論文2000-37CI-4-4

신경망을 이용한 실시간 멀티프로세서 스케줄링 알고리즘과 하드웨어 설계

(Real-Time Multiprocessor Scheduling Algorithm using Neural Network and Its Hardware Design)

李 在 衡*, 李 康 彰*, 趙 鏞 範**

(Jae-Hyoung Lee, Kang-Chang Lee, and Yong-Beom Cho)

요 약

본 논문은 실시간 멀티프로세서 스케줄링 문제를 효과적으로 해결하는 신경망 알고리즘을 제안한다. 제안된 알고리즘은 대표적인 신경망 모델인 홉필드 네트워크를 근간으로 태스크의 처리요구에 대해 지정된 시간 이내에 처리할 수 있는 실시간 시스템을 신경망의 장점인 병렬처리가 가능하도록 구현하였다. 본 알고리즘의 성능을 비교하기 위하여 기존에 실시간 멀티프로세서 스케줄링을 위해 연구되는 EDA와 LLA의 두 알고리즘과 비교한다. 제안된 알고리즘은 VHDL을 이용하여 하드웨어로 설계한다.

Abstract

This paper proposes a neural network algorithm for real-time multiprocessor scheduling problem. The proposed algorithm is developed base on Hopfield neural network for a benefit of parallel processing, in order to finish a requested task within a deadline time. To compare the performance of the proposed algorithm, we used EDA and LLA algorithm that has studied real-time multiprocessor scheduling before. The proposed algoritm is implemented hardware using VHDL.

I. 서 론

오늘날 반도체 설계 기술과 공정 기술의 발전에 힘입어 고성능의 프로세서가 나오고 있다. 그럼에도 불구하고 사용자의 요구는 나날이 증가하여, 짧은 시간 내에 많은 양의 데이터를 처리할 수 있는 멀티프로세서의 필요성이 크게 대두되고 있다. 멀티프로세서 시스템은 프로세서 수에 따라 선형적인 성능의 향상을 보일 경우 이상적인 시스템이 되나, 현재까지의 기술로는 그러한 시스템이 나오지 못하고 있다. 이러한 현상은 시스템의 구조적인 문제와 더불어 태스크의 스케줄링 문제 때문이다.

실시간 시스템은 태스크의 처리요구에 대해 지정된 시간 이내에 처리해 줄 수 있는 시스템을 말한다. 이러한 실시간 시스템에 속하는 실시간 멀티프로세서 시스템의 경우 여러 작업을 동시 수행하면서 특정 태스크를 지정된 시간이내에 수행하는 스케줄링 알고리즘이 요구된다. 또한 시스템의 성능향상을 위해서 문맥교환의 발생회수가 최소화되어야 함은 물론 임의의 태스크[1][2]에 대해서도 성공률이 보장되어야 한다. 최적화 문제는 주어진 모든 조건을 고려하여 최적의 해를 구하는 것을 말하는데, 실시간 멀티프로세서 스케줄링 문제는 주어진 태스크의 정보와 프로세서의 사용정보를 이용하여 시스템의 효율과 수행시간을 최적화하는 일종의 최적화 문제이며 또한 NP-complete 문제이다. NP-complete 문제는 지수적으로 증가하는 계산시간으로 인해 빠른 시간 안에 최적의 해를 찾기가 사실상 불가능하기 때

* 正會員, 建亞情報技術(주), 연구개발팀

(R&D Devision, KeonA Information Technology Inc.)

** 正會員, 建國大學校 電子工學科

(Dept. of Electronics Engineering, Konkuk University)

接受日字:1999年10月1日, 수정완료일:2000年7月6日

문에 대부분의 알고리즘은 수학적인 방법과 휴리스틱한 방법을 이용하여 해결을 하게 된다. 최근에 NP-complete 문제를 해결하기 위한 방법으로 신경망을 이용하는 연구가 활발히 진행중이다.

본 논문에서는 홉필드 네트워크[3][4]를 근간으로 멀티프로세서 스케줄링 문제를 해결하려 한다. 홉필드 네트워크는 에너지 함수의 최소화를 통해 문제를 해결하고, 에너지 함수는 주어진 조건을 고려하여 결정된다. 홉필드 네크워크는 리아프노프(Lyapunov)의 최급하강법을 이용하고 있어 항상 지역 최소점으로 수렴된다는 것이 증명되어 있고[5] 전역 최소점(global minimum)이 아니더라도 빠른 시간 내에 근사적인 해를 구할 수 있으므로 해결이 복잡한 최적화 문제에 응용되고 있다[8]. 홉필드 네트워크는 뉴런의 병렬처리를 통해 해를 산출하고, 문제의 크기가 증가할수록 뉴런의 수가 증가하여 병렬처리 능력이 향상되는 구조이므로 문제의 규모 증가에 따른 계산시간의 증가비율이 선형적인 방법에 비해 적다. 따라서 실시간 멀티프로세서 스케줄링 문제는 홉필드 신경망을 이용하여 해결하는 것이 적합하고 병렬처리를 실현하기 위해 하드웨어로 구현해야 한다.

본 논문에서는 홉필드 신경망을 이용하여 실시간 멀티프로세서 스케줄링 문제를 해결하고, 실시간 단일프로세서 스케줄링에서 최적이고 실시간 멀티프로세서 스케줄링을 위해 연구되는 EDA(Earliest Deadline Algorithm), LLA(Least Laxity Algorithm)의 두 알고리즘과 비교, 분석한다[7]. 설계된 알고리즘은 HDL의 일종인 VHDL(Very High Speed Integrated Circuit Hardware Description Language)을 이용하여 하드웨어로 설계한다.

본 논문에서는 2장에서 실시간[3] 멀티프로세서 스케줄링 문제를 정의하고 이를 모델링하기 위한 방법을 제시하며 3장에서 신경망 알고리즘을 제안한다. 4장에서는 시뮬레이션 결과고찰을 하고 5장에서 결론을 내린다.

Ⅱ. 실시간 멀티프로세서 스케줄링 문제

1. 온라인·오프라인, 선점·비선점 스케줄링

온라인 스케줄링의 특징은 미래에 발생될 태스크의 시점을 알 수 없다는 것에 있다. 이에 반해 오프라인 스케줄링은 태스크의 정보를 사전에 아는 상태에서 스케줄링 한다. 선점(preemption) 스케줄링은 태스크의 선점을 허용하는 스케줄링 방법이고 비선점 스케줄링은 태스크의 선점을 허용하지 않는다. 선점은 한 태스크의 수행이 끝나기 전에 수행되던 태스크를 정지하고 다른 태스크를 수행하는 것을 의미한다. 이 때 태스크의 교환을 문맥교환(context switch)이라 한다. 최적화(optimal) 알고리즘은 스케줄링 가능한 모든 태스크의 집합에 대해 스케줄링이 가능함을 뜻한다. 이와 비슷한 개념으로 부최적화(sub-optimal) 알고리즘이 있는데 이는 스케줄링이 가능한 모든 준비된 태스크(ready task)의 집합에 대해 스케줄링이 가능함을 뜻한다. 준비된 태스크는 현재 스케줄링 되고 있는 태스크, 즉 스케줄러가 스케줄링을 위한 정보를 가지고 있는 태스크를 뜻한다. 최적과 부최적의 차이는 발생될 태스크의 고려여부에 있는데 발생될 태스크를 고려하여 스케줄링하여 스케줄링 가능한 모든 태스크의 집합에 대해 스케줄링하는 알고리즘은 최적이고, 발생될 태스크를 고려하지 않고 스케줄링하여 스케줄링 가능한 모든 준비된 태스크의 집합에 대해서만 스케줄링 가능한 알고리즘은 부최적이다.

