는 최소의 CPU갯수라 할 때, 다음의 조건을 만족해야 한다.

$$Nc > (Rp)max \qquad (3)$$

2) Efficiency of CPU (Ec)

앞서 살펴 본 바 와같이 대상Task의 Tc에 따라 그 한계는 있지만, CPU수 를 늘릴 수록 Tp를 짧게 할 수 있다. 그러나 반면 CPU의 대기 시간이 길어지기 때문에, 그 경제성과 효율은 감소한다. 여기서 효율(Ec)는 다음과 같이 정의한다.

$$Ec = Tt / (N \cdot Tp) \qquad (4)$$

따라서 CPU의 갯수를 정하는 문제에 있어 Parallel Rate(Rp)와 효율(Ec)를 모두 고려하는 타협점 (Trade-off) 을 찾는 기법이 필요하다.

3) Task Scheduling Algorithm

하나의 Task를 여러개의 Sub-Task로 나누어 병렬처리하려면 주어진 CPU갯수(N) 에 이들을 적절히 배치하는 Scheduling Algorithm이 필요한데, N이 Nc인 경우, Optimal한 Schedule을 찾을 수 있으나, N이 Nc보다 작은 경우에 대한 Optimal Schedule을 찾는 것은 대단히 어려운 문제이다. 이는 각 Sub-Task들 간의 상관관계, 각 Task의 계산시간(Ts), Task의 Level에 따라 그 Optimal Schedule이 달라지기 때문이며, 더구나 실시간연산의 경우 각 Task의 계산시간(Tp)은 실제 데이터의 값에 따라 변화하기 때문에 더욱 더 그러하다. 그러므로 본 논문에서는 Sub-optimal Schedule을 찾는 알고리즘을 제시하고자 한다.

3. PUMA ROBOT의 Inv. Kinematics 식의 병렬처리도 분석

PUMA ROBOT의 역 Kinematics식을 기하학적방법을 통해 해를 구하면 부록.1과같다.[6] 이식은 35개의 실수덧셈, 46개의 실수곱셈/나눗셈, 4개의 제곱근, 11개의 초월함수 식으로 이루어져있으며, 이를 하나의 8086/87 시스템으로 연산할 때, 약 10mSEC의 시간이 필요하다. 그러나 이 연산블럭을 병렬처리하고자 할 때, 여러개의 Sub-task로 나누어 이들의 상관관계를 분석할 필요가 있다. 우선 Sub-task를 다음과 같은 기준에 의해 분류한다.

1) 두번이상 사용되는 항(Term)
2) 병렬로 처리될 수 있는 항.

이러한 기준에 의해 부록.1의 식들을 표.1과 같이 44개의 Sub-task로 정리할 수 있으며, 각 Task간의 상관관계는 Pre-task로 표현된다. 여기서 Pre-task는 하나의 Task가 수행되기전에 미리 수행되어야 할 Task로 정의하며, 만약 그 Pre-Task들이 같은 Path상에 존재하면, 최종의 Task가 Pre-task로 대표된다. 이 표.1에서 각 Task의 계산시간 (Ts)은 8087 Co-processor (4MHz)를 기준으로 산정하고, 실제 연산시 발생하는 데이터 Load/Store를 모두 고려한다. 그림.2는 이러한 44개의 Task간의 상관 관계를 도식적으로 보여주는데, 여기서 화살표는 Pre-task의 관계를 표시하고, 이중원은 Critical Path상의 Task를 표시한다.

4. Task Scheduling 알고리즘

여기서 타스크 스케듈링이란 병렬처리될 Sub-Task들을 수행될 순서를 정하여 각 각 CPU에 배정하는 것을 의미한다. 본 연구에서 제안한 알고리즘은 다음의 5단계로 이루어 진다.

