The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 15, No. 6, pp.115-121, Dec. 31, 2015. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2015.15.6.115
JIIBC 2015-6-16

고성능 멀티프로세서를 위한 유전 알고리즘 기반의 반복 데이터흐름 최적화 스케줄링 알고리즘

An Iterative Data-Flow Optimal Scheduling Algorithm based on Genetic Algorithm for High-Performance Multiprocessor

장정욱*, 인치호**

Jeong-Uk Chang*, Chi-Ho Lin**

요 약 본 논문에서는 멀티프로세서 아키텍처 상에 반복적인 데이터흐름 알고리즘을 스케줄링하는 방법을 제안한다. 기본적인 하드웨어 모델을 기반으로 멀티프로세서 아키텍처라는 세부적인 특성을 가지도록 확장하여 용량이 제한된 통신 네트워크상에 전송할 데이터를 라우팅 하는데 필요한 하드웨어 모델을 구현하고, 스케줄링 방법을 적용한다. 제안한 스케줄링 방법은 세 가지 계층으로 구성된다. 가장 상위 계층에 구현된 유전 알고리즘은 반복 데이터흐름 그래프의 최적화를 담당한다. 유전 알고리즘은 대상이 되는 연산들에 대해 서로 다른 조합을 생성한다. 그리고서 이 조합들은 중간 계층으로 전달된다. 이 중간 계층에는 전역 스케줄링이 위치하며, 연산들의 조합을 바탕으로 스케줄링에 관한 주요 결정을 이 스케줄이 내리게 된다. 마지막으로, 하부 계층에서는 하드웨어 세부사항을 고려하며 블랙-박스 스케줄링을 이용한다. 연산에 대한 스케줄링을 완료하고, 세부적인 하드웨어 모델이 이 결정을 준수하는지 확인한다. 스케줄 사이에 사이클을 삽입할 수 있는 두 가지 스케줄링을 통해 유효한 스케줄을 항상 빨리 찾아낼 수 있다. 본 논문에서 제안한 스케줄링 방법의 성능을 테스트하기 위하여 다섯 가지 필터들에 대한 벤치마크를 수행하여 합당한 시간 안에 양질의 스케줄을 찾아낼 수 있음을 입증한다.

Abstract In this paper, we proposed an iterative data-flow optimal scheduling algorithm based on genetic algorithm for high-performance multiprocessor. The basic hardware model can be extended to include detailed features of the multiprocessor architecture. This is illustrated by implementing a hardware model that requires routing the data transfers over a communication network with a limited capacity. The scheduling method consists of three layers. In the top layer a genetic algorithm takes care of the optimization. It generates different permutations of operations, that are passed on to the middle layer. The global scheduling makes the main scheduling decisions based on a permutation of operations. Details of the hardware model are not considered in this layer. This is done in the bottom layer by the black-box scheduling. It completes the scheduling of an operation and ensures that the detailed hardware model is obeyed. Both scheduling method can insert cycles in the schedule to ensure that a valid schedule is always found quickly. In order to test the performance of the scheduling method, the results of benchmark of the five filters show that the scheduling method is able to find good quality schedules in reasonable time.

Key Words : Genetic Algorithm, Scheduling Method, Multiprocessor, Iterative Data-Flow, DSP

*준회원, 세명대학교 컴퓨터학부
**정회원, 세명대학교 컴퓨터학부(교신저자)
접수일자: 2015년 10월 2일, 수정완료: 2015년 11월 2일
게재확정일자: 2015년 12월 11일

Received: 2 October, 2015 / Revised: 2 November, 2015 / Accepted: 11 December, 2015
**Corresponding Author: ich410@semyung.ac.kr
School of Computer, Semyung University, Korea