2. EDA와 LLA

EDA(Earliest Deadline Algorithm)와 LLA(Least Laxity Algorithm)는 실시간 단일프로세서 스케줄링 알고리즘으로 단일프로세서에서 최적 알고리즘이다. EDA는 다중프로세서에 적용시 문맥교환이 적게 일어나는 장점이 있으나 스케줄링 가능한 준비된 태스크 집합에 대해 스케줄링을 성공하지 못하는 경우가 있다. 반면 LLA는 멀티프로세서에서 부최적이기 때문에 스케줄링 가능한 준비된 태스크 집합에 대하여 스케줄링에 성공하나 문맥교환이 많이 발생한다. 다음은 멀티프로세서에서의 EDA와 LLA이다[7].

- EDA : n개의 프로세서 시스템에서 가장 가까운 제한시간을 갖는 n개의 태스크를 실행한다. 같은 값일 때는 임의 선택한다.
- LLA : n개의 프로세서 시스템에서 가장 작은 여유시간을 갖는 n개의 태스크를 실행한다. 같은 값일 때는 임의 선택한다.

다음은 멀티프로세서에서의 EDA와 LLA이다[7].

본 논문에서는 EDA와 LLA를 수행하기 위해서 꼭 필요한 태스크의 계산시간(computation time)과 제한시간(deadline time)만을 이용하였다. 태스크 A의 계산시간과 제한시간을 'A(계산시간, 제한시간)'으로 표시하고, 한번 스케줄링이 되어 한 프로세서에 할당되

어 수행을 하면 계산시간과 제한시간에서 각각 1을 감소시키고 할당되지 못하여 수행되지 못하면 제한시간에서만 1을 감소시킨다

여유시간(Laxity) = 제한시간 - 계산시간 (1)

본 논문에서 사용하는 태스크 모델을 이용하여 EDA와 LLA의 스케줄링 예를 다음에 보인다.

표 1의 문제는 EDA와 LLA의 특징을 잘 보여주고 있다. EDA의 경우 6번째 스케줄링시 태스크 A와 B를 선택하여 C가 제한시간이내에 수행되지 못하여 스케줄링에 실패한다. 반면 실패 시까지 문맥교환은 단 1회도 일어나지 않아 그 문맥교환이 적은 특성을 보여주고 있다. LLA의 경우는 성공적으로 스케줄링하는 결과를 보이고 있으나 6차례의 문맥교환을 이루는 것을 보아 EDA에 비해 많은 문맥교환이 일어나는 특성을 보여주고 있다.

표 2. EDA와 LLA를 이용한 준비된 태스크 스케줄링의 예

Table 1. Example of the ready task scheduling using a EDA, LLA.

구분 / 횟수	EDA				LLA			
	A	B	C	문맥교환	A	B	C	문맥교환
1	(7, 9)	(5, 8)	(6, 10)	-	(7, 9)	(5, 8)	(6,10)	-
2	(6, 8)	(4, 7)	(6, 9)	X	(6, 8)	(4, 7)	(6, 9)	X
3	(5, 7)	(3, 6)	(6, 8)	X	(5, 7)	(3, 6)	(6, 8)	O
4	(4, 6)	(2, 5)	(6, 7)	X	(4, 6)	(3, 5)	(5, 7)	O
5	(3, 5)	(1, 4)	(6, 6)	X	(3, 5)	(2, 4)	(5, 6)	O
6	(2, 4)	(0, 3)	(6, 5)	X	(2, 3)	(2, 3)	(4, 5)	O
7			X		(1, 2)	(1, 2)	(4, 4)	O
8					(0, 2)	(1, 1)	(3, 3)	O
9					-	(0, 0)	(2, 2)	X
10					-	-	(1, 1)	X
11					-	-	(0, 0)	X
	실 패			0	성 공			6

3.최적화 문제

최적화 문제는 크게 조건 만족 문제와 최소화 문제로 나누어진다. 조건 만족 문제는 몇 가지 기준에 만족하기만 하면 해가 되는 문제이다. 이 문제는 수식을 통하여 해를 구할 수 있고 쉽게 해인지 아닌지 검증할 수 있다. 최소화 문제는 수식을 통해 해를 구할 수 있으나 최적해인지 검증할 수 없는 문제이다. 따라서 일반적으로 최적화 문제는 조건 만족 문제보다 해결이 어렵다.

NP-complete 문제는 다항식 시간(polynomial time)이내에 처리할 수 있는 알고리즘이 존재하지 않는 것을 말한다. 멀티프로세서에서의 스케줄링 문제는 모든 작업의 수행 시간과 프로세서간의 통신비용이 같다는 가정 하에도 NP-complete 임이 증명되었다[1]. NP-complete 문제는 해를 구해야 할 시스템의 크기가 커질수록 경우의 수가 기하급수적으로 발생하여 순차적인 컴퓨터에서는 해를 구하는데 엄청난 계산 시간을 요구하는 것으로 알려져 있다. 그러나 1985년 홉필드 및 탱크에 의해 신경망의 병렬 처리 알고리즘을 이용할 경우 해를 빠른 시간 내에 구할 수 있음이 증명되었다[5]. 이 때 제안된 신경망 모델은 시스템을 에너지라는 물리적 양으로 나타내어 이 에너지의 표면을 따라 감소하면서 최소점을 찾는 방법을 사용하여 원하는 최적 해를 구한다. 홉필드 네트워크는 빠른 수렴 속도로 거의 근사적인 해를 구할 수 있기 때문에 최적화 문제에 많이 응용되고 있다.

