최적화된 에너지 소비를 위한 코드 생성 기술

Code Generation Techniques for the Optimized Energy Consumption

고광만*, 소경영**
상지대학교 컴퓨터정보공학부*, 전북대학교 응용시스템공학부**

Kwang-Man Ko(kkman@sangji.ac.kr)*, Kyoung-Young So(kyso@chonbuk.ac.kr)**

요약

최근 임베디드 시스템의 폭넓은 보급은 응용 소프트웨어 개발과 더불어 임베디드 소프트웨어 개발 도구의 필요성 및 중요성이 강조되고 있으며 임베디드 소프트웨어를 위한 컴파일러의 개발을 동시에 요구하고 있다. 특히, 임베디드 프로세서를 탑재한 모바일 장치에서는 제한된 전력/에너지의 하드웨어적인 관리 못지않게 소프트웨어적인 관리 기술의 중요성이 강조되고 있다. 본 논문에서는 검증된 재목적 컴파일러 후단부 도구인 EXPRESSION을 통해 최적화된 에너지 소비를 고려한 MIPS 코드 생성 기술을 제안하였다. 이를 위해, 효율적인 MIPS 코드 생성을 위한 코드 생성 규칙을 기술하였으며 생성된 코드에 대한 다양한 성능분석 결과를 제시한다.

■ 중심어 : | 임베디드 소프트웨어 개발 도구 | MIPS | 전력/에너지 최적화 관리 | 컴파일러 |

Abstract

Recently, together with a new advent of embedded processor developed to support specific application area, and it evolution, a new study of software development to support the embedded processor and its commercial use has been revitalized. Specially, In a mobile device that is built-in embedded processor, software management is as important as hardware management for the limited power/energy. In this paper, we suggest that the code generation technique considering the energy dissipation through the verified retargetable compiler backend tool, EXPRESSION. For this goals, we describes the efficient code generation patterns and showed the variable performance results

■ keyword : | Embedded Software Development Tools | MIPS | Power/Energy Optimizing Management | Compiler |

I. 서 론

최근 임베디드 시스템 분야는 프로세서 진화 및 개발, 응용 소프트웨어 및 소프트웨어 개발 환경 요구, 비약적인 시장 확대, 전문 인력 요구 및 배출 등 모든 관련 부분에서 중요성 및 필요성이 적극적으로 대두되고 있다. 특히, 유비쿼터스 컴퓨팅 환경에서는 임베디드·모바일 장치 사용이 더욱더 증가될 것이며 이를 위한 소프트웨어 지원의 중요성과 필요성 크게 부각되고 있다. 이를 반영하듯 2000년대 초반부터 미국·독일 등에서는 임베디드 시스템에 관련된 하드웨어 발전 못지않게 임베디드 시스템에 관련된 소프트웨어 개발 환경에 관한

접수번호 : #081030-003
접수일자 : 2008년 10월 30일
심사완료일 : 2008년 11월 20일
교신저자 : 고광만, e-mail : kkman@sangji.ac.kr

연구 및 개발 시도가 지속적으로 진행되고 있다.

임베디드 시스템은 범용 시스템에 비해 많은 시스템 자원이 제한적이다. 따라서 제한된 자원의 활용을 극대화할 수 있는 많은 기술이 제안되고 활용되고 있다. 특히, 임베디드 프로세서를 탑재한 모바일 장치 등에서는 제한된 전력/에너지의 하드웨어적인 관리 못지않게 소프트웨어적인 관리 기술의 중요성이 강조되고 있다. 즉, 빠른 실행시간 및 효율적인 메모리 관리와 더불어 제한된 전력/에너지의 소비를 최소화할 수 있는 소프트웨어적인 기술 요구가 임베디드 시스템에서는 중요한 요소이다[1].