Ⅰ. 서 론

최근 DSP(Digital Signal Processing)는 더욱 진보되어 어디에서나 흔하게 사용되고 있다. 또한 VLSI 기술의 진보로 인한 하드웨어 실행 속도가 엄청나게 빨라졌고, 이로 인해 매우 다양한 종류의 신호들을 디지털 회로를 이용해 처리할 수 있게 되었다. 얼마 전까지만 해도 디지털 신호 처리는 예를 들면 사람의 목소리처럼 낮은 대역폭의 신호에만 국한 되었다. 그러나 요즘은 DSP를 활용한 응용이 고품질의 오디오 데이터를 실시간 처리하는데 까지 확장되었을 뿐 아니라, 심지어는 움직이는 이미지를 처리하는 수준까지 확장되기에 이르렀다.[1-3]

고품질 오디오/비디오 데이터의 실시간 신호의 경우, 이를 처리하려면 과거 DSP 응용이 요구했던 것보다 훨씬 더 높은 수준의 컴퓨팅 파워를 필요로 한다. 추가로 필요한 신호처리력은 기술의 진보로 일부 해결되었으며, DSP 알고리즘의 특성에 적합하도록 특수 설계된 DSP 프로세서의 활용 또한 이를 뒷받침해주고 있다. 그럼에도 불구하고 여전히 더 나은 컴퓨팅 파워가 요구되는 상황이다. 이런 요구조건에 맞춰 멀티프로세서 상에서 병렬적으로 수행할 수 있는 알고리즘을 고려해보는 것이 유용하다. 그러나 이러한 멀티프로세서는 알고리즘의 스케줄링에 세심한 주의가 필요하다. 각각의 연산을 어느 하나의 프로세서에 배정하여야 하고, 이 연산의 수행을 언제 시작할지도 결정하여야 한다. 이러한 문제들은 그 자체만으로도 이미 어려운 문제인데, 멀티프로세서 아키텍처에서 고려해야 할 통신지연 및 다른 측면들까지 감안한다면 훨씬 복잡해질 수밖에 없다.[4-6]

본 논문은 복잡하고 자세한 하드웨어 모델에서 활용할 수 있는 고성능 멀티프로세서에서의 스케줄링 방법을 제안한다. 통신지연이 고려된 일반적인 하드웨어 모델을 기초로, 보다 특수한 멀티프로세서의 특성을 포함시키기 위하여 확장된 하드웨어 모델에 제안한 스케줄링 방법을 적용한다. 또한 제한적 용량을 가진 통신 링크를 통해, 데이터 전송의 라우팅을 필요로 하는 하드웨어 모델에 적용하여 본 논문에서 제안한 접근법이 실현가능하다는 것을 보여줄 것이다. 특별히 본 논문에서 다룰 DSP 알고리즘은 일반적으로 반복 수행되어 오버랩 되는 스케줄을 만들어낼 수 있어서, 병렬성을 충분히 활용할 수 있게 해주는 좋은 모델이 되기 때문에, 제안된 스케줄링 방법은 반복 데이터흐름을 가지는 알고리즘을 대상으로 고려하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 제안한 스케줄링 방법의 설계 및 구현에 대하여 기술한다. 3장에서는 구현된 스케줄링 알고리즘을 적용하여 얻어진 결과에 대하여 기술한다. 끝으로 4장에서는 결론을 맺는다.

Ⅱ. 혼합 유전 알고리즘 기반의 스케줄링 방법의 설계 및 구현

1. 스케줄링 개요

제안한 스케줄링 방법은 그림 1에 표현되어 있듯이 세 개의 계층으로 구성된다. 상위 계층에 위치한 유전 알고리즘은 주요 최적화 목표를 다루는 역할을 한다. 이것은 혼합 유전 알고리즘(Hybrid Genetic Algorithm)으로, 스케줄을 위한 연산의 순서를 이 유전 알고리즘이 제공하고, 스케줄 생성을 위하여 하위에 위치한 두 개의 문제-특화된 스케줄링 알고리즘이 최적화를 담당한다. 결론적으로 제안하는 유전 알고리즘은 이러한 순서를 변경하는 방식으로 스케줄을 최적화한다.