Ⅲ. 신경망을 이용한 멀티프로세서 스케줄링

본 논문에서는 홉필드 네트워크를 멀티프로세서 스케줄링 문제에 적용시켰다. 홉필드 네크워크를 스케줄링 문제에 적용하기 위해서는 문제에서 얻어진 코스트(cost) 함수와 제한 조건들(constraint)을 고려하여 에너지 함수를 만들어야 한다. 그러나 에너지 함수를 직접 만들기는 매우 까다로우며, 해당 문제에 적용할 때는 상태방정식으로 전환해야 한다. 따라서 뉴런의 상태방정식을 먼저 만들고 필요에 따라 이를 적분하여 에너지 함수를 얻는다. 홉필드 네트워크를 이용하여 최적화 문제를 해결하기 위해서는 수학적 에너지 함수 E를 최소화해야 한다. 에너지 함수를 최소화하기 위해 제1차 오일러 방법을 사용하여 새로운 입력값 U(t+1)을 구하고 이를 뉴런의 입출력 함수에 의해 새로운 뉴런의 상태 V(t+1)을 구한다. 그리고 출력된 전체 시스템의 안정상태를 조사하여 알고리즘을 종료하던가 다시 입력 변화값을 구하여 되풀이한다. 그림 1은 설계된 신경망 구조이다. m개의 태스크와 n개의 프로세서에 대한 스케줄링 문제는 $m \times n$ 뉴런의 구조로 이루어지며 뉴런(2,1)의 출력이 '1'이면 프로세서 1이 태

스크 2를 처리하도록 스케줄링함을 뜻한다. 뉴런(2,3)의 출력이 '0'이면 프로세서 3이 태스크 2를 처리하지 않음을 뜻한다.

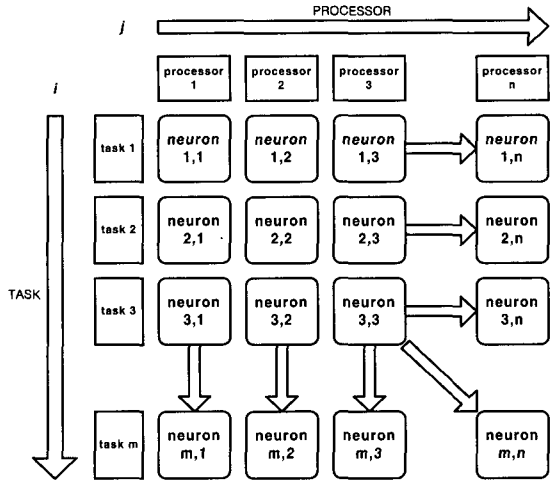

그림 1. 실시간 멀티프로세서 스케줄링을 위한 신경망 구조

Fig 1. Neural network structure for a real-time multiprocessor scheduling.

실시간 멀티프로세서 스케줄링을 위해 이론을 토대로 다음과 같은 제약조건을 정하였다.

① 프로세서 하나는 동시에 여러 개의 태스크를 처리할 수 없다.

② 하나의 태스크는 동시에 여러 개의 프로세서에서 처리할 수 없다.

③ 요구된 태스크의 수만큼만 프로세서를 사용하여야 한다. 단 프로세서 수가 요구된 태스크의 수보다 적으면 최대한 사용한다.

④ 부최적이어야 한다.

⑤ 선점은 허용하고 문맥교환의 수가 최소화되어야 한다.

설계된 신경망의 동작을 기술하는 상태방정식은 다음과 같다

$$\begin{aligned}\frac{dU_{ij}}{dt} = & -A\Big(\sum_{i=1}^{m} V_{ij}-1\Big)\cdot f\Big(\sum_{i=1}^{m} V_{ij}-1\Big)\cdot(\Omega_{\max}-\Omega_i)\\ & -B\Big(\sum_{j=1}^{n} V_{ij}-1\Big)\cdot f\Big(\sum_{j=1}^{n} V_{ij}-1\Big)\cdot(\Omega_{\max}-\Omega_i)\\ & -C\cdot f\Big(\sum_{j=1}^{n}\sum_{i=1}^{m} V_{ij}-R_{ef}\Big)-D\cdot f\Big(\Big(\sum_{j=1}^{n} f\Big(\sum_{i=1}^{m} V_{ij}\Big)\Big)-R_{ef}\Big)\\ & -E\Big(i\cdot f\Big(\sum_{i=1}^{m} V_{ij}-1\Big)\Big)-F\Big(j\cdot f\Big(\sum_{j=1}^{n} V_{ij}-1\Big)\Big)\\ & +G\cdot\Omega_i(1-\Theta)\cdot g\Big(\sum_{i=1}^{n} V_{ij}\Big)\cdot g\Big(\sum_{j=1}^{n} V_{ij}\Big)\\ & +\frac{1}{H}\cdot\Theta\cdot\Omega_i\cdot g\Big(\sum_{j=1}^{n} V_{ij}\Big)+I\cdot(1-\Theta)\cdot V_{ij}^{T-1}\end{aligned} \tag{2}$$

$$\Theta = g\Big(\Big(\sum_{j=1}^{n} f\Big(\sum_{i=1}^{m} V_{ij}\Big)\Big)-R_{ef}\Big)\cdot g\Big(\Big(\sum_{i=1}^{m} f\Big(\sum_{j=1}^{n} V_{ij}\Big)\Big)-R_{ef}\Big)\cdot g\Big(\sum_{j=1}^{n}\sum_{i=1}^{m} V_{ij}-R_{ef}\Big) \tag{3}$$

$$\Omega_i = \alpha\cdot\frac{1}{T_i^{dl}}+\beta\cdot\frac{1}{(T_i^{dl}-T_i^{c}+1)}+\gamma\cdot(\gamma'-T_i^{dl})\cdot f(T_i^{c})+\delta\cdot(\delta'-(T_i^{dl}-T_i^{c})) \tag{4}$$

$$f(x)=\begin{cases}1 & x>0\\ 0 & x\le 0\end{cases} \tag{5}$$

$$g(x)=\begin{cases}1 & x=0\\ 0 & x\ne 0\end{cases} \tag{6}$$

i :태스크 번호 j: 프로세서 번호

m: 최대 태스크 수 n: 프로세서 수

U :뉴런의 상태 V: 뉴런의 출력

T:스케줄링 시간

T_{dl}:태스크의 제한시간 T_c: 태스크의 계산시간

R: 처리를 요구하는 태스크(준비된 태스크)의 개수

R_{ef}: $n$if $R > n$, Rif $R \le n$

Ω_i: EDA와 LLA를 이용하여 태스크의 긴급도를 결정하는 항, 클수록 긴급

Θ: 충돌여부와 개수만족을 나타내는 항, 충돌이 없고 개수가 맞으면 1 A, B, C, D, E, F, G, H, I, α, β, γ, γ', δ, δ': 계수

이 상태방정식은 개별의 뉴런의 상태를 갱신하기 위해 사용되고 오일러 방법을 통해 뉴런의 새로운 상태를 계산할 수 있다. 설계된 상태방정식은 다음과 같이 작동한다.

