병렬 내장형 소프트웨어 개발환경을 위한 데이터 플로우 블록 클러스터링

조용우 권성남 하순회

서울대학교 전기컴퓨터공학부

ywcho@iris.snu.ac.kr ksn@iris.snu.ac.kr sha@iris.snu.ac.kr

Dataflow Block Clustering for Parallel Embedded Software Development Environment

Yongwoo Cho Seongnam Kwon Soonhoi Ha

Seoul National University, EECS

요 약

갈수록 복잡해지는 내장형 시스템을 개발함에 있어서 소프트웨어 개발의 중요성은 날로 커지고 있다. 기존 연구에서 소프트웨어 개발 효율을 높이기 위해 소프트웨어의 재사용 가능성을 높이고 병렬성 명세를 용이하게 하고자 중간단계코드(CIC)를 정의하였다. 이 중간단계 코드는 각 태스크의 순수 알고리즘을 기술하는 C형태의 태스크 코드와 그 외의 정보를 포함하는 XML형태의 아키텍쳐 정보 파일로 구성된다. 이 CIC는 사용자가 직접 기술할 수 있고 각종 모델로부터 자동 생성할 수도 있다. 이 논문에서는 후자에 초점을 두고 데이터 플로우 모델에 사용된 블록들을 클러스터링하여 태스크 코드를 생성하는 기법을 제안하였다. 이것을 위해 블록 클러스터링 알고리즘은 주어진 클러스터의 크기로 블록이 묶일 때까지 블록의 수행시간 정보를 고려하여 함수 병렬성을 최대한 보존하며 블록들을 묶어나간다. H.263 코덱 예제를 이용한 실험을 통해 제안하는 방법이 다양한 클러스터의 크기 조건에 대해서 다양한 클러스터링 결과를 제공함을 보였다.

1. 서 론

짧은 개발 주기 하에서 복잡한 시스템을 효율적으로 개발하고자 다양한 시스템 수준 설계 방법이 제안되었다. 과거 설계 시간의 대부분을 차지하던 하드웨어 설계는 플랫폼 기반 설계 방법에 기반한 하드웨어 플랫폼의 재사용을 통해 개발 시간을 상당부분 단축할 수 있게 되었다. 반면에 소프트웨어는 다양한 사용자 요구를 만족시키기 위해 점차 복잡해지고 있으며, 전체 시스템의 설계 시간을 결정짓는 중요한 요소가 되고 있다.

과거 연구에서는 소프트웨어의 설계 효율성을 향상시키고자 병렬 소프트웨어 개발을 위한 중간 단계 코드(Common Intermediate Code : CIC)를 정의하고 이것을 이용한 소프트웨어 개발 환경을 구축하였다. CIC는 특별히 알고리즘 개발과 그 구현을 분리하고자 크게 두 종류의 파일로 구성하였다. 하나는 소프트웨어의 알고리즘을 기술하기 위한 태스크 코드 파일로, 하나의 태스크 코드 파일은 하나의 태스크를 나타낸다. 다른 하나는 아키텍쳐 정보 파일로 XML 형식으로 기술되며, 각종 하드웨어 정보 및 태스크의 연결관계를 기술한다.

이렇게 CIC를 이용하여 기술한 소프트웨어는 각 태스크 코드를 프로세서에 매핑하는 과정을 거치게 된다. 이 때 앞에서 기술한 태스크 코드 파일이 매핑의 단위가 된다. 매핑된 태스크는 CIC 변환기에 의해 타겟 의존적인 코드로 변환이 된다.

CIC의 태스크 코드 파일 내부는 기본적으로 C로 기술되며, 기존의 C프로그래밍에 익숙한 개발자에 의해 큰 진입장벽 없이 기술이 가능하다. 또한 태스크 코드는 몇몇 모델로부터 자동 생성하여 얻을 수도 있다. 이 논문에서는 후자와 같이 데이터 플로우 그래프로부터 태스크 코드를 자동 생성하는데 초점을 두고 있다.

데이터 플로우 모델로부터 태스크 코드를 생성하는 방법은 크게 두 가지 장점을 갖는다. 하나는 병렬성의 명세가 용이하다는 것이다. C언어를 비롯한 대부분의 소프트웨어 개발 언어는 순차적인 수행을 가정하고 있다. 반면에 주로 신호처리 알고리즘을 개발하는데 사용하는 데이터 플로우 모델은 블록간 데이터 의존성이 명백히 들어나기 때문에 알고리즘의 병렬성이 분명하게 드러나게 되어 병렬적인 알고리즘을 개발하는데 용이하다.