No. of Task	Contents of Calculation	Pre-Tasks	Ts(μSEC)
1	A(1)=Px*Px		70
2	A(2)=Py*Py		70
3	A(3)=Pz*Pz		70
4	A(4)=A(1)+A(2)	1,2	67
5	A(5)=A(4)-d2*d2	4	67
6	A(7)=SQR(A(5))	5	73
7	A(8)=-Py*A(7)-Px*d2	6	179
8	A(9)=-Px*A(7)+Py*d2	6	179
9	TH(1)=ATAN2(A(8),A(9))	7,8	400
10	A(11)=A(5)+A(3)	3,5	67
11	A(12)=SQR(A(11))	10	73
12	A(13)=-Pz/A(12)	11	116
13	A(14)=-A(7)/A(12)	6,11	116
14	A(15)=(A(11)+b1)/(b2*A(12))	11	176
15	A(16)=SQR(1-A(15)*A(15))	14	125
16	A(17)=A(14)*A(13)-A(15)*A(16)	12,13,15	161
17	A(18)=A(14)*A(15)-A(13)*A(16)	12,12,15	161
18	TH(2)=ATAN2(A(17),A(18))	16,17	400
19	A(19)=(b3-A(11))*b4	10	114
20	A(20)=-SQR(1-A(19)*A(19))	19	143
21	A(21)=A(20)*b6-A(19)*b5	20	161
22	A(22)=A(19)*b6+A(20)*b5	20	161
23	TH(3)=ATAN2(A(21),A(22))	21,22	400
24	A(23)=COS(TH(1))	9	546
25	A(24)=SIN(TH(1))	9	549
26	A(25)=TH(2)+TH(3)	18,23	67
27	A(26)=COS(A(25))	26	546
28	A(27)=SIN(A(25))	26	549
29	A(28)=A(23)*A(26)	24,27	83
30	A(29)=A(24)*A(26)	25,27	83
31	A(30)=-Ax*A(28)-Ay*A(29)+Az*A(27)	28,29,30	257
32	A(31)=-A(23)*Ay+A(24)*Ax	24,25	179
33	TH(4)=ATAN2(A(31),A(30))	31,32	400
34	A(32)=COS(TH(4))	33	546
35	A(33)=SIN(TH(4))	33	549
36	A(34)=-A(30)*A(32)-A(31)*A(33)	34,35	179
37	A(35)=A(27)*(A(23)*Ax+A(24)*Ay)+A(26)*Az	24,25,27,28	286
38	TH(5)=ATAN2(A(34),A(35))	36,37	400
39	A(36)=-(A(24)*A(32)+A(28)*A(33))	34,35	179
40	A(37)=A(23)*A(32)-A(29)*A(33)	34,35	161
41	A(38)=A(33)*A(27)	35	83
42	A(39)=A(36)*Nx+A(37)*Ny+A(38)*Nz	39,40,41	239
43	A(40)=A(36)*Sx+A(37)*Sy+A(38)*Sz	39,40,41	239
44	TH(6)=ATAN2(A(39),A(40))	42,43	400

표.1 PUMA Robot의 Inv.Kinematics식의 Sub-task

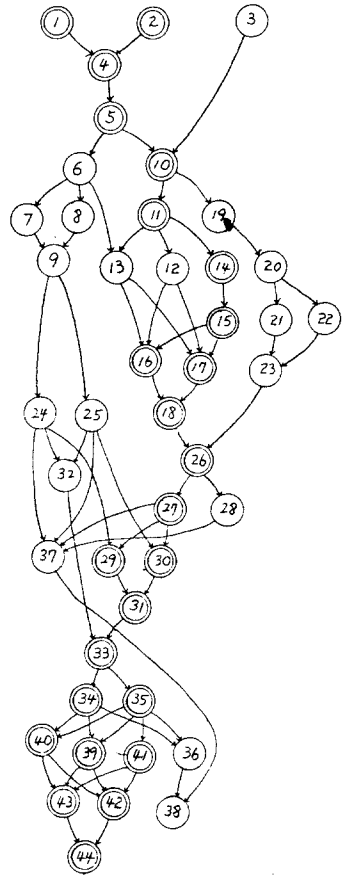

그림.2 Sub-task의 Tree 구조

(단계 1): Task Level의 결정

타스크 레벨은 타스크의 종속적인 관계를 표현하는 값으로 다음과 같은 룰에 의 해 결정 한다.

룰(1) : Critical Path상의 타스크는 레벨이 유일하게 결정되며, 다음과 같은 식으로 부터 구해진다

$$Lt = Max(Ltp1, Ltp2, \ldots) + 1 \qquad (5)$$

여기서 Lt는 타스크의 레벨이고 Ltpi(i=1,2,..)는 선행 타스크(Pre-Task)의 레벨이다.