A항은 하나의 processor에서 여러개의 태스크를 처리하지 않게 하도록 하는 항이다. 즉 1개의 processor에 2개이상의 태스크가 할당되는 것을 방지한다. B항은 가로축에 대하여 A항과 비슷하게 작동한다. 이것은 1개의 태스크가 2개의 processor에서 처리되지 않도록 방지하는 것이다. C항은 할당된 뉴런의 수가 처리할 수 있는 태스크의 수보다 클 때 태스크의 수를 맞추어 주는 항이다. D항은 사용하는 processor의 수가 요구된 태스크의 수보다 많을 때 뉴런을 억제하는 항이다. E항은 세로축의 뉴런 2개가 진동하는 것을 방지하는 항이다. F항은 가로축의 뉴런 2개가 진동하는 것을 방지하는 항이다. G항은 뉴런의 충돌이 있거나 할당된 뉴런의 수가 태스크 요구의 수와 같지 않을 때 태스크의 긴급도에 따라 할당되지 않은 프로세서에 대해 할당되지 않은 태스크를 할당하도록 뉴런을 활성화하는 방향으로 작동하는 항이다. H항은 뉴런의 충돌이 없고 할당된 뉴런의 수가 태스크 요구의 수와 일치할 때, 즉 global minimum이거나 적어도 local minimum에 위치하여 있을 때 적절한 해답인지 검증

하기 위해 에너지를 태스크의 긴급도를 참조하여 긴급도가 높은 태스크가 할당되어 않은 경우 할당되는 방향으로 유도한다. 즉 hill climbing 항이다. I항은 이전에 스케줄링된 뉴런이 활성화되는 방향으로 유도하여 문맥교환의 수를 줄여주는 항이다. 차후에 복잡한 태스크모델 적용시 Ω_i만을 변형하여 쉽게 변형이 가능하도록 하였다.

T_i^c가 0인 경우 태스크의 실행이 끝난 상태이고 새로운 태스크 요구가 없는 상태를 의미한다. 이 경우 뉴런의 갱신이 필요하지 않다. 이를 위해 $f(T_i^c)$를 상태방정식의 모든 항에 곱해주면 되나 상태방정식에 포함시키면 복잡하므로 오일러 방법을 변형하여 간단히 구현하였다.

$$U(t+1) = U(t) + \Delta U \cdot \Delta t \cdot f(T_i^c) \tag{7}$$

종료조건(Termination Condition)은 홉필드 신경망의 종료시점을 판단하는 기준을 말한다. 이것은 설계된 신경망의 성능을 좌우하는 중요한 요소 중 하나이므로 적절히 지정되어야 한다. 설계된 신경망의 종료조건으로는 뉴런 전체의 출력이 변하지 않는 횟수를 계산하여 지정된 횟수이상 변하지 않았을 때 다시 여유시간을 계산하여 여유시간이 0인 태스크에 프로세서를 할당하였는지 여부를 검사하고 할당하지 않았을 경우는 다시 할당을 계속하고, 모두 할당되었을 경우는 수행을 종료하도록 하였다. 이 종료조건은 홉필드 신경망이 항상 적절한 답에 가까운 방향으로 작동한다는 점과 여유시간이 0인 태스크에 프로세서가 할당되기만 하면 적절한 지역최소점인 것을 고하여 결정된 것이다. 최적화 문제는 일반적으로 적절한 종료조건의 결정이 어려운데 모의 실험에서 위의 조건의 성능을 알 수 있다.

현재 스케줄링의 결과가 이전 스케줄링의 결과와 같다면 문맥교환을 줄이는 효과를 가져온다. 그리고 스케줄링 되지 못한 태스크 중에 특별히 위급한 태스크가 없을 경우 문맥교환을 줄이는 방향으로 스케줄링하는 것이 바람직하다. 따라서 본 논문에서는 이전에 스케줄링 된 결과 즉 뉴런의 출력배열을 다음 스케줄링의 초기출력상태로 정하여 현재 스케줄링의 출력으로 적절한지를 hill climbing 항에 의해 우선 검토되도록 하였다. 단 이전 스케줄링의 결과 중 출력이 '1'인 것은 현재 T_i^c가 0인지를 우선 검사하여 0이 아닌 경우에만 이번 스케줄링의 초기출력상태로 정한다. 이것은 T_i^c가 0이 되어 경쟁이 필요 없는 뉴런의 출력이 '1'이 되어 필요하지 않은 경쟁을 하지 않도록 방지하는 것이다.

다음은 태스크의 스케줄링 요구에 대해 설계된 신경망 스케줄러가 어떻게 응용될 수 있는지에 대한 구상이다. 태스크 버퍼(task buffer)는 준비된 태스크에 대한 정보를 저장하고 태스크 분산기(task distributer)는 이미 처리를 끝내고 비어있는 태스크 버퍼에 스케줄링 요구된 태스크의 정보를 입력한다.

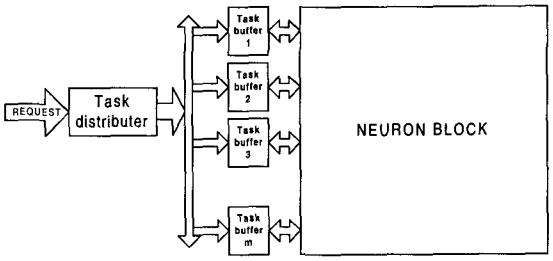

그림 2. 실시간 멀티프로세서 스케줄러의 구조
Fig 2. Structure of a real-time multiprocessor scheduling.

신경망은 병렬처리를 기본으로 한다. 그러나 우리가 사용하는 컴퓨터는 순차적(sequential)인 처리를 기본으로 하기 때문에 가상으로 병렬처리를 할 수 있는 알고리즘을 구현하여야 한다. 폰노이만 컴퓨터에서 홉신

표 2. 병렬처리 알고리즘
Table 2. Parallel processing algorithm.

① 상태방정식에 사용될 계수와 뉴런 모델에서의 각 점을 구체적으로 결정한다.
② 각 뉴런의 입력 $U_{i,j}$을 임의로 초기화하고 각 뉴런의 출력 $V_{i,j}$을 뉴런 모델에 따라 $U_{i,j}$에서 산출한다.
③ 상태방정식에 의해 각 뉴런의 ΔU을 계산한다.
④ 제 1차 오일러 방법에 의해 각 뉴런의 새로운 입력을 산출한다.
⑤ 각 뉴런의 출력 $V_{i,j}$을 뉴런 모델에 따라 $U_{i,j}$에서 산출한다.
⑥ 해답을 검사하여 적당한 해답이면 종료하고 아니면 2번부터 다시 수행한다.
⑦ 계속해서 적당한 해답을 찾지 못할 경우 1번부터 다시 수행한다.

경망은 병렬처리를 기본으로 한다. 그러나 우리가 사용하는 컴퓨터는 순차적(sequential)인 처리를 기본으로 하기 때문에 가상으로 병렬처리를 할 수 있는 알고필드 네트워크를 이용한 병렬처리 알고리즘을 모의 실험하기 위한 순서는 표 2에 따른다.

Ⅳ. 모의 실험 결과

1. 소프트웨어 모의 실험 결과

다음에 3×2, 15×10, 30×20, 300×200의 시스템에 대하여 모의 실험한 결과를 보인다. 최소 계산시간은 2 최대 계산시간은 9로 하여 임의의 값을 발생시켰고 제한시간은 9이내의 임의의 값을 발생시켜 계산시간과 합한 값으로 하였다. 임의의 문제를 발생시켜 문제가 모두 수행될 때까지 수행하여 문맥교환을 측정하였다. 한번 발생된 문제는 각각의 알고리즘에 모두 수행시켜 비교되도록 하였다.

