DOI: 10.3745/KIPSTD.2011.18D.1.045

임베디드 소프트웨어의 설계모델로부터 에너지 효율을 향상시키기 위한 태스크 도출

홍 장 의[†] · 김 두 환[††]

요 약

저전력을 소모하는 임베디드 시스템 개발의 중요성이 증대되고 있다. 저전력 소모의 시스템 개발은 하드웨어 측면에서 많이 연구되어 왔지만, 소프트웨어의 동작이 하드웨어 전력 소모를 유발하기 때문에 소프트웨어의 소모 전력을 분석하는 것 또한 중요한 이슈중의 하나이다. 본 논문에서는 임베디드 소프트웨어 개발 과정에서 작성되는 설계 모델을 이용하여 에너지 효율적인 태스크 도출 방법을 제시한다. 이를 위하여 먼저 태스크 분할 기준을 제시하고, 이를 이용한 UML 설계 모델의 분할 과정을 설명한다. 제안된 태스크 도출 방법은 성능과 함께 전력 소모에 많은 영향을 미치는 임베디드 소프트웨어 개발에 활용하여 선행적으로 에너지 소모량을 절감할 기회를 제공할 수 있다.

키워드 : 임베디드 소프트웨어, 태스크 식별, 소모전력, UML 설계 모델

Task Extraction from Software Design Models to Improve Energy Efficiency of Embedded Software

Jang-Eui Hong[†] · Doo-Hwan Kim[††]

ABSTRACT

The importance of low-power embedded system is being increased. The studies on low-power system have been performed in issues of hardware architecture and operating system. However because the behaviors of software control the working of hardware devices, the power analysis of software is one of critical issues in energy-efficient embedded system development. This paper proposes a technique to extract tasks from software design models with considering power consumption. We first define the criteria for task extraction, and then propose the way to separate out the task from UML 2.0 design models. Our technique can provide the chance to reduce the power consumption as well as to fulfill the performance requirement in the early phase of software development.

Keywords : Embedded Software, Task Extraction, Energy Consumption, UML Design Model

1. 서 론

임베디드 시스템의 응용 영역이 확산됨에 따라, 내장되는 소프트웨어의 기능 또한 복잡도가 증가하고 있다. 이러한 상황에서 휴대성이 강조되는 스마트 폰, PMP, MP3 플레이어와 같은 임베디드 시스템의 경우는 배터리를 통해 운용되기 때문에 제한된 용량 내에서 시스템의 구동 시간을 연장하는 것이 중요하다.

이러한 요구에 따라 임베디드 시스템의 저전력 소모를 위한 연구들이 다양한 관점에서 수행되고 있다. 초기에는 하드웨어 자체의 소모 전력을 줄여 전체 시스템의 소모 전력을 절감하는 방법으로 연구되어 왔으나, 최근에는 내장되는 소프트웨어의 소모 전력을 감소시키고자 하는 연구들도 활발히 진행되고 있다. 이는 하드웨어 플랫폼의 구성 자체는 크게 변하지 않는 반면 하드웨어의 동작을 제어하는 소프트웨어의 규모가 커지고, 기능 및 구조가 복잡해지기 때문이다. 또한 하드웨어 형상의 수정 및 변경을 통해 전력 소모를 절감하는 방법에 비해 소프트웨어 관점에서의 변경을 통해 소모 전력을 절감하는 것이 수정 용이성이나 절감 효과

※ 이 논문은 2009학년도 충북대학교 학술연구지원사업의 연구비 지원에 의해 연구되었음.
† 종신회원 : 충북대학교 컴퓨터공학 부교수(교신저자)
†† 준 회 원 : 충북대학교 컴퓨터과학과 박사과정
논문접수 : 2010년 10월 6일
수 정 일 : 1차 2010년 12월 20일
심사완료 : 2010년 12월 25일

측면에서 이점을 제공하기 때문이다[1].

임베디드 소프트웨어의 소모 전력 분석에 대한 연구들은 분석 대상에 따라 소스코드 기반의 분석기법, 모델 기반의 분석 기법 등으로 분류할 수 있다. 소스코드 기반의 분석기법[2,3,4]은 특정 하드웨어 플랫폼 상에서 각각의 명령어나 프로그램 코드가 소모하는 전력을 계측하고, 이를 기반으로 소모 전력량을 산출하는 방법 등을 통해 수행되었다. 그러나 최근 임베디드 소프트웨어의 개발에 있어서 요구사항의 분석 및 설계 과정을 통하여 소프트웨어의 품질을 높이고, 각 기능을 수행하는 모듈의 재사용성을 증대시키는 모델기반의 개발방법이 요구됨에 따라 코드 생성 이전 단계에서 소모 전력을 예측하고자 하는 모델 기반의 분석기법[5,6]이 연구되고 있다. 이러한 모델기반 분석기법의 연구는 코드기반의 분석방법이 분석 소요시간이 길고, 결과의 피드백을 위한 노력이 상대적으로 크다는 것에 기인하며[1], 또한 모델 기반의 분석 방법이 별도의 소모 전력 분석 프레임워크 없이 소프트웨어 개발 과정에서 작성되는 산출물을 이용하여 분석할 수 있다는 장점이 있다.

본 연구에서는 이러한 추세에 따라 모델 기반의 임베디드 소프트웨어 개발 과정에서 작성되는 UML 다이어그램을 이용한 소모전력 절감기법을 제안한다. 특히 멀티 프로세서(Multi-Processor) 플랫폼을 사용하는 지능형 서비스 중심의 임베디드 소프트웨어 개발에서, 설계 모델로부터 에너지 효율적인 태스크를 도출하기 위한 기법을 제안한다. 이러한 방법은 소프트웨어 설계 과정에서 태스크를 분할하고 분할된 태스크들을 프로세서에 할당하기 위한 전략에 활용될 수 있다.

논문의 구성은 다음과 같다. 2장에서는 기존에 소프트웨어 태스크 분할 기법에 대한 관련 연구들을 살펴보고, 3장에서 에너지 효율적인 태스크를 도출하기 위한 분할 기준을 제시하였다. 4장에서는 태스크 분할 절차와 함께 예제 시스템에서의 적용을 설명하고, 5장에서는 시뮬레이션 기반의 실험을 통해 제시한 방법의 유용성을 확인하고, 6장에서는 결론 및 향후 연구 내용을 제안한다.

2. 관련 연구

멀티프로세서 아키텍처 환경에서 소프트웨어 태스크를 처리하기 위해서는 사전에 태스크를 식별하고, 식별된 태스크를 프로세서에 할당하는 과정이 필요하다. 기존의 많은 연구들이 태스크를 프로세서에 할당하고 스케쥴링하는 과정에서 에너지 효율을 고려하였다. 그러나 본 연구에서는 태스크를 식별하는 과정에서 에너지 효율을 고려하는 방법에 대하여 제시하고 있으며, 이와 직접적으로 관련된 연구들은 좀처럼 찾기 어려운 상황이다.

기존의 많은 연구들이 주어진 태스크들을 프로세서에 할당하고 스케쥴링하는 연구들을 수행해 왔다. 대표적인 연구들은 Schmitz[7], Shin[8], Cong[9] 등의 연구가 있다. Schmitz[7]는 동적으로 전압을 변경할 수 있는(DVS, Dynamic Voltage Scalable) 프로세서 아키텍처에 주어진 태스크들을 에너지 효율적으로 스케쥴링하고, 프로세서에 매핑(mapping)하기 위한 알고리즘을 제시하였다. 이를 위하여 특히 각 태스크의 실행 시간에 대한 슬랙 타임(slack time)과 프로세서의 소모전력 프로파일을 고려하여 태스크를 할당하는 방법을 제안하였다. Shin의 연구[8]도 Schmitz의 연구와 유사하게 DVS 기반의 멀티프로세서 시스템에서 소모전력을 고려하는 태스크 스케쥴링 알고리즘을 제시하였다. 이 연구에서는 특히 복잡한 태스크의 우선순위를 고려하기 위하여 CTG(Conditional Task Graph)를 작성하고, 그래프 분할을 통하여 태스크들을 프로세서에 할당하는 알고리즘을 제시하였다.

