복수의 CPU로 제어되는 매니퓰레이터의 병렬계산 알고리즘

우 광 방, 김 현 기, 최 규 석°
(연세대학교 대학원 전기공학과)

Algorithm of Parallel Computation for a multi-CPU controlled Robot Manipulator

Kwang-Bang Woo, Hyun-Ki Kim, Gyoo-Suck Choi
(Department of Electrical Eng., Graduate School of Yonsei University)

ABSTRACT

The purpose of this paper is to develope the parallel computation algorithm that enables it to minimize the completion time of computation execution of the entire subtasks, under the constraints of the series-parallel precedence relation in each subtask. The developed algorithm was applied to the control of a robot manipulator functioned by multi-CPU's, and to obtain the minimum time schedule so that real time control may be achieved.

The completion time of computation execution was minimized by applying "Variable" Branch and Bound algorithm which was developed in this paper in determining the optimum ordered schedule for each CPU.

1. 서 론

산업용 매니퓰레이터(manipulator)는 최근에 산업자동화에 있어서 점점 더 중요시되고 있으며 마이크로 컴퓨터의 발달로 컨트롤러와 산업용 매니퓰레이터를 함께 하나의 유니트로 장착하는 것이 가능케 되었다.

상업적으로 이용가능한 PUMA 250과 같은 매니퓰레이터에 있어서,그 운동은 프로그램되어 각 관절의 구동모터로 계획된 궤적을 따라서 핸드가 움직일 수 있도록 하는 충분한 토오크를 제공해야한다.

그러나 로드가 미지이거나 변수이면 각관절의 운동은 가속 또는 감속이 되며 핸드는 목적한 경로를 정확하게 따라갈수 없게 된다.

이러한 문제를 해결하기 위해 필요한 일반화된·입력(input-generalized force)을 반복적으로 계산해내야 한다. 이러한 입력을 계산 하는데 있어서 복수의 CPU를 가진 컴퓨터를 사용하므로서 실시간제어가 가능토록 하고 최소의 계산완료시간을 성취하는데 병렬처리시스템을 이용한 병렬계산이 실행될 수 있다.

매니퓰레이터에서 인접링크간의 다이나믹 커플링 때문에 각각의 CPU내에서 실행되는 서브태스크간에 선행관계가 존재하게 된다. 복수의 CPU를 이용한 병렬계산을 수행하기 위해서는 각각의 CPU에 대한 스케듈링이 선행되어야한다.

병렬계산을 위해서는 Newton-Euler방식이 효율적이므로 Newton-Euler방식을 적용하였다. 본 논문에서는 직병렬 선행관계 제약하에서 계산완료시간이 최소가 되도록 복수의 CPU간에 순차적 순서로 로드를 분산시키므로서 태스크들의 계산을 스케듈링하는 "Variable" Branch and Bound 알고리즘을 전개한다.

2. 병렬계산과 매니퓰레이터

산업용 로보트는 end effector가 보통 그리퍼(gripper)인 직렬링크구조의 매니퓰레이터이다.

$i=1,2,..,n$ (n은 매니퓰레이터의 조인트수)에 대하여

$T_i(t)$ 를 조인트 i에 인가되는 일반화된 입력(applied input-generalized force)이라하고 $Q_i(t)$를 동일한 조인트상의 실변위(actual displacement)라 하면 매니퓰레이터를 제어하기 위해서는 $Q_i, \dot{Q}_i, \ddot{Q}_i$의 데이타를 이용하여 반복적으로 T_i를 계산할 필요가 있다.

최근의 멀티프로세서에 의한 분산제어 이론을 로보트 매니퓰레이터 제어에 도입하여 T_i의 계산을 전담하는 CPU(i)를 가진 multi-CPU 컴퓨터상에서 계산업무(computational task)가 수행된다고 가정하면 CPU(i)에서의 중간계산업무가 CPU(n-1) 와 CPU(n+1)로 부터의 계산결과를 필요로 할수 있다.