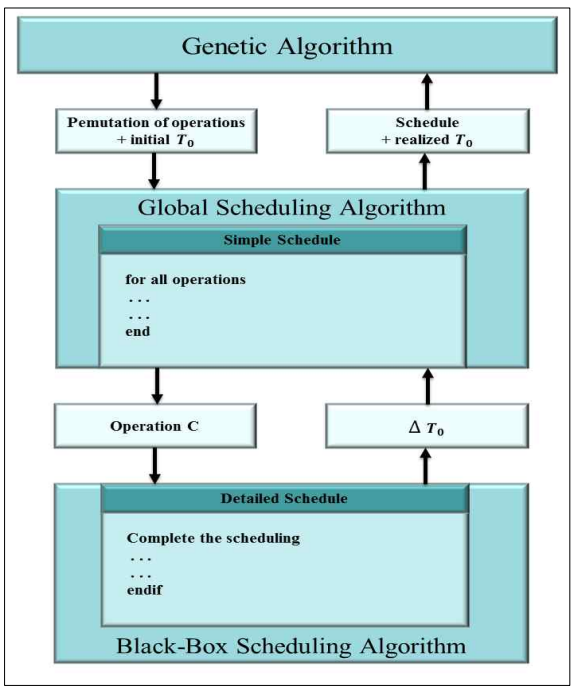

그림 1. 제안한 스케줄링 방법
Fig. 1. The proposed scheduling method

중간 계층은 전역 스케줄링으로 구성된다. 이 전역 스케줄링은 일반적인 멀티프로세서 구성에 바탕을 둔다. 이러한 일반적 멀티프로세서 구성은 모든 가용한 기능 유닛과 단순한 모델의 통신 네트워크를 포함하고 있다. 따라서 기능 유닛의 모든 쌍 사이의 통신지연을 최소화하는 모델라고 할 수 있다. 전역 스케줄링은 스케줄링을 위한 결정을 내릴 때에는 멀티프로세서 모델의 세부적인 특성은 무시한다. 결과로 얻은 스케줄이 멀티프로세서 상에 확실히 수행 가능한지 보장하기 위하여 전역 스케줄링은 세 번째 계층에 위치한 블랙-박스 스케줄링과 함께 쓰인다. 즉, 전역 스케줄링은 고정된 방식으로 블랙-박스 스케줄링과 통신하며 블랙-박스 스케줄링의 행위에 적응할 필요가 없다. 블랙-박스 스케줄링은 멀티프로세서 아키텍처의 세부사양을 활용하여 각 연산의 스케줄을 완료한다. 멀티프로세서 모델에 따라, 입력 피연산자는 기능 유닛의 입력으로 이동하여야 하며, 프로세서 간 데이터 전송이 라우팅 되어야 하며, 레지스터가 할당되어야 한다.

2. 전역 스케줄링

본 논문에서 제안하는 전역 스케줄링의 특성으로는, 첫째로 연산이 스케줄 되는 순서는 연산의 조합에 의해 결정되기 때문에 유전 알고리즘을 사용하여 최적의 조합을 탐색하게 된다. 두 번째로, 스케줄링은 반복 데이터들 간의 병렬성을 활용할 수 있다. 스케줄이 하나의 반복 주기 내로 들어가게 되면, 서로 다른 반복에 속한 연산이 동시에 고려될 수 있기 때문이다. 세 번째로, 이 스케줄링은 연산들 간의 선행성 제약 효과를 고려한다. 이것은 직접적 또는 간접적 선행성 제약을 위반하는 방식으로는 절대로 스케줄 되지 않게 보장한다. 하지만 어떤 연산을 스케줄 해야 할 때 스케줄 내에 빈 슬롯이 항상 충분히 남아 있다고 보장할 수는 없다. 따라서 빈 사이클을 스케줄 중간에 끼워 넣을 수 있다는 스케줄링의 네 번째 특성이 이 문제를 극복한다.