또 다른 장점은 태스크 코드의 크기를 변경하기가 용이하다는 점이다. 하나의 태스크 코드 파일은 매핑의 단위로서 작은 단위 태스크는 보다 세밀한 매핑이 가능해지지만, 이는 실행시간 증가 및 태스크간 버퍼의 증가 등, 소프트웨어의 전체 성능은 떨어질 수 있어, 효율적인 개발을 위해선 태스크 코드의 크기가 매우

중요하다. 만약 직접 태스크 코드를 작성했다면, 그 크기를 수정하기 위해 태스크 코드를 직접 수정해야 하나, 데이터 플로우 명세로부터 생성하였다면, 데이터 블록을 묶어 태스크로 만드는 크기를 조정하는 것을 통해 용이하게 변경이 가능하다.

데이터 플로우 모델로부터 효율적으로 코드를 생성하는 연구 자체는 이미 다양하게 수행되었다[1][2][3][4]. 따라서 생성된 소프트웨어의 효율성을 결정짓는 중요한 요소는 명세한 데이터 플로우 그래프의 블록을 어떻게 묶는가 하는 것이 된다. 데이터 플로우 그래프를 클러스터링 하는 연구 또한 다양하게 수행되었다[5]. 그러나 기존의 연구는 코드의 크기 및 수행시간 등 성능을 높이는 것을 목표로 하고 있기 때문에, 이 연구에서처럼 데이터 플로우 그래프를 병렬화 가능성을 고려하여 묶어 태스크 코드로 변환하는 것과는 차이를 보인다.

따라서 이 논문에서는 데이터 플로우 블록의 크기 및 버퍼의 크기를 고려하여 블록들을 클러스터링 하여 다양한 크기의 태스크 코드를 생성할 수 있는 기법을 제안한다. 제안하는 방법은 데이터 플로우 그래프에 사용된 블록을 주어진 클러스터링 크기가 될 때까지 묶어나간다. 이 때 각 블록의 수행시간 정보를 참조하여 함수 병렬성을 최대한 보존하는 방향으로 묶는다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 먼저 2장에서는 제안하는 설계 환경의 설계 흐름도를 살펴보고, 3장에서는 명세를 위한 데이터플로우 모델에 대해 알아보겠다. 4장에서는 제안하는 클러스터링 기법을 살펴본 후, 5장에서는 몇 가지 실험을 통해 제안하는 기법의 동작을 보이고, 6장에서는 논문을 결론지으며 추가 연구방향에 대해 설명하겠다.

2. 중간단계 코드를 이용한 설계 환경 및 문제 정의

중간단계 코드(CIC)는 제안하는 소프트웨어 개발 환경의 핵심으로서 크게 두 가지를 고려하여 설계 하였다.**오류! 참조 원본을 찾을 수 없습니다.** 하나는 소프트웨어의 재사용 가능성으로 알고리즘 명세와 알고리즘의 구현을 분리하였다. 이것을 위해 CIC를 각 태스크의 순수한 알고리즘만을 명세 하는 C 형태의 태스크 코드 파일과 하드웨어 정보, 각종 제약사항 정보, 그리고 태스크의 구조 및 매핑정보를 명세 하는 XML 형태의 아키텍쳐 정보파일의 두 종류 파일로 구성하였다.

다른 하나는 소프트웨어의 병렬성 명세로서, 함수 병렬성과 데이터 병렬성을 구분하여 명세하도록 하였다. 하나의 태스크 코드 파일은 하나의 태스크를 나타내며 매핑 단계에서 매핑의 단위가 된다. 따라서 복수의 태스크를 별도의 태스크에 매핑함으로써 함수 병렬성을 얻을 수 있으며, 태스크 코드 파일 자체는 함수 병렬성을 잠재적으로 표현하는 단위가 된다. 또한 하나의 태스크 내부에서 동일 코드가 여러 데이터에 대해 병렬적으로 수행할 수 있는 데이터 병렬성이 있음을 표시하기 위해 OpenMP 지시자를 이용하여 명세 하도록 하였다.

