실시간 제약조건을 갖는 다중태스크 다중코어 SoC의 하드웨어-소프트웨어 통합합성

(Hardware-Software Cosynthesis of Multitask Multicore SoC with Real-Time Constraints)

이 춘 승 † 하 순 회 ††

(Choonseung Lee) (Soonhoi Ha)

요 약 이 논문은 실시간 제약 조건을 갖는 다중태스크 응용을 여러 개의 코어를 갖는 SoC 위에서 동작시키고자 할 때, 시스템의 비용은 최소로 하면서 성능을 높일 수 있도록 프로세서 및 하드웨어 IP를 선정하고, 태스크를 매핑 하는 기법을 제안한다. 이와 같은 기법은 하드웨어-소프트웨어 통합합성 기법이라고 한다. 이전 연구에서 우리는 복잡한 통합합성 문제를 세 가지 하부 문제(프로세서 컴포넌트 선택문제, 태스크 매핑문제, 그리고 스케줄-가능성 검사문제)로 세분화 하고, 각 문제를 독립적으로 해결하는 기법을 제안하였다[1]. 하지만 많은 장점에도 불구하고 이전 연구에서는 한 태스크가 스케줄 될 때, 자신의 스케줄-길이를 최소로 줄이기 위해 시스템 전체 자원을 모두 점유하는 것을 가정하는 제약점이 있었다. 그러나 일반적으로 보다 향상된 성능을 얻기 위해서는, 서로 관련이 없는 태스크들은 서로 다른 프로세서에서 동시에 실행될 수 있어야 한다. 이 논문에서는 다중프로세서 환경에서 다양한 운영정책을 가지는 일반적인 시스템을 위하여 태스크 매핑회피 기법과 태스크 매핑전용 기법이라는 두 가지 매핑기법을 제시한다. 멀티미디어 실시간 응용 프로그램인 다채널 디지털 비디오 레코더(Digital Video Recorder)와 관련 논문에서 제공된 임의 생성 다중태스크 예제에 대해서 큰 성능 향상을 얻을 수 있었다.

키워드 : 하드웨어-소프트웨어 통합합성, 실시간, 다중코어, 다중태스크, 시스템-온-칩

Abstract This paper proposes a technique to select processors and hardware IPs and to map the tasks into the selected processing elements, aming to achieve high performance with minimal system cost when multitask applications with real-time constraints are run on a multicore SoC. Such technique is called to 'Hardware-Software Cosynthesis Technique'. A cosynthesis technique was already presented in our early work [1] where we divide the complex cosynthesis problem into three subproblems and conquer each subproblem separately: selection of appropriate processing components, mapping and scheduling of function blocks to the selected processing component, and schedulability analysis. Despite good features, our previous technique has a serious limitation that a task monopolizes the entire system resource to get the minimum schedule length. But in general we may obtain higher performance in multitask multicore system if independent multiple tasks are running concurrently on different processor cores. In this paper, we present two mapping techniques, task mapping avoidance technique(TMA) and task mapping pinning technique(TMP), which are applicable for general cases with diverse operating policies in a multicore environment. We could obtain significant performance improvement for a multimedia real-time application, multi-channel Digital Video Recorder system and for randomly generated multitask graphs obtained from the related works.

Key words : Hardware-Software Cosynthesis, Real-Time, Multicore, Multitask, System-on-a-Chip

· 본 논문은 국가 지정 연구실 프로그램(Grant No. M1-0104-00-0015), 두뇌한국 21, SystemIC 2010, IT-SoC, 정보통신부 IT선도 과제에 의해 지원 되었으며, 본 연구에 필요한 기자재들은 서울대학교 컴퓨터연구소와 IDEC에서 지원해 주었다.

† 학생회원 : 서울대학교 전기컴퓨터공학부
cslee@iris.snu.ac.kr
†† 정 회 원 : 서울대학교 전기컴퓨터공학부 교수
sha@iris.snu.ac.kr

논문접수 : 2006년 5월 24일
심사완료 : 2006년 8월 16일

1. 서 론

칩의 집적 기술의 향상과 더불어 응용 프로그램의 복잡도 및 실시간 제약사항으로 인해 점점 많은 프로세싱 컴포넌트를 하나의 시스템-온-칩(System-on-a-Chip, SoC) 내로 통합하는 것이 필요해지고 있다. 다중코어

SoC에서는 응용 프로그램을 설계할 때 단일태스크가 모든 자원을 독점하여 수행하기보다 전체 응용 프로그램의 실시간 제약-조건을 만족하면서 여러 태스크를 동시에 수행하는 방식이 일반적이다. 이 논문에서 관심을 갖고 있는 하드웨어-소프트웨어 통합합성 문제는 그러한 다중태스크 (Multitask)로 기술된 응용을 각 태스크의 실시간 제약-조건을 만족하면서 전체 시스템의 비용을 최소로 줄이기 위한 최적의 다중코어 SoC 구조를 찾는데 있다. 또한 태스크 사이에서의 하드웨어 자원 공유(Resource sharing)와 여러 응용 프로그램을 선별적으로 수행하는 다중모드(Multi-mode), 다중태스크 시스템을 고려하고자 한다. 이러한 하드웨어-소프트웨어 통합합성 문제(간단히, 통합합성 문제라고 한다)는 다항식 시간(Polynomial time)내에 결정적(Nondeterministic)으로 풀 수 없는 문제들 중 하나로 상용의 CAD 도구나 최적의 알고리즘이 현재까지 알려져 있지 않다. 따라서 본 논문에서는 주어진 큰 문제를 보다 간단한 세부문제로 구분하고, 각 문제를 독립적으로 해결하는 접근방식을 취했다.

우리는 이 논문에서 태스크는 비순환(Acyclic) 그래프로 표현하고, DCT(Discrete Cosine Transform), ME (Motion Estimation)와 같은 비교적 큰 단위의 기능블럭들을 노드(Node)라고 가정하며, 각 노드가 매핑의 기본대상이 되도록 하였다. 사용 가능한 프로세싱 컴포넌트들은 라이브러리 형태로 주어지며, 비용과 각 노드들을 수행할 때의 수행시간 정보를 담고 있다고 가정한다. 본 논문의 연구는 그림 1에 도시한 2-단계 설계공간탐색 환경의 일부로 구현되었으며, 회색으로 칠해진 부분에서 통합합성이 이루어진다.

통합합성의 입력정보는 시스템 동작에 대한 기능 및 알고리즘 명세와 노드 수행 정보가 담긴 데이타베이스 그리고 초기의 아키텍처 명세로 이뤄져 있다. 이로부터 적합한 프로세싱 컴포넌트를 선택하는 문제, 기술된 노드를 어떤 프로세서에서 실행할 것인지에 대한 매핑문제, 얻어진 결과가 주어진 시간 제약-조건을 만족하는지 여부(Schedulability, 스케줄-가능성)를 분석하는 문제를 해결하는 것이 통합합성의 목표이다.

프로세싱 컴포넌트의 선택과 태스크 내의 노드들에 대한 매핑이 결정되면, 각 프로세싱 컴포넌트에서 수행될 코드가 자동으로 합성된다. 매핑결과에 따라 소프트웨어에서 수행될 부분은 C/C++ 코드로 생성되고, 하드웨어에서 수행될 부분은 VHDL 코드로 합성된다. 생성된 코드는 Armulator와 같은 명령어 시뮬레이터와 ModelSim 과 같은 하드웨어 시뮬레이터에 의해서 통합 시뮬레이션 되며, 메모리 트레이스(Memory trace) 정보를 얻는다. 생성된 메모리 트레이스 정보와 각 기능블럭의 실행순서를 나타내는 스케줄 정보에 근거하여 통신구조 설계공간탐색(Communication Design Space Exploration) 루프를 수행한다. 통신구조의 설계공간탐색 기법은 [3]에 자세히 기술되어 있다.

하드웨어-소프트웨어 통합합성 단계에서 매핑 성능을 예측하기 위해서는 프로세싱 컴포넌트들 간의 통신비용(Communication overhead)을 고려하여야 한다. 통신비용은 동일한 전송 데이타라 할지라도 통신구조에 따라 서로 다른 비용을 갖기 때문에 반드시 통신구조가 변경되는 것에 대해 고려되어야 한다. 하지만 그림 1에서 제

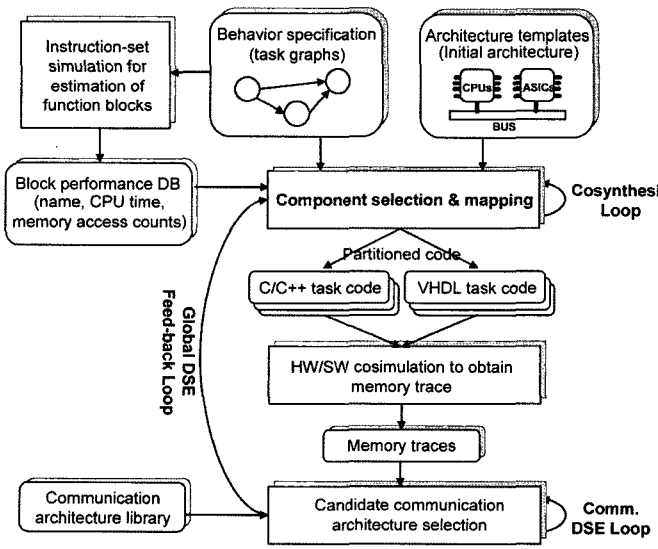


그림 1 2-단계 설계공간탐색 프레임웍

안된 것처럼 2-단계 설계공간탐색 프레임웍에서 노드의 프로세서 매핑단계는 통신구조 탐색 단계보다 이전이므로 정확한 통신비용을 고려할 수 없다. 따라서 통합합성 단계에서의 통신비용은 일단 데이타의 양과 한번 전송하는데 필요한 임의의 고정 값의 곱으로 표현한다. 이후에 통신구조가 결정되면 수정된 통신비용을 가지고 주어진 스케줄 결과가 타당한지를 다시 검증하고, 노드의 통신시간 정보를 갱신하기 위하여 전역설계공간탐색(Global DSE) 피드백 루프를 사용 하는데 그림 1에 도시되어 있다. 이렇게 통합합성 문제와 통신구조 최적화 문제를 나누어 품으로써 얻는 중요한 이득은 복잡한 문제를 보다 단순화시켜 쉽게 풀 수 있는 방법을 제시하고, 또한 각 설계 단계에서 제안된 서로 다른 다양한 구현 방법을 프레임웍 내로 쉽게 이식시킬 수 있는 설계 환경을 제공하는데 있다. 이 논문에서 우리는 2-단계의 설계과정 중 첫 번째로 통합합성 문제를 푸는데 초점을 두었다.