Chatzigiannakis[10]는 서로 다른 처리 속도를 갖는 멀티프로세스 아키텍처에서 태스크간의 의존성을 고려하는 태스크 스케쥴링 기법을 제시하였다. 이때 프로세스의 처리속도에 따른 태스크의 재할당을 통해 에너지를 절감할 수 있도록 하였다. 최근에 연구된 태스크 스케쥴링 알고리즘은 DVS 기반의 멀티프로세서 아키텍처에서 태스크의 실행 시간에 영향을 주는 입력 의존적 변수와 자원의 제약사항, 그리고 입력 지연 등의 특성을 반영하여 제시된 Cong의 연구[9]이다. 이러한 연구들은 모두 에너지 절감을 위한 목적을 가지고 수행되었으나, 태스크의 식별단계가 아닌 주어진 태스크의 스케쥴링 과정에서 태스크를 프로세서에 할당하는 과정에서의 연구들이다.

에너지를 고려한 태스크 식별 기법에 대한 연구는 2009년에 Goraczko[11]에 의해 수행되었다. 이 연구에서는 이종의 멀티프로세서 아키텍처에서 실행시점의 모드(mode) 스위칭에 의한 시간과 에너지 비용을 절감하기 위한 자원 모델을 제시하였다. 특히 애플리케이션을 기능 중심의 태스크 그래프로 표현하고, 그래프의 각 태스크들을 선형 프로그래밍 기법에 의하여 자원에 할당한다. 비록 이 연구에서 주어진 기능(그래프의 노드)을 프로세서에 할당할 때, 태스크의 병합 및 분리 등을 고려하고 있지만, 본 논문에서 제시하는 애플리케이션의 설계모델로부터 태스크를 분할하는 기법과는 차이가 있는 연구이다. 일반적으로 소프트웨어 설계모델로부터 태스크를 식별하는 연구는 전통적으로 Page-Jones[12]와 Gomaa[13]에 의해 제시되었다. 그리고 멀티프로세서 아키텍처를 위한 태스크 식별 기법에 대한 연구도 Jeon[14]에 의해 수행되었다. 그러나 이러한 연구들은 소프트웨어의 태스크 식별 과정에서 에너지 소모에 대한 특성을 고려하지는 않았다.

3. 태스크 도출을 위한 에너지 인자

3.1 전력 소모에 영향을 주는 요소

둘 이상의 프로세서가 존재하는 멀티프로세서 환경의 임베디드 시스템에서는 태스크의 분할과 할당 과정에 사용되는 방법에 따라 시스템의 전반적인 성능에 영향을 준다. 이러한 시스템의 성능은 태스크 수행 시간 관점에서 고려되어야 하지만, 임베디드 시스템의 경우는 수행시간과 함께 소모 전력에 대한 고려도 함께 이루어져야 한다. 멀티프로세서 아키텍처에서 프로세서 사용률을 최대로 하여 병렬 처리하는 경우, 수행 시간은 빨라질 수 있으나, 소모전력 측면에서는 나빠질 수 있다. 따라서 수행시간 즉, 데드라인(deadline)을 만족시키는 범위에서 프로세서의 부하를 균등화하는 것이 소모 전력을 감소시키기 위한 전략이 될 수 있다. 일반적으로 소프트웨어 시스템의 전력에 영향을 주는 요소들은 다음과 같이 구분할 수 있다[15].

1) 실행시간 : 실행 시간이 크다는 것은 소모 전력이 많음을 의미하는 것이기 때문에 가능한 빠른 시간내에 또는 주어진 데드라인내에 태스크가 처리될 수 있어야 한다.
2) 태스크 상호작용 : 태스크의 상호작용에 의한 소모전력은 (1) 태스크간의 메시지 전달의 횟수와 (2) 전달되는 매개변수(parameter)의 크기에 의존적이다. 이들은 상호작용 방법에 따라 메모리 접근을 유발하기 때문에 태스크간에 최소한의 상호작용을 유지하는 것이 필요하다.
3) 태스크 병렬성 : 태스크가 병렬 수행되면, 실행시간의 단축으로 소모전력이 감소될 수 있으나, 동작하는 프로세서의 수가 증가하고, 부가적인 상호작용 오버헤드가 늘어날 수 있다. 따라서 적절한 병렬성을 유지하면서 실행 시간을 단축하는 것이 요구된다.

3.2 태스크 식별 기준

3.1절에서 설명한 실행시간, 태스크 상호작용, 그리고 태스크 병렬성 요소중에서 실행시간은 태스크 상호작용 축소 및 병렬성이 향상되면 자연히 줄어들게 된다. 따라서 본 논문에서는 태스크 상호작용의 축소와 태스크 병렬성 측면에서 UML[16] 모델 기반의 태스크 식별 기준을 정의하였다.

[기준 1. 메시지 전송 데이터 크기] 시퀀스(Sequence) 다이어그램에 나타나는 두 객체 Ti와 Oi가 상호작용할 때, 전달되는 메시지의 매개변수 사이즈에 따라 상호작용의 오버헤드가 달라진다.

[1.1] 객체간의 메시지 전송에서 매개변수가 기초 데이터 타입으로 정의된 경우, 두 객체는 별도의 독립된 태스크로 식별한다. 이는 성능을 고려하는 병렬성 향상을 위한 것이다.

[1.2] 메시지 전송의 매개변수가 복합 데이터 타입을 갖는 경우, 전송되는 메시지의 크기에 따라 두 객체를 통합하여 하나의 태스크로, 또는 분리하여 두 개의 태스크로 정의한다. 이때 분리 기준은 메시지 전송에 따른 소모전력이 매개변수의 메모리 접근에 따른 소모전력보다 작아진다면 분리하게 된다.

<표 1>은 하나의 매개변수 전달에 대한 메시지 타입별 소모전력과 메시지를 송수신하는 경우의 소모 전력을 나타낸다. 소모전력의 산출은 StrongARM 기반 SA1100 마이크로 프로세서에서 수행되는 EMSIM 시뮬레이터[17]를 이용하여 산출하였다.

〈표 1〉 메시지 전달 방식에 따른 소모전력

유형	메시지 타입	메시지 크기(bytes)	소모 전력(nJ)
메모리 접근	단일 정수형	4	14.4
	정수형 배열	40	149.2
메시지 전달	메시지 전달	C	4.0C+4554
	메시지 수신	C	4.4C+4810

<표 1>과 같이 메모리 접근에 의한 매개변수 전달의 소모전력은 배열의 경우 149.2nJ 이지만, 메시지 전송의 형태를 취하는 경우는 대략 62배 이상의 소모전력이 요구된다. 따라서 <표 1>에 제시된 결과에 따라 대략 메시지의 크기가 1,418Bytes 이상이면 메시지 전송을 위해 요구되는 소모전력이 높아지기 때문에 두 객체를 하나의 태스크로 정의하는 것이 유리하다.

[기준 2. 제어 병렬성] 시퀀스 다이어그램이나 IOD 다이어그램에는 처리 흐름의 병렬성을 나타내는 연산이 있다.