스케줄링 중간에 추가 사이클을 삽입할 경우, 스케줄의 질이 떨어질 수 있는데 스케줄링의 최종 단계에 사이클이 삽입될 때 자주 발생될 수 있다. 그러나 새로운 사이클 삽입으로 인해 항상 수행 가능한 스케줄을 만들 수 있다. 이러한 특성은 유전 알고리즘의 효율성에 긍정적 영향을 미친다. 더 나아가, 초기의 반복 기간을 스케줄에 제공할 수 있다.

```
int function_get_pref_time_bounded(int c, int R_nc)
    {/* "Determine the preferred
        start time t_pref for operation c.
        R_nc is the range of valid start times of
        the operation and it is bounded." */
    begin:
    {/* "Global variables the are used:
        N_pred_s : The number of direct predecessor operations that
                   have already been scheduled.
        N_succ_s : The number of direct successor operations that
                   have already been scheduled.
        N_pred_t : The total number of direct predecessor operations.
        N_succ_t : The total number of direct successor operations. " */

 if (N_pred_s < N_succ_s)
    t_pref = R_nc_max;
 else if (N_pred_s > N_succ_s)
    t_pref = R_nc_min;
 else if (N_pred_t < N_succ_t)
    t_pref = R_nc_min;
 else if (N_pred_t > N_succ_t)
    t_pref = R_nc_max ;
 else
    t_pref = 0;
 return t_pref ;
    }
 }
```

그림 2. 유효 스케줄링 인스턴스 코드
Fig. 2. The pseudocode of effective scheduling instance

스케줄링 문제가 주어졌을 때, IPB(Iteration Period Bound)가 단지 DFG(Data-Flow Graph)의 단순한 특성과 하드웨어 구성에만 기반을 두어 계산되기 때문에 주어진 스케줄링 문제에 대한 반복주기의 최소값은 IPB로 정해지는 이론적 반복주기 값보다 커질 수 있다. 그리하여 이 초기 반복 기간이 IPB보다 큰 값으로 선택되었으나, 이것이 반복 기간이 실질적으로 가질 수 있는 최소값보다는 여전히 같거나 낮은 값이라면, 스케줄의 질 저하는 줄어들 수 있을 것이다.

그림 2와 그림 3은 각각 유효 스케줄링 인스턴스의 시작시간에 대한 의사코드와 스케줄에 어떻게 연산이 삽입될 수 있는지를 결정하는 알고리즘을 나타내었다.

3. 블랙-박스 스케줄링

전역 스케줄링이 연산을 FU에 할당하며 시작 시간을 선택하여 작업을 진행할 때, 연산들 사이의 선행관계와 최소 통신지연을 고려하지만 전역 스케줄링은 멀티프로세서 아키텍처로 인한 추가적 제약을 무시한다.

```
int function_get_num_free(int schedule, int fu, int col);
    /* "Returns the number of consecutive empty cycles in the schedule
     on FU fu, starting at column col." */
int function_col_delta(int schedule, int col1, int col2);
    /* "Returns the number of cycles that column col2 lies beyond
        column col1." */
int function_add_time(int time, int delta);
    /* "Returns the time time increased by delta." */
int function_how_to_insert(int schedule, int c, int fu, int starttime) {
begin:
    /* "Can operation c be scheduled at the proposed starttime or is it
        in the middle of another operation and should it be moved?" */
    c2 += function_get_num_free(schedule, fu, col_starttime);
    if ( c2=EMPTY || col(c2.starttime)==col(starttime) )
        shift == 0;
    else
        shift = c2.duration - col.delta(schedule, col(starttime), col(c2.starttime));
end:
    N1+= shift;
        /* Insert as many cycles as operation c is shifted, to avoid
           that any precedence relations might be violated." */
        /* Is there already enough room in the schedule or should extra
           cycles be inserted?" */
    F = function_get_num_free(schedule, fu, time(starttime, shift));
    N2 = max(0, c2.duration - N1 - F);
    N = N1 + N2;
        /* The operation can be scheduled with a start time that is shift
           time cycles to the right of the proposed start time. N cycles
           have to be inserted before column col(starttime)." */
return shift, N;
```