또한 매니퓰레이터의 인접 링크간에 존재하는 다이나믹 커플링으로 인한 서브태스크간의 직병렬 선행관계 재약에 의해서 n개의 CPU들이 독립적으로 계산을 수행하지 못할 수 있다. 따라서 n개의 CPU가 n개의 T_i를 계산하는데 걸리는 시간이 최소가 되도록하는 n개의 순차적인 계산순서를 스케듈링할 필요가 있다.

매니퓰레이터에 대한 Newton-euler 공식에 근거하여 조인트 i에 인가되는 입력을 병렬계산을 수행하기 위해 $i = 1,2,\ldots,n$ 에 대하여 Task(i)를 다음과 같이 6개의 서브태스크로 나눌 수 있다.

$$(1/i) : A^{\bullet}_i f_i = A_i^{i+1}(A^{\bullet}_{i+1} f_{i+1}) + A^{\bullet}_i F_i \quad \ldots\ldots (2.1)$$

$$(2/i) : \mathrm{VEC}\ 9 = (A^{\bullet}_{i+1} P_i^{*}) * (A^{\bullet}_{i+1} f_{i+1}) \quad \ldots\ldots (2.2)$$

$$(3/i) : \mathrm{VEC10} = A_i^{i+1}(A^{\circ}_{i+1} n_{i+1} + \mathrm{VEC}\ 9)$$

$$(4/i) : \mathrm{VEC11} = A_i (p_i^{*} + \hat{s}_i) * (A^{\bullet}_i F_i)$$

$$(5/i) : A^{\bullet}_i n_i = \mathrm{VEC}\ 10 + \mathrm{VEC}\ 11 + (A^{\bullet}_i F_i)$$

$$(6/i) : T_i = \begin{cases} (A^{\bullet}_i n_i)'(A_i^{i-1} z_o) + b_i \dot{q}_i) & \ldots\ldots (2.3) \\ \text{if link i is rotational.} & \\ (A^{\bullet}_i f_i)'(A_i^{i-1} q_o) + b_i \dot{q}_i & \\ \text{if link i is translational} & \end{cases}$$

3. 알고리즘 해석및 프로그래밍

1) "Variable" Branch and Bound 알고리즘

본 논문에서 전개하는 "Variable" Branch and Bound 방법은 solution tree의 branch들과 관련된 값들이 idle time의 variable length에 따라서 고정되어 있지 않다는 점에서 기존의 branch and bound technique과 구별된다.

이 방법에 있어서 선행관계제약들은 branching에 의해서 보호되고 반면에 Bound를 산출하는데 idle time의 생략이 가능하다.

이방법은 전진탐색과 후진탐색 과정을 포함하는데 초기시간으로 부터 출발하여 전진탐색 과정은 실행가능계획을 만드는데 적용되며 그것은 최적한 계획을 위한 실행시간의 상위한계를 설정한다. 실행가능계획에서 서브태스크들이 방금 완료된 각각의 시각은 분기점으로서 사용된다. 실행가능계획이 얻어진 다음 후진탐색과정을 적용함으로서 그 이전의 분기점으로 거슬러 올라간다.

동일한 시각에 여러개의 서브테스크를 실행 하는 것이 가능하기 때문에 하나이상의 가지(branch)그룹들(각각의 그룹은 하나의 특정한 CPU와 관련된다.)이 하나이상의 분기점으로 부터 열거될 수 있다. 실행시간의 최소값은 가지가 모든 가능한 idle time들을 무시함으로서 산출될 때의 여러가지 가지그룹들에 일치한다. 만일 산출된 값이 현재 설정된 상위한계를 초과한다면 이러한 가지들은 제거되고 프로세스는 그다음에 선속하는 분기점으로 거슬러간후, 전진탐색과정에 의해서 가지들이 완전히 수색된다.

새로운 실행가능계획이 설정되고 그실행시간이 현재의 상위한계보다 작을 때 그것을 최적 계획을 위한 현재의 후보(candidate)로 설정하고 그러한 과정을 가지들이 다 소비될 때 까지 반복하면 마지막 후보(candidate)가 최소시간계획이 된다.