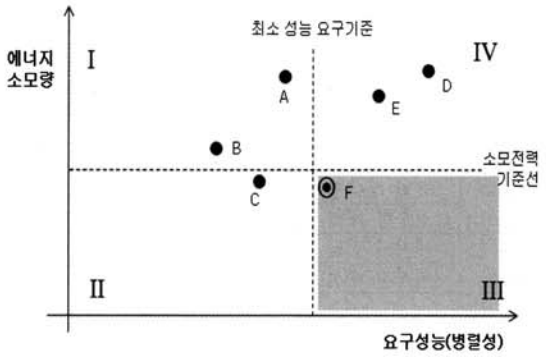
[2.1] 시퀀스 다이어그램에 나타나는 "par" Combined Fragment에 걸쳐진 객체들은 하나의 태스크로 통합한다.

[2.2] 통합된 태스크에 대하여 "par" Combined Fragment 내부의 연산들을 병렬 수행이 가능하도록 다른 태스크로 분할한다. 분할은 클래스의 멤버 함수만을 복사한 별도의 클래스로 정의된다. 이때 분할되는 태스크는 각각 50.7개 이상의 연산이 포함되어야 한다.

[2.3] IOD 다이어그램에 나타나는 Fork/Join 연산에 대하여 생성되는 프로세스의 수만큼 별도의 태스크로 분리한다. IOD에 나타난 Fork()에 의해 생성되는 프로세스는 하나의 시퀀스 다이어그램으로 표현되며, 이때 시퀀스 다이어그램에는 최소 50.7개 이상의 연산이 포함되어 있어야 한다.

병렬성을 갖는 태스크를 분할함에 있어서 분할된 태스크의 사이즈가 작다면 병렬처리를 위한 프로세스 생성의 오버헤드 등으로 순차처리에 비해 소모 전력의 효과가 감소된다. 따라서 적절한 크기의 태스크가 생성될 때, 병렬 수행의 효과가 발생한다는 것이다. (그림 1)은 소프트웨어의 에너지 소모량과 병렬성을 고려하는 성능간의 관계를 그래프로 표시한 것이다. (그림 1)에서와 같이 에너지 소모량과 요구성

능에 대한 기준선을 통하여 최적의 태스크 크기를 결정하는 것이 중요하다. (그림 1)의 III 사분면은 정의된 요구 성능을 유지하면서 에너지를 절감할 있는 영역이다. 특히 III 사분면에 존재하는 F 지점에서의 태스크 크기를 찾는 것이 필요하다. II 사분면의 지점 C가 비록 에너지 소모량이 적기는 하지만 성능 요구사항을 만족하지 못하는 경우이다.



(그림 1) 소모전력과 요구성능과의 관계

다시 말해서, (그림 1)에 표현된 F 지점을 찾기 위해서, 동일한 크기의 태스크에 대하여 병렬 수행하는 경우와 순차 수행하는 경우의 에너지 소모량에 대한 수식은 다음과 같이 정의할 수 있다. 여기서 Pi (i=1..n)는 하나의 태스크가 소모하는 전력량이고, 함수 f와 w는 각각 fork()와 join() 연산의 소모 전력이다.

$$(P_1+P_2+..+P_n)/n+(n-1)f+(n-1)w<P_1+P_2+..P_n \quad (1)$$

식 (1)에서 모든 Pi 가 동일한 크기라고 가정하면, 식 (1)은 식 (2)와 같이 표현할 수 있다.

$$\begin{aligned} & nP_i/n+(n-1)f+(n-1)w<nP_i \\ & \Rightarrow (n-1)f+(n-1)w<P_i(n-1) \\ & \Rightarrow f+w<P_i \end{aligned} \quad (2)$$

결국 태스크의 병렬성을 찾기 위하여 한 태스크가 소모하는 전력량이 fork()와 join() 연산에서 소모하는 전력량과 같아지는 태스크의 크기를 결정하면 된다. 임베디드 리눅스에서 제공하는 fork()와 waitPID() 시스템 함수에 대한 알고리즘 - join() 함수는 waitPID 함수로 대체 사용됨 -을 입력으로 EMSIM 2.0[17]에서의 소모전력 시뮬레이션 값은 각각 132202.6nJ과 25077.3nJ이며, 이들의 합은 157279.9 nJ이다. 따라서 태스크가 이에 해당되는 소모 전력을 갖는 크기를 식별할 수 있다.

이를 위하여 일반적인 사칙연산을 수행하는 임의의 C 프로그램을 작성하여 소모전력이 157988.9 nJ이 되는 태스크를 생성하였으며, 이때 태스크가 갖는 어셈블리 명령어수는 6015개, 소스코드 수준에서 1014개의 명령어로 구성되었다. 모델 수준으로 추상화하는 경우 평균적으로 객체간의 상호작용과 멤버함수의 로직을 표현하는 Execution Occurrence를 포함하여 약 50.7개 정도의 연산이 포함될 때 태스크를 분할하는 것이 좋다. 이는 UML 다이어그램의 한 모델 요소가 대략 C 언어로 구현될 때, 평균적으로 20개의 소스코드 명령어로 구현됨에 따른 것이다.

[기준 3. 데이터 독립성] 시퀀스 다이어그램의 수행 흐름상에서 명시적으로 병렬 처리를 나타내지 않지만, 병렬 수행이 가능한 부분을 식별한다.

[3.1] 시퀀스 다이어그램에 나타나는 "loop" Combined Fragment에 걸쳐진 객체들중에서 loop 행위를 제어하는 핵심 객체를 하나의 태스크로 정의한다.

[3.2] 정의된 태스크를 기준으로 "loop" Combined Fragment에서 루프내의 처리 로직에 사용되는 데이터가 각 루프의 실행에 영향을 주지 않는 독립성을 갖는다면 해당 루프는 적절하게 병렬 수행으로 분리되어 별도의 태스크로 식별 가능하다. 별도의 태스크로 분리되는 경우, 기준 [2.2]와 같이 복사된 클론(clone)을 생성한다.

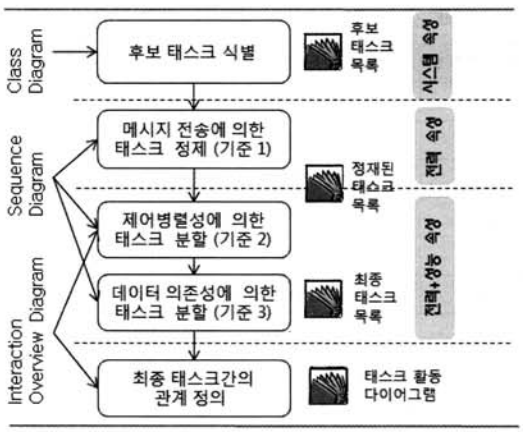
기준 [3.1]에 제시된 처리 로직간의 데이터 의존성에 대한 연구는 Kyriskopoulos[18]와 Jacobson[19]에 의해 이미 연구되었다. 따라서 루프(loop)의 반복 횟수를 적절하게 분리하여 병렬화 시킨다. 즉, 루프가 for (k=1; k=60; k++)일 때, for (k1=1; k1=20; k1++), for (k2=21; k2=40; k2++), 그리고 for (k3=41; k3=60; k3++) 와 같이 3개의 병렬 루프로 분리하여 수행하면, 보다 우수한 성능을 보일 수 있다. 다만 이때 분할된 한 개의 병렬 루프내에는 기준 3에서 제시한 조건을 만족시켜야 한다.