그림 3. 연산의 삽입 알고리즘
Fig. 3. The insertion algorithm of operation

이러한 제약을 고려하기 위해, 전역 스케줄링은 블랙-박스 스케줄링을 호출하는데, 이 블랙-박스 스케줄링은 멀티프로세서 모델의 정확한 세부사항을 활용하고, 연산의 스케줄링을 완료한다. 이를 테면 이것은 현재 연산과 이것의 선행 및 후행 연산과의 모든 통신을 스케줄 한다.

블랙-박스 스케줄링은 전역 스케줄링에 의해 연산이 스케줄 되는 방식을 반드시 따라야 한다. 어떤 연산을 다른 FU 상에 스케줄 하는 편이 보다 편리하다 할지라도 다른 FU에 연산을 스케줄해서는 안 된다. 그러나 블랙-박스 스케줄링은 자신의 내부와 세부적인 스케줄을 변경하는 것은 허용된다. 이것은 연산을 스케줄 할 수 있도록 보장하기 위해서 반드시 필요한 과정이다. 이런 과정의 발생 여부는 블랙-박스를 지원하는 멀티프로세서 모델에 따라 달라진다.

블랙-박스 스케줄링은 항상 연산을 스케줄 할 수 있어야 하며, 그 결과로 얻은 스케줄은 항상 유효한 것이어야 한다. 이것은 블랙-박스 스케줄링이 스케줄 내에 새로운 사이클을 삽입할 수 없다면 항상 보장되지 못한다. 그러므로 블랙-박스 스케줄링은 스케줄 내에 사이클 삽입이 허용된다. 예를 들면 통신 네트워크상에 경쟁이 존재할 때 이것이 필요할 수 있다. 연산에 대한 스케줄링이 완료되면, 스케줄에 삽입한 사이클의 수를 전역 스케줄링에 리턴 한다. 그 후, 전역 스케줄링은 스케줄 내에 주어진 위치에서 연산의 스케줄 필요 여부(시작 시간과 FU)를 결정한다. 또는 현재의 연산을 스케줄 하는데 있어 또 다른 가능성이 존재하는지 알아보기 위해 탐색을 계속할지를 결정할 수 있다.

```
void genetic_algorithm() {
    generations = 0;
    evaluations = 0;
/* "Initialize population" */
    new.population = {};
    for (i=1; i<population_size; i++) {
        organism.genotype = random_genotype();
        organism.phenotype = evaluate(organism.genotype);
        organism.fitness = calculate_fitness(organism.phenotype);
        evaluations = evaluations + 1;
        new.population = new.population + organism;
    }
/* "Evolve population" */
    while(new.population>generation.limit) {
        generations = generations + 1;
        old.population = new.population;
        new.population = {};

        for (i=1; i<population_size; i++) {
            parent1 = select.parent(old.population);
            parent2 = select.parent(old.population);
            child.genotype = crossover(parent1, parent2);
            child.phenotype = evaluate(child.genotype);
            child.fitness = calculate_fitness(child.phenotype);
            evaluations = evaluations + 1;
            new.population = new.population + child;
        }
    }
    return new.population;
}
```

그림 4. 유전 알고리즘의 프레임워크
Fig. 4. A framework of genetic algorithm

4. 유전 알고리즘 구현

유전 알고리즘의 구현을 위하여, 각 유기체의 유전형은 단 하나의 염색체로 구성된다. 이 염색체는 스케줄 되어야 할 연산들의 조합을 나타내는 시퀀스 염색체이다. 따라서 이것의 길이는 DFG 상에 나타난 연산 노드의 수

와 동일하고, 유저가 제공하는 하드웨어 상에 수행할 수 있다.