임베디드 시스템에 관련된 응용 분야의 비약적인 확장은 다양하고 시간에 민감한(time-to-market) 프로세서의 개발 및 소프트웨어의 개발을 요구한다. 즉, 프로세서의 개발 주기 및 새로운 프로세서 개발 요구가 빨라지고 이를 지원할 수 있는 소프트웨어 개발 도구인 컴파일러, 최적화기, 시뮬레이터 및 디버거 등의 지원이 중요한 요소가 되었다. 다양한 프로세서에 대한 소프트웨어 개발 환경의 구축은 많은 전문지식, 비용, 인력을 요구한다. 이러한 요구사항에 대처하기 위해 기계 기술 언어(Architecture Description Language; ADL)로부터 목적기계에 대한 컴파일러 후단부를 생성하는 재목적 기술이 대안으로 등장하여 중요성이 부각되고 있다[2][3].

본 논문에서는 MIPS R4000[4] 프로세서에 대한 컴파일러 및 시뮬레이터를 ADL로부터 생성하는 검증된 재목적 시스템인 EXPRESSION[5]을 개선하여 전력 소비가 최적화된 MIPS R4000 코드 생성을 목표로 한다. 이를 위해, 첫째, EXPRESSION에서 어플리케이션의 중간표현(*.defs, *.procs)으로부터 MIPS R4000 코드로 변환하는 오퍼레이션 매핑 부분을 재작성하여 생성된 코드 질의 효율성을 입증한다. 둘째, 중간표현으로부터 생성되는 MIPS R4000 코드가 에너지 소비를 최소화할 수 있는 코드 변환 기법을 제안한 후 생성된 MIPS R4000 코드의 에너지 소비 효율성을 입증한다. 마지막으로, 중간표현에 대해 실행속도 및 메모리 활용을 고려한 기법과 최적화된 에너지 소비를 고려한 기법의 비교를 통해 모바일 임베디드 프로세서에 적합한 목적코드 생성 방안을 제시한다.

본 논문은 제 2장에서 ADL을 통한 재목적 컴파일러 후단부 생성 기술과 소프트웨어의 최적화 에너지 소비 관리 기법을 고찰한다. 제 3장에서는 본 논문에서 제안하는 MIPS R4000 코드 생성 방법과 에너지 소비를 최적화하기 도입된 기법을 기술한다. 제 4장에서는 본 논문에서 제안한 기법에 대한 실행 검증과 기존 시스템과 다양한 성능비교 및 분석결과를 기술한다.

II. 관련연구

ADL은 목적기계에 대한 프로세서 구조, 명령어 집합, 메모리 등과 같은 정보를 정형화된 방법으로 기술하여 코드 생성기, 어셈블러, 최적화기, 명령어-셋 시뮬레이터 등을 생성하는 재목적 기술이다. 재목적 기술의 핵심 요소인 ADL을 이용하여 범용 프로세서를 위한 재목적 컴파일러 후단부 개발은 LCC[6], Zephyr[7], LANCE[8] 등을 주목할 수 있다. DSP와 VLIW에 관련되어서는 IMPACT[9]가 관심을 받고 있으며 EXPRESSION, LISA[10][11], CHESS[12]는 ASIP를 위해서 연구용 및 상용화 제품으로 발표되었다.

본 논문의 기반이 되는 EXPRESSION은 구조/행위 정보를 가진 LISP-like한 ADL로부터 컴파일러와 시뮬레이터를 생성한다. 특히, 계층적 메모리 구조 표현과 프로세서의 구조를 시각적으로 확인할 수 있는 기능을 보완하였다. 전체적인 구조는 오퍼레이션 명세, 인스트럭션 기술, 오퍼레이션 매핑을 표현하는 행위 부분과 컴포넌트 명세, 파이프라인/데이터-전송 경로, 메모리 서브시스템을 표현하는 구조 부분으로 구성되어 있다.

하드웨어적인 관리를 통해 프로세서의 전력/에너지 최소화 소비에 대한 다양한 연구와 더불어 최근에는 명령어 선택, 스케쥴링, 전력/에너지 소비에 대한 컴파일러 최적화 기술에 관한 다양한 연구가 시도되고 있다[13]. Uli Kremer는 임베디드 프로세서의 전력/에너지 관리를 위해 컴파일러 기술의 중요성을 강조하고 있으며 임베디드 프로세서를 탑재하는 모바일 장치에서 다양한 전력/에너지 관리를 위한 최적화 기법을 발표하였

다[14-16]. R. Leupers는 재목적 기술에 관련된 다수의 논문 발표를 통해 임베디드 프로세서를 위한 컴파일러 기술에서 전력/에너지 소비에 관한 최적화 기술과 재목적 기술의 중요성을 역설하였다[17].