이전 연구에서 우리는 통합합성 문제를 프로세싱 컴포넌트 선택문제, 태스크 매핑문제, 스케줄-가능성 검사 문제로 세분화하고 각 문제를 독립적으로 해결하는 기법을 제안하였다[1,2]. 그리고 각 태스크를 실행할 때 마다 자신의 스케줄-길이를 최대로 줄이기 위해 전체 시스템 자원을 독점하는 것을 가정하였다. 즉, 다중태스크를 실행하기 위해 단일프로세서에서 동작되는 것처럼 모든 태스크들에게 시분할(Time Division Multiplexing) 스케줄링을 적용한 것이다. 이와 같은 실행 구조는 단일프로세서에서 단일운영체제를 수행하는 시스템에 적합한 방법인데 반해, 다중프로세서에서 다중태스크를 수행시키기 위한 최선의 방법은 아니다. 왜냐하면 다중태스크를 허용하는 시스템이 높은 성능을 내기 위해서는 동시에 수행 가능한 태스크들을 서로 다른 프로세싱 컴포넌트에서 동시에 수행시켜야 됨이 자명하기 때문이다. 이 논문에서는 이러한 단점을 해결하기 위한 두 가지 태스크 매핑기법과 새로운 스케줄-가능성 검사 기법을 제안한다.

이 논문의 구성은 다음과 같다. 다음 장에서는 예제를 가지고 이전 통합합성 기법의 문제점을 제시한다. 그리고 3장에서 몇몇 관련 연구들이 제안하는 기법을 설명한다. 4장에서는 제안된 기법이 어떻게 동작이 되는지에 대한 설명과 수식들에 관해 기술하고, 5장에서 실험결과들을 정리한다. 마지막으로 6장에서 남은 과제들을 검토하고 결론을 맺는다.

2. 기존 통합합성 기법과 동기부여 예제

이전 연구에서 개발된 통합합성 기법은 그림 2에서 보인 것과 같이 독립적인 3개의 세부단계로 나뉘어 문제를 해결하고, 각 단계는 서로 반복적인 루프를 돌면서 수행된다. 통합합성 문제에서의 입력은 태스크 그래프, 프로세싱 컴포넌트 라이브러리, 그리고 각 프로세서에서의 노드 수행시간 정보를 갖고 있는 Module-PE profile 테이블이다. 그림 3(a)는 두 태스크 그래프 $Task_1$, $Task_2$로 이뤄진 간단한 다중태스크 예제를 보여주고 있다. 노드 사이에서의 간선은 데이타-의존성(Data dependency)을 표현하고 간선 위에 있는 수는 통신비용을 나타낸다. 두 태스크의 주기(Period)는 각각 40과 60으로 정했고, 데드라인(Deadline)은 주기와 동일하다고 가정하였다.

제안하는 기법은 PE-할당-제어기(Processing Ele-

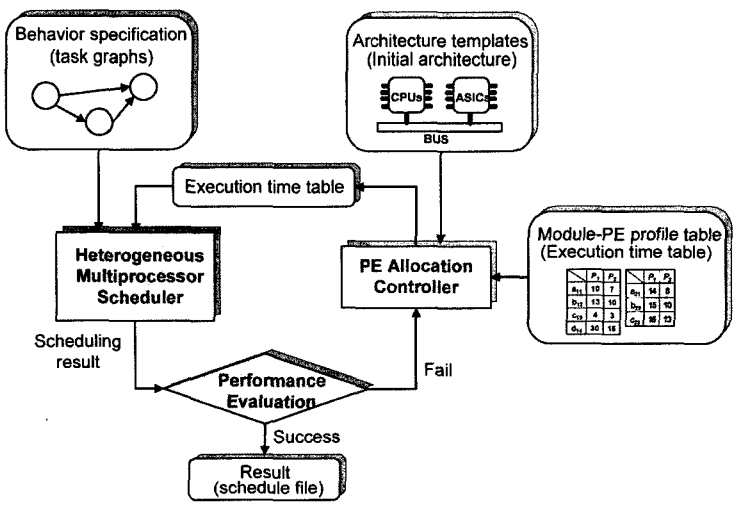
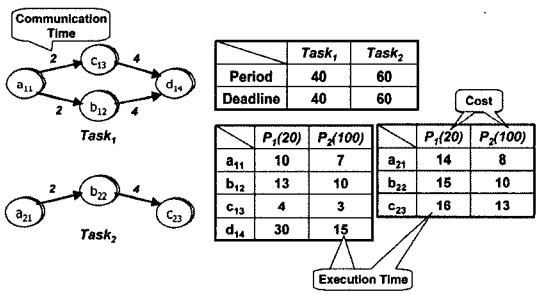


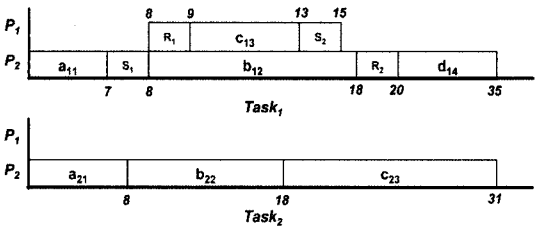
그림 2 기존의 통합합성을 위한 설계루프

ment Allocation Controller)에 의해 시작된다. PE-할당-제어기는 사용가능한 프로세싱 컴포넌트들로부터 가장 싼 프로세싱 컴포넌트 집합을 선택하고, 선택된 컴포넌트 집합에 대해 Module-PE profile 테이블을 재구성한다. 설계의 목적이 시스템 전체의 비용을 줄이는 것이기 때문에 가장 싼 프로세서를 먼저 선택하고, 시스템 설계상 프로세서 선택에 제약이 따를 경우, 예를 들면, '몇 Mhz 이하의 성능은 사용될 수 없다.'와 같은 제약-조건이 따를 경우 PE-할당-제어기에서 선택되지 않도록 처리한다. 그림 3(a)는 사용가능한 프로세서 P_1, P_2가 선택되어졌을 때 재구성된 Module-PE profile 테이블을 보여주고 있다. 테이블내의 각 항목은 노드가 선택된 프로세싱 컴포넌트 위에서 수행될 때 소요되는 실행 시간을 나타낸다.

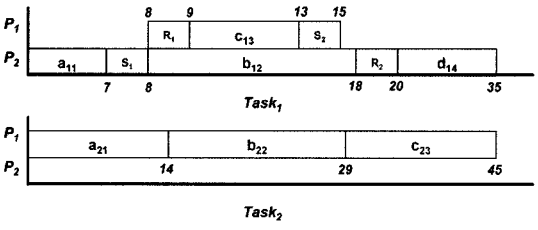
프로세싱 컴포넌트가 선택된 후 이형 다중프로세서 스케줄러(Heterogeneous Multiprocessor Scheduler, HMS)



(a) 두 태스크와 통신비용을 가지는 예제



(b) 시간 제약-조건을 만족하지 못하는 스케줄 결과



(c) 시간 제약-조건을 만족하는 스케줄 결과

그림 3 동기부여 예제에 대한 HMS 스케줄 결과
(*S는 통신을 위한 송신(Send) 노드를 나타내고, R은 수신(Receive) 노드를 나타낸다)

는 각 태스크의 스케줄-길이가 최소가 되도록 노드들을 프로세서로 매핑 하는 일을 수행한다. 이는 HMS 자체의 목적 함수(Objective function)가 태스크 스케줄의 길이를 가급적 최소한으로 줄이는 방향으로 설계되었기 때문이다. HMS에서 스케줄을 하고 나면, 그림 3(b)에서 보인 바와 같이 각 태스크 별로 최소의 길이를 가진 스케줄을 얻을 수 있다. 주의 깊게 봐야 할 점은, HMS는 모든 선택된 프로세싱 컴포넌트들을 충분히 사용하지 못하고 있다는 점이다. 프로세서 P_1과 P_2는 그 성능이 서로 다르기 때문에 각 태스크 입장에서 보면 자신의 스케줄-길이를 최소로 줄이기 위해 되도록이면 빠른 프로세서를 선택할 수밖에 없고, 그렇게 되면 상대적으로 느린 프로세서들은 다른 태스크들을 수행할 수 있음에도 불구하고, 휴지상태(Idle state)가 많아 시스템 내의 프로세서 이용률의 불균형이 발생한다. 태스크-의존성이 없는 다중태스크 예제에서는 태스크 실행의 병렬도와 프로세서 간의 성능차를 고려하여 느린 프로세서라고 할지라도 동시에 수행하면 성능의 이득이 크다는 것은 분명하다.

프로세서의 선택과 매핑을 마쳤으면 그 결과가 주어진 실시간 제약-조건을 만족했는지를 검사하는 성능 평가 단계를 거친다. 이 스케줄-가능성 검사 결과, 제약-조건을 만족하면 통합합성 루프를 빠져나와 결과를 출력하도록 한다. 만일 스케줄-가능성 검사 결과 주어진 시간 제약-조건을 만족하지 않는다면, PE-할당-제어기에게 통보하여 현재 가정한 시스템 비용보다 더 큰 비용을 가진 다른 프로세싱 컴포넌트들을 선택하도록 지시한다. 그런 다음 HMS는 새롭게 구성된 프로세싱 컴포넌트들을 가지고 매핑 과정을 태스크 별로 처음부터 다시 수행하게 된다.

그림 3(b)의 두 태스크내의 노드들은 자신의 스케줄-길이를 최대한 줄이기 위해 가급적 빠른 프로세서인 P_2에 할당 되었다. 하지만 그림 3(b)는 프로세서 P_2의 전체 이용률의 한계 기준인 1(35/40 + 31/60 = 1.4)을 넘어서기 때문에 실제로 스케줄 될 수 없는 잘못된 매핑 결과이다. 그러면 HMS는 PE-할당-제어기에게 현재 시스템 가격인 120 보다 더 큰 가격을 요구할 것이고, 더 이상 새로운 프로세서를 할당할 수 없는 할당기는 '매핑 불가'를 사용자에게 통보할 것이다. 하지만 만일 $Task_2$가 그림 3(c)와 같이 휴지상태가 큰 P_1으로 매핑 된다면 스케줄 가능한 결과를 얻을 수 있다. 이렇듯, HMS 알고리즘 자체의 문제로 인해 실제로 매핑을 효율적으로 하지 못하는 단점이 발생한다. 이 논문에서는 $Task_2$를 P_1으로 보낼 수 있도록 '분할회피' 기법을 제안하여 이러한 단점을 극복하였다.

