논문 2009-46SD-6-9

리스트 스케줄링을 통한 Coarse-Grained 재구성 구조의 맵핑 알고리즘 개발

(A Resource-Aware Mapping Algorithm for Coarse-Grained Reconfigurable Architecture Using List Scheduling)

김 현 진*, 홍 혜 정*, 김 홍 식**, 강 성 호***

(HyunJin Kim, Hyejeong Hong, Hong-Sik Kim, and Sungho Kang)

요 약

재구성 구조를 위한 자동화된 툴의 개발에 있어서 명령들을 재구성 구조에 맵핑하기 위한 알고리즘의 개발은 가장 중요한 부분 중의 하나이다. 본 논문에서는 리소스가 한정된 Coarse-Grained 재구성 구조에 명령들을 맵핑하기 위한 알고리즘을 개발하고 이를 위한 휴리스틱을 제시하였다. 제안된 알고리즘에서는 하드웨어 리소스 사용에 대한 명령 할당과 라우팅 경로 할당을 사이클 기반의 타이밍 모델을 통해서 동시에 고려하였다. 제안된 알고리즘은 통신에 사용되는 리소스의 사용 및 전역 메모리 접근을 리스트 스케줄링을 기반으로 최소화한다. 리스트 스케줄링에서 맵핑되어야 할 명령들은 대상 어플리케이션의 데이터 플로우의 일반적인 특성들로 우선순위가 결정되게 된다. 제안된 맵핑 알고리즘의 대한 평가를 통해서 볼 때 전역 메모리 자원의 소모 및 수행 시간면에서 상당한 성능향상을 얻을 수 있었다.

Abstract

For the success of the reconfigurable computing, the algorithm for mapping operations onto coarse-grained reconfigurable architecture is very important. This paper proposes a resource-aware mapping system for the coarse-grained reconfigurable architecture and its own underlying heuristic algorithm. The operation assignment and the routing path allocation are simultaneously performed with a cycle-accurate time-exclusive resource model. The proposed algorithm minimizes the communication resource usage and the global memory access with the list scheduling heuristic. The operation to be mapped are prioritized with general properties of data flow. The evaluations of the proposed algorithm show that the performance is significantly enhanced in several benchmark applications.

Keywords : Reconfigurable architecture, Instruction mapping, List scheduling

Ⅰ. 서 론

Coarse-Grained 재구성 하드웨어 (reconfigurable hardware)는 풍부한 하드웨어 자원과 내재된 병렬성 때문에 많은 계산량이 요구되는 연산에서 코프로세서 (coprocessor)로써 사용될 수 있다. 많은 연구에도 불구하고 자원 할당의 부분에서 coarse-grained 재구성 구조에 대한 맵핑 알고리즘의 개발은 부족한 점이 많다[1].

메쉬 (mesh) 또는 토러스 (torus) 배열과 같은 정형화된 coarse-grained 재구성 하드웨어는 재구성 구조 (reconfigurable architecture)의 골격으로 많이 사용되고 있다[1~4]. 이는 풍부한 통신 자원과 이웃된 처리소자 간의 링크를 제공하기 때문이다. 또한 정형성과 추상화된 정보를 바탕으로 상위 수준의 명령의 할당을 수행할 수 있다. 그러나 일반적으로 맵핑의 경우 NP-hard한 문제중의 하나이기 때문에 확정된 (exact) 해를 찾는 것은 매우 작은 문제에 국한된다[4]. 그러므로 몇몇 기존

* 학생회원, ** 정회원, *** 평생회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
접수일자: 2008년11월12일, 수정완료일: 2009년5월12일

연구는 각각의 설계 공간에 대한 휴리스틱 (heuristic)을 제안하려고 하였다[2~5].

명령 맵핑의 경우 거대한 설계 공간으로 인해서 다루기 힘든 문제로 알려져 있다[4]. 또한 자원의 제약사항은 재구성 구조의 자원의 유연성과 풍부한 특성으로 인해서 명령 맵핑의 문제를 더욱 복잡하게 만든다. 이 경우 리스트 스케줄링(list scheduling)이 설계 공간을 탐색하기 위한 휴리스틱으로 많이 사용되어 왔다[3~5]. 리스트 스케줄링에서는 주어진 몇몇 법칙에 의해서 모든 명령들에게 우선순위를 할당하며 그 우선순위가 큰 명령들부터 차례대로 스케줄링을 실시하는 것이다. 이를 통해서 선형적인 시간에 명령들을 스케줄링을 할 수 있다.