룰(2) : Critical Path상에 있지 않는 타스크는 레벨이 일정치 않고 다음식과 같이 최대값과 최소값을 갖는다.

$$(Lt)min = Max(Ltp1, Ltp2, \ldots) + 1 \qquad (6)$$
$$(Lt)max = Min(Lts1, Lts2, \ldots) - 1 \qquad (7)$$

여기서 Ltsi(i=1,2,..)는 후행 타스크(Post-task)의 레벨이다.

룰(3) : 선행 타스크가 없는 경우 (Lt)min은 1로 하고 후행 타스크가 없는 경우 (Lt)max는 가장 큰 Lt로 한다.

이와같은 룰에 의해 그림.2의 타스크들의 레벨을 정해보면, 1번 타스크는 룰(1)과 룰(3)에 의해 1이되며, 4번 타스크는 룰(1)에 의해 2가 된다. 12번 타스크는 룰(2)에 의해 최소값이 6이고 최대값이 7이된다. 이 때 레벨값이 여럿이 있을 수 있는 타스크의 레벨을 우선 최소값으로 놓는다.

(단계 2) : 타스크의 우선순위(Priority) 결정

일단 각 타스크의 레벨이 결정되면, 그 값이 작은 타스크가 모두 먼저 수행되고 난후 다음 레벨의 타스크가 수행되어야 한다. 따라서 레벨값이 가장 작은 타스크부터 CPU 배정우선권을 갖는다. 그리고 같은 레벨의 타스크가 여럿이 될 경우, 타스크의 계산시간(Ts)이 긴 타스크가 더 높은 우선권을 갖는다. 이러한 룰에의해 표.1의 44개의 타스크의 우선순위를 정한다.

(단계 3) CPU 상태(State) 결정

타스크의 우선순위가 정해지면, 일단 가장 우선순위가 높은 타스크, 즉 1 레벨의 가장 계산시간(Ts)이 긴 타스크부터 배정권을 얻게된다. 그러면 남은 문제는 그 타스크를 어느 CPU에 배정하는가 하는 것이다. 이를 결정하기위한 사전 정보로서 각 CPU의 상태를 알아 볼 필요가 있다. 각 CPU의 상태는 다음의 3가지로써 정의한다.

가) 완료상태 (Completed state) : 선행타스크의 수행이 막 완료된 상태
나) 대기상태 (Waiting state) : 선행타스크가 이미 완료되어 대기중인 상태
다) 분주상태 (Busy state) : 선행타스크가 진행중인 상태

(단계 4) 타스크의 배정(Scheduling)

CPU의 상태가 결정되면 다음과같은 룰에 의해 타스크를 CPU에 배정한다.

(룰1) - 완료상태의 CPU가 있으면, 무조건 배정.
(룰2) - 대기상태의 CPU가 있으면, 가장 대기시간이 짧은 CPU에 배정.
(룰3) - 분주상태의 CPU가 있으면, 가장먼저 완료될 CPU에 배정.

여기서 룰은 번호 순으로 우선적용한다. 예를 들어 완료상태의 CPU와 분주상태의 CPU가 동시에 존재하면 (룰1)이 (룰3)보다 우선 하므로 (룰1)의 적용을 받는다. 이러한 룰에의해 한 타스크가 배정되면, 다음 우선순위의 타스크에 대해, (단계 3) 및 (단계 4)를 되풀이하여 배정을 하고, 우선순위가 매겨진 모든 타스크의 배정을 마칠 때 까지 계속한다.

(단계 5) 타스크 레벨의 변경

앞서 (단계 1)의 타스크의 레벨결정시 여러 레벨을 갖는 타스크의 레벨을 우선 최소값으로 한 바 있다. 이러한 레벨하에 (단 4)까지 거치면서 모든 타스크의 배정이 완료되면 병렬수행시간(Tp)가 결정되는데 이는 각 CPU의 마지막 타스크의 수행완료시간중 가장 긴시간으로 구해진다. 그러나 레벨을 달리하면 각 타스크들의 우선순위가 달라지고, 따라서 CPU 배정순위가 달라지므로 전체 병렬수행시간(Tp)가 달라지게되는데 원하는것은 가장 짧은 Tp를 갖는 스케줄을 찾는 것이다. 따라서 레벨을 달리할 수 있는 타스크의 레벨을 변경하여 (단계2)에서 (단계4)를 되풀이 하여 병렬시간을 조사하고 그값이 최소가 될 때 까지 반복한다.