또 다른 문제점은 이전 연구는 스케줄-가능성 분석을

전체 프로세서 이용률만 가지고 검사했다는 것이다. 프로세서 이용률 기반 분석은 우선순위를 가진 다중태스크의 최악 성능 범위를 보장하는 방법이긴 하지만 그 근본 원리가 단일프로세서서서 시스템을 가정하고 고안된 것이기 때문에 스케줄-가능성 검사도 HMS 목적함수처럼 전체 다중프로세서 시스템 전체를 하나의 프로세서로 간주하여 분석하는 한계를 가지고 있었다. 즉, 노드 매핑 결과가 그림 3(c)의 경우와 같을지라도 이용률에 근거한 분석 방법을 적용할 경우, 전체 이용률이 1.63(35/40 + 45/60)이 되어 실제로 스케줄 가능한 결과임에도 불구하고 스케줄이 불가능하고 분석된다. 따라서 우리는 동시에 실행 가능한 다중태스크의 새로운 스케줄-가능성 분석 방법이 필요하였고, 이 논문에서 Timed Multitasking (TM)모델[4]을 위해 고안된 정적 스케줄에 알맞은 스케줄 기반의 다중태스크 스케줄-가능성 분석 방법을 구현하였다.

3. 관련 연구

다중태스크를 통합합성 하는 연구는 이질적인 내장형 분산(Distributed Heterogeneous Embedded, DHE) 시스템에서 많이 연구되어 왔다[5-8]. R. P. Dick과 N. K. Jha의 MOGAC[5]과 그 확장 버전인 EMOGAC[6]은 유전 알고리즘(Genetic algorithm)을 사용하여 다중태스크 문제를 풀었고, B. P. Dave와 G. Lakshminarayana는 COSYN[7]이라고 불리는 반복적인 휴리스틱 방법을 이용하여 문제를 해결하였다. 그들의 연구는 우리와 비슷한 분야에 속한 것이긴 하지만, 문제의 크기가 작고 무엇보다 태스크를 매핑 하는 기본 단위에 차이가 있다. 그들의 연구는 매핑 단위가 태스크인데 반해 우리의 매핑 단위는 그 보다 더 작은 태스크내의 기능블럭인 노드 단위로 하였다. 또한 이들은 다중모드, 비주기 태스크에 대해서 고려할 수 없을 뿐만 아니라 스케줄-가능성 분석 방법도 매핑 알고리즘과 매우 관련되어 있어 다양한 예제로의 적용이 어려운 단점이 있다. 즉 제시된 알고리즘을 적용하려면 특별한 태스크 스케줄링 환경이어야만 동작이 가능하다는 것을 의미한다. 반면에 우리의 접근 방식은 매핑문제와 스케줄-가능성 분석방법이 분리, 구현되어 있어 서로 다른 매핑 방법 또는 스케줄-가능성 분석 알고리즘의 비교를 쉽게 할 수 있는 장점이 있다.

스케줄-가능성 분석 방법에 대해서도 그들의 방법은 모든 태스크가 자신의 하이퍼-주기(Hyper-Period) 또는 최소 공통 배수(Least Common Multiple, LCM) 주기만큼에 대해 모든 태스크가 완전히 정적 스케줄 하는 것으로 가정하였다. 따라서 그들의 알고리즘 복잡도는 문제의 크기가 태스크의 크기, 태스크의 수, 하이퍼-주기의 곱에 비례하게 되고, 실행 가능한 스케줄을 얻는 시간도 매우 길다. 이를 통합설계 플로우에 적용한다고 가정하면 생성된 스케줄에 따라 코드를 합성해야 하는데, 최악의 경우 각 태스크가 자신의 하이퍼-주기까지 매번 서로 다른 프로세서에서 수행되어야 한다. 즉, 한 태스크의 컴파일 된 코드 이미지가 모든 프로세서내의 메모리에 중복되어 적재될 수도 있다. 이것은 응용 프로그램의 크기가 커질 경우 시스템 설계에 치명적일 수 있다. 반면에 우리의 접근 방법은 오직 태스크 수의 정수배만큼의 복잡도를 갖기 때문에 태스크의 수가 커지더라도 그에 비례한 매핑 시간을 갖게 될 뿐이고, 프로세서에 적재될 태스크 코드 이미지도 오직 매핑 결과로 얻어진 태스크들에 대해서만 고려되면 되기 때문에 알고리즘 자체는 응용 프로그램 크기에 둔감하다.

P. Pop *et al.* [9]은 branch-and-bound 방법을 사용하여 time-triggered와 event-triggered 응용이 포함된 분할과 매핑문제를 풀었고, (E)MOGA, COSYN과 마찬가지로 태스크 종단간의 최악 응답시간을 매핑의 한 파라미터로 포함시켜, 매핑문제와 스케줄-가능성 분석 문제를 혼합하여 풀도록 하였다. 이런 접근 방법은 고정된 실행 시간을 갖으면서 실시간성을 갖는 다중태스크가 다중프로세서로, 최적의 매핑 결과를 갖기 위한 알고리즘은 NP-완전 문제임을 증명한 Leung과 Whitehead [10]에 따른 것이다.

한편 과거 수십 년 동안 다중프로세서 시스템에서 다중태스크의 최악 응답시간을 분석하는 방법이 연구되어 왔다[11-14]. 그들 중 Tindell의 방법은 선구자적 역할을 하였으며, 이후 다른 논문들의 근간이 되었다. Tindell *et al.* [11]에 의하면 고정적 옵셋(Offset)을 갖는 태스크 집합에 대해 최악 응답 시간을 계산하는 방법을 제시하였다. 그 논문에서 시스템은 주기성을 갖은 트랜잭션(Transaction)이라고 하는 태스크 집합을 가지고 있으며, 태스크의 최악 응답 시간을 구하기 위해서 모든 트랜잭션 시나리오에 대해 자신 보다 우선순위가 높은 모든 태스크들은 자신을 선점할 수 있다는 가정을 하였다. 이는 최악의 경우 우선순위가 가장 낮은 태스크 내에 있는 한 노드의 선점시간은 상위 태스크들의 전체 스케줄 시간의 합이 되고, 결과적으로 우선순위가 가장 낮은 태스크의 전체 선점시간은 상위 태스크의 수, 상위 태스크내의 노드 수, 자신의 태스크 내에 있는 노드 수의 곱에 비례하게 된다. 이는 노드단위의 매핑을 가정하는 우리의 경우, 아주 비관적인 가정이며 실용성이 거의 없다.

F. Slomka[13]는 태스크들 간에 데이타-의존성이 있는 응용에 대한 시뮬레이션 기반의 접근 방법으로 하위 우선순위 태스크의 가장 늦은 실행 시각을 구하기 위해 상위 태스크로부터 가능한 모든 선점 시나리오를 고려

하는 방식이다. Slomka의 방법은 이전 Tindell의 분석적인 방법에 비해서 다중태스크 시스템에서 보다 현실적인 성능을 나타내지만, 태스크의 부모가 둘 이상이 되는 그래프 형태에 대해서는 적용될 수 없다는 치명적 단점이 있다. 즉, 제한적인 태스크 그래프에 대해서만 적용 가능하다는 의미이다.

결론적으로, 스케줄 상 루프가 존재하는 경우 및 태스크 내에 존재하는 노드가 매핑의 단위로 세분화될 경우, 위의 제시된 방법들로 스케줄-가능성을 분석할 수 없다. 이를 위해 이 논문에서는 우리의 태스크 모델과 가장 유사한 모델인 TM 모델을 사용하여 스케줄-가능성 분석 방법을 구현하였다. 실제로 TM 모델은 다음의 몇 가지 가정을 가지고 있으며 아래와 같다.

- 노드의 실행 시간은 런타임에 변하지 않고, 통신은 노드 실행 마지막에 발생한다.
- 노드 실행 중간에 공유 자원을 접근할 수 없다.
- 자신이 수행되기 전에 필요한 모든 데이타는 이미 도착되어 있어야 한다.
- 태스크 스케줄은 정적 컴파일 시간에 작성되며, 노드의 수행 순서는 변하지 않는다.
- 만일 스케줄 된 시간 보다 먼저 노드가 끝날 경우, 다음 스케줄 될 노드들은 정해진 원래 시간까지 기다린다.

이러한 TM 모델은 본 논문에서 가정하는 노드 스케줄 규약과 매우 비슷하기 때문에 이 논문내의 스케줄-가능성 분석단계에서 구현하여 사용하였다.

4. 제안하는 매핑 기법

2장에서 설명되었듯이 우리는 기존의 통합합성 기법을 다음 두 가지 방법으로 개선하였다. 첫째, 이전의 기법은 분할에 참여하는 모든 태스크들이 자신의 스케줄-길이를 최소로 줄이기 위해 분할에 필요한 모든 자원을 독점한데 반해, 이 논문에서는 이미 수행된 우선순위가 높은 상위 태스크들의 분할결과를 이용하여 하위 태스크 안에 있는 노드의 분할 위치를 결정하는 방식을 제시한다. 이 기법은 '매핑회피 기법'이라고 불리며, 성능이 높은 프로세서에 모든 태스크가 몰려 오히려 전체 성능이 저하되는 현상을 회피함으로써 시스템 전체적인 성능을 향상 시킬 수 있다. 둘째, 태스크 할당을 프로세서에 전용시킴으로써 통신 횟수를 줄임과 동시에 보다 향상된 성능을 나타내는 기법을 추가로 제안한다. 이것은 '매핑전용 기법' 이라고 불리며, 4.2에서 자세히 설명하였다.

4.1 태스크 매핑 회피 기법 (Task Mapping Avoidance Technique, TMA)

다중프로세서 시스템을 위한 태스크 매핑의 중요 목표는 전체적인 프로세서 이용률의 균형을 맞추는 일이다. 그림 4는 수정된 통합합성 루프를 도시하고 있으며, 본 논문에서 제시하는 기법은 다음과 같다.