이러한 알고리즘을 programming하기 위해서 여러개의 push-pop 구조의 STACK과 FIFO(First In First Out) 구조를 갖는 Queue를 운용하였으며 Language는 Pascal을 사용하였다.

2) 시뮬레이션을 위한 데이타구조

i) 입력 데이타구조

각각의 데이타는 CPU name, Task name, Time, Sign, next의 5개의 정보를 가진 Record로 구성한다. 여기서

Subtask	Predecessors	Successorss	Amount of Computation
(1/3)	(1/4)	(1/2), (2/2)	4M , 5A
(2/3)	(1/4)	(3/3)	6M , 2A
(3/3)	(2/3), (4/4)	(5/3) '	4M , 4A
(4/3)	-----	(5/3)	2M
(5/3)	(3/3), (4/3)	(3/2)	6A
(1/4)	(1/5)	(1/3), (2/3)	4M , 5A
(2/4)	(4/5)	(4/4)	4M , 2A
(3/4)	-----	(4/4)	4M , 1A
(4/4)	(2/4), (3/4)	(3/3)	6A
(1/5)	-----	(1/4)	4M , 5A
(2/5)	(2/6)	(4/5)	4M , 2A
(3/5)	-----	(4/5)	4M , 1A
(4/5)	(2/5), (3/5)	(2/4)	6A
(1/6)	-----	(2/6)	6M , 3A
(2/6)	(1/6)	(2/5)	3A

표 1에 의한 입력데이타를 근거로 하여 이 알고리즘에 의해 컴퓨터가 출력한 결과는 표 2 에 나타난 바와 같다.

표 2 스탠포드 매니퓰레이터에 적용한 결과
Table 2 The result of applying to Stanford Manipulator

초기에 얻은 Feasible Schedule

CPU[1] taskname	CPU[1] time	CPU[2] taskname	CPU[2] time
idle	0.00132	idle	0.00120
1	0.00172	1	0.00132
2	0.00196	2	0.00142
4	0.00220	4	0.00166
idle	0.00198	idle	0.00266
3	0.00334	3	0.00274
5	0.00358	5	0.00298

CPU[3] taskname	CPU[3] time	CPU[4] taskname	CPU[4] time
idle	0.00080	idle	0.00040
1	0.00120	1	0.00030
2	0.00158	idle	0.00130
4	0.00168	2	0.00153
idle	0.00206	3	0.00182
3	0.00242	4	0.00206
5	0.00266		

CPU[5] taskname	CPU[5] time	CPU[6] taskname	CPU[6] time
1	0.00040	1	0.00042
idle	0.00054	2	0.00054
2	0.00082		
3	0.00106		
4	0.00130		

Entire Execution Time = 0.00358 s

최종적으로 얻은 Feasible Schedule

CPU[1] taskname	CPU[1] time	CPU[2] taskname	CPU[2] time
4	0.00024	4	0.00024
idle	0.00132	idle	0.00120
1	0.00172	1	0.00132
2	0.00196	2	0.00142
idle	0.00260	idle	0.00228
3	0.00296	3	0.00236
5	0.00320	5	0.00260

CPU[3] taskname	CPU[3] time	CPU[4] taskname	CPU[4] time
4	0.00010	3	0.00024
idle	0.00080	idle	0.00040
1	0.00120	1	0.00080
2	0.00158	idle	0.00116
idle	0.00168	2	0.00144
3	0.00204	4	0.00168
5	0.00228		

CPU[5] taskname	CPU[5] time	CPU[6] taskname	CPU[6] time
1	0.00040	1	0.00042
3	0.00064	2	0.00054
2	0.00092		
4	0.00116		

Entire Execution Time = 0.00320 s

표 2에서 알 수 있듯이 초기에 얻은 실행가능 계획을 사용하여 조인트 i(i=1,2,..,6)에 인가된 힘 또는 토오크를 계산하는데 걸리는 전체 실행시간은 3.58ms 이다.