어플리케이션의 실행은 필연적으로 하드웨어적인 동작으로 인한 전력 소비와 더불어 저수준의 명령어 실행을 위해 기억장소 접근, 명령어 디스패치, 레지스터 이용 등으로 전력의 소비가 발생된다. 에너지는 이러한 명령어 실행 등으로 발생되는 전체 전력 소비량으로서 각 명령어의 실행에 소비되는 전력량을 줄임으로서 에너지 소비를 감소시킬 수 있다[13]. 어플리케이션 실행에 소비되는 전력(P)은 P = I(전류) × Vcc(전압)로 측정된다. 에너지(E)는 어플리케이션 실행으로 전력이 소비되는 시간(T)에 연관되므로 E = P × T로 측정한다. 어플리케이션의 실행시간은 T = N × τ로 측정되며 N은 명령어 실행에 필요한 클럭-싸이클 수를 의미하며 τ는 클럭의 주기를 의미한다. 따라서 어플리케이션의 실행에 필요한 에너지 소비량(단위: Joules)은 (식)-1과 같이 표현된다[18]. 본 논문에서는 소비되는 에너지 소비량 측정을 위해 EXPRESSION의 SIMPRESS[8] 시뮬레이터를 이용하며 에너지 소비량 측정을 위해 (식)-1이 이용된다.

$$E = I \times Vcc \times N \times \tau \quad \text{-----} \quad \text{(식)-1}$$

또한 소프트웨어를 위한 명령어-수준의 전력 분석과 최적화를 위해서 각 명령어가 실행되는데 필요한 에너지를 측정하여 어플리케이션의 총 에너지 소비량을 측정하고 명령어 순서 재조정, 메모리 접근 최소화를 통해 소프트웨어의 에너지 소비를 최적화하는 기법이 제안되었다[19].

Ⅲ. 에너지-드리븐 코드 생성

3.1 에너지-드리븐 코드 생성 모델

본 논문에서 어플리케이션의 중간표현에 대해 목적 코드를 생성하는 모델은 [그림 1]과 같은 EXPRESSION 코드 생성 모델을 적용하여 에너지-드리븐 코드 생성기(energy-driven code generator)를 구축한다. MIPS R4000 프로세서에 대한 코드를 생성하기 위해 전단부, 툴킷 생성기, 시뮬레이터는 변경없이 사용하지만 ADL(MIPS.xmd)에서 어플리케이션의 중간표현에 대해 MIPS R4000 오퍼레이션으로 매핑 관계를 기술하는 부분을 재작성 하였다. 새롭게 기술된 MIPS.xmd를 통해 실행속도 향상, 메모리 접근 횟수 감소, 에너지 소비 최적화할 수 있는 에너지-드리븐 코드 생성기를 생성한다.

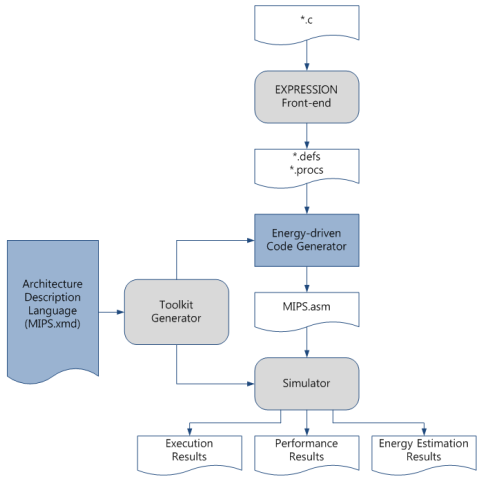

그림 1. 에너지-드리븐 코드 생성 모델

생성된 코드에 대한 실행 검증 및 다양한 성능 분석을 위해 EXPRESSION의 시뮬레이터를 이용한다. 따라서 MIPS R4000에 대한 ADL에서 시뮬레이터에 관련된 부분에 대해서는 변경없이 사용하였다.