4. 소모 전력을 고려한 태스크 식별

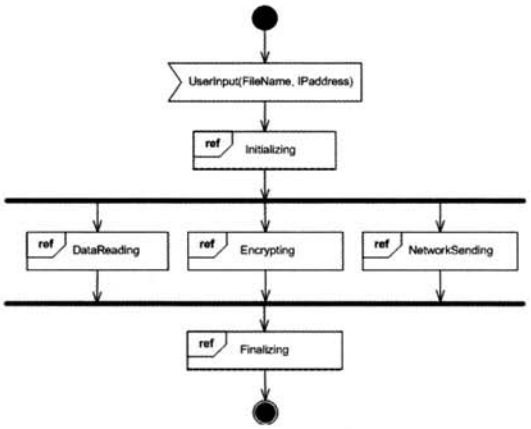
앞서 3.2절에서 정의한 기준에 근거하여 임베디드 소프트웨어의 태스크 식별 절차를 정의하고, 예제를 통해 식별 과정을 설명한다.

4.1 태스크 식별 절차

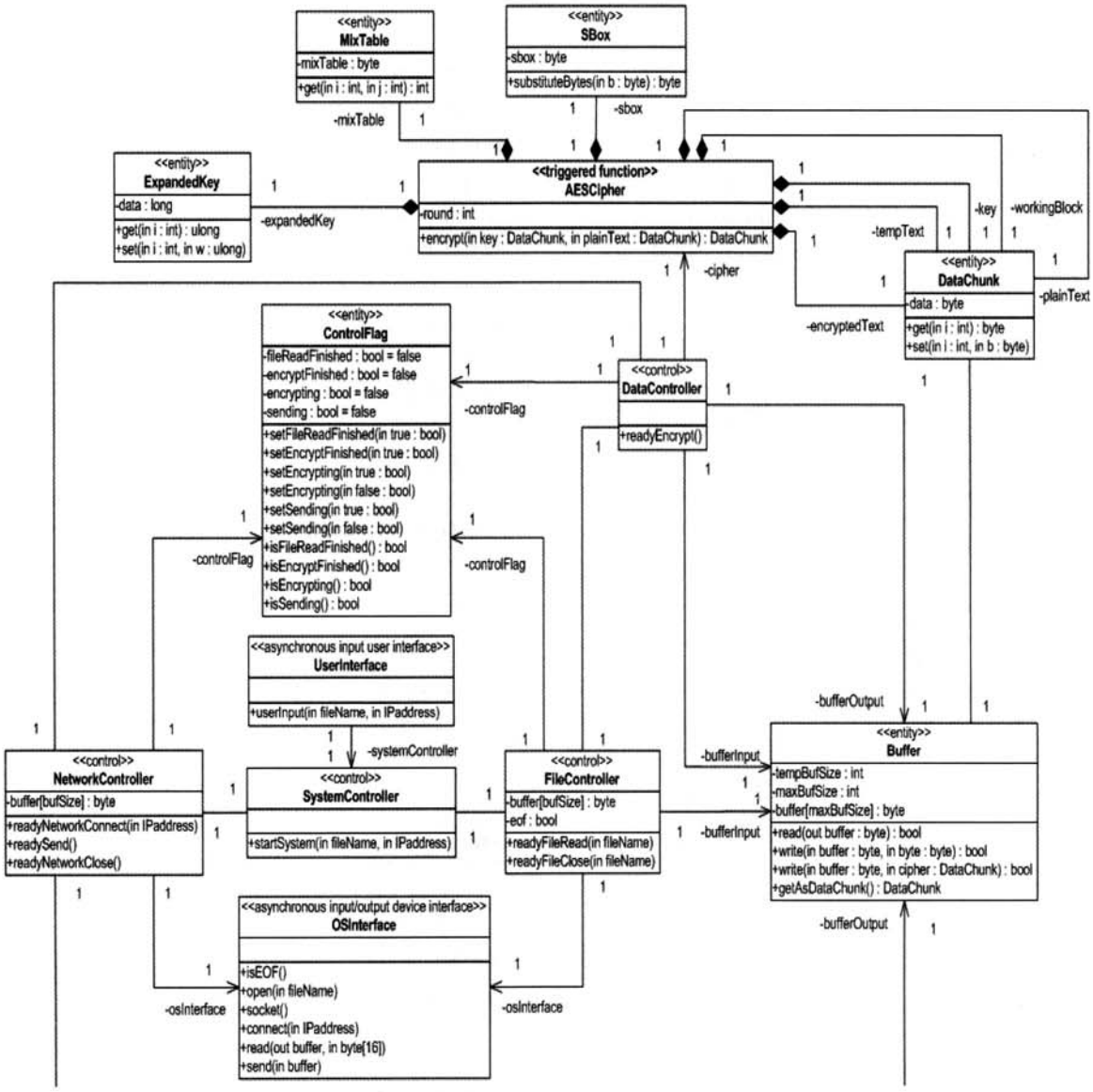
태스크를 식별하기 위한 기본 입력 모델은 설계단계에서의 클래스 다이어그램이다. 클래스 다이어그램에 정의된 클래스의 타입을 기준으로 후보 태스크를 식별한다. 즉, 클래스의 스테레오타입이 <<Control>>, <<Asynchronous>>, 그리고 <<Periodic>>으로 정의된 경우, 객체가 자율성을 갖는 액티브(active) 태스크로 구현되기 때문에 이들은 후보 태스크가 된다. (그림 2)는 태스크 식별 절차를 도식화 한 것이다.



(그림 2) 소모전력을 고려하는 태스크 식별 절차



(그림 3) SFT 시스템의 최상위 모델



(그림 4) SFT 시스템의 클래스 다이어그램

(그림 2)와 같이 태스크 식별을 위한 기본 입력은 클래스 다이어그램이다. 클래스 다이어그램을 입력으로 시퀀스 다이어그램에 나타난 객체간의 상호 작용에 대한 정보를 이용하여 클래스를 통합하거나 분할한다. 또한 IOD가 갖는 제어 정보를 이용하여 클래스간의 병렬성 정보를 이용한다. 전체적으로 태스크 식별을 위한 과정은 3장에서 제시한 기준을 순차적으로 적용하면서 이루어진다. 기준 1에서는 객체간의 상호작용에 발생하는 메시지의 크기에 따른 전력 속성 반영하는 태스크 식별이며, 기준 2와 기준 3에서는 전력 속성과 성능(병렬성)의 두 가지 측면을 고려하는 태스크 분할 기준을 적용하는 과정이다.

태스크를 도출하는 과정을 통한 산출물은 먼저 식별된 후보태스크의 목록과, 기준의 적용에 따른 최종 태스크 목록이며, 이들은 IOD의 구조에 따라 태스크 활동(Activity) 다이어그램의 형태로 표현된다.

〈표 2〉 예제 시스템의 후보 태스크 목록

태스크 ID	태스크 명	타입	비고
AT1	SystemController	control	Active tasks
AT2	FileController	control	
AT3	DataController	control	
AT4	NetworkController	control	
AT5	OSInterface	asyn.	
AT6	UserInterface	asyn.	
ET1	Buffer	entity	Entity
ET2	ControlFlag	entity	
ET3	DataChunk	entity	
ET4	ExpendedKey	entity	
ET5	SBox	entity	
ET6	MixTable	entity	

4.2 예제 시스템의 태스크 분할

3장에서 정의된 태스크 식별 기준에 의거하여 SFT (Secure File Transfer) 시스템에 대한 태스크 도출 과정을 설명한다. SFT 시스템은 특정한 사용자에게 안전한 파일 전송을 위하여 FIPS-PUB-197 AES(Advanced Encryption Standard)[20]에 정의된 알고리즘을 이용하여 파일을 암호화 한다. (그림 3)은 SFT 시스템의 최상위 IOD이며, (그림 4)는 클래스 다이어그램이다.