따라서 초기에 얻은 스케듈에 비해서 최종적으로 얻은 최소시간의 스케듈이 약 10.6%[(3.58-3.20)/3.58]의 계산 시간을 단축시켰음을 알 수 있다. 또한 한 개의 CPU

Time은 각각의 서브태스크의 실행시간길이를 나타내며 Sign은 그 서브태스크를 실행완료했는가를 검사하기위해 사용된다. 선속자 (Predecessor)는 Time에 음부호를 붙이므로서 후속자(Successor)와 구별한다. 서브태스크와 선속자,후속자간의 연결은 파스칼 언어의 포인터 (Pointer)구조를 이용한 연결 리스트(linked list)로써 구성한다

```
Record type
  pointer = ^noder
  noder = record
  taskn,cpun,sign : integer ;
              time : real ;
              next : pointer ;
   end ;
  infotype = record
     taskn,cpun : integer ;
           time : real ;
   end ;
  ptr = ^celltype ;
  celltype = record
     info : infotype ;
     next : ptr ;
    end ;
```

ii) 리스트와 스택에 들어가는 원소의 데이타구조

알고리즘의 프로그래밍을 위해서 다수의 큐구조의 리스트와 스택을 운용했다. 컴퓨터 알고리즘의 보다 보편적인 자료객체로서는 스택(Stack)과 큐(Queue)가 있다. 스택은 톱(top)이라 불리는 한쪽끝에서 자료의 삽입과 삭제가 일어나는 순서리스트이고 큐는 뒤(rear)에서 자료가 삽입되고 앞(front)에서 제거되는 순서리스트다.

```
Type
    qtype = record
         front,rear : ptr ;
      end ;
    bptype = record
         tk : real ;
         nk : integer ;
        elt : array[1..cn] of infotype ;
      end ;
    stacktype = record
         elements : array[1..maxi] of infotype ;
             top  : 0..maxi ;
         end ;
    bstacktype = record
         elements : array[1..maxi] of bptype ;
             top  : 0..maxi ;
        end ;
    Var
    rlist : array[1..m,1..n] of qtype ;
    alist : array[1..m,1..n] of qtype ;
    candidate : array[1..n] of qtype ;
    sublist : qtype ;
    bufferlist : qtype ;
    timelist : qtype ;
    i,j,k : integer ;
```

iii) 리스트와 스택의 프로그램 구현

리스트와 스택을 만들기위해서 다음과 같은 기본적인 프로시져(procedure)와 함수(function)를 사용했다.

(x : element_type)

a) 리스트(Queue)

```
Ready-List[k,j] :   clear_rlist(k,j),empty_rlist(k,j),
  en_rlist(k,j,x),de_rlist(k,j,x)  { x : infotype }
Alternate-List[k,j]:clear_alist(k,j),empty_alist(k,j),
  en_alist(k,j,x),de_alist(k,j,x)  { x : infotype }
Subtask-List:clear_sublist,empty_sublist,en_sublist(x)
  ,de_sublist(x)  { x:infotype }
Buffer-List:en_bufferlist(x),de_bufferlist(x)
  clear_bufferlist,empty_bufferlist { x : infotype }
Time-Pointer-List : clear_timelist,empty_timelist,
  en_timelist(x),de_timelist(x)   { x : infotype }
Candidate_List : clear_candidate,empty_candidate,
  en_candidate(x),de_candidate(x)   { x : infotype }
```

b) 스택(Stack)

```
Solution-Stack[j]:clear_solstack(j),empty_solstack(j)
  push_solstack(j,x),pop_solstack(j,x) {x:stacktype }
Branching-Point-Stack:clear_bpstack,empty_bpstack,
pop_bpstack(x),push_bpstack(x)   { x : bstacktype }
```

4. 매니퓰레이터에 대한 적용및 결과고찰

본절에서는 위에서 기술한 알고리즘을 조인트가 6개인 Stanford 매니퓰레이터에 적용한다. 여기서 각각의 조인트에 대하여 각각 한 개의 조인트를 할당한 병렬처리 시스템을 이용하여 각각의 조인트에 인가되는 입력(힘 또는 토오크)을 계산하는데 이 알고리즘을 적용한다. 매니퓰레이터의 각각의 조인트에 인가되는 입력에 관한 Newton Euler계산방정식을 분석하므로써 표 1과 같은 선행관계및 산술연산의 수를 알 수 있다. 여기서 A는 덧셈을 나타내고 M은 곱셈을 나타내며 곱셈에 대한 계산실행시간은 50us, 덧셈에 관한 계산실행시간은 40us로 간주한다.