3.2 MIPS R4000 기계 기술 언어

ADL의 구조는 오퍼레이션 명세, 인스트럭션 기술, 오퍼레이션 매핑에 관련된 행위 부분과 컴포넌트 명세, 파이프라인/데이터-전송 경로, 메모리 서브시스템을 표현하는 구조 부분으로 구성되어 있다.

오퍼레이션 명세부에서는 프로세서의 인스트럭션-셋이 유사한 성격을 갖는 그룹 단위로 MIPS R4000 명

령어 대한 명세를 기술하는 부분으로서 [그림 2]와 같이 명령어가 갖는 오퍼란드 종류, 자료형과 같은 속성을 기술하는 VAR_GROUPS 부분과 실제로 명령어의 명세와 동작, 출력 형식을 정의하는 OPCODE 부분으로 구성되어 있으며 MIPS R4000의 모든 명령어에 대해 본 논문에서는 EXPRESSION 기술 내용을 변경없이 이용하였다.

```
(OPERATIONS_SECTION
(VAR_GROUPS
 (float_any (DATATYPE FLOAT)
        (REGS FPRFile))
 (int_mem (DATATYPE INT) (REGS
L1))
 (int_any (DATATYPE INT) (REGS
GPRFile[1-28]))
 (any_call_param (DATATYPE INT) (REGS
GPRFile[4-12]))
 // ...
 )
(OPCODE div
 (OP_TYPE DATA_OP)
 (OPERANDS (_SOURCE_1_ int_any)
(_SOURCE_2_ int_any) (_DEST_ int_hilo))
 (BEHAVIOR "_DEST_ = _SOURCE_1_ /
_SOURCE_2_")
)
(OPCODE mult
 (OP_TYPE DATA_OP)
 (OPERANDS (_SOURCE_1_ int_any)
(_SOURCE_2_ int_any)
                (_DEST_ int_hilo))
 (BEHAVIOR "_DEST_ = _SOURCE_1_ *
_SOURCE_2_")
)
// . . .
```

그림 2. MIPS R4000 오퍼레이션 명세

본 논문에서 제안하는 ADL의 핵심이 되는 오퍼레이션 매핑 부분에서는 EXPRESSION의 전단부로부터 생성된 중간표현의 제너릭 오퍼레이션(*.procs)을 MIPS R4000 명령어로 변환하는 매칭 패턴을 메모리 접근 감소, 실행속도 향상, 에너지 소비의 최소화에 적합한 코드 생성을 고려하여 새롭게 기술하였다. 매칭 패턴을 기술하기 위해 메모리 접근대신 레지스터에 피연산자를 저장하고 읽어올 수 있도록 제너릭 패턴에 대해 목적 패턴을 [그림 3]과 같이 기술하였다. 메모리 접근 형식을 레지스터 접근으로 변경하거나 추가한 패턴은 EXPRESSION 명세서에서 제공하는 92개의 제너릭 명령어 모두에 대해 진행하였다. 메모리 접근을 레지스터 접근으로 변경은 전력/에너지 소비를 최소화할 수 있는 장점을 제공하지만 제한된 레지스터를 효율적으로 사용하기 위한 다양한 시도는 실행속도의 저하를 유발할 수 있다. 따라서 본 논문에서 제안한 패턴의 효과를 검증하기 위해 기존 매칭 패턴을 사용하지 않고 본 논문에서 제안한 패턴만으로 성능을 측정하였으며 4장 2절에서 측정 방법, 내용, 결과 분석을 기술하였다. 또한 다수의 순차적인 제너릭 명령어 패턴(N)을 분석하여 1:1 매칭 방식이 아닌 N:1 또는 N:M 방식으로 MIPS R4000 명령어를 생성할 수 있는 패턴을 기술하여 추가하였다. 추가된 패턴은 기존 EXPRESSION의 패턴 저장 부분과 별개의 테이블을 생성한 후 저장할 수 있도록 하였다.