이렇게 기술된 CIC는 그림 1의 과정을 거쳐 타겟에서 수행 가능한 최적화된 코드로 변환된다. 먼저 각 태스크 코드를 각 프로세서에 매핑하는 과정을 거치게 된다. 이 때 매핑 결과는 아키텍쳐 정보 파일에 반영이 된다. 그 뒤에 CIC 변환기에 의해 타겟에서 수행가능한 코드로 변환이 되며, 매핑 결과를 고려하여 태스크의 스케줄링 코드 또한 생성이 된다. 이 때 각 태스크는 기본적으로 병렬적으로 수행되는 것을 가정하며, 각 태스크간 통신에 필요한 통신 채널 또한 자동 생성된다. 최종 생성된 코드를 수행해서 원하는 성능을 보이지 않는다면 그림 1의 피드백 간선이 보여주는 것과 같이 태스크 매핑을 다시 수행하거나 CIC를 변경하여 작성하여야 한다.

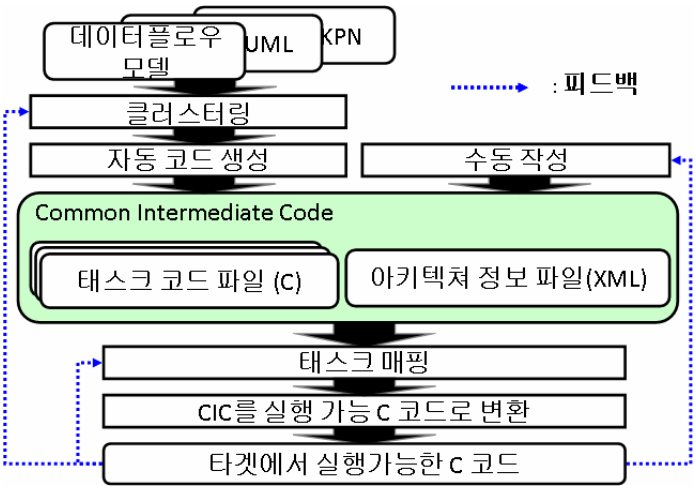

그림 1. 제안하는 소프트웨어 개발 흐름도

그림 1이 보여주는 것처럼 CIC의 태스크 코드는 소프트웨어 개발자에 의해서 직접 작성될 수도 있으며, 각종 모델로부터 자동 생성할 수도 있다. 자동 생성할 때에 있어서 여러 개의 블록을 묶어 하나의 태스크로 생성할 수도 있다. 이 때 작은 크기로 다수의 태스크를 생성한다면 매핑 과정에서 보다 유연하게 매핑을 수행할 수 있다. 그러나 각 태스크는 최종 코드에서 병렬적으로 수행되기 때문에 이 경우 수행 시간 부담이 증가하게 되며, 또한 태스크간 통신을 위해 필요한 채널이 증가하게 되어 시스템 메모리 측면에서도 부담이 된다. 반면에 태스크를 크게 묶어 소수의 태스크를 생성한다면 수행시간 및 메모리 측면에서는 이익을 얻을 수 있어도 매핑 시 선택의 폭이 좁아지게 된다.

따라서 효율적인 태스크 코드 파일을 얻기 위해선 모델에 사용한 블록을 태스크로 적절하게 묶어주는

클러스터링 단계가 필요하다. 이 때 적절한 태스크의 크기는 태스크 매핑 알고리즘의 요구사항에 따라 다를 수 있으며, 클러스터링 기법은 이러한 태스크 크기 요구사항에 맞게 블록을 묶을 수 있어야 한다.

또 하나 고려해야 할 점이 그래프의 함수 병렬성이다. 그림 2는 클러스터링의 변화에 따른 함수 병렬성의 변화를 보여준다. 그림 2(a)와 같은 그래프가 존재할 때 (b), (c), (d)와 같이 다양하게 클러스터링을 수행할 수 있다. (b)의 경우는 블록 A와 B를 묶은 클러스터와 C, D의 클러스터간에 의존성이 존재하지 않으므로 병렬적으로 수행이 가능하다. 반면 (c)의 경우는 클러스터의 개수가 2개로 줄어들었지만 두 클러스터간에 의존성이 존재하게 되어 함수 병렬성을 잃게 되었다. 이 경우 블록의 크기가 작다면 (d)와 같이 하나의 클러스터로 묶는 편이 성능이 좋을 수도 있다. 이와 같이 클러스터링 알고리즘은 블록의 크기뿐 아니라 블록을 묶음으로 잃어버리게 되는 함수 병렬성을 고려하여야 한다.