4.2.1 후보 태스크 추출

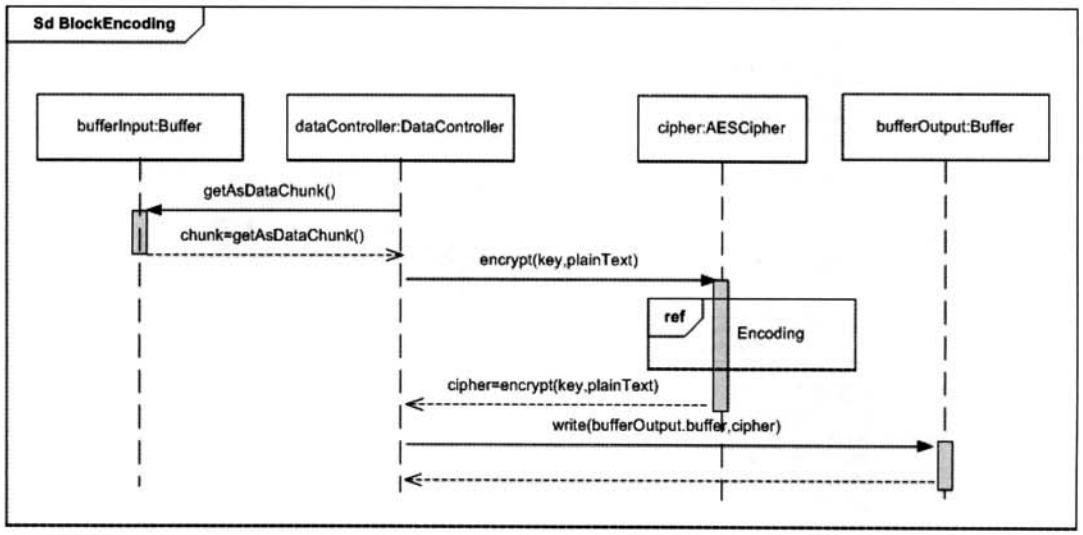
먼저, 클래스가 갖는 스테레오타입을 기준으로 예제 시스템에 대하여 식별된 후보 태스크는 <표 2>와 같다. (그림 4)에 나타난 <<entity>> 타입의 클래스는 추후 메모리로 저장되는 데이터에 해당되기 때문에 별로 분리 가능하다.

4.2.2 객체 상호작용에 의한 태스크 정제

두개의 객체가 상호작용할 때 발생하는 메시지 전송에 따라 태스크를 통합할 수 있다. 본 논문에서는 SFT 시스템에서 파일 암호화를 위한 인코딩 과정의 시퀀스 다이어그램 (그림 5)을 대상으로 태스크 정제 과정을 설명한다.

(그림 6)에 의하면 두 개의 객체 "dataController"와 "ciper" 사이에 데이터를 Encrypt 하기 위한 메시지를 주고받는다. 이 메시지는 상위의 시퀀스 다이어그램에서 파일을 모두 읽을 때 까지 반복적으로 수행되는 루프안에 포함되어 있다. 따라서 대략적으로 400KB 이상의 이미지나 파일을 반복 처리하는 이 메시지에 의하여 기준 [1.2]를 적용하여 두 객체는 통합되어야 한다.

또한 (그림 5)에 나타난 객체 "ciper"는 Reference Fragment "Encoding" 과정에서 (그림 4)에 나타난 4개의 엔티티 클래스



(그림 5) SFT 암호화를 위한 인코딩 과정의 시퀀스 다이어그램

"DataChunk," "ExpendedKey," "SBox," 그리고 "MixTable"와 상호작용을 한다. 비록 적은 양의 메시지가 전송되지만, 서로 다른 메모리로 접근하는 방지하기 위하여 4개의 엔티티 클래스는 하나의 태스크로 병합하여 하나의 메모리로 할당될 수 있도록 한다. 이에 대한 태스크 식별 결과는 <표 3>과 같다.

〈표 3〉 예제 시스템의 후보 태스크 목록(기준 1)

태스크 ID	태스크 명	병합 객체
AT1	SystemController	-
AT2	FileController	-
AT3	DataController	AESCipher
AT4	NetworkController	-
AT5	OSInterface	-
AT6	UserInterface	-
ET1	Buffer	-
ET2	ControlFlag	-
ET3	DataChunk	ExpendedKey, SBox, MixTable

4.2.3 제어 병렬성에 의한 태스크 정제

3.2절에 제시한 [기준 2]의 제어병렬성 기준에 따라 "par" Fragment의 행위는 분리되어 수행될 수 있다. 따라서 (그림 6)에 제시된 시퀀스 다이어그램은 PlainText1과 PlainText2에 대한 인코딩 과정을 병렬로 수행하도록 모델링되었다.

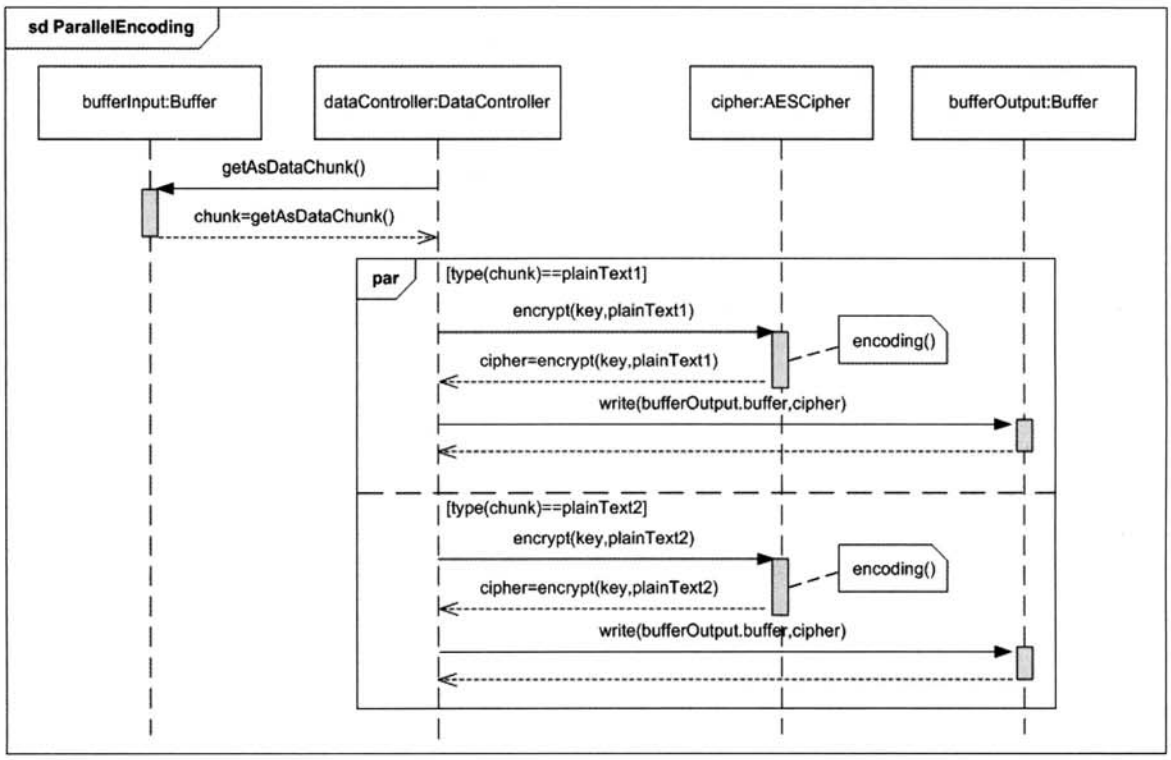
(그림 6)의 "par" Fragment에 포함된 객체 "dataController"와 "ciper"는 [기준 2]에 따라 하나의 태스크로 통합된 후, 병렬 태스크로 분리된다. 객체간의 상호작용 역시 encoding()에 대한 행위를 포함하고 있으며, 상위 시퀀스 다이어그램에서 루프 구조내에 포함되는 부분이기 때문에 [기준 2.3]에 제시한 50.7개 이상의 연산을 포함하고 있다. 이와 같은 기준의 적용에 따라 정제된 태스크 목록은 <표 4>와 같다.