3.3 MIPS R4000 코드 생성 결과

본 본문에서 제안한 패턴이 올바른 결과를 생성하는 과정을 검증하기 위해 EXPRESSION에서 실행검증 및 성능평가와 같은 벤치마킹을 위해 사용하였던 어플리케이션(LL1.c~LL19.c) 모두 사용하였다. 실제로 임의의 어플리케이션에 대해 중간표현과 생성된 MIPS R4000 코드는 [그림 4][그림 5]와 같다.

EXPRESS 패칭 패턴(변경전)	본 논문 제안 패턴(변경후)
;IADD ((GENERIC ((IADD DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))) (TARGET ((addu DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))))	;IADD ((GENERIC ((IADD DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))) (TARGET ((addu DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = REG(3)))))
;ISUB ((GENERIC ((ISUB DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))) (TARGET ((subu DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))))	;ISUB ((GENERIC ((ISUB DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))) (TARGET ((subu DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = REG(3)))))
;IVLOAD ((GENERIC ((IVLOAD DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))) (TARGET ((lw DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))))	;IVLOAD; ((GENERIC ((IVLOAD DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = IMM(3)))) (TARGET ((lw DST[1] = REG(1) SRC[1] = REG(2) SRC[2] = REG(3)))))
;IVSTORE ((GENERIC ((IVSTORE SRC[1] = REG(1) SRC[2] = IMM(2) SRC[3] = REG(3)))) (TARGET ((sw SRC[1] = REG(1) SRC[2] = IMM(2) SRC[3] = REG(3)))))	;IVSTORE ((GENERIC ((IVSTORE SRC[1] = REG(1) SRC[2] = IMM(2) SRC[3] = REG(3)))) (TARGET ((sw SRC[1] = REG(1) SRC[2] = REG(2) SRC[3] = REG(3)))))

그림 3. 레지스터 접근을 위한 매칭 패턴 기술

```
; LL1.c
   main() {
            register int      k;
            int   cksum[21];
            int   x[1002], y[1002], z[1002];
            int   r, t;
            register int q;
            r = 4; t = 276; q = 0;
            for(k=1; k<=20; k++) {
              x[k]=y[k]=z[k]=k;
            }
            for (k=1; k<=4; k++) {
                    x[k] = q+y[k]*(r*z[k+10]+t*z[k+11]);
            }
   }

; LL1.procs
(PROC_BEGIN                 _main
(LABEL _main)
                vars= 8104, regs= 2/0, args= 48, extra= 0
                (ISUB           $sp $sp 8160)
                (IVSTORE        8156 $sp $31)
                (IVSTORE        8152 $sp $fp)
                (IASSIGN        $fp $sp)
                move            $fp,$sp
(LOCATION 7)
(LOCATION 11)
                (IASSIGN        $4 $0)
                move            $4,$0
                (IADD           $7 $fp 136)
                addu            $7,$fp,136
                (IADD           $3 $fp 4144)
                (IASSIGN        $6 $7)
                (IASSIGN        $5 $3)
(LABEL L6)
(LOCATION 12)
                (IVSTORE        0 $5 $4)
                sw              $4,0($5)
                (IVSTORE        0 $6 $4)
                sw              $4,0($6)
(LOCATION 11)
                (IADD           $6 $6 4)
                addu            $6,$6,4
                (IADD           $5 $5 4)
                addu            $5,$5,4
                (IADD           $4 $4 1)
                addu            $4,$4,1
                (ILT            $2 $4 10)
                (INE            $cc0 $2 $0)
                (IF             $cc L6)
                (LOCATION 19)
                (IVLOAD         $2 4148 $fp)
                sw              $2,140($fp)
(LOCATION 20)
                (IADD           $4 $7 8)
                $4,$7,8
                $5,$3,8
; 생략
```