본 연구의 목적은 어플리케이션의 수행시간을 최소화하기 위한 하드웨어 리소스를 고려한 맵핑 알고리즘을 제안하는 것이다. 이는 리스트 스케줄링에 기반을 두고 있다. 맵핑시에는 각 명령을 처리 소자에 할당하고 데이터를 주고받기 위한 사이클 기반의 시간 모델하의 통신 링크의 할당이 동시에 고려된다. 또한 최적화에 가깝도록 통신 자원의 사용을 최소화하고 전역 메모리의 접근 또한 처리 소자의 구성에 따른 시간대별의 분리 및 데이터 전달에 의해서 최소화 한다. 제안된 알고리즘의 평가를 통해서 볼 때 전역 메모리 자원의 소모 및 수행 시간면에서 상당한 성능향상을 얻었다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 알고리즘의 설명에 필요한 정의 및 모델을 제공할 것이다. Ⅲ장에서는 제안된 알고리즘의 세부 사항을 설명한다. Ⅳ장에서는 제안된 알고리즘에 대한 평가를 제시한다.

Ⅱ. 정의 및 하드웨어 모델

1. 정의

가. 어플리케이션 그래프

어플리케이션 그래프 의 경우 directed acyclic graph (DAG)들로 표현된다. 이 때 DAG의 꼭지점 $v_i, v_j \in V$는 각각의 명령을 나타내고 각 모서리 $e_{ij} \in E$는 소스 명령에 해당되는 와 도착지 명령에 해당되는 사이간의 데이터 통신을 의미한다. 이때 i와 j는 각 명령의 인덱스이다. 각 명령끼리 주고받는 연산수(operand)의 단위는 하나의 워드 크기로 고정되어 있다.

나. 어플리케이션 그래프

플랫폼 그래프 는 단방향 그래프로써 각각의 꼭지점 $n_\alpha, n_\beta \in N$ 는 각각의 처리소자를 나타낸다. α와 β는 처리소자의 인덱스를 의미한다. 각 모서리 $i_{\alpha\beta} \in I$는 처리소자 α와 β에 연결된 라우터간의 상호연결을 표현한다. 이 때 처리소자 α와 처리소자 β와 연결된 라우터들 사이에는 다른 라우터가 존재하지 않는다. 통신 링크 $c_{ij} \in C$의 경우 여러 개의 상호연결로 이루어 질 수 있다. 이때 i와 j는 데이터 통신을 이루는 두 명령의 인덱스이다. 즉 통신 링크는 연결된 상호연결의 집합이다.

다. 명령 맵핑

명령 맵핑은 각 명령을 한 처리 소자에 맵핑하는 것으로 한 통신링크가 여러개의 라우터간의 상호연결에 맵핑되는 것을 의미한다. 명령 맵핑 함수의 결과물은 처리소자 및 통신 자원에 대한 시간대별의 사용결과를 의미하게 된다.

2. 하드웨어 모델

실제 환경 및 최근에 연구중인 coarse-grained 재구성 구조들을 고려하여 다음과 같은 공통사항들을 일반적으로 포함하고 있다고 가정하도록 한다.

전역 메모리 변수의 접근은 모든 처리소자에서 직접적인 상호연결로 가능하다. 이는 병렬화 작업을 통해서 전역 메모리 변수의 처리를 가속화 할 때 적용될 수 있다. 그러나 전역 메모리 변수의 경우 큰 오버헤드를 포함한다.

처리소자들은 입력과 출력 버퍼를 가지고 있다. 출력은 처리소자의 출력 버퍼로부터 라우터로 전송된다. 상호연결은 경우 각 사이클마다 네트워크 인터페이스에서 구성이 바뀔 수 있다. 처리소자는 한 시간대에서는 구성이 변경되지 않는다.

라우터를 효율적으로 사용하기 위해서 라우터 사이의 상호연결은 동기화되고 파이프라이닝이 적용된다. 한 통신 링크에 대해서 파이프라이닝 단계의 개수는 상호연결의 숫자에 비례한다.

각 처리소자의 연산은 동종의 coarse-grained 재구성 구조에서 미리 정해진 처리 지연을 통해서 나타낼 수 있다.