〈표 4〉 예제 시스템의 태스크 목록(기준 2)

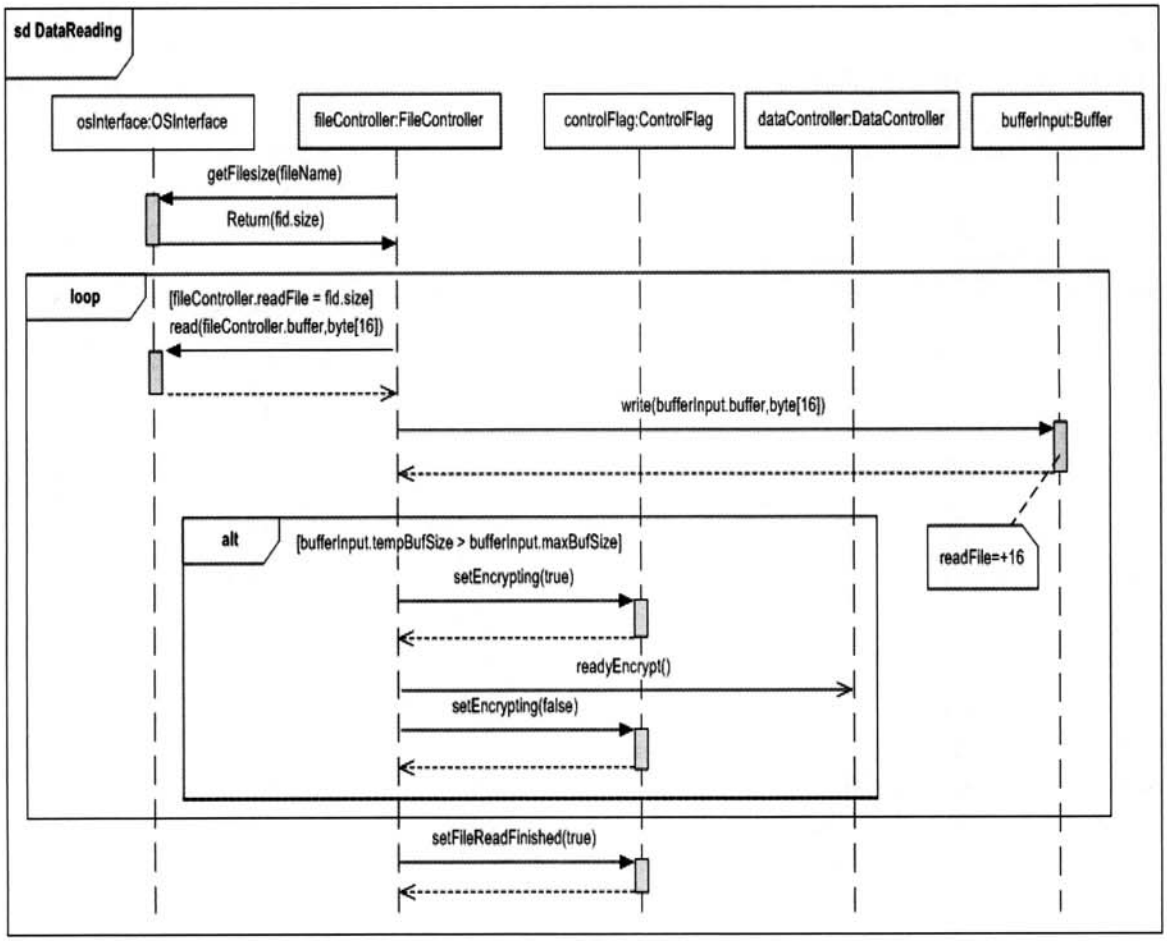
태스크 ID	태스크 명	병렬 객체
AT1	SystemController	-
AT2	FileController	-
AT3	0: DataCntl_Ciper	1: DCnCiper_pText1 2: DCnCiper_pText2
AT4	NetworkController	-
AT5	OSInterface	-
AT6	UserInterface	-
ET1	Buffer	-
ET2	ControlFlag	-
ET3	DataChunks	-

4.2.4 데이터 병렬성에 의한 태스크 정제

데이터의 처리에 대한 선후행 연산이 독립적으로 이루어질 수 있는 경우, [기준 3]에 근거하여 병렬 태스크로 분리



(그림 6) "par" Fragment를 갖는 시퀀스 다이어그램



(그림 7) 데이터 입력에 대한 루프 연산을 갖는 시퀀스 다이어그램

할 수 있다. (그림 7)은 암호화/인코딩를 위한 파일을 버퍼로 읽는 과정에 대한 시퀀스 다이어그램이다.

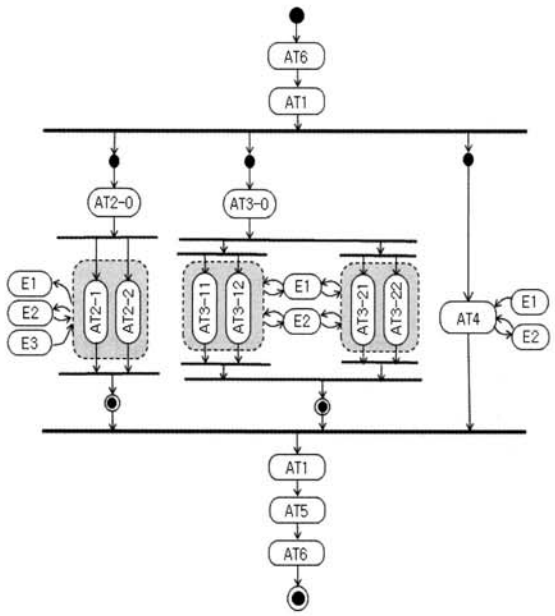
(그림 7)에 나타난 "loop" Fragment는 입력할 파일의 크기인 fid.size 변수를 N 등분하여 N개의 병렬 태스크를 생성할 수 있다. 다만 [기준 3.2]에서 제시한 바와 같이 루프내에서의 연산은 50.7개 이상이 되어야 한다. 400KB 크기의 파일에 대하여 16바이트 단위의 버퍼에 기록한다고 할 때, 25,000번의 연산이 요구된다.

제시된 SFT 시스템에서 데이터의 병렬성을 찾을 수 있는 또 하나의 부분은 인코딩 부분이다. 인코딩 알고리즘도 블록단위로 암호화 키를 이용하여 인코딩을 수행하기 때문에 이들 부분도 선후행 연산간의 데이터 의존성이 존재하지 않는다. 따라서 최종적으로 식별될 수 있는 태스크 목록은 <표 5>와 같다.

<표 5>에 나타난 SFT 시스템의 최종 태스크들에 대하여 (그림 3)의 시스템 레벨 IOD에 나타난 제어 구조 및 각 시퀀스 다이어그램에 나타난 행위를 기준으로 태스크간의 실행 관계를 나타내면 (그림 8)과 같은 태스크 활동 다이어그램을 얻을 수 있다.