- 단계(1)
  주어진 응용에서 태스크들의 매핑 순서를 정하기 위해 상위 우선순위를 가진 태스크들로부터 하위 우선순위를 가진 태스크들로 정렬한다.
- 단계(2)
  스케줄 되지 않은 태스크들 중 가장 우선순위가 높은 태스크를 취해 매핑 한다.
- 단계(3)

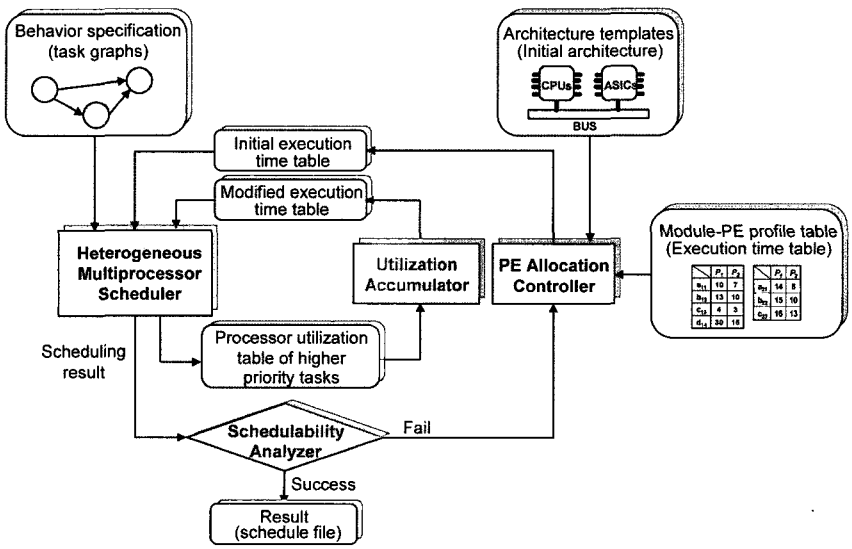


그림 4 수정된 통합합성 루프

매핑된 태스크가 수행된 스케줄 결과에 따라 실행된 상위 태스크들에 의한 프로세서 이용률을 계산하고, 이후에 수행될 하위 태스크의 Module-PE profile 테이블의 수행시간 값을 갱신한다. 이 작업은 그림 4에서 프로세서 이용률 누적기(Utilization Accumulator)에 의해 수행된다.

• 단계(4)

모든 태스크가 매핑 될 때까지 단계(2)~(3)을 반복한다.

단계(1)에서 태스크들을 정렬하는 기준은 주기가 짧은 태스크에 높은 우선순위를 주는 방식을 택하였고, 같은 우선순위를 가진 것들에 대한 우선순위 부여는 그들 중 임의로 선택하였다. 동일 우선순위를 갖는 태스크들에 의한 스케줄 성능의 차이는 이 논문에서 다루지 않고 이후 연구 과제로 남겨 두었다. 왜냐하면 다중프로세서 시스템에서 다중태스크를 스케줄 할 경우, 같은 우선순위를 같더라도 하더라도 어느 태스크를 먼저 스케줄 하느냐에 따라 매핑결과의 성능이 달라지기 때문에 이 연구의 주제 범위를 넘어선 또 다른 문제이기 때문이다. 실제로 많은 실시간 응용들의 경우, 짧은 주기나 사용자에 의해서 긴급으로 부여된 태스크들에 대해서 높은 우선순위를 주는 것이 일반적이다.

단계(3)은 이 논문이 제안하고 있는 핵심 알고리즘이다. 아이디어는 현재까지 사용된 프로세서들의 이용률에 기반하여 Module-PE profile 테이블 값을 변경하는 것이다. 다시 말해 높은 우선순위 태스크가 높은 성능의 프로세서를 사용한다면, 낮은 우선순위 태스크들은 그 프로세서에서의 실행시간 값을 상대적으로 증가시킴으로써 그 결과 상위 태스크가 매핑 된 프로세서로의 매핑을 회피시키는 것이다. 이 과정을 수식으로 정리하면 표 1의 범례와 식 (1), (2)와 같다.

상위 태스크에 대한 프로세서 이용률이 식 (1)에 의해서 얻어졌으면 하위 태스크의 프로세서 P_k에서의 실행시간 변경 값은 식 (2)에 의해서 얻어진다. 식 (2)가 암시하는 내용은, 우선순위가 가장 높은 태스크는 이전에 실행된 태스크가 없기 때문에 값이 0이 되어 노드의 수행시간은 데이타베이스로부터 얻은 값에서 변경되지 않는다. 만일 어느 하위 우선순위 태스크가 매핑 될 때는 식 (1)에 의하여 어떤 프로세싱 컴포넌트의 누적된 프로세서 이용률이 1에 근접하게 되면, 그 프로세서에서 수행되는 노드의 수행시간은 식 (2)에 의해 무한대로 늘어나게 되어 결국 그 프로세서로의 매핑이 불가능해짐을 의미한다.

$$util(t_i,P_k)=\sum_{i\in hp(t_i)}\left(\frac{1}{T_i}\times\sum_{p(n_{ij})\cap P_k}\left(E(n_{ij},P_k)+comm(n_{ij})\right)\right) \quad (1)$$

$$E^{new}(n_{ij},P_k)=E(n_{ij},P_k)\times 1/(1-util(t_i,P_k)) \quad (2)$$

이제 그림 3의 예제로 돌아가서 제안된 휴리스틱을 적용해 보자. $Task_1$이 상위 태스크라고 하면, 그림 3(b)처럼 최소 스케줄-길이를 얻기 위해 b_{12}와 c_{13}은 동시에 서로 다른 프로세서로 매핑 되어 실행될 것이다. $Task_1$이 매핑 된 이후 누적된 프로세서 이용률 값은 다음과 같이 계산되어진다.

$$\begin{aligned} util(t_1,P_1) &= \{E(c_{13},P_1)+comm(c_{13})\}/T_1 \\ &= \{(4+3)\}/40=0.18 \\ util(t_1,P_2) &= \{(E(a_{11},P_2)+comm(a_{11}))+ \\ &\quad (E(b_{12},P_2)+comm(b_{12}))+ \\ &\quad (E(d_{14},P_2)+comm(d_{14}))\}/T_1 \\ &= \{(7+1)+(10+0)+(15+2)\}/40 \\ &= 0.88 \end{aligned}$$

이후 $Task_2$가 매핑을 시도할 경우, $Task_2$의 Module-PE profile 테이블은 그림 5에서 보인 것처럼 갱신된다. 갱신된 테이블 값을 가지고 HMS는 $Task_2$에 대해서 매핑을 수행할 것이고, 그 결과는 그림 3(c)에 보인 것과 같이 P_1에 스케줄 될 것이다. 회색으로 칠한 부분은 HMS에 의해 결정된 매핑 결과를 나타낸다.

표 1 용어 정의

n_{ij}	태스크 인덱스 i 내의 j번째 노드 번호
T_i	태스크 i의 주기
$p(n_{ij})$	매핑이 결정된 노드 n_{ij}의 프로세서 인덱스
t_i	인덱스 i를 갖는 태스크
$hp(t_i)$	태스크 i보다 우선순위가 높은 태스크의 집합
$E(n_{ij}, P_k)$	노드 n_{ij}가 프로세서 인덱스 k를 갖는 P_k에서 얼마의 수행시간을 갖는지를 나타내는 값으로 그림 1의 노드 수행성능예측 단계에서 측정된다. 측정된 값은 데이타베이스에 저장되어 다른 응용에 사용될 수 있도록 라이브러리화 된다.
$E^{new}(n_{ij}, P_k)$	프로세서 P_k에서 노드 n_{ij}의 예측된 수행시간으로 매핑 단계에서 매핑 결정을 위해 동적으로 임시로 계산되어지는 수행시간 값을 의미한다. 수정된 값은 데이타베이스에 저장되지 않는다.
$comm(n_{ij})$	노드 n_{ij}의 통신비용을 말한다. 이 값은 송신비용과 수신비용 값을 포함하고 있으며, 각 비용은 통신비용의 반으로 가정하였다.
$util(t_i, P_k)$	태스크 t_i 이전에 수행된 상위 태스크들의 프로세서 P_k에서의 누적된 이용률을 의미하며, 식 (1)에 의해 계산되어진다.

	P_1	P_2
a_{21}	14	8
b_{22}	15	10
c_{23}	16	13

	P_1	P_2
a_{21}	17	66
b_{22}	18	83
c_{23}	19	108

그림 5 갱신된 $Task_2$의 Module-PE profile 테이블

만일 $Task_2$보다 우선순위가 낮은 $Task_3$이 주어진다면 누적기에 의해 누적된 각 프로세서 이용률은 아래와 같이 다시 계산될 것이다. $Task_2$는 P_2를 사용하지 않았기 때문에 $util(t_3, P_2)$는 변하지 않고, 오직 $Task_3$에 대한 P_1의 이용률만 다시 계산되어 진다.

$$\begin{aligned} util(t_3,P_1) &= \{E(c_{13},P_1)+comm(c_{13})\}/T_1 + \\ &\quad \{(E(a_{21},P_1)+comm(a_{21}))+ \\ &\quad (E(b_{22},P_1)+comm(b_{22}))+ \\ &\quad (E(c_{23},P_1)+comm(c_{23}))\}/T_2 \\ &= \{(4+3)\}/40+ \\ &\quad \{(14+0)+(15+0)+(16+0)\}/60 \\ &= 0.175+0.75=0.925 \end{aligned}$$

여기서 a_{21}, b_{22}, 그리고, c_{23}의 통신비용이 0이 된 것은 이들 모두가 서로 P_1으로 매핑되었기 때문이다.

Module-PE profile 테이블을 갱신하는 제안된 알고리즘은 그림 6에 정리하였다.

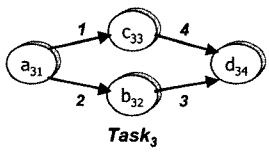
```
1 : Algorithm I : change module-PE profile table
2 : While if a node n_ij to be scheduled exists do
3 :   For each processor P_k
4 :     If (P_k is HW) then
5 :       If (P_k implements n_ij) then E^new(n_ij, P_k) <= E(n_ij, P_k);
6 :       Else E^new(n_ij, P_k) <= INFINITE;
7 :       End If
8 :     Else /* is SW */
9 :       weight <= get accumulated processor utilization ();
10:       E^new(n_ij, P_k) <= E(n_ij, P_k)* 1/(1-weight);
11:     End If
12:   End For
13: End While
14: End Algorithm I
```

그림 6 Profile 테이블을 변경시키는 알고리즘

그 결과, 그림 2의 이전 통합합성 기법이 그림 4와 같이 HMS에 프로세서 이용률 누적기가 포함된 기법으로 변경되었다.