그림 1은 두 개의 라우터와 두 개의 처리소자로 이루어진 통신 모델을 묘사하고 있다. 각각의 라우터는 상호연결을 구성하도록 설정된다. 라우터간의 상호연결은

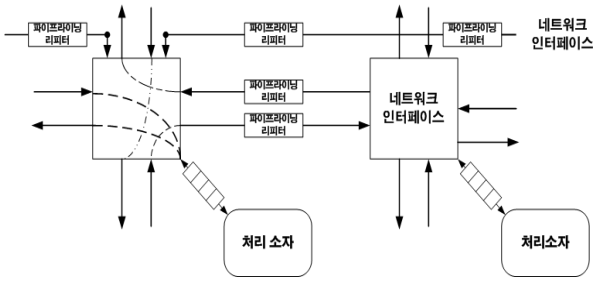


그림 1. 라우터 및 통신모델
Fig. 1. Routers and communication model.

리피터를 포함하고 있고 이를 통해서 통신을 파이프라이닝하고 클럭 주파수를 더 빠르게 가져갈 수 있다. 라우터는 입력 버퍼를 가지고 있지 않다. 이는 입력 버퍼에 따른 제어 신호를 삭제하여 단순화 하는 장점이 있다. 하지만 제어 신호를 미리 계산하여 그에 따른 데이터 플로우를 설정하도록 라우터의 구성을 계속 변경해야한다. 이를 위해서 제안된 알고리즘이 통신 링크를 위한 라우팅 경로의 설정을 미리 결정한다. 각 처리소자는 입력과 출력 데이터를 네트워크 인터페이스로부터 버퍼링한다. 처리소자간의 통신 경로는 소스에 해당하는 처리소자의 출력 버퍼로부터 시작된다. 적절한 데이터 출력이 처리되어야 할 시간 또는 사이클과 해당 라우터의 구성은 제안된 알고리즘에 의해서 생성된다. 통신 링크의 데이터는 목적지 처리 소자의 입력 버퍼에 값이 저장되면서 끝이 난다. 만약 어플리케이션 그래프의 꼭지점의 수가 플랫폼 그래프의 꼭지점 수보다 크다면 처리 소자내의 메모리를 넘어서기 때문에 전역 메모리의 접근이 필요로 할 수 있으므로 전역 메모리 접근을 시간대 분리 (time division)를 통해서 최소화 한다.

그림 2에서 도트는 명령 A와 B의 전처리된 명령들과 전처리된 명령들로부터의 통신이 모두 맵핑되었다는 것을 의미한다. 분리된 시간대 n에서의 명령의 수행이 끝난후에 시간대 $n+1$에서의 명령 A와 B는 명령 C에 대해서는 전처리된 명령이 되게 된다. 시간대 $n+1$에서의 명령 A와 B의 처리소자들의 출력은 전역 메모리에 저장되지 않고 처리소자의 출력 버퍼에 유지된다. 만약

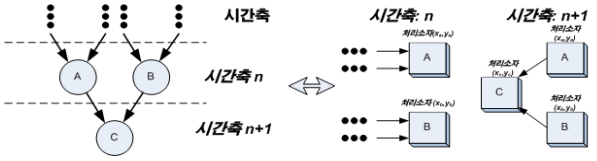


그림 2. 대상 DAG 및 시간대 분리 기법의 적용 예
Fig. 2. Target DAG and its time division method.

시간대 분리가 수행되지 않는다면 데이터를 시간대 $n+1$에서 전역 메모리에 저장한 후 나중에 이를 다시 불러와야 하므로 큰 오버헤드가 생기게 된다.

Ⅲ. 제안된 알고리즘

제안된 알고리즘은 명령들에 대해 우선순위를 부여한 우선순위에 따라 명령들을 맵핑 및 스케줄링한다.

가. 우선순위 부여 법칙

한 어플리케이션에서 추출할 수 있는 일반적인 정보를 통해서 명령들에 대한 우선순위 리스트를 생성한다. 제안된 알고리즘에서는 한 사이클의 지연을 하나의 단위로 삼는다. 어떤 명령 v의 수행 시간 $P(v)$는 그 단위로 표현된다. 하나의 어플리케이션을 구성하는 DAG들에서 사용가능한 특성을 나열하면 다음과 같다.