그림 4. LL1.c 및 LL1.procs

```
                ; ...
sw              ()
                sw          ()
                move        ($259)      ($258)
                move        ($252)      ($229)
                addu        ($253)      ($259,136)
                addu        ($254)      ($259,4144)
                move        ($255)      ($253)
                move        ($256)      ($254)
L6:
                sw          ()
                sw          ()
                addu        ($255)      ($255,4)
                addu        ($256)      ($256,4)
                addu        ($252)      ($252,1)
                slt         ($257)      ($252,10)
                sne         ($257)      ($257,$229)
                bnez        ()
                lw          ($257)      (4148,$259)
                sw          ()
                addu        ($253)      ($253,8)
                addu        ($254)      ($254,8)
                li          ($255)      (4)
L11:
                lw          ($256)      (-4,$253)
                lw          ($257)      (0,$254)
                addu        ($257)      ($256,$257)
                sw          ()
                addu        ($253)      ($253,4)
                addu        ($254)      ($254,4)
                addu        ($255)      ($255,-1)
                sge         ($257)      ($255,$229)
                bnez        ()
                lw          ($241)      (160,$259)
                jal         ()
; 생략
```

그림 5. LL1.c에 대한 MIPS R4000 코드 생성

Ⅳ. 성능평가 및 분석

4.1 성능분석 환경

본 논문에서 설계한 에너지 소비를 최소화하는 코드를 생성하기 위한 패턴의 실행검증과 성능분석을 위해 Windows XP, Pentium4(3.00GHz, 2G RAM) 환경에서 EXPRESSION에서 제공하는 SIMPRESS 시뮬레이터를 이용하였다. 모든 성능분석 환경을 동일한 조건으로 설정하였으며 동일한 어플리케이션(LL1.c~LL19.c)을 EXPRESSION과 본 논문에서 제안한 패턴을 이용하는 코드 생성기에 적용하였다.

SIMPRESS는 어플리케이션에 대해 MIPS R4000 코

드가 실제로 실행되는 과정에서 실행속도, 메모리 접근 횟수, 에너지 소비 등의 측정결과를 제시한다.

4.2 성능분석 및 평가

실제로 LL1.c에 대한 실행속도, 캐쉬메모리 접근 횟수, 에너지 소비량을 기존 EXPRESSION의 코드 매칭 패턴을 적용한 결과와 본 논문에서 에너지 소비를 줄일 수 있도록 제안한 패턴의 성능평가 결과는 [그림 6]과 같다.

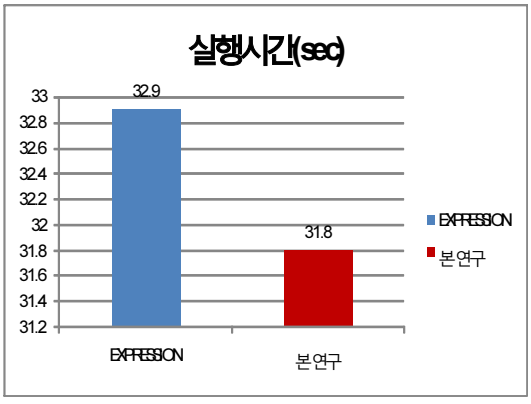

그림 6. 실행속도 감소 예(LL1.c)

본 논문에서 제안한 패턴을 적용하였을 경우 LL1.c 어플리케이션의 경우 실행시간이 다소 감소(1.1sec)되는 효과를 얻을 수 있었지만 19개의 어플리케이션 전체에 대해서는 현저한 실행시간 감소(15개 어플리케이션) 또는 증가(4개 어플리케이션)의 결과를 확인할 수 없었다. 어플리케이션중 LL17.c 어플리케이션의 경우는 [그림 7]과 같이 실행속도가 증가(0.8sec)되는 결과를 확인하였지만 상대적으로 에너지 소비에는 큰 변화가 없어 본 논문에서 제안한 패턴이 실행속도에는 큰 영향을 주지 않고 에너지 소비는 감소할 수 있는 결과를 확인할 수 있었다.