4.2 태스크 매핑 전용 기법 (Task Mapping Pinning Technique, TMP)

'매핑회피' 기법이 이전 연구보다 다중프로세서에서 좋은 성능을 제안하기는 하지만 몇 가지 면에서 개선할 사항이 있다. 첫째, '매핑회피' 기법은 프로세서 이용률에 따라 태스크 내에 있는 노드를 분산 매핑 시키는 방법이므로 통신횟수가 비교적 많다. 늘어난 통신비용을 감수하더라도 이전 매핑 알고리즘에 비해 향상된 성능을 내는 것만으로도 '매핑회피' 기법은 상당히 매력적인 방법이다. 하지만, 다중코어가 있는 SoC에서 태스크를 기능별로 나눈 다음, 특정 태스크를 특정 프로세서에 전용 (Pinning) 시켜 수행하는 것과 같은 일반적인 해법을 제시하지 못하는 한계가 있다. 또한 통신횟수가 많아지면 그 만큼 통합시뮬레이션 속도도 동기화(Synchronization) 문제로 인해 느려지기 때문에 통신횟수를 줄이는 방향으로 시스템이 설계되는 것이 더 바람직하며, 매핑 된 결과를 가지고 코드를 합성할 때도 태스크가 프로세서에 전용되어 있으면 합성된 코드 안에 통신을 위한 코드가 삽입될 필요가 없기 때문에, 코드 크기도 작아지며 생성되는 코드의 복잡도도 많이 줄어들 것이다. 이 논문에서는 4.1에서 제안된 '매핑회피' 기법을 개선한 태스크 '매핑전용' 기법에 대해 간단한 동기 예제를 가지고 설명할 것이다. 만일 그림 3에 소개된 예제에 그림 7과 같은 동작을 가지는 $Task_3$이 추가로 주어진다고 가정해 보자.



	$Task_3$
Period	120
Deadline	120

	$P_1(20)$	$P_2(100)$
a_{31}	6	3
b_{32}	8	6
c_{33}	10	9
d_{34}	5	3

그림 7 추가된 $Task_3$ 예제 그래프

제안된 '매핑전용' 방식의 핵심 아이디어는 우선순위가 낮은 태스크를 성능이 낮은 프로세서에 우선적으로 전용시키는 방법이고, 전용방법을 이용하여 스케줄이 불가능한 태스크의 경우에 한해서 4.1에서 제안된 '매핑회피' 방식을 적용하는 것이다.

그림 12는 제안된 알고리즘을 3-단계로 구분한 것으로 첫 번째 단계는, 선-스케줄 단계(Pre-schedule)로 PE-할당-제어기에 의해 주어진 각각의 프로세서에 대해 각 태스크들이 전용될 수 있는지를 검사하는 단계이다. 만일 프로세서에 태스크가 전용될 수 있으면 그림 8에 나와 있는 것처럼 프로세서 인덱스, 태스크 인덱스, 이용률의 주요키(Primary key)를 가진 전역-레코드 테이블에 저장된다. 이 전역-레코드 테이블은 주어진 프로세서들에 대해 각 태스크들이 전용될 수 있는지 없는지를 저장하고 있는 자료 구조로써 통합합성이 끝날 때까지 그 내용이 유지된다. 따라서 제일 싼 프로세서인 P_1 (현재 시스템비용=20)이 할당되면, 그림 8(a)처럼 $Task_2$, $Task_3$만 저장되고, $Task_1$은 프로세서 이용률이 1(57/40

current system cost
20

Index	Proc. Id	Task Id	Makespan	Period	Utilization
0	1	2	45	60	0.75
1	1	3	29	120	0.24

(a) P_1이 사용 가능할 경우, 현재 시스템 가격 = 20

current system cost
100

Index	Proc. Id	Task Id	Makespan	Period	Utilization
2	2	1	35	40	0.88
3	2	2	29	60	0.48
4	2	3	23	120	0.19

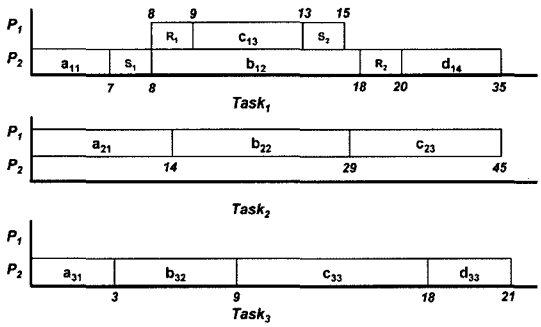
(b) P_2가 사용 가능할 경우, 현재 시스템 가격 = 100

그림 8 전역-레코드 테이블 내용

	P_1	P_2
a_{31}	6	3
b_{32}	8	6
c_{33}	10	9
d_{34}	5	3

	P_1	P_2
a_{31}	80	25
b_{32}	106	50
c_{33}	133	75
d_{34}	67	25

(a) '매핑회피' 방식에 의한 $Task_3$ 분할 결과



(b) 동기예제 분할 결과

그림 9 '매핑회피' 방식에 의한 동기예제 분할 결과

= 1.43)을 넘기 때문에 제외된다. P_1만 가지고 그림 11의 이형 다중프로세서 전용 스케줄러(Heterogeneous Multiprocessor Pinning Scheduler, HMPS)는 모든 태스크를 스케줄 할 수 없으므로, PE-할당-제어기에게 더 비싼 프로세서 조합을 요구할 것이다. PE-할당-제어기는 다음으로 비싼 P_2(현재 시스템 비용=100)를 선택할 것이고, 또 다시 그림 8(b)처럼 $Task_1$, $Task_2$, 그리고 $Task_3$에 대해서 레코드가 추가로 채워질 것이다. P_1과 마찬가지로 P_2에 대해서 HMPS는 모든 태스크를 스케줄 할 수 없으므로, PE-할당-제어기에게 더 비싼 시스템 자원을 요구하고, PE-할당-제어기는 120의 비용을 가진 P_1, P_2 프로세서 조합을 선택한다. P_1과 P_2는 이미 전역-레코드에 있기 때문에 그림 12의 첫 번째 단계인 선-스케줄 단계를 거치지 않고 두 번째 단 계인 태스크 전용 스케줄 단계로 넘어간다.

전용 스케줄 단계는 프로세서 이용률 누적기 상태를 고려해 태스크를 실행할 프로세서를 선택하는 단계로 우선순위가 낮은 태스크를 가장 싼 프로세서에 우선적으로 매핑 한다. 따라서 그림 8의 테이블 내용 중 가장 우선 순위가 낮은 $Task_3$이 가장 싼 프로세서 P_1에 매핑되고, 뒤를 이어서 $Task_2$가 매핑 된다. $Task_2$가 매핑 될 때 레 코드 테이블에 의하면 P_1, P_2에 매핑 될 수 있지만, P_2보다 싼 프로세서 P_1에서 스케줄 가능하므로 P_1으로 매핑 된다. 그리고 마지막으로 $Task_1$은 P_2에서만 스케줄 가능하므로 P_2로 매핑 된다.

그림 10(a), (b), (c)는 태스크가 레코드 테이블에 의해 전용 스케줄 될 때 프로세서 이용률 누적기의 상태를 나타낸 그림이고, 그림 10(d)는 최종적으로 매핑 된 결과이다. 만일 $Task_1$이 레코드 테이블에 존재하지 않는다면, $Task_1$은 4.1에서 제안된 '매핑회피' 방식이 적용될 것이다. 제안된 기법의 성능비교를 위해 주어진 예제를 순수한 '매핑회피' 기법을 이용해 매핑하면 그림 9(a)에 나와 있는 것처럼 $Task_3$의 내부 노드들은 HMS 스케줄에 의해 모두 P_2로 매핑될 것이다. 왜냐하면 $Task_2$가 매핑 된 후, 두 프로세서 P_1, P_2의 util 값은 아래와 같게 되고,

$$util(t_3, P_1) = 0.925,\ util(t_3, P_2) = 0.88$$

이후 $Task_3$의 Module-PE profile 테이블 값은 아래에 도시된 것처럼 변경될 것이다. 결국 $Task_3$은 P_2로 매핑 된다. 이렇게 되면 P_2의 프로세서 이용률은 1을 넘어 스케줄이 불가능할 것이다. 결론적으로, 태스크를 프로세서에 전용시킴으로써 줄어든 통신횟수의 이득을 바탕으로 우선순위가 낮은 태스크를 성능이 낮은 프로세서에 우선적으로 전용시킴으로써 더 좋은 매핑성능을 얻을 수 있다.

이때 고려할 점은, 우선순위가 낮은 태스크를 빠른 프로세서에 우선 매핑 하는 경우와 우선순위가 높은 태스크를 빠른 프로세서에 우선 매핑 하는 경우를 추가로

	$Task_1$	$Task_2$	$Task_3$	Total
P_1	-	-	0.24	0.24
P_2	-	-	-	-

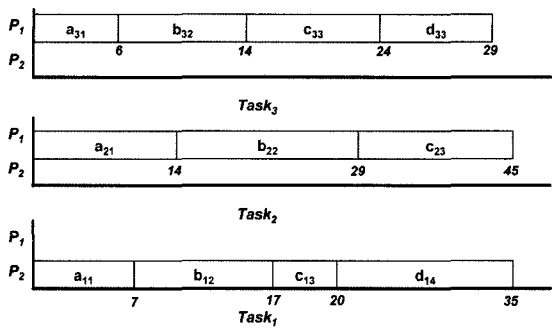
(a) $Task_3$가 스케줄 된 후 누적기 상태

	$Task_1$	$Task_2$	$Task_3$	Total
P_1	-	0.75	0.24	0.99
P_2	-	-	-	-

(b) $Task_2$가 스케줄 된 후 누적기 상태

	$Task_1$	$Task_2$	$Task_3$	Total
P_1	-	0.75	0.24	0.99
P_2	0.88	-	-	0.88

(c) $Task_1$이 스케줄 된 후 누적기 상태



(d) HMPS에 의한 최종 분할 결과

그림 10 동기예제의 분할과정에서의 프로세서 누적기 내용

생각해 볼 수 있는데, 전자의 경우, 우선순위가 낮은 태스크를 시스템 비용이 많이 들어간 빠른 프로세서에서 먼저 이용하는 것은 설계의 비효율성을 의미하고, 후자의 경우, 이후에 매핑 되는 우선순위가 낮은 태스크들 입장에서는 상대적으로 빠른 프로세서를 사용할 수 있기보다는 느린 프로세서 2개가 있는 것과 같은 효과를 본다. 결국, 우선순위가 낮은 태스크를 느린 프로세서에 우선 매핑 시키면, 이후에 매핑 해야 되는 우선순위가 높은 태스크들은 비교적 빠른 프로세서와 느린 프로세서의 여유분을 충분히 사용할 확률이 높아지기 때문에 더 효과적이다. 그림 12는 설명된 '매핑전용' 기법을 나타내는 알고리즘이고, 그림 11은 제안된 태스크 전용할당을 위해 '매핑회피' 기법에서 수정된 통합합성 루프를 나타낸다. 회색으로 표시된 부분은 HMS를 대체한 HMPS로 구현된 알고리즘이 수행되는 부분이며, 이전 분할 방법인 HMS를 비롯한 제시된 두 알고리즘은 마련된 통합설계 환경에서 사용자 입력(GUI)에 의해 선택적으로 사용되어 질 수 있도록 구현되었다.