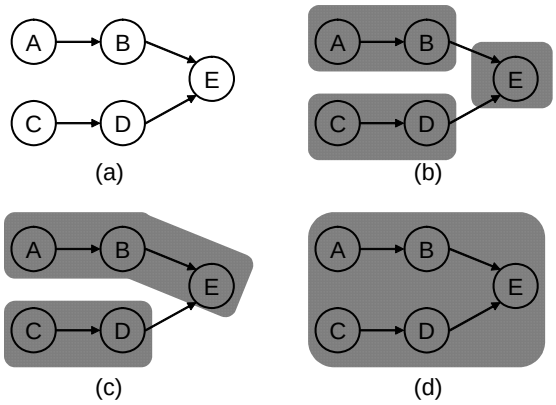

그림 2 클러스터링에 따른 함수 병렬성의 변화

3 명세를 위한 모델

제안하는 설계 플로우에서 CIC를 생성하기 위해서 태스크 모델과 데이터 플로우 모델의 두 가지 모델을 사용하고 있다. 태스크 모델에서 하나의 블록은 하나의 태스크를 나타내며 각 태스크는 하나의 쓰레드나 프로세스와 유사하게 병렬적으로 동작하게 된다. 각 태스크의 내부는 데이터 플로우 모델을 이용하여 명세된다. 각 태스크간 통신은 통신 블록을 통해 이루어지며 통신블록간에는 1:1로 통신 채널이 생성된다. 그림 3 (a)는 알고리즘 명세의 예를 보여주며 Task2 내부에는 통신을 위해 Rcv.라는 통신블록이 사용되고 있다.

데이터 플로우 모델에서 하나의 노드 혹은 블록은 입력 데이터 스트림을 변환하여 출력 스트림을 만들어내는 기능 블록을 의미한다. 각 단위 블록의 기능명세는 C 혹은 VHDL과 같은 상위수준 언어로 이루어진다. 블록들을 연결하는 간선(arc)은 원시 노드로부터 목적지 노드에 데이터 샘플의 스트림을 전달하는 채널을 나타낸다. 데이터 플로우 모델에서 블록이 수행되기 위해 필요한 입력 샘플의 개수와 실행 후 발생하는 출력 샘플의 개수를 각각 입력 혹은 출력 ***샘플 레이트***(input/output sample rate)라고 하며 샘플 레이트가 고정된 데이터 플로우를 특별히 SDF[4]라고 한다. 그림 3(a)의 Task2는 SDF로 명세된 태스크의 예를 보여주는데, 각 간선은 블록이 수행될 때 소모되고 생성되는 샘플의 개수가 표시되어 있다.

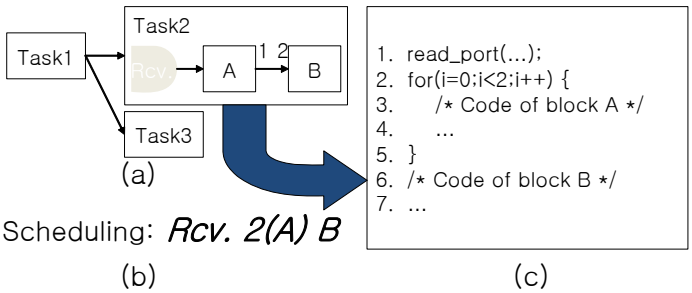

그림 3 (a) 알고리즘 명세의 예, (b) 태스크 2의 스케줄링 예, (c) 스케줄링을 고려한 태스크 2의 자동 생성된 코드

SDF 의 특징 때문에 블록의 수행 순서나 필요한 메모리 요구량 등을 컴파일 시간에 정적으로 결정할 수 있다. 특히 블록의 수행 순서를 ***스케줄링***이라고 하며, 코드 생성을 위해 필요하다. 어떤 스케줄에서 한 노드가 호출되는 회수를 그 노드의 ***반복횟수***(repetition count)라고 한다. 그림 3(b)는 Task2 명세의 가능한 여러가지 스케줄 가운데 하나를 보여주고 있으며, 여기서 2(A)의 의미는 블록 A 가 2 번 수행되는 것을 의미한다. 이 스케줄링 결과에 따른 코드는 그림 3(c)와 같이 생성된다.