$D_{end}(v)$: DAG의 가상 싱크 (virtual sink) 꼭지점에서 명령 v의 수행 종료 시간까지의 최대 거리. 여기서 거리(distance)는 각 명령의 수행시간을 합한 것으로 표현된다. 가상 싱크의 경우 출력에 해당되는 모서리가 없는 꼭지점이 가상의 모서리를 통해 연결되는 가상의 꼭지점을 의미한다. D는 지연시간 (delay)를 의미한다.

임계 경로 (critical path): DAG의 가상 소스 (virtual source) 꼭지점에서부터 가상 싱크 꼭지점 전까지의 최대 수행시간. 여기서 경로의 길이는 거리와 마찬가지로 각 명령의 수행시간을 합한 것으로 표현된다. 가상 소스의 경우 입력에 해당되는 모서리가 없는 꼭지점들이 가상의 모서리를 통해 연결되는 가상의 꼭지점을 의미한다.

$Slack(v)$: 슬랙 (slack)은 임계 경로에 대해서 한 명령의 상대적인 여유로 다음 식으로 계산된다:

$$|D_{\text{임계 경로}} - D_{end}(v) - P(v)|.$$

$G(v)$: 명령 v의 그룹 인덱스를 의미하며 이는 어플리케이션을 구성하는 각 DAG의 인덱스를 의미하게 된다. 이 때 G는 그룹(group)을 나타내며 그 DAG에 속하게 된다.

$O(v)$: 명령 v에 대한 입력과 출력 모서리의 개수.

리스트 스케줄링에 앞에 열거된 특성들을 사용하기

```
우선권 부여(A(V, E))
{
    Sg= 독립적인_그룹들_생성(V), Sg∋ s;
    벡터(S) = Φ;
    소트(Sg, 최대(s, 임계 경로));
    Foreach 각 소트된 그룹 리스트 소자 sg ∋ vs
        Sd = 소트(s, Dend(v)), Sd∋ sd;
        Foreach 각 소트된 리스트 소자 sd ∋ vs
            Ss = 소트(sd, Slack(v)), Ss∋ ss;
            Foreach 각 소트된 리스트 소자 ss ∋ vs
                Se = 소트(ss, O(v));
            EndForeach
        EndForeach
    EndForeach
        벡터(s) 에 sg 추가;
    벡터(s) 반환;
}
```

그림 3. 우선순위 부여 알고리즘의 의사코드
Fig. 3. Pseudocode of algorithm for building priority list.

```
맵핑(A(V, E), P(N, I))
{
    벡터(s) = 우선권 부여(A(V, E));
    자원_테이블 = Φ;
    Foreach 각 벡터 (s)의 소자 s
    {
        리스트(v) = s;
        While (리스트(v) == Φ)
            While (time < 최대시간)
                Foreach 리스트(v) 에 속한 각 명령 v
                    If (가능한_경로_검색(v, 자원테이블, time))
                        자원 테이블에 가능한 경로 자원 추가;
                    EndIf
                EndForeach
                time++;
            EndWhile
        EndWhile
    EndForeach
    자원테이블 반환;
}
```

그림 4. 제안된 맵핑 알고리즘의 의사코드
Fig. 4. Pseudocode of proposed mapping algorithm.

위해서 각 특성들에 대한 적용에 대한 우선순위를 부여한다. 예를 들어서, $G(v)$는 $Slack(v)$ 보다 우선시되는데 다른 DAG에 속한 명령들 사이에서는 데이터 의존성이 존재하지 않기 때문이다. 위의 특성들에 따라 규칙을 만듦으로써 데이터 의존성을 유지시키면서 우선순위를 명령들에 부여하도록 한다. 그림 3은 명령에 우선순위를 부여하는 알고리즘이다.

우선순위 부여 알고리즘에서는 단순히 한 특성만을 사용만을 하지 않고 다양한 어플리케이션의 특성이 사용되었다. 첫번째로 독립적인 그룹들 생성을 위해서 각 꼭지점들이 속한 DAG들의 그룹을 구성한다. 이 DAG들은 각 DAG의 임계 경로에 따라 소트된다. 즉 임계 경로가 큰 DAG가 우선시 된다. 각 DAG내의 명령들은 우선 D_{end} 특성에 의해서 소트된다. 이 때 D_{end} 값이 같은 명령들에 대해서는 슬랙을 통해서 소트가 되며 슬랙의 값이 같은 명령들에 대해서는 각 명령에 대한 입력 및 출력 모서리의 개수를 기준으로 소트가 된다. 이와 같이 소트가 끝나면 DAG들에 대한 벡터를 구성하게 된다. 그렇기 때문에 각각의 DAG로 구분된 이차원적인 리스트가 얻어지게 된다. 이는 데이터 연관성이 없는 명령들을 차례로 맵핑하는 것을 방지한다.