〈표 5〉 예제 시스템의 최종 태스크 목록(기준 3)

태스크 ID	태스크 명	분할 / 병렬 객체
AT1	SystemController	-
AT2	0: FileController	1: FileController_1
		2: FileController_2
AT3-0	DataCntl_Ciper	-
AT3-1	DCnCiper_pText	1: DCnCiper_pText1_1
		2: DCnCiper_pText1_2
AT3-2	DCnCiper_pSound	1: DCnCiper_pText2_1
		2: DCnCiper_pText2_2
AT4	NetworkController	-
AT5	OSInterface	-
AT6	UserInterface	-
ET1	Buffer	-
ET2	ControlFlag	-
ET3	DataChunks	-



(그림 8) SFT 시스템의 태스크 활동 다이어그램

5. 실험 및 결과 분석

5.1 실험 설계

앞의 4장에서 제시한 SFT 시스템에 대한 태스크 식별 결과에 대한 유용성을 확인하기 위하여 다음과 같은 실험하였다.

- 실험은 동일한 SFT 시스템에 대하여 소모전력을 고려하는 경우와, 그렇지 않은 경우로 구분하여 태스크를 식별하였다.
- 두 가지 방법에 의해 식별된 태스크들에 대하여 각각 소모전력을 예측하고, 비교하였다.
- 실험 결과는 두 가지 측면에서 검토하였다. 첫 번째는 실행 시간만을 고려하는 경우, 두 번째는 소모전력량을 추가로 고려하는 경우로 실험하였다.

5.2 실험 환경

실험을 수행하기 위한 환경은 C 언어로 구현된 SFT 알고리즘에 대하여 EMSIM 2.0을 기반으로 시뮬레이션을 수행하였다. EMSIM 2.0은 리눅스 운영체제상에서 동작하는 소모전력 예측용 시뮬레이터로써, EMSIM에서 채택하는 하드웨어 구조는 앞서 설명한 것처럼 StronARM 기반의 SA1100 마이크로프로세서와 임베디드 리눅스 커널 2.4.x를 사용한다. 특히 실행 시간 측면에서는 리눅스 상에서의 타이머 명령어를 삽입하여 측정하였다.

특히, 태스크 실행에 대한 소모전력 예측을 위하여 식별된 태스크들이 프로세서상에 할당되어야 하는 필요성이 발생한다. 이에 대하여 다음과 같은 가정을 두었다.

- 멀티프로세서 아키텍처가 프로세스의 개수를 한정적으로 가지고 있으나, 본 논문에서는 스케쥴링에 대한 부분을 고려하지 않기 때문에 특별히 프로세서의 개수를 제한하지 않았다.
- EMSIM 2.0은 단일 프로세서를 기반으로 소모전력을 예측하는 도구이다. 따라서 소모전력 분석을 위해 단위 태스크별로 소모전력을 분석하고, 이를 리눅스상에서 병렬처리 하는 것으로 에뮬레이션 한다.

5.3 실험 데이터 준비

실행시간과 소모전력을 고려하는 실험을 수행하기 위하여 다음과 같은 두가지 실험 데이터를 준비하였다.

(1) 태스크 그룹 A: 이는 태스크 분할에서 태스크간의 응집력을 고려하는 일반적인 속성으로 태스크를 식별한 후, 태스크간의 상호 작용을 줄이는 방향으로 도출된 태스크 목록이다[14]. 식별된 최종 태스크 목록은 <표 6>과 같다.

〈표 6〉 태스크 응집력을 고려한 태스크 도출

태스크 ID	태스크 명	병합된 객체
rAT1	SystemController	none
rAT2	FileController	none
rAT3	DataController	AESCipher
rAT4	NetworkController	none
rAT5	OSInterface	none
rAT6	UserInterface	none
rET1	Buffer	-
rET2	ControlFlag	-
rET3	DataChunk	-
rET4	ExpendedKey	-
rET5	SBox	-
rET6	MixTable	-

(2) 태스크 그룹 B: 소모전력 특성을 함께 고려하는 태스크 도출 결과이며, SFT 시스템에 대하여 <표 5>와 같은 태스크를 도출하였다. 전체적으로 실행 태스크의 수는 (그림 8)에서와 같이 12개이며, 엔티티 태스크는 3개이다.

5.4 결과 분석

5.4.1 실행 시간

<표 5>와 <표6>에 주어진 태스크에 대하여 리눅스(Fedora 7) 커널 2.6.21에서 실행 시간을 분석하였다. 실행 시간은 태스크의 병렬수행 정도에 의존적이다. C언어를 이용하여 구현된 태스크 그룹 A와 그룹 B의 실행 결과는 <표

7>과 같다. 실험에서는 암호화를 위한 입력 파일의 크기에 따라 다양하게 측정되었다.

〈표 7〉 태스크 실행 시간(second) 비교

태스크 그룹	입력 파일의 크기		
	200K	400K	600K
A	0.415	0.743	1.083
B	0.381	0.704	0.889
편차(%)	8.2	5.3	17.9

<표 7>로부터 병렬성을 고려하여 태스크를 식별한 그룹 B가 실행 시간 측면에서 약 10.5%의 향상을 보였다. 입력 파일의 크기가 작은 경우는 수행 시간의 효과가 8.2%에 불과했으나, 데이터의 크기가 커짐에 따라 18%의 수행시간 향상을 보였다. 이는 많은 데이터의 병렬처리가 유용함을 보여주는 결과라 판단된다.

5.4.2 소모전력

태스크 그룹 A와 그룹 B에 대한 소모전력 분석을 위하여 SFT 소프트웨어의 코드 수정을 통하여 병렬 태스크를 생성하고, 이들에 대한 소모 전력을 EMSIM 2.0을 이용하여 분석하였다. 분석결과는 <표 8>과 같이 암호화 대상 파일의 사이즈를 고려하여 측정하였다.

〈표 8〉 SFT 시스템의 태스크 소모전력 비교

태스크 그룹	입력 파일 크기에 따른 소모전력 (mJ)		
	200K	400K	600K
A	6075.2	12150.1	18246.4
B	6076.1	12147.1	18220.4
편차	-0.9	3.0	26

<표 8>로부터 입력파일의 크기가 200K인 경우는 데이터 병렬성에 의해 처리되는 데이터의 양이 적어서 - 400K 보다 적음 - 루프의 반복처리가 작아지는 경우에 해당된다. 입력 파일의 크기가 큰 경우에 소모전력의 감소효과가 더 큰 것으로 나타났다. 600K의 파일 사이즈에 대하여 소모 전력의 효과는 26mJ 정도의 감소가 생겼으며, 이는 명령어 "add" 연산이 2,367,942번 또는 fork() 연산이 196.7번 수행하는 전력을 감소하는 효과와 같다고 할 수 있다.

5.4.3 에너지 절감 효과 확인

<표 8>에 제시한 SFT 시스템의 암호화 알고리즘에 대한 소모전력 현상을 확인하기 위하여 JPEG 2000, H.263 등을 지원하는 동영상 압축 시스템에 대한 추가 실험을 진행하였다. 태스크 분할을 위해 사용한 부분은 부호화를 위한 호프만 코딩 알고리즘[21]이다. 이 알고리즘의 모델로부터 도출된 태스크는 그룹 A의 경우, Hoff_coding, HC_tree1, HC_tree2의 3개 태스크이고, 그룹 B의 경우는 Hoff_coding, HC_tree11, HC_tree12, HC_tree21, 그리고 HC_tree22의 5개 태스크로 도출되었다. 실험을 위해 사용된 데이터는 600K이며, 이는 120 프레임 정도의 영상 이미지를 갖는 데이터이다. 도출된 태스크들에 대하여 시뮬레이션된 소모전력은 <표 9>와 같다.