SDF 그래프의 블록을 효율적으로 클러스터링 하기 위해선 그래프의 구조뿐 아니라 스케줄링 또한 중요하다. 예를 들어 스케줄링상 반복횟수에 의해 묶인 루프 구조의 블록들은 하나의 클러스터로 묶는 것이 효과적이다. 따라서 모델로 명세된 그래프의 구조뿐 아니라 스케줄링 정보 또한 XML 형태의 파일을 통해 블록 클러스터링 알고리즘으로 넘겨진다.

4. 블록 클러스터링

블록 클러스터링을 위해서는 블록의 크기를 결정할 수 있는 지표가 필요하며, 본 연구에서는 각 블록의 수행 시간 정보를 지표로 사용한다. 각 블록의 수행시간 정보는 제안하는 설계 환경에서 성능 데이터베이스에 존재하며 이 정보 또한 XML형태로 전달된다.

주어진 정보를 이용하여 그림 4와 같이 클러스터링을 수행한다. 먼저 블록의 스케줄링 정보를 이용하여 블록을 1차적으로 묶어 작은 클러스터들을 만들어 낸다. 그 뒤에 작은 클러스터들을 점차 병합해 가며 원하는 크기가 될 때까지 병합을 반복해 나간다. 클러스터링의

최종 결과는 XML형태로 코드 생성기에 전달된다. 각각의 단계에 대한 자세한 설명은 세부 섹션에 설명되어 있다.

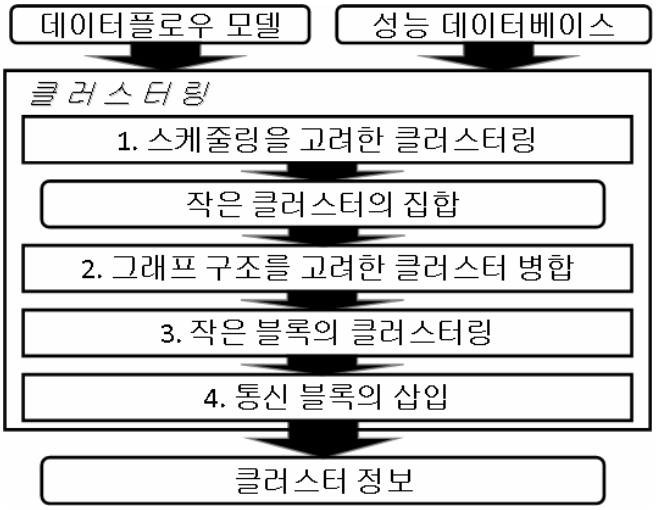

그림 4 블록 클러스터링 흐름도

4.1 스케줄링을 고려한 클러스터링 및 도달 가능성 분석

이 단계에서는 스케줄링 정보 가운데, 반복횟수로 묶여 루프 구조가 형성된 블록들을 묶어나간다. 이 때 루프 안에 있는 블록의 수행 시간 합이 원하는 클러스터의 크기보다 작다면 묶는데 성공하게 되며, 크다면 실패하게 된다. 실패할 경우에는 루프 안에 중첩된 루프가 존재하는지 체크하여 존재하면 묶는 과정을 반복해 간다. 이 과정을 통해 얻어지는 클러스터의 집합은 다음 단계인 클러스터 병합 단계에서 하나의 노드로 취급된다.

4.2 그래프 구조를 고려한 클러스터 병합

이 단계에서는 전 단계의 클러스터들을 하나의 노드로 취급하여 보다 크기가 큰 클러스터로 묶는 과정을 수행하며, 전 단계와 다르게 그래프의 구조를 고려하여 함수 병렬성을 해치지 않는 범위에서 클러스터링을 수행한다. 그림 2의 예제를 살펴보면, 입력과 출력이 한 개씩인 노드들을 병합할 때에는 함수 병렬성이 줄어들지 않으며, 입·출력이 2개 이상인 노드를 병합할 때 줄어듦을 알 수 있다. 이 단계에서는 그래프의 이러한 특성을 고려하여 입, 출력이 1개인 노드의 간선을 따라가며 먼저 클러스터의 후보를 작성한다. 이 때 후보의 크기가 원하는 클러스터링의 크기를 넘어가거나 입출력이 1개가 아닌 노드를 만나면 멈추게 된다.