나. 맵핑 알고리즘

맵핑 알고리즘은 명령들에게 부여된 우선순위 리스트를 가지고 명령을 처리소자에 할당하고 라우팅 경로의 통신자원을 사이클 기반의 자원 사용을 고려하여 설정한다. 그림 4와 같이 자원의 한계치에 따른 처리 소자와 통신 자원의 사용량이 고려된다.

각 그룹에 있는 명령들은 다른 그룹에 있는 명령들과는 데이터 의존적이지 않다. 그래서 한 그룹의 명령들은 우선순위에 따라 차례대로 스케줄링 된다. 여기서 변수 time을 통해서 맵핑이 되는 시간을 기록한다. 이 때 각 시간대에서 맵핑이 된 명령들은 우선순위 리스트에서 삭제된다. 만약 해당 시간대에서 가능한 라우팅 경로를 찾을 수 없다면 이 라우팅 경로와 연결된 목적지에 해당되는 명령은 현 시간대에서는 절대 맵핑될 수 없기 때문에 time 변수 값이 증가된 경우에 라우팅 경로 할당이 가능한지 알아본다. 이 경우 명령의 입력에 해당되는 라우팅 경로가 할당된 경우 명령을 처리소자에 맵핑할 수 있다.

하나의 명령을 맵핑하기 위해서 여러 가지 라우팅 경로중의 하나가 선택된다. 여러 가지 라우팅 경로 중 전체적인 시스템의 수행시간을 줄이기 위해서 비용 함수(cost function)를 고려하여 하나의 라우팅 경로를 선택한다. 이 선택을 위해 현 시간대에서 자원의 분배상 가능한 모든 경로의 리스트를 추출한다. 제안된 알고리즘의 비용 함수의 경우는 통신 자원의 사용을 나타내는데 목적이 있다. 여기에서 라우터간의 물리적인 거리인 도약 (hop)을 비용 함수를 도출하는데 사용할 것이다. 이웃한 라우터 A와 라우터 B간의 물리적인 거리는 1이며 만약 4x4의 라우터와 처리소자의 배열이 존재한다면 우측 상단의 라우터와 좌측 하단의 라우터간의 도약의 수는 6이다. 도약 1-2 통신 모델의 경우는 가로 혹은 세로 방향으로 라우터 상의 물리적인 거리가 1인 라우터들과 2인 라우터 간에 직접적인 상호연결이 있다는 것이다. 비용 함수의 의미를 고려하면 해당 명령을 맵핑

하는 처리소자의 위치에 대한 부분이 고려된다. 또한 해당 처리소자에 연결하기 위한 다양한 라우팅 경로들의 비용을 고려하여 최저의 비용을 나타내는 위치의 처리소자 및 비용이 최저인 라우팅 경로를 선택한다.

만약 명령의 개수가 해당 재구성 구조의 처리소자의 개수보다 많을 경우 시간대 분리 기법이 적용되어야 한다. 전역변수에 값을 저장후 불러오는 부분의 오버헤드가 매우 커질 수 있으므로 최우선적으로 시간대 분리 기법의 적용시 전역변수의 접근을 줄여야 한다.

Ⅳ. 실험 결과

제안된 알고리즘을 구현하기 위해서 C++ 언어 및 부스트 라이브러리[6]를 사용하였다. 제안된 실험환경은 ExpressDFG 벤치마크 회로에 대해서 수행되었다[7]. ExpressDFG는 기본적인 멀티미디어상의 루프 및 멀티미디어 벤치마크인 Mediabench상의 루프 몸체중에서 추려진 벤치마크 회로이다[8].