표 3의 문맥교환 발생시점 수 평균값으로부터 EDA의 문맥교환 회수는 매우 적게 일어남을 알 수 있다. 반면 LLA의 문맥교환은 이론에서 보인 것과 같이 EDA비해 많이 일어나는 결과를 보인다. 제안된 신경망 알고리즘은 문맥교환의 수가 LLA보다 적게 일어나지만 EDA보다는 우수하지 못하였다. 성공률에 있어서 EDA는 스케줄링 가능한 문제에 대해 스케줄링하지 못하는 경우가 발생하였으나, LLA나 신경망 알고리즘은 주어진 문제 모두에 대한 스케줄링에 성공하여 부최적 알고리즘임을 알 수 있다. 표 3의 결과는 전체적으로 프로세서 수의 증가로 인한 시스템 성능 향상에 비해 부하의 증가가 부족하여 전체적으로 시스템의 크기가 증가할수록 성공률이 높게 나오고 또한 문맥교환의 수의 증가가 작아지는 현상을 보였다.

표 3의 신경망 모의 실험 결과 중 첫 스케줄링 수행회수 평균은 신경망 알고리즘이 스케줄링을 위해 소요하는 시간을 의미한다. 3×2 시스템에서 평균 70번 이내에 문제를 해결하고 15×10 시스템에서 300번, 30×20 시스템에서 500번, 300×200 시스템에서 1500 이내에 문제를 해결한 것을 알 수 있다. 3×2 시스템에서 전체 규모가 100배 증가한 30×20 시스템에서 수행회수는 7배 증가하고 또 100배가 증가한 300×200 시스템에서 3배의 수행회수증가를 보였다. 주어진 문제가 시스템의 증가에 따라 부하가 적어지는 특성을 보인다는 점을 감안하더라도 신경망을 이용한 알고리즘은 규모증가에 따른 계산속도의 저하가 적음을 알 수 있다. 두 번째 스케줄링 수행회수 평균은 이전 결

표 3. 모의 실험 결과
Table 3. Simulation results.

조건 / 항목		시스템의 규모				비고
		3×2	15×10	30×20	300×200	
문제의 총수		1000	1000	1000	100	
적합하지 못한 문제의 수		50	0	0	0	
EDA	스케줄링 실패회수	82	92	38	0	성공률 저조, 문맥교환 우수
	스케줄링 실패확률(%)	6.316	9.200	3.8	0	
	문맥교환 발생시점 수 평균	0.868	2.347	2.841	3.02	
LLA	스케줄링 실패회수	0	0	0	0	부최적, 문맥교환 저조
	스케줄링 실패확률(%)	0	0	0	0	
	문맥교환 발생시점 수 평균	2.685	4.517	4.779	4.95	
신경망	스케줄링 실패회수	0	0	0	0	부최적, 문맥교환 우수
	스케줄링 실패확률(%)	0	0	0	0	
	문맥교환 발생시점 수 평균	1.094	2.832	3.462	3.8	
	첫 스케줄링 수행회수 평균	26.717	131.755	180.666	464.22	
	두 번째 스케줄링 수행회수 평균	3	3	3.334	7.05	
	스케줄링 종료 시까지의 수행회수 합계 평균	67.942	284.589	475.772	1457.83	

과의 반영으로 인해 수행회수가 매우 적게 일어나면서 대부분 이전 결과를 적합한 결과로 수용하는 것을 알 수 있다. 이는 문맥교환을 줄이는 효과를 나타낸다. 스케줄링 종료 시까지의 수행회수 합계는 여러 번의 스케줄링을 거쳐 주어진 문제를 모두 수행하기까지의 스케줄링을 위한 수행회수의 합계를 문제의 수로 나누어 평균을 낸 것으로 시스템의 규모와 문제의 부하에 따라 변하는 수치이다.

표 4. 3×2 시스템에서의 모의 실험 예
Table 4. Example of simulation on 3×2 system.

	EDA			LLA			신경망		
1	(6,8)	(4,9)	(8,12)	(6,8)	(4,9)	(8,12)	(6,8)	(4,9)	(8,12)
2	(5,7)	(3,8)	(8,11)	(5,7)	(4,8)	(7,11)	(5,7)	(4,8)	(7,11)
3	(4,6)	(2,7)	(8,10)	(4,6)	(4,7)	(6,10)	(4,6)	(4,7)	(6,10)
4	(3,5)	(1,6)	(8,9)	(3,5)	(3,6)	(6,9)	(3,5)	(4,6)	(5,9)
5	(2,4)	(0,5)	(8,8)	(2,4)	(3,5)	(5,8)	(2,4)	(4,5)	(4,8)
6	(1,3)	(0,4)	(7,7)	(1,3)	(2,4)	(5,7)	(1,3)	(4,4)	(3,7)
7	(0,2)	(0,3)	(6,6)	(1,2)	(1,3)	(4,6)	(1,2)	(3,3)	(2,6)
8	(0,1)	(0,2)	(5,5)	(0,1)	(1,2)	(3,5)	(1,1)	(2,2)	(1,5)
9	(0,0)	(0,1)	(4,4)	(0,0)	(0,1)	(2,4)	(0,0)	(1,1)	(1,4)
10	(0,0)	(0,0)	(3,3)	(0,0)	(0,0)	(1,3)	(0,0)	(0,0)	(0,3)
11	(0,0)	(0,0)	(2,2)	(0,0)	(0,0)	(0,2)			
12	(0,0)	(0,0)	(1,1)						
13	(0,0)	(0,0)	(0,0)						
비고	스케줄링 성공 문맥교환 1개, 1회			스케줄링 성공 문맥교환 6개, 6회			스케줄링 성공 문맥교환 3개, 3회		

표 4는 표 3에 포함된 결과 중 하나로 3×2 시스템에서의 스케줄링 예이다. 이 예는 EDA 알고리즘이 스케줄링에 성공한 한 예로 불필요한 문맥교환은 한번도 일어나지 않고 단지 태스크의 수행종료로 인한 문맥교환만이 발생하였다. LLA 알고리즘은 불필요한 문맥교환이 많이 일어나는 특징을 보여주고 있다. 신경망 알고리즘은 LLA보다는 적은 문맥교환이 발생하나 EDA에 못 미치는 결과를 보이고 있다.

표 5는 모의 실험 결과중 한 예로 표 3에 포함되지 않은 결과이나 각각의 알고리즘의 특징을 잘 보여주는 경우이다. 6×4 시스템에서 최소 계산시간은 2 최대 계산시간은 9로 하여 임의의 값을 발생시키고 제한시간은 5이내의 임의의 값을 발생시켜 계산시간과 합한 값으로 하여 표 3의 실험보다 부하를 증가시켜 실험하였다. 부하가 증가되면 EDA는 스케줄링에 성공하지 못하는 경우가 발생하게 된다. 위의 예는 그 중 하나이다. LLA는 문맥교환이 많이 발생하지만 스케줄링에 성공하는 결과를 보이고 있다. 신경망 알고리즘은 문맥교환은 LLA보다 적으면서도 스케줄링에 성공하는 결과를 보이고 있다.