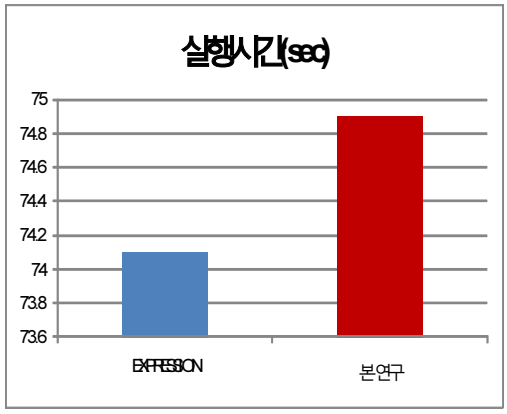

그림 7. 실행속도 증가 예(LL17.c)

에너지 소비를 최적화하기 위한 패턴을 적용하면 기존 방식에 비해 캐쉬 메모리에 대한 읽기, 쓰기 동작이 감소되는 효과와 명령 싸이클이 감소되는 결과를 시뮬레이터를 통해 확인할 수 있었으며 소비되는 에너지의 양도 감소되는 결과를 [그림 8]과 같이 확인하였다.

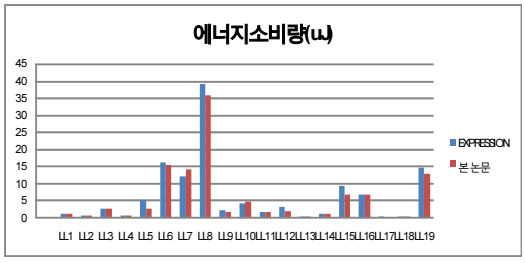

그림 8. 에너지 소비량 비교

이와 같은 실험을 EXPRESSION에서 벤치마킹을 위해 적용한 19개의 어플리케이션에 모두 적용하여 에너지 소비가 평균 9% 정도의 감소 효과를 확인하였다.

V. 결론 및 향후연구

임베디드 시스템에 탑재되는 프로세서는 범용 시스템의 프로세서에 비해 많은 시스템 자원이 제한적이다. 따라서 제한된 자원의 활용을 극대화할 수 있는 많은 기술이 제안되고 활용되고 있다. 특히, 임베디드 프로세

서를 탑재한 모바일 장치 등에서는 제한된 전력/에너지의 하드웨어적인 관리 못지않게 소프트웨어적인 관리 기술의 중요성이 강조되고 있다. 즉, 빠른 실행시간 및 효율적인 메모리 관리와 더불어 제한된 전력/에너지의 소비를 최소화할 수 있는 소프트웨어적인 기술요구가 임베디드 시스템에서는 중요한 요소이다

본 논문에서는 모바일 장치에 탑재되는 프로세서를 위해 에너지 소비를 줄일 수 있는 코드를 생성할 수 있도록 중간표현에 대한 패턴을 기술하였다. 또한 생성된 코드의 평가를 위해 EXPRESSION과 동일한 조건, 환경에서 결과를 확인하고 성능평가를 진행하였다.

본 연구를 통해서 전력자원이 연속적인 환경에서는 실행속도 향상, 메모리 사용의 최소화 등을 통해 양질의 목적코드를 생성하는 코드 생성기, 코드 최적화기의 개발이 중요한 쟁점이었지만 모바일 장치와 같이 전력자원이 제한적인 경우는 빠른 실행시간, 메모리 사용빈도 못지않게 에너지 소비를 줄일 수 있는 코드 생성기술의 중요성과 발전된 기술 개발의 필요성을 확인할 수 있었다.

앞으로 생성된 목적코드에 대해 명령어 순서 재조정(rescheduling)을 적용하여 에너지 소비를 줄일 수 있는 방법을 보충할 예정이며 궁극적으로 다양한 임베디드 프로세서의 컴파일러 개발을 자동화할 수 있는 재목적 기술의 핵심인 ADL에 반영할 예정이다.

참 고 문 헌

[1] K. Uli, "Compilers for Power and Energy Management," ACM SIGPLAN PLD, 2003(6).

[2] L. Rainer, "Compiler Design Issues for Embedded Processors," IEEE Design & Test of Computers, 2002(7).