〈표 9〉 호프만 코딩의 태스크 소모전력 비교

입력 크기	태스크 그룹에 따른 소모전력(mJ)		
	단일 태스크	그룹 A	그룹 B
600K	11,862,995.9	11,630,387.9	11,458.510.2
편차	-	232,608	171,877

<표 9>로부터 본 논문에서 제시한 태스크 도출 기준을 적용한 그룹 B의 소모 전력이 그룹 A 보다 1.44%에 해당되는 171,877mJ 감소한 것을 확인하였다. 비록 SFT 시스템의 예와 같이 많은 비율의 소모전력 절감 효과는 제시하지 못했지만, 제시된 태스크 식별 기준의 유용성은 확인할 수 있었다. 제시한 암호화 알고리즘과 부호화 알고리즘의 실험에서부터 우리는 데이터 처리의 양이 많은 응용에서 보다 큰 전력 절감 효과를 얻을 수 있다는 사실도 알 수 있었다.

6. 결 론

임베디드 소프트웨어의 기능이 복잡해지고, 서비스가 지능화됨에 따라 소프트웨어의 크기가 증가하고 있으며, 또한 다중 프로세서와 같은 하드웨어 환경에서 동작하는 시스템이 늘고 있다. 소프트웨어 개발에 있어서 태스크의 식별 및 스케쥴링에 대한 연구는 다중 프로세스의 성능과 소모 전력에 대한 측면에서 많이 연구되고 있다. 본 논문에서는 임베디드 소프트웨어에 대한 설계 모델 기반의 태스크 식별 방법에 대하여 제안하였다. 특히 태스크 도출 과정에서 태스크의 상호 작용과 병렬성에 대한 고려를 통해 보다 에너지 효율적인 태스크를 식별할 수 있도록 실험을 통한 기준을 제시하였다.

SFT 예제 시스템과 부호화 알고리즘을 이용하여 제시한 방법의 유용성을 성능, 특히 실행시간과 소모전력 측면에서 실험하였다. 예제 시스템의 소모전력 양을 얻기 위한 시뮬레이션 시간이 너무 커서 - 부호화 알고리즘 예제의 경우 42시간 이상 - 다양한 다른 예제 시스템에 대한 실험이 추가적으로 이루어지지 않았지만, 실험을 통하여 얻은 결과는 제안하는 방법을 통해 태스크를 도출하는 경우, 그렇지 않은 경우에 비하여 소모전력의 절감 효과가 존재한다는 것을

보였다. 이는 제시한 방법이 임베디드 소프트웨어의 설계 과정에서 태스크를 어떻게 식별할 수 있는가에 대한 가이드라인을 제공할 수 있을 것으로 판단한다. 따라서 향후에는 이에 대한 연구의 보완책으로써, 멀티 프로세서상에서 소모 전력을 시뮬레이션할 수 있는 환경 구축과 도구 개발이 필요하다.

참 고 문 헌

[1] T. K. Tan, A. Raghunathan, et al., "Software Architectural Transformations: A New Approach to Low Energy Embedded Software," Proceeding of Design, Automation & Test in Europe, pp.1046-1051, 2003

[2] G. Qu, et. al., "Code Coverage-Based Power Estimation Techniques for Microprocessors," Journal of Circuits, Systems, and Computers, Vol. 11, No. 5, pp. 1-18, 2002.

[3] T. K. Tan, et. al, "High-Level Energy Macromodeling of Embedded Software," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 21, No. 9, pp.1037-1050, Sep. 2002.

[4] E. Senn, et. al., "SoftExplorer: Estimating and Optimizing the Power and Energy Consumption of a C Program for DSP Application," EURASIP Journal on Applied Signal Processing, Vol.16, pp.2641-2654, 2005.

[5] J.P. Kim, D.H. Kim, and J.E. Hong, "Estimating Power Consumption of Mobile Embedded Software Based on Behavioral Model, Proceedings of ICCE 2010, pp. 105-106, Jan. 2010.

[6] H. Jun, L. Xuandong, and W. Chenghua, "Modelling and Analysis of Power Consumption for Component-Based Embedded Software," Proc. of EUC Workshop, pp.795-804, 2006.

[7] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems," in Proc. Design, Automation & Test in Europe Conference, pp. 514 - 521, Feb. 2002.

[8] D. Shin and J. Kim, "System Level Issues: Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems," in Proc. of the 2003 International Symposium on Low Power Electronics and Design, pp.408-413, August 2003.

[9] J. Cong and K. Gururaj, "Energy Efficient Multiprocessor Task Scheduling under Input-dependent Variation," in Proc. of DATE'09, pp.411-416, April 2003.

[10] I. Chatzigiannakis, G. Giannoulis, and P.G. Spirakis, "Energy and Time Efficient Scheduling of Tasks with Dependencies on Asymmetric Multiprocessors," in Proc. of the PODC'08, Aug., pp. 1-12, 2008

[11] M. Goraczko, et. al., "Energy-Optimal Software Partitioning in Heterogeneous Multiprocessor Embedded Systems," Proc. of DAC 2001, pp.191-196, June 2008

[12] Meilir Page-Jones, The Practical Guide to Structured System Design, Yourdon Press, 1988

[13] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison-Wesley, 2000

[14] S.U. Jeon, I.G Song, D.H. Bae, and J.E. Hong, "Developing platform specific model for MPSoC architecture from UML-based embedded software models", Journal of Systems and Software, Vol.82, pp.1695-1708, 2009

[15] R. Jain, D. Molbar, and Z. Ramzan, "Towards a Model of Energy Complexity for Algorithms," Proceedings of the IEEE WCNC 2005, pp.1884-1890, 2005

[16] OMG, UML Superstructure Specification, V2.1.2. Object Management Group, 2007

[17] T. K. Tan, A. Raghunathan, et al., "EMSIM: An Energy Simlator Framwork for an Embedded Operating System", Proc. of the International Symposium of Circuits and Systems, pp. 464-467, 2002.

[18] K. Kyriakopoulos and K. Psarris, "Data Dependence Analysis for Complex Loop Regions," Proceedings of Parallel Processing, pp.195-204, Sept., 2001

[19] T. Jacobson and G. Stubbendieck, "Dependency Analysis for For-Loop Structures for Automatic Parallelization of C Code," Proceedings of the MICS 2003, Duluth, pp.1-13, April 2003

[20] Federal Standards Publisher, Advanced Encryption Standard(AES) : FIPS-PUB-197, Nov. 2001

[21] H.C Huang and J.L. Wu, "Novel real-time software-based video coding algorithms," IEEE Transactions on Consumer Electronics, Vol.39(3), 1993, pp. 570-580

홍 장 의

e-mail : jehong@chungbuk.ac.kr

1988년 충북대학교 전산학과(이학사)

1990년 중앙대학교 컴퓨터공학과 (공학석사)

2001년 한국과학기술원 전산학(공학박사)

2002년 국방과학연구소 선임연구원

2002년~2004년 (주)솔루션링크 기술연구소장

2004년~현 재 충북대학교 컴퓨터공학 부교수

관심분야 : 소프트웨어공학, 임베디드 소프트웨어, 소프트웨어 품질공학, 서비스 기반 개발

김 두 환

e-mail : dhkim@selab.cbnu.ac.kr

2007년 충북대학교 컴퓨터공학과(학사)

2007년~2009년 충북대학교 컴퓨터과학과 (석사)

2010년~현 재 충북대학교 컴퓨터과학과 박사과정

관심분야 : 임베디드 소프트웨어 모델링, 모델기반 전력분석, 소프트웨어 품질공학