다음의 표 6a, 표 6b, 표 6c는 15×10 시스템에서의 신경망 알고리즘 스케줄링 결과로 신경망 알고리즘의 작동상태를 보여준다. 30×20 시스템과 300×20 시스템에서의 예는 지면관계상 생략한다.

홉필드 신경망에서 초기 입력의 결정이 중요하여 결과에 많은 영향을 미친다. 그러나 위에서 보여진

표 5. 6×4 시스템에서의 모의 실험 예
Table 5. Example of simulation on 6×4 system.

	EDA						LLA						신경망					
1	8,10	5,7	8,11	6,10	7,8	4,6	8,10	5,7	8,11	6,10	7,8	4,6	8,10	5,7	8,11	6,10	7,8	4,6
2	8,9	4,6	8,10	5,9	6,7	3,5	7,9	4,6	8,10	6,9	6,7	3,5	7,9	4,6	7,10	6,9	7,7	3,5
3	8,8	3,5	8,9	4,8	5,6	2,4	7,8	3,5	7,9	6,8	5,6	2,4	6,8	3,5	6,9	6,8	6,6	3,4
4	8,7	2,5	8,9	3,8	4,6	1,4	6,7	3,4	7,8	5,7	4,5	1,3	5,7	2,4	5,8	6,7	5,5	3,3
5							5,6	2,3	6,7	5,6	3,4	1,2	4,6	1,3	5,7	6,6	4,4	2,2
6							5,5	1,2	5,6	4,5	3,3	0,1	3,5	1,2	5,6	5,5	3,3	1,1
7							4,4	1,1	4,5	3,4	2,2	0,0	2,4	1,1	5,5	4,4	2,2	0,0
8							3,3	0,0	4,4	2,3	1,1	0,0	2,3	0,0	4,4	3,3	1,1	0,0
9							2,2	0,0	3,3	1,2	0,0	0,0	1,2	0,0	3,3	2,2	0,0	0,0
10							1,1	0,0	2,2	0,1	0,0	0,0	0,1	0,0	2,2	1,1	0,0	0,0
11							0,0	0,0	1,1	0,0	0,0	0,0	0,0	0,0	1,1	0,0	0,0	0,0
12							0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
비고	스케줄링 실패 문맥교환 없음						스케줄링 성공 문맥교환 11개, 7회						스케줄링 성공 문맥교환 6개, 5회					

모든 결과는 초기 입력을 0으로 초기화한 후 수행한 결과이다.

이것은 종료조건이 우수한 것과 설계된 신경망의 동작이 안정적인 것을 증명해 준다. 모의 실험을 통해 신경망 알고리즘의 특징과 EDA와 LLA과의 비교를 통해 각각의 장단점을 알 수 있었다.

표 6a. 15×10 시스템에서의 신경망 알고리즘 스케줄링 결과

Table 6a. Result of neural network scheduling on 15×10 system.

1	7,1	5,6	4,1	7,1	3,1	7,7	3,7	8,1	5,1	2,4	8,8	5,6	4,1	2,4	8,8	
2	6,1	4,5	4,1	7,1	3,1	6,6	2,6	7,9	5,1	1,3	7,7	4,5	4,9	1,3	7,7	
3	5,9	3,4	4,9	7,1	3,9	5,5	1,5	6,8	5,9	0,2	6,6	3,4	4,8	0,2	6,6	
4	4,8	2,3	4,8	7,1	3,8	4,4	0,4	5,7	4,8	0,1	5,5	2,3	3,7	0,1	5,5	
5	3,7	1,2	3,7	7,1	3,7	3,3	0,3	4,6	3,7	0,0	4,4	1,2	2,6	0,0	4,4	
6	2,6	0,1	2,6	7,9	3,6	2,2	0,2	3,5	2,6	0,0	3,3	0,1	1,5	0,0	3,3	
7	1,5	0,0	1,5	6,8	2,5	1,1	0,1	2,4	1,5	0,0	2,2	0,0	0,4	0,0	2,2	
8	0,4	0,0	0,4	5,7	1,4	0,0	0,0	1,3	0,4	0,0	1,1	0,0	0,3	0,0	1,1	
9	0,3	0,0	0,3	4,6	0,3	0,0	0,0	0,2	0,3	0,0	0,0	0,0	0,2	0,0	0,0	
10	0,2	0,0	0,2	3,5	0,2	0,0	0,0	0,1	0,2	0,0	0,0	0,0	0,1	0,0	0,0	
11	0,1	0,0	0,1	2,4	0,1	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	
12	0,0	0,0	0,0	1,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
13	0,0	0,0	0,0	0,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	

표 6b. 15×10 시스템에서의 신경망 알고리즘 스케줄링 결과

Table 6b. Result of neural network scheduling on 15×10 system.

1	5,6	8,1	8,1	4,4	2,9	8,1	7,1	7,8	8,1	2,7	2,9	8,1	3,6	1,1	2,9	7,1
2	4,5	7,9	7,9	3,3	2,8	7,9	6,9	6,7	8,1	1,6	2,8	7,1	2,5	1,0	2,8	7,1
3	3,4	6,8	6,8	2,2	2,7	6,8	5,8	5,6	8,1	0,5	2,7	6,1	1,4	0,9	2,7	7,1
4	2,3	5,7	5,7	1,1	2,6	5,7	4,7	4,5	7,1	0,4	2,6	5,1	0,3	0,8	2,6	7,9
5	1,2	4,6	4,6	0,0	2,5	4,6	3,6	3,4	6,9	0,3	2,5	4,9	2,7	2,5	7,8	
6	0,1	3,5	3,5	0,0	2,4	3,5	2,5	2,3	5,8	0,2	2,4	3,8	1,6	2,4	6,7	
7	0,0	2,4	2,4	0,0	1,3	2,4	1,4	1,2	4,7	0,1	2,3	2,7	0,5	2,3	5,6	
8	0,0	1,3	1,3	0,0	0,2	1,3	0,3	0,1	3,6	0,0	1,2	1,6	0,4	2,2	4,5	
9	0,0	0,2	0,2	0,0	0,1	0,2	0,2	0,0	2,5	0,0	0,1	0,5	0,3	1,1	3,4	
10	0,0	0,1	0,1	0,0	0,0	0,1	0,1	0,0	1,4	0,0	0,0	0,4	0,2	0,0	2,3	
11	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,3	0,0	0,0	0,3	0,1	0,0	1,2	
12	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2	0,0	0,0	0,2	0,0	0,0	0,1	

표 6c. 15×10 시스템에서의 신경망 알고리즘 스케줄링 결과

Table 6c. Result of neural network scheduling on 15×10 system.