표 1 인가되는 입력(힘/토오크)계산을 위한 선행관계 테이블
Table 1 Precedence Table computing Applied Forces/Torques

Subtask	Predecessors	Successorss	Amount of Computation
(1/1)	(1/2)	-----	4M , 5A
(2/1)	(1/2)	(3/1)	4M , 1A
(3/1)	(2/1),(5/2)	(5/1)	4M , 4A
(4/1)	-----	(5/1)	4M , 1A
(5/1)	(3/1),(4/1)	-----	6A
(1/2)	(1/3)	(1/1),(2/1)	3A
(2/2)	(1/3)	(3/2)	2M
(3/2)	(2/2),(5/3)	(5/2)	2A
(4/2)	-----	(5/2)	4M , 1A
(5/2)	(3/2),(4/2)	(3/1)	6A

를 사용하였을 때는 전체 계산실행시간이 6.6ms가 걸린다

따라서 동시성 수준(level of concurrency)이 (3.2/6.6)임을 알 수 있으며 한 개의 CPU를 사용할 때와 비교하여 6개의 CPU를 사용하면 그 계산시간을 약 51.5%[(6.6-3.2)/6.6] 단축시킬 수 있음을 알 수 있다.

5. 결 론

본 논문에서는 직병렬 선행관계 제약을 갖는 서브태스크들을 여러개의 CPU를 갖는 병렬처리시스템이 병렬계산을 수행할때 각각의 CPU에 대한 idle time을 최소로 줄이고 CPU효율을 극대화시키므로써 전체태스크에 대한 계산실행시간을 최소로 할 수 있는 최적한 순서계획을 결정해주는 "Variable" branch and bound 알고리즘을 개발하였다.

여러개의 CPU를 가진 컴퓨터로 산업용 매니퓰레이터를 제어한다고 할때 인접 링크간의 다이나믹 커플링때문에 각각의 CPU내에서 실행되는 서브태스크간에 직병렬 선행관계가 존재하게 되므로 매니퓰레이터의 각각의 조인트에 인가되는 입력(applied torques/forces)을 Newton-Euler공식에 의해서 계산하는데 이 알고리즘을 적용하여 전체 계산실행시간이 최소가 되도록 하는 스캐듈을 얻음으로써 실시간제어가 가능하도록 했다.

또한 이 알고리즘에서 추정실행시간이 현재 설정된 upper bound 보다 더 긴 모든 실행가능 계획을 나타내는 가지들을 제거함으로써 불필요한 탐색을 감소시켜 기존의 Branch and bound 알고리즘을 수정 개선하였다.

참 고 문 헌

1. Horowitz and Shani, Fundamentals of Data structures in pascal
2. S.E.Goodman and S.T.Hedetniemi, Introduction to the design and analysis of algorithms
3. 이석호 역, 자료구조론
4. J.Y.S.Luh, M.W.Walker, and P.C.Paul, "On-line computation scheme for michanical manipulators," ASME Trans.J.Dynamic Syst.,Meserment an Contr., vol 102, no. 2, pp. 69-76, June 1980.
5. S.Kasaha,T.Yokota,T.Yasui,and S.Narita,"Gernal purpose parallel processing scheme of robot control computation using scheduling theory." in Proc.Ann.Conv.IECE of Japan, Mar., 1984, no.1523(preprint)
6. J.Y.S.Luh and C.S Lin,"Scheduling of parallel computation for a computer-controlled mechanical manipulator," IEEE Trans. Syst., Man, Cybern. vol.12,no 2,pp.214-234,March/April 1982.
7. J.M.Hollerback,"A recursive Lagrangian formulation of manipulator Dynamics formulation complexity," IEEE Trans. Syst., Man, Cybern., vol.10,no 11,pp.730-736,Nov. 1980.