유전형을 바탕으로 스케줄 구성을 위해 필요한 평가 함수는 블랙-박스 스케줄링 알고리즘과 함께 결합된 전역 스케줄링 알고리즘이 사용된다. 모든 유기체의 적합도는 이것이 나타내는 스케줄의 두 가지 특성인 반복주기와 지연에 의해 결정된다. 적합도 함수는 지연을 최소화하는 것보다 반복주기를 최소화하는 것이 비용절감 효과가 크다.

유전 알고리즘은 G.P.W. Williams, Jr.가 만든 GECO(Genetic Evolution through Combination of Objects) 패키지 버전 2.0에 구현되어 있다.[7-8] 본 논문에서는 그림 4의 유전 알고리즘의 프레임워크같이 Visual C++를 이용하여 유전 알고리즘을 프로토타이핑 하는 프레임워크로 확장하여 적용하였다.

III. 실험 및 결과

제안된 스케줄링 방법의 성능을 테스트하기 위해 다섯 개의 DFG를 사용하였다. 다섯 가지 필터에 대한 약칭으로 Second-order digital filter[9]는 Second, Third-order digital filter[10]는 Third, Fourth-order all-pole lattice filter[11]는 Lattice, Fourth-order Jaumann wave digital filter[12]는 Jaumann, Fifth-order wave digital elliptic filter[13]는 Elliptic로 각각 표기하여 구분하였다.

실험에 필요한 연산의 덧셈과 뺄셈은 1TU를 필요로 한다. 두 연산은 동일한 수행기간을 가지기 때문에, 따로 떼어 생각하지 않아도 무방하다. 즉 "+" 기호로 이 두 연산을 모두 나타낼 것이다.

곱셈의 경우, 수행기간은 언제나 동일하지 않다. 그러나 이 스케줄링 문제에서는 각 필터가 고정적인 곱셈 수행기간을 가지는 것으로 하였다. 이렇게 하면, 각 필터에 대해 성능 한계의 집합이 단 하나만 존재하게 된다는 장점을 가진다. 각 필터에 대한 곱셈의 수행기간은 표 1에서 찾을 수 있다. 이 표는 또한 성능 한계인 PDB(periodic delay bound)와 IPB 각각을 결정하는 critical path와 critical loop를 보여준다. IPB, PDB, PB(processor bound) 값은 표 1에 나와 있다. PB를 계산하기 위해서는 연산의 수가 필요하다. 그래서 이 값이 주어진다. 성능 한계는 찾아낸 스케줄의 질을 결정하는데 도움이 된다.

표 1. 다섯 개의 벤치마크 필터에 대한 성능 한계
Table 1. The performance bounds for the five benchmark filters

DFG	Dur.		Num.		IPB	PDB	PB
	+	*	+	*			
Second	1	2	4	4	3	3	4
Third	1	2	6	6	3	4	6
Lattice	1	5	11	4	14	28	3
Jaumann	1	5	13	4	16	9	3
Elliptic	1	2	26	8	16	14	3

스케줄링 방법의 설정은 표 2에서 제시한 것과 동일하며, 실험 결과는 Visual C++로 구현한 프로그램을 HP Z640-E5 Workstation 서버 상에 실행하여 확인하였다.

표 2. 벤치마크 수행에 사용되는 스케줄링 방법 설정
Table 2. Configuration of the scheduling method used to benchmark test

Genetic algorithm	Population size	60
	Crossover	80% UNIPERM (copy bias=0.50)
	Mutation	0%
	Selection	tournament (size=6)
	Termination	margin=10%, fraction=90% or max number of evaluations=5000
Heuristics	$T_{0,initial}$	IPB, unless stated otherwise
Results	Number of runs	50

스케줄링 결과는 표 3에서 찾을 수 있다. 평가의 횟수의 평균, 그리고 수행(Run)당 평균 시간은 실행 속도를 나타내주는 지표가 된다. 다음으로, 찾아낸 스케줄 중 가장 좋은 것을 설명한다. 첫째, 최소의 T_0을 스케줄을 생성한 수행의 분수와 함께, 찾아낸 최소 반복주기가 주어진다. 두 번째로, 최소 T_0를 가지는 스케줄에 대해, 찾아낸 지연값 중 최소값을 나타낸다. 최소의 T_0와 최소의 지연을 가지는 스케줄을 만들어낸 수행의 분수가 또한 주어진다.