그림 5는 특정 그래프에서 어떻게 클러스터링 후보를 만들 수 있는지를 보여준다. 이 그래프는 4개의 클러스터링 후보가 만들어진다. 이 때 블록 E는 4개의 모든 클러스터링 후보에 포함되어 있는 것을 볼 수 있으며, 블록 E가 어떤 그래프에 포함되느냐에 따라 전체 소프트웨어의 함수 병렬성이 달라질 수 있다. 현재 구현에서는 이 경우를 처리하기 위해 특별한 분석을 하고 있지는 않으며, 단지 클러스터링에 과정에서 먼저 선택된 클러스터에 포함되도록 하였다. 블록 E가 하나의 클러스터에 포함이 되면, 다른 쪽 클러스터 후보에서는 이 블록이 제외 된다. 그 뒤에 클러스터 결과를 노드로 취급하여 원하는 클러스터의 크기를 얻을 때까지 위의 과정을 반복하게 된다.

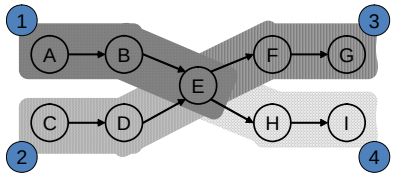

그림 5 클러스터링 후보 생성의 예

4.3. 작은 블록의 클러스터링 및 통신 블록의 삽입

클러스터 병합과정이 종료된 후에도 수행시간이 매우 작지만 병합되지 못한 블록이 존재할 수 있다. 이 단계에서는 최종적으로 작은 블록들을 클러스터에 병합하는 과정을 수행한다. 이 때 그 블록은 병합하려는 클러스터들 중 수행시간이 가장 짧은 곳에 병합된다.

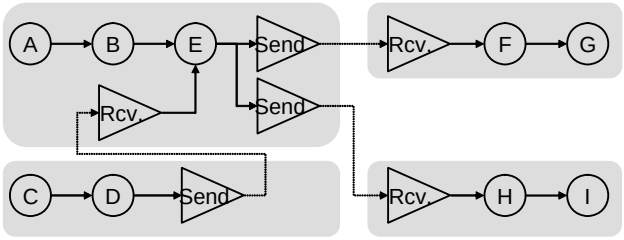

그림 6 최종 클러스터링 결과의 예

모든 블록이 클러스터링되면 하나의 클러스터는 하나의 태스크 코드로 생성되어야 하며, 클러스터의 경계점에서는 태스크간 통신이 발생하게 된다. 이것을 위해서 경계점에 Send/Receive역할을 수행하는 통신 블록을 적절히 삽입하여 최종 클러스터링을 완성하게 된다. 그림 6은 그림 5 예제가 4개의 클러스터로 묶인 결과의 예를 보여준다.

5. 실험

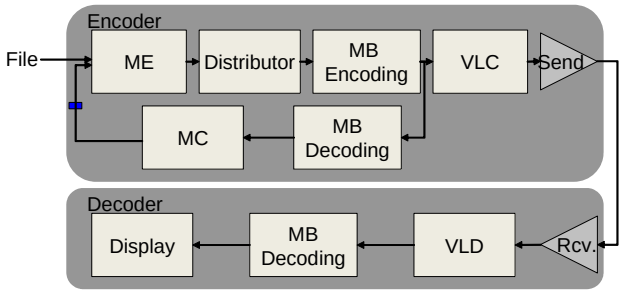

그림 7 H.263 코덱 예제의 간략한 명세

실험을 위해서 H.263 코덱 예제를 태스크 모델과 SDF모델을 이용하여 명세하였다. H.263 코덱 예제는 그림 7과 같이 H.263 인코더와 디코더의 2개 태스크로 구성되어 있으며 각각 내부는 SDF를 이용하여 명세하였다. H.263 인코더와 디코더는 각각 41개와 17개의 데이터 플로우 블록으로 구성되어 있다. 각 블록의 수행시간 정보로 RealView Development Suite 2.2[7]에서 제공해주는 시뮬레이터를 이용하여 Arm926ej-s 프로세서에 대해 측정한 값의 평균을 사용하였다.