이웃하고 있는 라우터끼리의 상호연결의 경우 한 사이클의 지연시간을 가진다고 가정하였다. 이는 리피터를 이용한 파이프라이닝과도 일치하는 점이다. 즉 n 개의 도약을 통해서 라우터를 통해서 데이터가 전달되며 이는 $n+1$개의 파이프라이닝 단계를 가지는 것으로 요약될 수 있다. 정형화된 메쉬 구조의 경우 상하좌우의 네 방향 까지 라우터가 상호연결을 다른 라우터와 연결할 수 있으나 대각선 (oblique) 모드를 지원하는 경우 주변의 여덟 방향 까지 이웃하는 라우터와 연결된다.

ExpressDFG에서 전임 (predecessor), 즉 한 명령의 입력과 데이터 의존성을 가지는 소스 명령은 기존에 제시된 지그재그 전환 법칙[3] (zigzag traversal rule)에 의해서 기본적으로 맵핑된다고 가정한다. 이 법칙은 리스트 스케줄링에 기반한 정적인 맵핑 법칙이다. 또한 각 명령의 경우 단일 사이클에서 처리 소자가 수행할 수 있다고 가정하였다. 그러나 통신의 경우 라우팅 경로가 확보되지 않은 경우 계속 지연이 생길 수 있으며 한 라우터와 라우터 사이의 상호연결에 따른 지연은 단일 사이클에서 가능하다고 가정한다. 그러므로 라우팅 경로가 확보된 경우 명령들 사이에서의 데이터 전달 시간은 해당 라우팅 경로의 도약 값과 일치한다고 가정한다.

제안된 알고리즘의 성능을 측정하기 위해서 몇 가지 비교 대상을 구현하였다. 우선은 이상적인 하드웨어 모델로써 VLIW (very long instruction word)를 들 수 있다. VLIW과 재구성 구조와 차별되는 점은 VLIW의 경우 완전한 크로스바 (crossbar)를 통해서 레지스터 파일 (register file)에 접근을 할 수 있다는 점이다. 여기서 통신 지연과 명령 지연은 한 사이클에 이루어진다고 가정하였다. 실제 VLIW 구현의 경우 크로스바와 통합된 레지스터 파일에 대한 접근으로 인해서 복잡한 파이프라인 구조 및 하드웨어가 크게 증가할 수 있다. 이상적인 VLIW에서 이러한 오버헤드는 무시되었다. 또한 처리소자의 개수에도 제한이 없다고 가정하였다. 다른 비교대상으로 그룹에 대한 추출을 하지 않은 우선순위 부여 방식을 추가하였다. DAG의 그룹에 따라 맵핑을 순차적으로 진행하는 제안된 2차원적인 방법과 차별화하기 위해 이는 1-D 알고리즘이라 명하였다. 이를 통해서 각 데이터 연관성이 없는 DAG를 나누는 것에 대한 효과를 알아 볼 수 있다. 2-D 알고리즘은 어플리케이션을 몇 개의 DAG로 나눈다는 의미로 제안된 알고리즘을 가리키게 된다. 마지막으로 비용함수가 고려되지 않은 알고리즘을 구현하였다. 이 경우 맵핑 시에 가능한 처리 소자 및 라우팅 경로의 경우 임의적으로 선택되게 된다. 기타 우선순위 부여 및 맵핑 방법등은 제안된 알고리즘과 동일하다. 이를 통해서 비용 함수의 유용성에 대해서 알아 볼 수 있을 것이다. 즉 비용 함수를 통해 전체 도약의 숫자를 계산함으로써 통신 자원의 사용을 줄이는 것이 얼마나 효과적인지를 알아본다.

그림 5의 경우 4x4의 정형화된 처리소자와 라우터 배열에서 각 라우터는 이웃한 라우터와 연결되어 있다라는 가정하에서의 수행 시간당 처리명령에 대해서 제시하였다. 이 경우 데이터 통신을 위해서는 이웃 라우터와의 상호연결로 이루어지기 때문에 이는 1-도약 통신 모델을 적용한 것이다. 또한 VLIW와 그룹리스트를 생성하지 않은 1-D, 및 비용 함수를 적용하지 않은 2-D 알고리즘등을 제안된 2-D 알고리즘과 비교하였다.

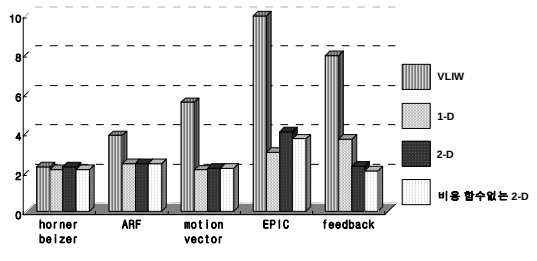


그림 5. 1-도약 통신 모델에서의 처리명령수의 비교
Fig. 5. Instruction level parallelisms with 1-hop communication model.