표 3. 다섯 가지 벤치마크 필터에 대한 스케줄링 결과
Table 3. Scheduling results for the five benchmark filters

	Num.	Dur.		Avg.	Avg.	Best Schedule	
	FUs	+	*	evals	run-time	IP	Latency
Second	4	1	2	324	5s	3 1.00	3 0.98
Third	6	1	2	406	11s	3 1.00	4 0.36
Lattice	3	1	5	237	6s	14 1.00	29 1.00
Jaumann	3	1	5	217	6s	16 1.00	9 1.00
Elliptic	3	1	2	834	76s	17 1.00	14 1.00

두 성능한계를 제외하고 모든 것이 충족되었음을 알 수 있다. 필터 Lattice에 대한 지연은 PDB에 비해 한 시간유닛이 더 길다. 그러나 찾아낸 스케줄은 최적이다.

최상의 스케줄은 시간 중 오직 36%에서만 발견되었다. 왜냐하면 T_0의 최소화, 그리고 지연의 최소화는 서로 상충되는 목표이기 때문이다. 또한 T_0을 최소화하면서도 최적의 지연을 가지는 스케줄은 그렇게 많지 않다. 알고리즘의 런타임은 통신지연을 무시한 스케줄링 문제를 해결하는데 할애된 문제의 런타임에 비해 몇 배가 더 크기 때문에, 다섯 문제 모두에 대해 최적의 해를 찾았다는 결과는 본 논문에서 제안한 스케줄링 방법으로 찾아낸 해의 질이 최적임을 입증한 것이다.

Ⅳ. 결 론

본 논문에서는 멀티프로세서 아키텍처 상에 반복적인 데이터흐름 알고리즘을 스케줄링하는 방법을 제안하였다. 기본적인 하드웨어 모델을 기반으로 멀티프로세서 아키텍처라는 세부적인 특성을 가지도록 확장하여 용량이 제한된 통신 네트워크상에 전송할 데이터를 라우팅하는데 필요한 하드웨어 모델을 구현하고, 오버랩 되는 데이터흐름의 스케줄을 생성하는 스케줄링 방법 세 계층으로 분할하여 설계하였다.

상단 계층은 최적화를 담당하는 유전 알고리즘으로 되어 있다. 이것은 하드웨어 모델에 어떠한 제약도 가하지 않는다. 다음 계층이 사용하게 될 연산의 조합을 탐색할 뿐이다. 중간 계층에는 전역 스케줄링이 포함되어 있다. 전역 스케줄링은 단순한 일반적인 하드웨어 모델에 기반을 둔다. 하단 계층에는 블랙-박스 스케줄링이 위치한다. 오직 블랙-박스 스케줄링만이 세부적인 하드웨어 모델을 사용한다. 지원해야 하는 하드웨어 모델의 세부사항이 달라지는 경우, 이 모델을 지원하려면 블랙-박스 스케줄링만 적절하게 교체하면 된다는 장점이 있다. 두 가지 스케줄링은 중간 스케줄에 사이클을 삽입할 수 있다. 이렇게 함으로써 스케줄링 방법이 어딘 가에서 더 진행을 못하고 멈춰 있지 않도록 하며, 가용한 자원이 없거나 또는 통신 지연의 최소값을 충족시키지 못할 때, 이 문제를 해결하기 위해 사이클 삽입이 가능하다.