표 1은 다양한 클러스터링 크기 조건에 따른 클러스터 결과를 보여준다. 클러스터링 크기 조건이 커짐에 따라 클러스터의 개수는 줄어들며 그에 따라 클러스터간 채널의 개수도 줄어듦을 볼 수 있다. 따라서 유연한 태스크 매핑과 생성된 코드의 효율성을 모두 고려하여 클러스터링의 크기를 결정하는 것이 매우 중요함을 알 수 있다.

표 1 클러스터 크기 조건의 변화에 따른 클러스터링 결과의 변화

	H.263 Encoder		H.263 Decoder	
클러스터 크기 (cycle)	클러스터 개수	통신채널 개수	클러스터 개수	통신채널 개수
1520250	30	73	8	21
2027000	29	68	8	21
2533750	23	71	8	21
3040500	21	66	6	19
3547300	21	60	6	19
4054050	21	60	6	19
4560800	21	60	2	15
5067550	21	59	1	0
10135100	17	48	1	0
50675400	1	0	1	0

6. 결론

데이터 플로우 명세로부터 태스크 코드를 생성하기 위해선 데이터 플로우 블록을 클러스터링하는 과정이 필요하다. 블록을 작은 크기로 클러스터링 하면 태스크 매핑을 보다 유연하게 수행할 수 있으나, 태스크 개수의 증가로 인한 생성된 코드의 수행시간이 증가하게 되며, 태스크간 채널이 증가하게 되어 시스템 메모리 요구량이 증가하게 된다. 따라서 다양한 클러스터링 크기 조건에 따라 다양한 클러스터링 결과를 제공해주는 클러스터링 도구가 필요하다.

이 연구에서는 데이터 플로우 블록을 묶어 각 블록의 수행시간 정보를 이용하여 주어진 클러스터링 크기 조건에 맞는 클러스터링을 수행하는 방법을 제안하였다. 또한 제안하는 방법은 함수 병렬성이 줄어들지 않는 방향으로 최대한 블록을 묶어나간다.

제안하는 방법의 효용성을 높이기 위해선 몇 가지 추가 연구가 필요하다. 현재는 클러스터를 병합할 때 블록이 여러 클러스터에 병합 가능하면 가장 처음 병합을 시도하는 클러스터에 병합시킨다. 그러나 보다 효율적인 클러스터링을 위해선 보다 많은 그래프 정보를 참조하여 병합시킬 클러스터를 결정할 필요가 있다. 또한 어떤 루프가 전부 하나의 클러스터로 묶일 수 있다면 몇 가지 제약조건 하에서 OpenMP 코드를 생성하여 데이터 병렬성을 갖는 태스크로 만들 수 있다. 이것에 대한 추가 연구가 필요하다.

7. 감사의 글

본 연구는 BK21 프로젝트, 교육과학기술부 도약연구지원사업(R17-2007-086-01001-0)에 의해 지원되었다. 또한 서울대학교 컴퓨터연구소와 IDEC은 본 연구에 필요한 기자재들을 지원해 주었다. 본 연구는 또한 한국전자통신연구원의 SoC 핵심설계인력양성사업에 의해 부분적으로 지원되었다.

8. 참고문헌

[1] Synopsys Inc., 700 E. Middlefield Rd., Mountain View, CA94043, USA, COSSAP User's Manual

[2] Signal Processing Designer(SPW) product site of CoWare, http://www.coware.com/products/signalprocessingSPW

[3] H. A. Andrade and S. Kovner, "Software synthesis from dataflow models for G and LabVIEWTM," in Proc. of Asilomar Conf. on Signals, Systems & Computers, vol. 2, pp. 1705-1709, Nov. 1998

[4] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," Intl. Journal of Computer Simulation, special issue on "Simulation Software Development," vol. 4, pp. 155-182, 1994

[5] A. Gerasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors," Intl. Journal of Parallel and Distributed Computing, vol. 16, pp.276-291, 1992

[6] Soonhoi Ha, "Model-based Framework of Embedded Software Design for MPSoC", ASP-DAC 2007, pp.330-335, Jan. 2007

[7] RealView Development Suite official homepage, http://www.arm.com/products/DevTools/RealViewDevSuite.html