4.3 스케줄 기반의 스케줄 가능성 분석

앞에서 설명한 바와 같이 제안된 기법에서는 스케줄-가능성 분석이 태스크 매핑 알고리즘과 독립적으로 수행된다. 이는 태스크 수행 정책에 따른 다양한 분석 기법을 통합설계 환경 안에서 사용자가 응용의 목적에 맞게 비교 및 적용할 수 있다는 장점이 있다. 이전 기법에서는 프로세서-이용률 기반의 분석 방법만을 제시하였는데 그것은 시스템이 단일 운영체제를 갖고 하나의 제어 프로세서에 의해 동작하게 되는 가정에서만 유효한 분석 방법이었다. 따라서 이 문제를 해결하기 위해 이

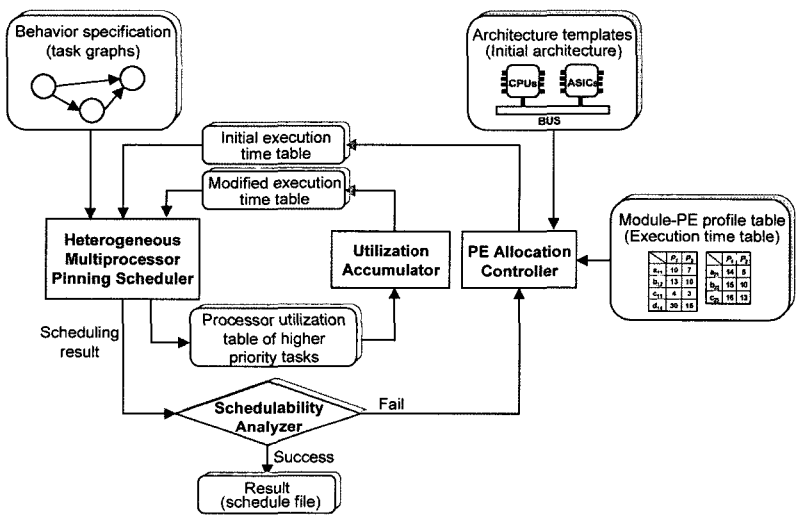


그림 11 제안된 태스크 전용할당을 위한 통합합성 루프

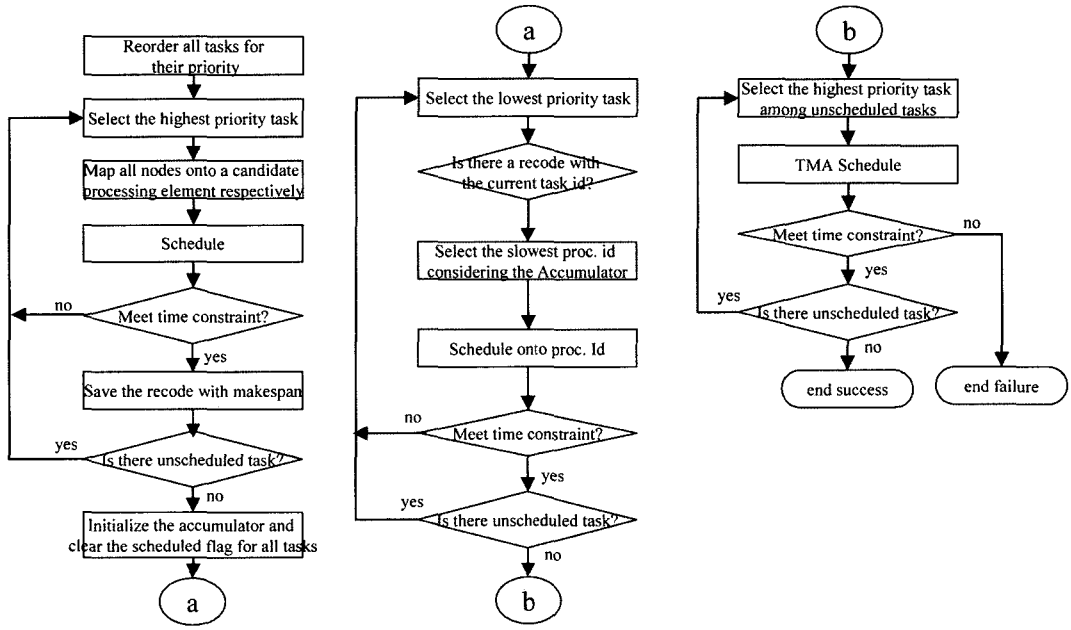
논문에서는 TM 계산 모델의 실행 시나리오를 적용한 스케줄 기반의 분석 방법을 구현 적용하였다.

아래 열거된 내용은 현재 이 논문에서 가정하고 있는 태스크 모델의 중요 명세 조건들을 정리한 것으로 다음과 같다.

- 태스크는 주기성을 갖는다: 산발(Sporadic) 태스크는 주기적 태스크 실행 사이에 짧은 실행 간격을 가진 태스크로 모델 될 수 있다.
- 태스크의 우선순위는 정적으로 컴파일-시간에 유일하게 정해진다. 태스크 내에 있는 노드들의 우선순위 및 주기는 자신이 속해 있는 태스크의 속성을 그대로 상속 받는다.
- 각 태스크의 시작 시각은 고정적으로 정해졌으며, 예제에서 보인 것과 같이 모든 태스크는 0에서 시작한 것으로 가정한다.
- 태스크 내의 각 노드들은 하나의 프로세싱 컴포넌트로 매핑 될 수 있으며, 한번 매핑 된 노드-프로세서 조합은 그 태스크가 끝날 때까지 변경되지 않고 끝까지 유지된다.

제안된 스케줄-가능성 분석 방법은 모든 태스크를 그들의 하이퍼-주기만큼 스케줄을 분석하는 방법으로 우선순위가 높은 태스크부터 순차적으로 스케줄 한다. 하위 태스크는 이미 상위 태스크가 실행되고 남은 휴지시간을 가지고 스케줄 되기 때문에 상위 태스크에 의한 선점시간은 스케줄 과정 중에 자연스럽게 계산되어 진다. 아래는 구현된 간단한 알고리즘이다.

- 단계(1)
  모든 태스크는 자신들의 우선순위가 높은 순서로 정렬된다.
- 단계(2)
  그들 중 가장 높은 우선순위를 가진 태스크 하나가 선택되고, 하이퍼-주기를 자신의 주기로 나눈 만큼 반복적으로 HMS에서 매핑 된 결과에 따라 실행된다. 그림 3의 예를 들면, 그들의 하이퍼-주기는 120이므로 $Task_1$은 3번 수행되어야 하며, $Task_2$는 2번 수행되어야 한다.
- 단계(3)
  남은 태스크들 중 사이에서 우선순위가 가장 높은 태스크를 선택하여, 상위 태스크가 남은 시간-슬롯(*Time slot*)을 이용해 스케줄 한다. 그림 13은 $Task_2$가 $Task_1$ 이후에 스케줄 된 타이밍 다이어그램의 일부이다. 노드 a_{21}은 P_1의 0에서 시작될 수 있고, 구간 [8, 15] 사이에서 $Task_1$에 의해 선점되었다. P_2의 구간 [35, 40] 사이는 두 태스크가 사용하지 않는 빈 여유시간(Slack time)을 나타낸다.
- 단계(4)
  모든 태스크가 스케줄 될 때까지 단계(3)을 반복한다.
- 단계(5)



(a) 선-스케줄 단계 (b) 태스크 전용 스케줄 단계 (c) 매핑회피 단계

그림 12 제안된 3-단계 태스크 전용 할당 방법

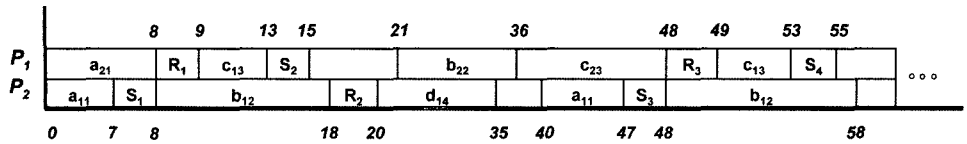


그림 13 예제 태스크 그래프 $Task_1$과 $Task_2$의 스케줄 다이어그램

만일 어떤 태스크가 주어진 실시간 제약-조건을 만족하지 않는다면, 에러 메시지를 사용자에게 알려주며 PE-할당-제어기에게 다른 프로세서를 사용하도록 요청한다.

위 결과에서 보인 것처럼 자신의 하이퍼-주기까지 태스크의 정적 스케줄을 분석한다. TM 모델에서는 런타임에도 노드의 스케줄 순서를 변경하지 않으며, 노드의 실행시간이 미리 주어진 양보다 먼저 끝날 경우, 예를 들어, a_{21}이 실제로 21 이전에 끝났을지라도 b_{22}는 21 이전에 실행 될 수 없다. 이 모델은 노드를 실제로 실행할 때 발생하는 지터(Jitter)의 영향을 노드 수행시간 안에 포함 시켜 분석할 수 있는 장점이 있으며, 입력 태스크 그래프가 SDF(Synchronous Data Flow) 모델[17]로 작성되었을 경우, SDF의 실행 조건(Firing condition)과 유사하여 그 스케줄-가능성을 분석하는데 매우 적합하다.