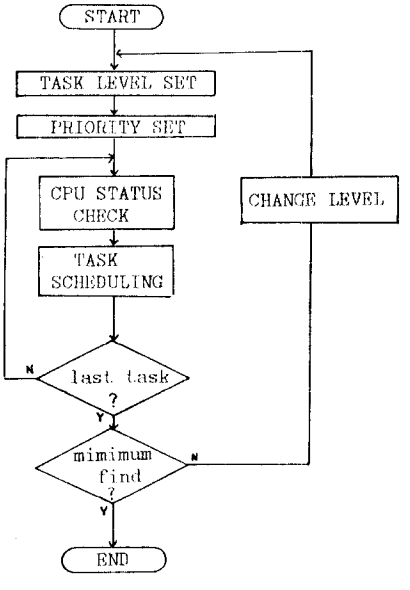

Fig.3 Flow chart of proposed task scheduling S/W

5. 전산 시뮬레이션

위에서 소개한 타스크 스케줄링 알고리즘을 전산 시뮬레이션을 통해 그 효용성을 고찰하고자 한다. 대상은 표.1과 그림.2로 표현되는 PUMA 역 Kinematic변환을 푸는 것으로 하였다. 4장에서 기술한 알고리즘을 흐름도(Flow Chart)로 표시하면 그림.3과 같다. 이 알고리즘을 적용시 고려할 점은 레벨을 변경할 타스크를 정하는 것이다. 그림.2의 타스크구조를 살펴보면 Critical Path상에 있지 않은 타스크는 19개나 되어 이를 모두를 레벨을 변화시켜 알고리즘을 적용하는것은 천문학적인 반복횟수가 필요하여 불가능하다. 따라서 레벨변경 가능 타스크를 줄여서 선정해야 한다. 그런데 병렬수행시간에 큰 영향을 미치는것은 비교적 계산시간이 긴 타스크의 레벨을 어디에 두는가에 있으므로, 표.2에 정리한 6개의 타스크를 선정한다. 이 6개의 타스크들은 표.2에 표기한 각 타스크의 레벨범위에 따라 총 440가지의 경우의 수를 갖는다. 이에 대해 알고리즘을 각각 적용하여 가장 짧은 병렬수행시간(Tp)을 찾는다. 이러한 방법으로

알고리즘을 전산 시뮬레이션을 통해 얻은 Sub-optimal 스케줄링 결과를 표.3에 표시한다. 이때 CPU는 Intel 8086/87을 사용하는것으로 가정하고 CPU를 3개 쓰는것으로 하였다. 표.3와 같이 스케줄된 타스크들은 각각 수행될 CPU및 시작시간과 완료시간이 정해진다.

Task No.	Minumum Level	Maximum Level	Ts(μSEC)
9	6	10	400
24	Level(9)+1	11	546
25	Level(9)+1	11	549
23	8	9	400
28	11	12	549
38	17	18	400

표.2 병렬처리시간에 영향이 큰 타스크

CPU	Task No.	Start time	End time
1	1	.0000	70.0000
1	4	70.0000	137.0000
1	5	137.0000	204.0000
1	6	204.0000	277.0000
1	7	277.0000	456.0000
1	20	456.0000	599.0000
1	9	599.0000	999.0000
1	23	999.0000	1399.0000
1	24	1399.0000	1945.0000
1	32	1945.0000	2124.0000
1	30	2124.0000	2207.0000
1	31	2207.0000	2464.0000
1	33	2464.0000	2864.0000
1	35	2864.0000	3413.0000
1	36	3413.0000	3592.0000
1	38	3592.0000	3992.0000
2	2	.0000	70.0000
2	10	204.0000	271.0000
2	8	277.0000	456.0000
2	12	458.0000	574.0000
2	13	574.0000	690.0000
2	22	690.0000	851.0000
2	17	920.0000	1081.0000
2	18	1081.0000	1481.0000
2	26	1481.0000	1548.0000
2	27	1548.0000	2094.0000
2	29	2094.0000	2177.0000
2	40	3413.0000	3574.0000
2	41	3574.0000	3657.0000
2	42	3657.0000	3896.0000
2	44	3896.0000	4296.0000
3	3	.0000	70.0000
3	19	271.0000	385.0000
3	11	385.0000	458.0000
3	14	458.0000	634.0000
3	21	634.0000	795.0000
3	15	795.0000	920.0000
3	16	920.0000	1081.0000
3	25	1081.0000	1630.0000
3	28	1630.0000	2179.0000
3	37	2179.0000	2465.0000
3	34	2864.0000	3410.0000
3	39	3413.0000	3592.0000
3	43	3657.0000	3896.0000