본 논문의 효용성을 입증하기 위하여 통신 지연과 제한된 용량을 가지는 통신 네트워크를 포함한 다섯 가지 하드웨어 모델에 대하여 이 스케줄링 방법을 벤치마킹한 결과, 생성된 스케줄의 질이 훌륭하다는 것을 알 수 있었다. 또한 고려 대상이 된 모든 스케줄링 문제에 있어서, 본 논문에서 제안한 스케줄링 방법으로 찾은 최상의 스케줄이 최적의 것이 아니라는 것을 나타내는 어떠한 사인도 존재하지 않았다. 따라서 이 스케줄링 방법은 허용 가능한 런타임 내에 양질의 스케줄을 생성할 수 있었음을 입증하였다.

References

[1] A. Rao and S. Pande. "Storage assignment optimizations to generate compact and efficient code on embedded DSPs", Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation, pp. 128-138, 1999.

[2] A. Darte, R. Schreiber, and G. Villard. "Lattice-based memory allocation". IEEE Transactions on Computers, Vol.54(10), pp. 1242-1257, October 2005.

[3] B. So, M. W. Hall, and H. E. Ziegler. "Custom data layout for memory parallelism", Proceedings of the International Symposium on Code Generation and Optimization, pp. 291-302, 2004.

[4] S. Leventhal, L. Yuan, N. K. Bambha, S. S. Bhattacharyya and G. Qu, "DSP address optimization using evolutionary algorithms", Proceedings of the 2005 workshop on Software

and compilers for embedded systems, pp. 91-98, 2005.

[5] J.-Y. Lee and I.-C. Park. "Address code generation for DSP instruction-set architectures", ACM Transactions on Design Automation of Electronic Systems, Vol.8(3), pp. 384-395, 2003.

[6] G. Chen and M. Kandemir. "Optimizing Address Code Generation for Array-Intensive DSP Applications", Proceedings of the international symposium on Code generation and optimization, pp. 141-152, 2005.

[7] GECO, "Genetic Evolution through Combination of Objects", http://common-lisp.net/project/geco/, October 2011.

[8] I.-K. Park, "A Study on the Prediction of the Nonlinear Chaotic Time Series Using Genetic Algorithmbased Fuzzy Neural Network", The Journal of The Institute of Internet, Broadcasting and Communication(JIIBC), Vol.11(4), pp.91-97, 2011.

[9] Ch. Dai and W. Weirong "Seeker Optimization Algorithm for Digital IIR Filter Design", IEEE Transactions on industrial electronics, Vol.57(5), 2010.

[10] Meyer-Baese, Uwe. "Infinite Impulse Response (IIR) Digital Filters." Digital Signal Processing with Field Programmable Gate Arrays. Springer Berlin Heidelberg, pp.225-304, 2014.

[11] Huang, Chaogeng, Yang Choon Lim, and Gang Li. "A generalized lattice filter for finite wordlength implementation with reduced number of multipliers." Signal Processing, IEEE Transactions on, Vol.62(8), pp.2080-2089, 2014.

[12] Bodong Li, Xieping Gao, "A method for initializing free parameters in lattice structure of linear phase perfect reconstruction filter bank", Signal Processing, Vol.98, pp.243 - 251, May 2014.

[13] Lars Wanhammar, Ya Jun Yu, "Digital Filter Structures and Their Implementation", Academic Press Library in Signal Processing, Vol.1, pp.245 - 338, 2014.

저자 소개

장 정 욱(준회원)

- 2005년 : 세명대학교 컴퓨터학과 이학사
- 2007년 : 세명대학교 일반대학원 이학석사 (전산정보학 전공)
- 2007년 ~ 현재 : 세명대학교 일반대학원 박사과정 (전산정보학 전공)

<주관심분야 : CAD, SoC, ASIC, Embedded System, RTOS, USN>

인 치 호(정회원)

- 1985년 : 한양대학교 공과대학 전자공학과 공학사
- 1987년 : 한양대학교 대학원 공학석사 (CAD 전공)
- 1996년 : 한양대학교 대학원 공학박사 (CAD 전공)
- 1992년 ~ 현재 : 세명대학교 컴퓨터학과 교수

<주관심분야 : SOC CAD, ASIC 설계, CAD 알고리즘, SOC 설계, RTOS 및 내장형 시스템>