5. 실 험

제안된 통합합성 기법의 검증을 위해 우리는 실제적인 시스템 예제인 4-채널 DVR(Digital Video Recorder)과 관련 논문에서 인용된 임의 그래프를 수행하여 이전 기법과의 성능을 비교하였다. 제안된 기법은 연구실에서 개발하고 있는 통합설계 환경 안에서 구현하였으며, 그림 14는 시스템 수준에서 예제를 명세한 그림이다.

DVR 예제는 4개의 채널로 구성되었으며 각 채널은 H.263 인코더 태스크로 구현되어 주기 및 데드라인과 같은 자신만의 실시간 제약-조건을 갖고, H.263 인코더 명세를 위해 참조코드로 TMN2.0이 사용되었다.

통합합성을 위한 입력으로 사용되는 초기 아키텍처의 명세를 위해 서로 성능이 다른 ARM 프로세서를 명세하였고, 하드웨어 프로세싱 컴포넌트를 위해 하나의 FPGA (Field Programmable Gate Array)를 사용하였다.

사용된 ARM 프로세서는 L1 캐시를 갖는 133Mhz의 ARM720T, 266Mhz ARM926ej-s, 그리고 399Mhz의 ARM1020T를 각각 10개, 총 30개의 프로세서를 사용하였다. FPGA는 HDL(Hardware Description Language)로 작성된 하드웨어 IP 코어를 수행하기 위한 프로세서로 H.263 인코더 알고리즘내의 DCT, IDCT(Inverse DCT), MC(Motion Compensation), ME(Motion

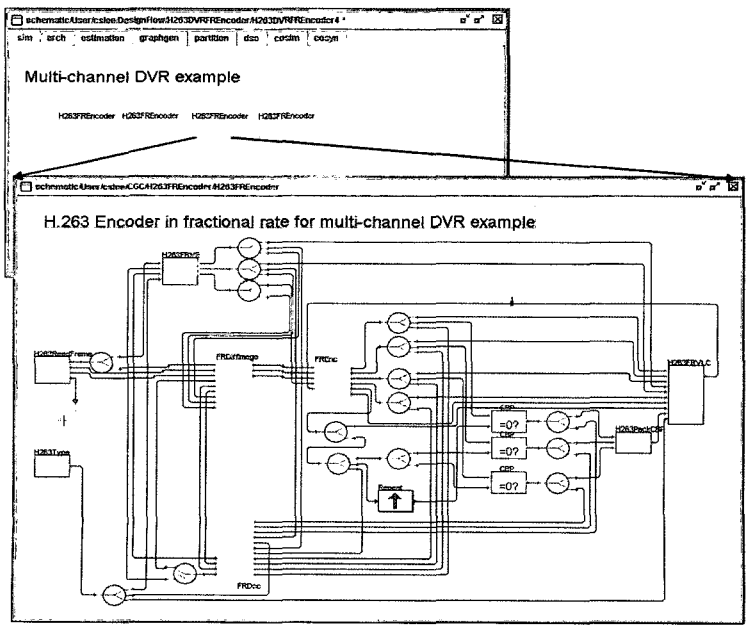


그림 14 시스템-수준에서의 4-채널 DVR 명세

Estimation)를 수행한다. 그림 1에서 보인 것처럼 모든 후보 프로세싱 컴포넌트들은 133Mhz의 버스에 연결되어 있는 간단한 구조를 사용하였고, 버스구조는 매핑 이후 통신구조탐색 단계에서 탐색된다.

매핑을 수행하기 전, 노드들에 대한 각 ARM 프로세서에서의 수행시간 예측이 필요한데 이 논문에서는 소프트웨어 프로세서 시뮬레이터인 ADS(ARM Development Suite) 1.2에 포함된 'armcc' 컴파일러에서 -02 옵션을 사용하여 코드를 컴파일 하였고, Xeon 2.8Ghz 듀얼 CPU 에서 수행시켰을 때 걸리는 사이클 수를 측정하였다. 하드웨어 IP의 경우 소프트웨어로 수행될 때보다 약 10배 정도 빠르다고 가정하였다.

표 2는 DVR 예제를 위해 H.263 인코더가 하드웨어 IP 없이 후보 프로세싱 컴포넌트들 위에서 소요되는 스케줄-길이를 나타내고 있으며, 후보 프로세싱 컴포넌트 가격은 그 성능에 비례하게 임의로 정하였다. 예를 들어 ARM1020T는 ARM720T에 비해 4배가 빠르기 때문에 가격도 4배 비싸게 책정했다. DVR 예제의 입력파일로 압축되지 않은 FOREMAN.QCIF를 사용했고, 실험 결과는 시스템의 데드라인을 줄여가면서 제안된 각 알고리즘이 얼마만큼의 비용을 가지고 스케줄 되는가를 측정하였다. 최소 스케줄 길이가 짧을수록 단위시간당 래코딩 하는 프레임 수가 많기 때문에 스케줄 길이가 짧으면 짧을수록 더 높은 성능을 나타내는 것을 의미한다.

표 3과 표 4는 4-채널 DVR을 가장 싼 프로세서에서 실행 시킬 수 있는 시간 제약-조건에서부터 점차 조건을 강화시키면서 어떻게 태스크 내 노드들이 여러 프로세서로 매핑 되는지를 보여준다. 이전 기법은 가급적 짧은 스케줄-길이를 얻기 위해 주어진 후보 프로세싱 컴포넌트 중 가장 빠른 프로세서를 사용하도록 설계되어

표 2 DVR 예제를 위한 후보 프로세싱 컴포넌트들에서의 노드 수행시간 예측

Candidate processing components		Schedule length (Bus cycles)
Core type	Cost	
ARM720T	100	45105050
ARM926ej-s	200	19113426
ARM1020T	400	10480947
DCT	2000	-
IDCT	2600	-
MC	4000	-
ME	4000	

있기 때문에 매핑 된 빠른 프로세서의 이용률이 급격히 증가하게 된다. 따라서 비교적 큰 시간 제약-조건하에서도 매핑이 불가능해짐을 알 수 있다.

전체적으로 이전 결과보다 '매핑회피' 나 '매핑전용' 방법이 더 우수한 성능을 나타냄을 볼 수 있고, 시간 제약 구간 [190000000, 10400000]에서 '매핑전용' 방식이 '매핑회피' 방식보다 더 좋은 성능을 나타냄을 알 수 있다. 이는 '매핑전용'이 소프트웨어 노드 간 통신하는 것을 없애는 방식으로 설계되었기 때문에 통신비용이 발생하지 않았기 때문이다. 반면, 시간 제약-조건이 구간 10100000이하부터 '매핑회피' 기법과 '매핑전용' 기법의 결과가 동일한 이유는 표 2에서 보인 것과 같이 가장 빠른 프로세서에서의 스케줄-길이 보다 더 짧은 시간 제약-조건에서는 한 태스크를 여러 프로세서로 전용시킬 수 없기 때문에 모든 태스크들에 대해서 '매핑회피' 기법이 사용되었기 때문이다. 시간 제약-조건을 7000000 이하로 주어졌을 경우, 제안된 방법의 알고리즘이 스케줄 불가로 나온 이유는 스케줄-길이가 시간 제약-조건을 넘어섰기 때문이고, 만일 더 빠른 프로세서나 하드웨

표 3 시간 제약-조건에 따른 4-채널 DVR의 스케줄 및 매핑 결과 (스케줄-길이, 통신횟수)

Timing constraints	Previous		TMA		TMP	
	Schedule length	# of comm.	Schedule length	# of comm.	Schedule length	# of comm.
190000000	180420200	0	180420200	0	180420200	0
180000000	76453740	0	76453740	0	76453740	0
76000000	41923788	0	41923788	0	57340278	0
41000000	39011524	64	38304487	111	38226852	0
38000000	37006180	88	37744840	107	31442841	0
37000000	36101424	240	35435224	76	31442841	0
31000000	26435224	76	27052325	82	19113426	0
19000000	18272732	248	16067138	188	10480947	0
16000000	15470105	244	15956329	222	10480947	0
10400000	N.A.	N.A.	10275697	242	10151485	48
10100000	N.A.	N.A.	8139147	243	8139147	243
9100000	N.A.	N.A.	8139147	243	8139147	243
8000000	N.A.	N.A.	7009193	262	7009193	262
7000000	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

표 4 시간 제약-조건에 따른 4-채널 DVR의 스케줄 및 매핑 결과 (사용한 프로세서, 비용)

Timing constraints	Previous		TMA		TMP	
	Used processors	Cost	Used processors	Cost	Used processors	Cost
190000000	7(1)	100	7(1)	100	7(1)	100
180000000	9(1)	200	9(1)	200	9(1)	200
76000000	10(1)	400	10(1)	400	7(1), 9(1)	300
41000000	9(1), 10(1)	600	9(3)	600	9(2)	400
38000000	9(1), 10(1), DCT	2600	9(3)	600	9(1), 10(1)	600
37000000	9(1), 10(3), IDCT	3400	10(2)	800	9(1), 10(1)	600
31000000	10(1), ME	4400	7(1), 10(2)	900	9(4)	800
19000000	9(1), 10(4), ME	5800	10(6)	2400	10(4)	1600
16000000	9(1), 10(2), DCT, IDCT, ME	9000	9(1), 10(6)	2600	10(4)	1600
10400000	N.A.	N.A.	9(1), 10(6), ME	6600	10(4), IDCT	3600
10100000	N.A.	N.A.	9(1), 10(7), ME	7000	9(1), 10(7), ME	7000
9100000	N.A.	N.A.	9(1), 10(7), ME	7000	9(1), 10(7), ME	7000
8000000	N.A.	N.A.	9(4), 10(8), MC, ME	12000	9(4), 10(8), MC, ME	12000
7000000	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

표 5 Hou & Wolf 예제

Example	EMOGAC		COSYN		Previous		TMA		TMP	
	Price	CPU Time(s)	Price	CPU Time(s)	Price	CPU Time(s)	Price	CPU Time(s)	Price	CPU Time(s)
H&W 1&2(u)	140	600	170	-	170	0.1	150	0.13	150	0.02
H&W 1&3(u)	170	4440	170	-	170	0.1	170	0.13	150	0.02
H&W 3&4(u)	140	600	N/A	-	170	0.1	170	0.13	150	0.02

어 IP를 사용한다면 현재보다 더 짧은 스케줄-길이를 가진 매핑결과가 생성될 것이다. 또한 표 4에서 제안된 방법이 기존연구 결과와 달리 최소 스케줄-길이를 얻기 위해 MC, ME만 사용되고, 또 다른 사용가능한 하드웨어인 DCT, IDCT는 사용되지 않은 결과처럼 보이는데, 그 이유는 DCT나 IDCT를 사용하여 얻는 노드 수행시간 단축의 이득보다 그들로의 분할로 인한 부가적인 통신시간의 증가율 즉, 통신횟수가 늘어남에 따라 버스에서 충돌되어 노드의 시작시간이 밀려 시스템 전체에 생기는 스케줄 시간소모 비율이 더 컸기 때문이다.