(Time unit:μSEC)

Table.3 The result of Task Scheduling with 3 CPUs

6. 병렬처리 시스템의 설계

1) CPU 갯수의 결정

서론에서 언급한 바와 같이 가장 효율성이 높으면서 병렬수행시간을 짧게 할 수있는 CPU수를 정하는 일이다. 그림.4는 제안된 알고리즘을 사용하 CPU수를 변화시켜가면서 병렬수행시간(Ts)을 조사한 결과이다. 그림에서 O표시는 440가지의 레벨조합중 레벨결정이 가장 잘된경우를,X표시는 가장 안된 경우를 나타낸다. CPU갯수(N)가 6이면 레벨변경 가능 타스크의 레벨위치에 관계없이 최소의 병렬수행시간을 얻을 수있는 스케줄을 얻을 수 있고, N이 5인경우 본 알고리즘을 통해 레벨을 결정하면 최소 Ts를 얻을 수 있음을 알 수있다. 그림.5는 이에따른 CPU의 효율(Ec)을 조사한것으로 CPU수를 증가시킬 수록 효율이 감소함을 보여준다. 그림.6은 병렬도를 나타내며 CPU수에 따라 증가하다가 N=6에서 일정해짐을 보여준다. 그림.7은 다음과 같이 정의한 유효병렬도(Re) 곡선으로 N=3에서 최대임을 보여준다.