1	5,13	7,14	7,13	8,16	6,10	8,8	7,7	4,12	2,7	8,11	7,13	8,12	6,12	8,9	5,10	
2	5,12	7,13	6,12	8,15	5,9	7,7	6,6	4,11	1,6	7,10	7,12	7,11	5,11	7,8	4,9	
3	5,11	7,12	5,11	8,14	4,8	6,6	5,5	4,10	0,5	6,9	7,11	6,10	4,10	6,7	3,8	
4	5,10	7,11	4,10	8,13	3,7	5,5	4,4	4,9	0,4	5,8	6,10	5,9	3,9	5,6	2,7	
5	5,9	7,10	3,9	8,12	2,6	4,4	3,3	4,8	0,3	4,7	5,9	4,8	2,8	4,5	1,6	
6	5,8	7,9	2,8	8,11	1,5	3,3	2,2	4,7	0,2	3,6	4,8	3,7	1,7	3,4	0,5	
7	5,7	6,8	1,7	8,10	0,4	2,2	1,1	4,6	0,1	2,5	3,7	2,6	0,6	2,3	0,4	
8	4,6	5,7	0,6	8,9	0,3	1,1	0,0	3,5	0,0	1,4	2,6	1,5	0,5	1,2	0,3	
9	3,5	4,6	0,5	7,8	0,2	0,0	0,0	2,4	0,0	0,3	1,5	0,4	0,4	0,1	0,2	
10	2,4	3,5	0,4	6,7	0,1	0,0	0,0	1,3	0,0	0,2	0,4	0,3	0,3	0,0	0,1	
11	1,3	2,4	0,3	5,6	0,0	0,0	0,0	0,2	0,0	0,1	0,3	0,2	0,2	0,0	0,0	
12	0,2	1,3	0,2	4,5	0,0	0,0	0,0	0,1	0,0	0,0	0,2	0,1	0,1	0,0	0,0	
13	0,1	0,2	0,1	3,4	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0	
14	0,0	0,1	0,0	2,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
15	0,0	0,0	0,0	1,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
16	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	

2. 하드웨어 모의 실험 결과

홉필드 네트워크를 이용하여 문제를 해결하는 경우 하드웨어의 설계는 필수적이다. 기존의 연구에서는 소프트웨어 모의 실험을 이용하여 알고리즘만을 증명하나 알고리즘이 복잡하여 하드웨어로의 구현이 불가능할 경우 그 알고리즘은 시스템으로의 구현이 불가능하다. 홉필드 네트워크의 소프트웨어적인 구현은 현재의 폰노이만 컴퓨터에서는 이점을 가져오지 못한다. 따라서 홉필드 네트워크를 이용한 알고리즘 설계시에는 상태방정식의 작성시 하드웨어로의 구현 가능성을 검토하고 하드웨어로의 설계가 용이하도록 하여야 한다. 홉필드 네트워크는 같은 구조의 뉴런을 반복 배치하는 구조를 지니므로 하드웨어로 구현시 모듈화가 간편하고, 더 큰 문제의 해결을 위해서 모듈을 추가하는 방식으로의 설계가 가능하다. 그러나 반복 배치하는 특성으로 인해 뉴런의 설계가 잘못되면 많은 하드웨어의 손실을 가져온다. 따라서 하드웨어로의 구현시 뉴런에 공통적인 부분은 뉴런의 외부로 빼내어 뉴런의 규모를 줄이고 효율성을 높이는 작업이 필요하다. 신경망의 하드웨어는 뉴런의 설계시 하드웨어의 낭비가 없도록

주의하여야 한다. 뉴런은 시스템의 규모만큼 반복되어 사용되므로 반복되어 사용될 필요가 없는 부분은 추출하여 별도의 블록으로 생성하여 하드웨어의 효율성을 높여 낭비가 없도록 해야 한다. 다음은 VHDL로 기술된 뉴런의 대략적인 내부구조이다.

표 7. VHDL로 기술한 뉴런의 내부구조
Table 7. Structure of neural network design by VHDL.

```
architecture BEHAVIORAL of NEURON is
  begin
        process
        begin
              wait until clk'event and clk='1';
              if rst='1' then
--                  Δu의 초기화
              else
--                  Δu의 계산
        end process;
        process
        begin
              wait until clk'event and clk='0';
              if rst='1' then
--            U의 초기화
              elsif en='0' then
--                  U의 갱신
              end if;
        end process;
        process(rst, U)
        begin
              if rst='1' then
--                  V값의 초기화
              elsif (U > UTP) then
--                  V<='1';
              elsif (U < LTP) then
--                  V<='0';
              end if;
        end process;
end BEHAVIORAL;
```

VHDL 기술시 구조를 최적화하기 위해 뉴런 내부의 U 의해 출력인 V의 출력이 동기 되어 동기신호의 1주기마다 전체 동작이 수행되도록 하였다. 이를 위해 process 문을 이용하여 적절히 나누어 설계하였다. 그림 3은 VHDL로 기술된 하드웨어를 Synopsys 소프트웨어를 이용하여 모의 실험한 결과이다. 그림 3에서 세로방향의 점선10개는 각각의 계산 종료시점을 의미하여 그때의 출력은 각 태스크에 대한 프로세서의 할당 결과이다. 점선 10개는 주어진 문제가 10번의 스케줄링을 통해 모두 수행될 수 있음을 의미한다. 20ns의 동기신호를 이용한 수행에서 1000번 정도의 최대수행회수를 보인다면 20μs이내에 수행될 수 있음을 의미한다. 위의 결과는 20ns의 동기신호를 이용한 각각의 스케줄링에서 수행회수 250번, 약 5μs이내에 수행되는 것을 볼 수 있다. 따라서 설계된 블록 중 가장 복잡한 뉴런이 20ns이내에 동작되도록 제작된다면 5μs정도의 계산시간을 갖는 6×4 시스템의 구현이 가능하다.

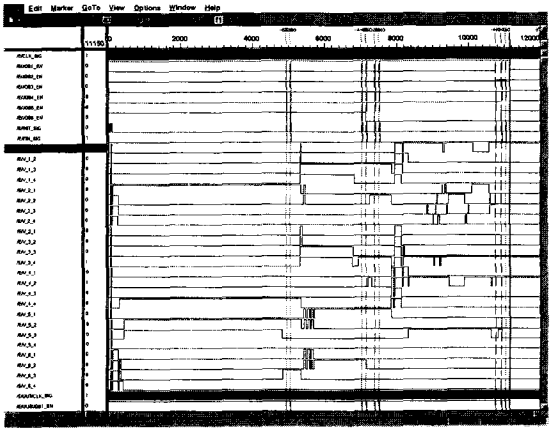

그림 3. VHDL로 기술된 하드웨어 모의 실험 결과
Fig 3. Result of a hardware simulation.