[3] B. Luca, M. Kandemir, and J. Ramanujam, *Compilers and Operating Systems for Low Power*, Kluwer Academics Publishers, 2003.

[4] H. Joe, MIPS R4000 Microprocessor User Manual 2nd, MIPS Technologies Inc., 1994.

[5] A. Halambi, P. Grün, V. Ganesh, A. Khare, N. D. Dutt, and A. Nicolau, "EXPRESSION: A Language for Architectural Exploration through Compiler/Simulator Retargetability and User Manual," University of California, Irvine, 2003.

[6] W. F. Christopher and R. H. David, *A Retargetable C Compiler: Design and Implementation*, Addison-Wesley, 1995.

[7] A. Appel, J. Davidson, and N. Ramsey, "The Zephyr Compiler Infrastructure," Supercomputing98, 1998(11).

[8] R. Leupers, *Retargetable Code Generation for Digital Signal Processors*, Kluwer Academic Publishers, 1997.

[9] P. Chang, S. Mahlke, W. Chen, N. Warter, and W. Hwu, "IMPACT: An Architectural Framework for Multiple Instruction Issue Processors," 18th Int. Symposium on Computer Architecture, 1991.

[10] S. Pees, A. Hoffman, V. Zivojnovic, and H. Meyr, "LISA-Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures," 36th DAC, 1999.

[11] A. Hoffman, A. Nohl, G, Braun, and H. Meyr, "A Survey on Modeling Issues Using the Machine Description Language LISA," ICASSP, 2001.

[12] G. Goosens, "CHESS: Retargetable Code Generation for Embedded DSP Processors," Chap. 5 Kluwer Academic Publishers, 1997.

[13] M. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power Analysis and Minimization Techniques for Embedded DSP Software," IEEE Trans. on VLSI Systems, Vol.5, No.2, 1997.

[14] K. Uli, "Compilers for Power and Energy Management," Tutorial ACM SIGPLAN PLDI2003, 2003(6).

[15] K. Uli, "Low Power/Power Compiler

Optimizations," In Low-Power Electronics Design, CRC Press, 2005.

[16] C. H. Hsu and K. Uli, "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction," ACM SIGPLAN PLDI2003, 2003(6).

[17] R. Leupers, "Compiler Design Issues for Embedded Processors," IEEE Design & Test of Computers, 2002(7).

[18] V. Tiwari, S. Malik, and A Wolfe, "Compilation Techniques for Low Energy: An Overview," IEEE Symposium on Low Power Electronics, 1994.

[19] V. Tiwari, S. Malik, and A Wolfe, "Instruction Level Power Analysis and Optimization of Software," Journal of VLSI Signal Processing, 1996.

저 자 소 개

고 광 만(Kwang-Man Ko) 정회원

- 1991년 2월 : 원광대학교 컴퓨터공학과(공학사)
- 1993년 2월 : 동국대학교 컴퓨터공학과(공학석사)
- 1998년 2월 : 동국대학교 컴퓨터공학과(공학박사)
- 1998년 3월 ~ 2001년 8월 : 광주여자대학교 컴퓨터과학과 전임강사
- 2003년 1월 ~ 2월 : QUT(호주) 객원연구원
- 2008년 7월 ~ 8월 : UQAM(캐나다) 객원연구원
- 2001년 9월 ~ 현재 : 상지대학교 컴퓨터정보공학부 부교수

<관심분야> : 프로그래밍언어론 및 컴파일러

소 경 영(Kyoung-Young So) 정회원

- 1986년 2월 : 원광대학교 컴퓨터공학과(공학사)
- 1989년 2월 : 원광대학교 컴퓨터공학과(공학석사)
- 2000년 8월 : 원광대학교 컴퓨터공학과(공학박사)
- 1991년 12월 ~ 2008년 2월 : 국립익산대학 컴퓨터정보과 교수
- 2008년 3월 ~ 현재 : 전북대학교 응용시스템공학부 교수

<관심분야> : 소프트웨어공학 및 프로그래밍언어론.