$$Re = Rp \cdot Ec \qquad (8)$$

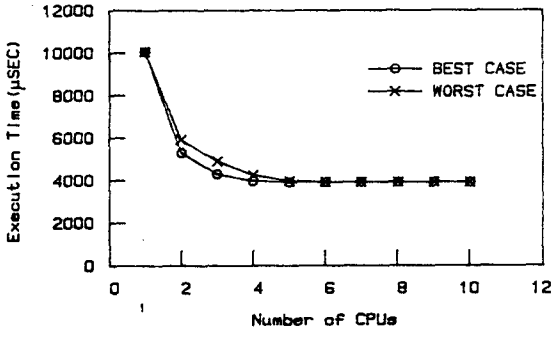

Fig.4 Execution Time on No.of CPUs

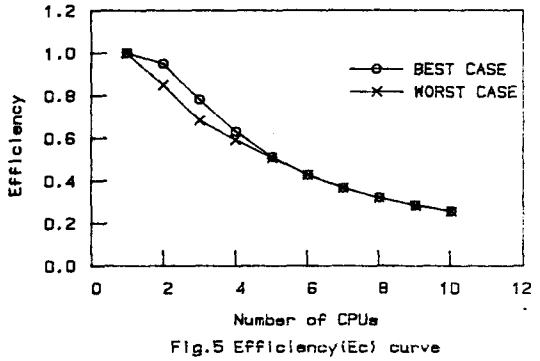

Fig.5 Efficiency(Ec) curve

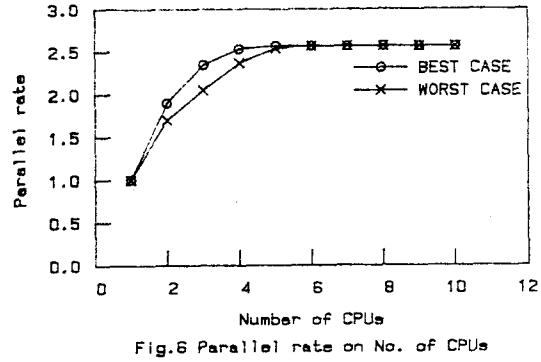

Fig.6 Parallel rate on No. of CPUs

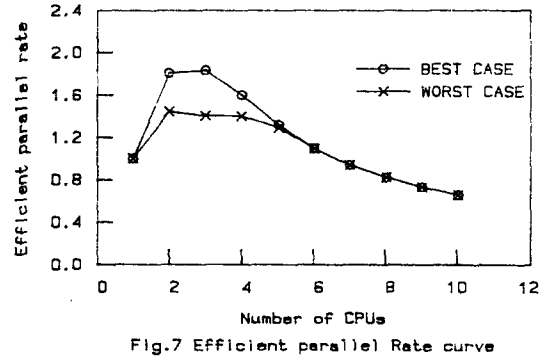

Fig.7 Efficient parallel Rate curve

따라서 경제적면서 병렬처리효과를 크게 얻을 수있는 CPU 수(N)는 유효병렬도(Re)가 최대인 3개로 선정한다. 즉 N이 3이면 그림.5에서 보다시피 N이 6일 때에 비해 Tp는 170 μSEC밖에 늘어나지 않은 반면 효율은 42.78%에서 78.13%로 크게 증가함을 알 수있다.

(2) CPU간의 통신

병렬처리 시스템에서 CPU간의 통신은 상당한 시간을 소요하며 이들을 연결해주는 Net-work의 형태가 효율적이지 못할 때는 더욱 그러하다. 예로써 N개의 CPU가 단일한 공용버스로 연결되어 사용된다면, Bus Contention으로 인하여 대단히 CPU간의 통신이 비효율적이 된다. 이와는 반대로 모든 CPU들을 서로 완전하게 연결(Completely connected)하는 경우, N(N-1)/2개의 연결선이 필요하게 되어 실현이 힘들어 진다. 따라서 로봇의 운동제어를 위한 병렬처리에 적합한 하드웨어 구조로서, 그림.8과 같이 CPU의 수행상태를 파악하여 타스크들의 수행여부를 정하여 주는 구조를 제안한다.

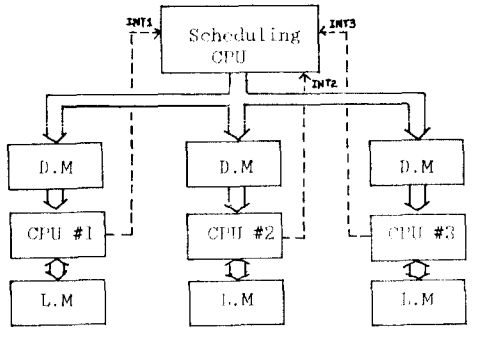

그림.8 제안된 하드웨어 구조

여기서 Scheduling CPU는 앞서 스케쥴링한 타스크를 수행 시작시간순으로 메모리 버퍼에 저장하고, 수행CPU부터 인터럽트를 통해 CPU의 상태를 판단하고, 선행타스크의 완료여부에 따라 타스크를 배정한다. 이때 전달되는 정보는 타스크번호및 선행타스크에서 계산된 데이터로 D.M (Dual port memory)을 통해 수행 CPU에 보내지게 된다.

7. 결 론

이상에서 살펴본 바와 같이 로봇의 운동제어에 있어 실시간 버든(Burden)이 가장 큰 역 Kinematics 변환을 병렬처리의 예로들어 본 연구에서 제안한 타스크 스케쥴링 알고리즘을 전산 시뮬레이션을 통해 그유용성을 알아 보았다. 또한 효율과 병렬 처리 시간의 단축을 고려한 CPU수를 정하는 방법을 제시하였고, CPU간의 통신을 위한 하드웨어구조를 제시하였다. 본 연구에서 제시한 알고리즘은 CPU갯수를 Critical Path상의 계산시간을 얻을 수없는 Nc이하인 경우에 대해서도 Sub-optimal한 타스크 스케쥴을 찾을 수 있으며, 앞으로 좀더 개선하여 거의 실시간으로 Optimal한 스케쥴을 찾는 알고리즘을 개발할 필요가 있다. PUMA 로봇의 조인트각 인산은 8086/87의 CPU를 3개를 써서 수행하면 약 4.25mSEC 으로 계산시간을 줄일 수 있음을 알 수 있고, 로봇 제어 알고리즘의 하나인 Computed Torque Method에 대해서도 병렬처리로 수행하면 실시간 실현이 가능하리라 생각된다.