표 8은 설계된 하드웨어의 모의실험 결과를 소프트웨어 모의실험 결과와 비교한 것이다. 신경망 하드웨어 모의 실험 결과에서 괄호 왼편의 번호는 프로세서 번호이다. 신경망 하드웨어 모의 실험 결과는 신경망 소프트웨어 모의 실험 결과와 성능에 있어 비슷하지만 다른 결과를 보이고있다. 소프트웨어 구현은 사용하기에 따라 얼마든지 많은 bit의 수를 사용할 수 있지만 하드웨어는 그 bit수가 많아질수록 자원의 낭비가 많아지고 또한 가격도 올라가게 되므로 되도록 최대한 bit수를 적게 사용하는 것이 좋다. 이와같은 이유로 하드웨어로 구현시 bit수가 줄어들었고 그에 따라 몇개의 계수가 변경되었기 때문이다. 또한 신경망 알고리즘의 소프트웨어 모의실험에서는 랜덤하게 초기화를 시키기 때문에 특정한 값으로 초기화를 시킨 하드웨어 모의실험과는 그 시작부터 차이가 생기게 된다.

표 8. 6×4 시스템에서의 소프트웨어 모의실험과 하드웨어 모의실험의 비교

Table 8. Compare a software simulation with a hardware simulation in 6×4 system.

	신경망 소프트웨어						신경망 하드웨어					
1	8,10	5,7	8,11	6,10	7,8	4,6	8,10	1)5,7	8,11	4)6,10	2)7,8	3)4,6
2	7,9	4,6	7,10	6,9	7,7	3,5	8,9	1)4,6	8,10	4)5,9	2)6,7	3)3,5
3	6,8	3,5	6,9	6,8	6,6	3,4	3)8,8	3,5	4)8,9	4,8	1)5,6	2)2,4
4	5,7	2,4	5,8	6,7	5,5	3,3	3)7,7	3,4	4)7,8	4,7	1)4,5	2)1,3
5	4,6	1,3	5,7	6,6	4,4	2,2	3)6,6	2)3,3	4)6,7	4,6	1)3,4	0,2
6	3,5	1,2	5,6	5,5	3,3	1,1	3)5,5	2)2,2	4)5,6	4,5	1)2,3	0,1
7	2,4	1,1	5,5	4,4	2,2	0,0	1)4,4	2)1,1	4)4,5	3)4,4	1,2	0,0
8	2,3	0,0	4,4	3,3	1,1	0,0	1)3,3	0,0	4)3,4	3)3,3	2)1,1	0,0
9	1,2	0,0	3,3	2,2	0,0	0,0	1)2,2	0,0	4)2,3	3)2,2	0,0	0,0
10	0,1	0,0	2,2	1,1	0,0	0,0	1)1,1	0,0	4)1,2	3)1,1	0,0	0,0
11	0,0	0,0	1,1	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,0
12	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
비고	스케줄링 성공 문맥교환 6개, 5회						스케줄링 성공 문맥교환 9개, 4회					

V. 결 론

본 논문에서는 기존의 실시간 단일프로세서 스케줄링 알고리즘인 EDA와 LLA를 m개의 태스크와 n개의 프로세서에 대해 m×n개의 뉴런을 상호 연결한 신경망 홉필드 네트워크로 결합하고 보완하여, 멀티프로세서 스케줄링 알고리즘으로 응용하고 평가하였다. 고안된 알고리즘은 실시간 멀티프로세서에서 부최적인 LLA와 같은 스케줄링 성공률을 보였으므로 부최적임을 알 수 있었다. 문맥교환은 스케줄링 성공률이 떨어지는 EDA보다는 많이 발생하였으나 부최적인 LLA보다 적게 발생하는 결과를 보였다. 시스템의 규모를 100배로 증가시켜 실험한 결과 계산시간은 7배미만의 증가를 보여 설계된 알고리즘은 대규모의 실시간 멀티프로세서 시스템에 적합한 알고리즘임을 알 수 있었다. 또한 VHDL을 이용한 하드웨어 구현을 통해 하드웨어에서의 신경망 알고리즘의 성능을 검증하였다.

설계된 알고리즘은 기존의 EDA와 LLA의 장점만을 취한 형태의 알고리즘으로 상태방정식에 포함된 EDA의 성질과 이전 스케줄링의 결과를 현재 스케줄링에 이용하는 기법을 통하여 문맥교환을 줄일 수 있었으며, LLA의 성질과 홉필드 네트워크의 특성을 통해 실시간 멀티프로세서 스케줄링 알고리즘으로써의 부최적성을 확보하였다. 설계된 신경망 6×4 실시간 멀티프로세서 스케줄링 하드웨어는 소프트웨어 모의실험에서 얻은 결과를 20ns의 동기신호를 통한 구동으로 20μs정도의 빠른 시간에 해결할 수 있었다.

향후 실제적인 태스크 모델의 적용하여 알고리즘의 실제 시스템 적용가능여부에 대한 검증과 과부하 시에 대한 추가 연구가 요구된다. 또한 준비된 태스크를 통해 검증된 부최적성에 그치지 않고 동적인 임의의 태스크의 스케줄링 요구 통한 최적성에 대한 성능평가도 필요하다. 추가 연구로 개선된 알고리즘은 하드웨어로 구현하여 그 계산시간을 통해 실제 시스템 적용가능성이 검토되어야 한다.

참 고 문 헌

[1] T. G. Lewis and H. Ei-Rewini, "Parallax : A Tool for Parallel Program Scheduling", IEEE Parallel & Distributed Technology, May 1993.

[2] J. D. Ullman, "NP-complete scheduling problems", J. Comput.Syst.Sci, oct. 1975.

[3] 김희수, 조용범, 최종욱, "전문가 시스템", 집문당, 175-186, 204-213, 1995

[4] 김대수, "신경망 이론과 응용(I)", 145-165, 1994

[5] J. J. Hopfiled and D. Tank, "Neural Computation of Decisions in Optimization Problems", Biological Cybernetics, vol. 52, pp. 141-152, 1985.

[6] Y. Takefuji and K. C. Lee, "A super parallel sorting algoritm based on neural networks," IEEE Transactions on Circuits and Systems, vol. 37, pp. 1425-1429, Nov. 1990.

[7] S. K. Lee and J. T. Lim, "A Perfomance Analysis of Real-Time On-Line Scheduling Algorithms", 한국정보과학회 학술발표논문집 Vol. 22, No 1, 1995.

[8] Y. B. Cho, "Silicon Neural Networks for Optimization Problems", Ph. D. Thesis, Case Western Reserve University, May 1992.

저 자 소 개

李 在 衡(正會員)

1974년 9월 28일생. 1997년 2월 서경대학교 컴퓨터과학과 학사. 2000년 2월 건국대학교 전자공학과 석사. 2000년 2월~현재 건아정보기술 연구개발팀. 주관심 분야는 시스템 프로그래밍 및 네트워크 통신 프로그램

李 康 彰(正會員)

1973년 11월 4일생. 1996년 2월 건국대학교 전자공학과 학사. 1998년 2월 건국대학교 전자공학과 석사. 1998년 2월~현재 건아정보기술 연구개발팀. 주관심 분야는 시스템 하드웨어 및 프로그래밍

趙 鏞 範(正會員) 第 35券 C編 第 5號 參照