그림 5에서 보는 바와 같이 VLIW의 경우보다 약 50% 정도의 성능이 저하된 것을 볼 수 있었다. 이는 처리소자의 개수 및 라우팅 경로에 대한 부분도 이상적이므로 이를 감안하였을 때 명령 처리의 병렬화를 이룰 수 있다고 볼 수 있다. 또한 1-D 방법에 비해서는 feedback의 경우를 제외하고는 평균 12.9%, 또한 비용 함수를 채택하지 않은 2-D 방법에 비해서는 4.6% 정도의 성능 향상을 볼 수 있었다. feedback과 EPIC의 경우는 전역 변수에 값을 저장하고 이를 다시 불러와서 사용해야 하는 문제가 있었다. 본 시뮬레이션은 이 값을 포함하지 않았기 때문에 이에 대한 전역 변수에 관한 지연을 감안하였을 경우 위와 같은 많은 부분의 성능 감소가 줄어 들 수 있다는 점을 감안해야 한다. 그러므로 제안된 DAG를 여러 개의 그룹으로 병렬화 시키는 방법은 어느 정도의 성능 향상을 이끌 수 있었다고 말할 수 있으며 비용 함수의 경우 또한 통신 자원을 효율적으로 사용하도록 지원하였다고 할 수 있겠다.

그림 6에서는 제안된 알고리즘과 그룹 리스트를 생성하지 않은 1-D 방식과 비교하였다. 여기서 다양한 구성의 통신 모델을 사용하였다. 이 경우 그림 5와 마찬가지로 feedback 과 EPIC의 경우 수행시간 비율이 1을

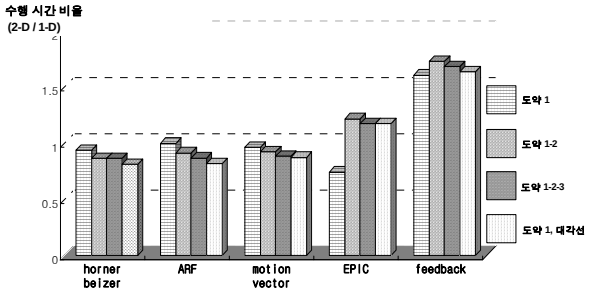


그림 6. 재구성 구조의 통신 모델에 따른 제안된 알고리즘과 1-D 방식과의 수행시간 비율 비교
Fig. 6. Execution time ratio of proposed algorithm to 1-D mapping.

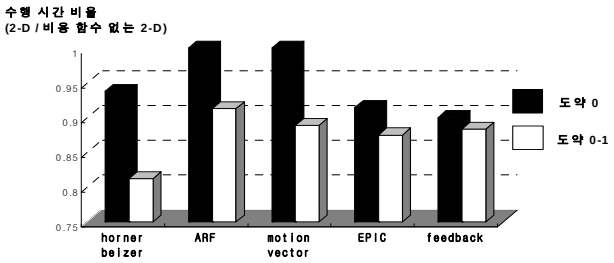


그림 7. 비용 함수의 채택에 따른 수행 시간 비율
Fig. 7. Execution time ratio depending on cost function.

넘었다. 이는 전역 변수에 값을 저장해야 할 필요성이 있기 때문이다. 전역 변수에 값을 저장하지 않는 다른 벤치마크에서는 각 처리소가가 여러 가지 도약을 통해서 다른 처리소자와 연결된 경우 더 성능이 향상되었다. 즉 라우터들과 직접적인 상호연결을 많이 채택함으로써 수행 시간의 비율이 더 줄어든 것을 볼 수 있었다. 이 경우 대각선의 직접적인 상호연결을 및 도약 1인 경우가 그러나 도약 1-2-3의 경우보다 수행시간을 더욱 줄일 수 있었다. 그러므로 도약의 개수가 많은 것보다는 대각선의 직접적인 상호연결이 더욱 효율적이다.