8. 참 고 문 헌

1) T.Watanabe,M.Kametani,"Improvement in the computing time of robot manipulator using a multimicroprocessor", Trans. of ASME, J. of Dynamic systems,Measurement, and Control, Vol.108,Sept. 1986,pp190-197
2) Luh, J.Y.S and C.S.Lin,"Scheduling of Parallel computation for a computer controlled mechanical manipulator",IEEE Trans,SMC, Vol. 12-2,Mar/Apr. 1982,pp 214-234
3) C.S.G. Lee,Po Rong Chang,"Efficient Parallel Algorithm For Robot Inverse Dynamics Computation", IEEE Trans. SMC, Vol.SMC 16, No 4, JULY/AUG. 1986,pp532-542
4) S.A. Jacklin,J.A. Leyland and W. Warmbrodt, "Integrating Computer Architectures into the Design of High-Performance Controllers",IEEE Magagine,Control Systems, Vol.6, No 3 ,June, 1986,pp3-8
5) Shigeru Kokaji,"Collision Free Control of a Manipulator with a Controller Composed of 64 Microprocessors",IEEE Magagine,Control Systems,Vol.6, No.5,Oct,1986,pp9-14
6) C.S.G. Lee,"Robot Arm Kinematics", Tutorial on Robotics,Computer Society Press,pp47-65

부록.1

$$\vartheta_1 = atan2\left[\frac{p_y\sqrt{p_x^2+p_y^2-d_2^2}-p_x d_2}{p_x\sqrt{p_x^2+p_y^2-d_2^2}+p_y d_2}\right]$$

$$\sin\alpha = -\frac{p_z}{\sqrt{p_x^2+p_y^2+p_z^2-d_2^2}}$$

$$\cos\alpha = -\frac{\sqrt{p_x^2+p_y^2-d_2^2}}{\sqrt{p_x^2+p_y^2+p_z^2-d_2^2}}$$

$$\cos\beta = \frac{p_x^2+p_y^2+p_z^2+a_2^2-d_2^2-(d_4^2+a_3^2)}{2a_2\sqrt{p_x^2+p_y^2+p_z^2-d_2^2}}$$

$$\sin\beta = \sqrt{1-\cos^2\beta}$$

$$\sin\vartheta_2 = \sin\alpha\cos\beta+\cos\alpha\sin\beta$$

$$\cos\vartheta_2 = \cos\alpha\cos\beta-\sin\alpha\sin\beta$$

$$\vartheta_2 = atan2\left[\frac{\sin\vartheta_2}{\cos\vartheta_2}\right]$$

$$\cos\varphi = \frac{a_2^2+(d_4^2+a_3^2)-R^2}{2a_2\sqrt{d_4^2+a_3^2}}$$

$$\sin\varphi = \sqrt{1-\cos^2\varphi}$$

$$\sin\beta = \frac{d_4}{\sqrt{d_4^2+a_3^2}}\quad;\quad \cos\beta = \frac{|a_3|}{\sqrt{d_4^2+a_3^2}}$$

$$\sin\vartheta_3 = \sin\varphi\cos\beta-\cos\varphi\sin\beta$$

$$\cos\vartheta_3 = \cos\varphi\cos\beta+\sin\varphi\sin\beta$$

$$\vartheta_3 = atan2\left[\frac{\sin\vartheta_3}{\cos\vartheta_3}\right]$$

$$\vartheta_4 = atan2\left[\frac{(C_1a_y-S_1a_x)}{(C_1C_{23}a_x+S_1C_{23}a_y-S_{23}a_z)}\right]$$

$$\vartheta_5 = atan2\left[\frac{(C_1C_{23}C_4-S_1S_4)a_x+(S_1C_{23}C_4+C_1S_4)a_y-C_4S_{23}a_z}{C_1S_{23}a_x+S_1S_{23}a_y+C_{23}a_z}\right]$$

$$\vartheta_6 = atan2\left[\frac{(-S_1C_4-C_1C_{23}S_4)n_x+(C_1C_4-S_1C_{23}S_4)n_y+(S_4S_{23})n_z}{(-S_1C_4-C_1C_{23}S_4)s_x+(C_1C_4-S_1C_{23}S_4)s_y+(S_4S_{23})s_z}\right]$$