결과적으로, 기존의 기법을 사용할 경우 최종적으로 얻어진 최소 스케줄-길이는 15470105 사이클로 133Mhz 버스의 성능비로 환산한다면, 한 프레임을 처리하는데 걸리는 시간은 0.1125초($1.5\times10^{7}\times7.5\times10^{-9}$)이며 초당 프레임 수는 채널당 약 8.9프레임/초에 해당되는 성능이다. 이것은 8.9프레임/초 보다 더 좋은 성능을 얻기 위해서는 단일운영체제를 가지고 불가능함을 의미한다. 하지만 제안된 매핑방법처럼 TM 스케줄 정책을 가진 4-채널 인코딩 태스크를 가급적 여러 프로세서에서 동시에 수행하도록 한다면, 채널당 최대 19.0프레임/초를 처리할 수 있고, 이전 기법보다 약 2.1배의 성능향상을 보인 결과이다. 표 4의 7, 9, 10은 ARM720T, ARM-926ej-s, 그리고 ARM1020T를 나타내는 약어이고, 괄호안의 숫자는 사용된 프로세서의 개수를 의미한다.

표 5는 관련 논문에서 구현된 다른 매핑 방법에 대해 Hou 그래프[15]를 가지고 성능을 비교한 결과이다. 시스템 성능지표는 주어진 실시간 제약-조건을 만족하면서 비용이 저렴한 결과를 내는 것이 목적이며, 사용 가능한 프로세싱 컴포넌트의 비용으로 20, 50, 100의 3가지 다른 프로세서를 적어도 두 개 이상 사용할 수 있다고 가정하였다. 제시된 논문에서 몇 개까지 사용가능한지 명확히 나와 있지 않았지만, EMOGAC의 결과가 140도 나온 것으로 보아 최소 2개 이상은 사용했을 것으로 생각되어 우리는 이 실험을 위해 세 종류 프로세서에 대해 각각 10개씩을 사용할 수 있다고 가정하였다. N/A(Not Available)로 표시된 항목은 알고리즘 개발자가 자신의 논문에 값을 표시하지 않은 경우이다.

EMOGAC이 시스템 비용면에서 가장 좋은 성능을 나타내긴 하지만 유전 알고리즘을 사용했기 때문에 결과를 얻기까지의 시간이 매우 많이 걸려 우리가 제시한 방법보다 결과가 더 좋다고 주장할 수 있지만, 실제 내장형 시스템 환경에서 사용될 수 없는 방법이다. 따라서 성능은 조금 나쁠지라도 탐색시간에서 월등한 성능을 보인 우리의 결과가 더 유용한 방법이다. 더군다나 제시된 H&W 1, H&W 2, H&W 3 태스크 그래프는 각각 10개의 노드로 구성된 아주 간단한 예제이다. 따라서 한 태스크 당 5249개, 총 20996개의 노드가 있는 4-채널 DVR 예제처럼 문제 크기가 커지면 그 수행 시간은 지

금보다 훨씬 많이 늘어날 것이다. 반면에 제안된 두 알고리즘의 탐색 시간은 아주 빠른 시간 안에 결과를 얻을 수 있다. 더욱이 유전 알고리즘의 특성상 초기 유전자 집단에 해당하는 Population을 생성시키는 임의 함수의 기초 값(Seed)에 의해 그 성능이 크게 좌우되고, 풀려는 문제마다 서로 상이한 Population을 제공해야 때문에 시스템 동작 명세에서부터 프로토타이핑까지 연결된 통합설계 환경 내에 실제로 적용하기 매우 어렵다.

Hou 그래프는 각 노드가 자신의 시간 제약-조건을 갖는 모델인 반면, 우리의 태스크 모델은 노드의 시간 제약-조건은 자신이 속한 태스크로부터 상속 받기 때문에 서로 다르다. 이를 위해 Hou 그래프의 노드 중 가장 짧은 시간 제약-조건을 우리 태스크 모델의 전체 시간 제약-조건 값으로 삼았다. 만일 Hou 그래프 중 가장 느슨한 시간 제약-조건을 채택할 경우, '매핑전용' 기법으로 비용이 100을 가진 실행 가능한 스케줄도 얻을 수가 있었다.

6. 결 론

이 논문에서는 실시간 제약-조건을 갖는 다중태스크를 다중코어 SoC 환경에서 어떻게 태스크를 매핑 해야 좋은 성능을 얻을 수 있을 것인가에 대한 알고리즘을 개선해 이전 연구보다 더 좋은 결과를 얻었다. 이전의 연구는 각 태스크를 매핑 할 때 전체 시스템 자원을 모두 독점한 반면, 제안된 알고리즘들은 각 태스크들을 서로 다른 프로세서로 병렬화 시키고, TM 기법으로 스케줄-가능성을 분석함으로써 4-채널 DVR 예제의 경우 성능을 2.1배 향상시켰고, 또한 통신량이 많은 예제의 효과적인 매핑을 위해 각 태스크를 하나의 프로세서로 전용하여 매핑 하는 방법을 추가로 제시하여 더 많은 이득을 얻을 수 있음을 보였다. 제시된 알고리즘들은 자원공유를 적용하고 있으며, 여러 태스크가 하나의 응용 모드를 갖는 다중모드에 대해서도 스케줄-가능성 분석 기능을 제공하고 있다. 이런 결과는 하드웨어를 포함한 현재 사용가능한 프로세서 자원을 가지고 목적하고자하는 시스템의 성능을 예측 및 프로세서 구조를 탐색할 수 있는 충분한 판단근거로 제시될 수 있으며, 더욱이 다중코어 SoC를 설계할 때 용이하게 쓰일 것이다.

앞으로의 남은 과제는 여러 태스크가 하나의 응용 모드를 갖는 다중모드에 대해서 모드별로 최적화된 태스크 매핑 기법을 제안하는 일이 수행될 것이다.

이 논문에서 제안된 모든 알고리즘은 통합설계 환경인 PeaCE[18] 위에서 구현되어 실험하였다.

참 고 문 헌

[1] H. Oh and S. Ha, "A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessor Scheduling," CODES, Rome, Italy, May, 1999.

[2] H. Oh and S. Ha, "Hardware-Software Cosynthesis of Multi-Mode Multi-task Embedded Systems with Real-Time Constraints," CODES, Estes Park, Colorado, USA, May, 2002.

[3] Sungchan Kim, Chaeseok Im, and Soonhoi Ha, "Efficient Exploration of On-Chip Bus Architectures and Memory Allocation," CODES+ ISSS, PP 248-253, Sep. 2004.

[4] J. Liu and E. A. Lee, "Timed Multitasking for Real-Time Embedded Software," IEEE Control Systems Magazine, Vol.23, Issue 1. Feb. 2003, pp 65-75.

[5] R. P. Dick and N. K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware- Software Cosynthesis of Distributed Embedded Systems," IEEE Transaction on CAD, vol. 17, no. 10, pp. 920-935, Oct 1998.

[6] R. P. Dick and N. K. Jha, "Multiobjective Synthesis of Low-Power Real-Time Distributed Embedded Systems," A Dissertation of Dept. of Electronical Engineering, Princeton Uinv. 2002.

[7] B. P. Dave, G. Lakshminarayana and N. K. Jha, "COSYN: Hardware-Software Co-synthesis of Embedded Systems," in Proc. Design Automation Conference, pp. 703-708, June 1997.

[8] J. Axelsson, "A Hardware/Software Codesign Approach to System-Level Design of Real-Time Application," SNART97, August 21-22, 1997.

[9] P. Pop, P. Eles, Z. Peng, and V. Izosimov, "Schedulability-Driven Partitioning and Mapping for Multi-Cluster Real-Time Systems," Euromicron 2004.

[10] Y-T, Leung, J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real- Time Tasks," Performance Evaluation, 2(4): 237-250, Dec. 1982.

[11] K. Tindell, and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," Microprocessing and Microprogrammig, Vol. 50, p. 117-134 April 1994.

[12] J. C. Palencia, and M. Gonzalez Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," Real-Time System Symposium, 1998, Page(s):26-37.

[13] F. Slomka, J. Zant, and L. Lambert, "Schedulability Analysis of Performance Message Sequence Chart," CODES/CASHE, March 1998, Page(s): 91-95.

[14] T. Pop, P. Eles, and Z. Peng, "Schedulability Analysis for Distributed Heterogeneous Time/Event Triggered Real-Time Systems," ECRTS'03.

[15] J. Hou and W. Wolf, "Process Partitioning for Distributed Embedded Systems," CODES/CASHE. pp.70-76, Mar. 1996.

[16] J. Axelsson, "A Hardware/Software Codesign Approach to System-Level Design of Real-Time Application," SNART97, August 21-22, 1997.
[17] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing," IEEE Transactions on Computers, January, 1987.
[18] PeaCE: Ptolemy extension as Codesign Environment, http://peace.snu.ac.kr

이 춘 승

2000년 2월 동국대학교 컴퓨터공학 학사. 2002년 2월 서울대학교 컴퓨터공학 석사. 2002년 3월~현재 서울대학교 전기,컴퓨터공학부 박사과정. 관심분야는 하드웨어-소프트웨어 통합설계, 시스템-수준 설계, 하드웨어-소프트웨어 분할

하 순 회

1985년 2월 서울대학교 전자공학과 학사 1987년 2월 서울대학교 전자공학과 석사 1992년 미국 U. C. Berkely 전기컴퓨터공학과 박사. 1993년~1994년 현대전자 근무 1994년~현재 서울대학교 전기, 컴퓨터공학부 교수. 관심분야는 하드웨어-소프트웨어 통합설계, 내장형 시스템을 위한 설계 방법론