그림 7에서는 제안된 알고리즘과 비용 함수를 채택하지 않은 방식과 비교하였다. 여기서 두가지의 통신 모델을 사용하였다. 다른 경우에서는 다른 라우터들과 직접적인 상호연결을 많이 채택할 수 있는 모델에서 수행 시간의 비율이 평균적으로 약 15% 줄어든 것을 볼 수 있었다. 그러므로 비용 함수의 효과가 직접적인 상호연결을 많이 채택한 경우 더욱 크게 나타났다.

V. 결 론

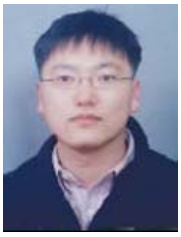
본 논문에서는 리소스가 한정된 coarse-grained 재구성 구조에 명령들을 맵핑하기 위한 알고리즘을 개발하였다. 제안된 알고리즘은 통신 자원 리소스의 사용 및 전역 메모리 접근을 리스트 스케줄링 및 시간대 분리 기법을 통해 최소화함으로써 성능을 향상시켰다. 또한 추출된 이차원적인 우선순위 리스트를 기반으로 순차적인 맵핑을 수행하게 된다. 이로써 선형적인 시간에 명령들을 순차적으로 맵핑할 수 있다. 제안된 맵핑 알고리즘의 대한 평가를 통해서 볼 때 전역 메모리 사용이 최적화 된 경우 상당한 성능 향상을 얻을 수 있었다.

참 고 문 헌

[1] R. Hartenstein, “A decade of reconfigurable computing: a visionary retrospective,” *in Proc. of Design and Test Conf.*, pp. 642-649, March 2001.
[2] B. Mei et al., “Architecture exploration for a reconfigurable architecture template,” *IEEE Design & Test of Computers*, Vol. 22, pp. 90-101, 2005.
[3] N. Bansal et al., “Network topology exploration of mesh-based coarse-grain reconfigurable architectures,” *in Proc. Design Automation and Test in Europe*, pp. 474-479, Feb. 2004.

[4] N. Bansal et al., "Interconnect-aware mapping of applications to coarse-grain reconfigurable architectures," *Lecture Notes in Computer Science*, Vol. 3203, pp.891-899, 2004.
[5] Y. Yi et al., "System-level scheduling on instruction cell based reconfigurable systems," *in Proc. Design, Automation and Test in Europe*, pp. 1-6, 2006.
[6] J.G. Siek, L.Q. Lee, and A. Lumsdaine, *The boost graph library user guide and reference manual*, Addison-Wesley Professional, 2001.
[7] G. Wang et al., "Design space exploration using time and resource duality with the ant colony optimization," *in Proc. of Design Automation Conf.*, pp. 24-28, July, 2006.
[8] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," *in Proc. Int. Symp. Microarchitecture*, pp. 330-335, 1997.

저 자 소 개



김 현 진(학생회원)
1997년 연세대학교 전기공학과 학사 졸업.
1999년 연세대학교 전기 및 컴퓨터 공학과 석사 졸업.
2005년 삼성전기 중앙연구소 선임연구원.
2009년 현재 연세대학교 전기전자공학과 박사과정.
<주관심분야: SoC 설계 및 응용, CAD>

김 홍 식(정회원)
1997년 연세대학교 전기공학과 학사 졸업.
1999년 연세대학교 전기 및 컴퓨터 공학과 석사 졸업.
2004년 연세대학교 전기전자 공학과 박사 졸업.
2004년~2005년 Virginia 공대 박사 후 연구원.
2006년 삼성전자 시스템 LSI 사업부 책임연구원.
2009년 현재 연세대학교 TMS 사업단 연구교수.
<주관심분야 : SoC 설계, 테스트>

홍 혜 정(학생회원)
2006년 연세대학교 전기전자 공학과 학사 졸업.
2009년 현재 연세대학교 전기전자 공학과 석박통합과정.
<주관심분야: SoC 설계 및 응용, 테스트>

강 성 호(평생회원)
1986년 서울대학교 제어계측 공학과 학사 졸업.
1988년 The University of Texas, Austin 전기 및 컴퓨터 공학과 석사 졸업.
1992년 The University of Texas, Austin 전기 및 컴퓨터 공학과 박사 졸업.
1992년 미국 Schlumberger Inc. 연구원.
1994년 Motorola Inc. 선임 연구원.
2009년 현재 연세대학교 전기전자공학과 교수.
<주관심분야 : SoC 설계, SoC 테스트>