ARM 프로세서 하에서의 코드 최적화 전략에 대한 성능 평가

최광원, 선주택, 이정근
한림대학교 컴퓨터공학과
e-mail : sodagu@hallym.ac.kr

Performance Evaluation of Code Optimization Strategies for ARM Processors

Kwang-Won Choi, Ju-Teak Sun, Jeong-Gun Lee
Dept. of Computer Engineering, Hallym University

요 약

임베디드 소프트웨어의 경우, 시스템 제약으로 인해서 일반 소프트웨어에 비해 최적화의 역할이 상대적으로 매우 중요하다. 본 논문에서는 임베디드 시스템 환경에서 다양한 코드 최적화 기법을 적용하고 각 최적화 기법의 성능 향상기여도를 평가하여, 다양한 최적화 전략에 대한 효율성을 분석하였다. 이는 향후 임베디드 소프트웨어 엔지니어를 위한 코드 최적화 가이드라인으로 활용되거나, 자동화된 최적 임베디드 코드 생성을 위한 기초 자료로 사용될 수 있을 것으로 기대한다.

1. 서론

최근 x86 시스템과 같이 대량의 리소스를 가진 시스템에서 코드의 최적화는 PC성능의 향상과 더불어 상대적으로 그 중요성을 약화되고 있다. 하지만, 임베디드 시스템과 같이 실시간, 저전력, 저용량 등의 다양한 설계 제약 조건을 가진 시스템의 경우 코드의 최적화는 성능 향상 및 단가 절약을 유도하여 개발 제품의 시장 경쟁력을 강화시키는 핵심 기술이다.

프로그램 최적화시에 가장 중요한 것은 첫째, 프로그램을 이루는 각각의 모듈들 중에서 어떤 부분이 느리게 작동하거나 큰 메모리를 소비하는 지 찾아내는 것이다. 이후, 성능저하에 큰 비중을 차지하는 각각의 모듈을 최적화하여 전체적 프로그램의 성능을 향상시킨다. 이러한 모듈단위 최적화는 최적화를 위한 부분을 비교적 쉽게 찾고, 쉽게 해결할 수 있다는 장점을 가지게 된다. 정형적으로 이루어지는 다양한 최적화 룰을 적용하여 기계적인 변형(transformation)이 가능하다. 다만, 상황에 따라 어떠한 최적화 기법을 적용하는 것이 성능향상에 큰 효과를 가져 오는지는 사용하는 CPU와 시스템의 구성에 따라 달라질수 있다.

일반적으로 속도를 향상시키기 위해 코드가 증가될 가능성이 있으며, tradeoff관계를 형성할 수 있으므로, 무조건 적인 속도향상 최적화는 이후 메모리 사용량을 증대시키고 전력소모의 증대 및 궁극적 성능저하를 유도할 수 있다. 본 연구에서는 성능향상과 코드의 크기 모두 고려한 최적화를 수행한다.

본 논문에서는 ARM프로세서(PXA255A) 환경에서 일반 C코드와 최적화된 C코드를 각각 모듈화 시켜 실행시켜 본 후 그 시간을 측정하여 각각의 상황에서 성능의 차이가 생기는지 확인하고 이를 비교/분석하였다.

2. 코드 최적화 실험의 활용성

많은 문헌에서 코드 최적화에 관한 부분을 다루고 있지만 현실적으로 각각의 시스템의 성능의 차이라는 관점이 있기에 어떤 부분이 얼마나 속도차이가 존재하는 지 보여주는 직접적 실험 데이터는 찾아보기 어렵다. 따라서 본 논문에서는 ARM환경에서 일반적으로 사용하는 소스와 codeproject.com [1]의 최적화 소스들을 비교하고 성능을 평가하여, 그에 따라 나오는 차이를 기록/분석하였다. 실험을 통하여 획득한 두 코드간의 성능 차를 기초 데이터를 기반으로 보다 큰 사이즈의 임베디드 소프트웨어 최적화시에 최적화 규칙을 적용한다며, 효과적인 소프트웨어를 작성할 수 있을 것이다.

3. 실험

3.1 시스템 구성

<표 1 시스템 사양>

ARM	CPU - PXA255A (200MHz)
	SDRAM - 128MB SDRAM (32BIT)
	FLASH - 32MB (32BIT)
	NAND - 1GB
	OS - Linux Kernel Source (2.6.21)
x86(PC)	CPU - Intel(R) L7500 1.60GHz
	RAM - 2G
	OS - Windows XP Professional sp2

ARM 시스템은 (주)Huins 사의 XSTATION-AP Kit를 사용하였다. 위 보드는 ARM ATC 인증용 임베디드 장비로 인텔사의 PXA255칩을 사용하고 있다. x86은 ARM 시스템과 비교하기 위해 셋팅하였으며 동일 소스를 사용하여 성능 평가하였다.

3.2 실험 목록

C코드 최적화 실험은 아래 <표 2>와 같이 실시하였다.

<표 2 최적화 전략 룰 [1]>

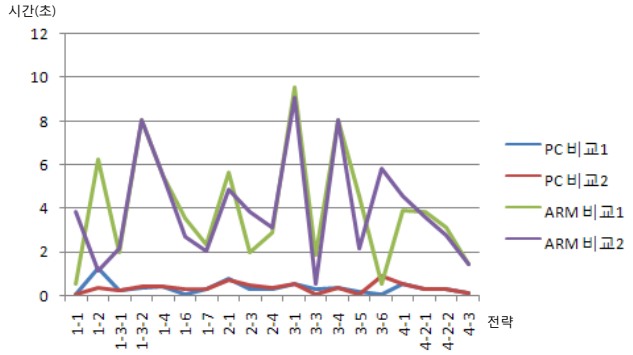
1. 데이터 연산
   1.1 정수
      int 형과 float 형의 연산 속도 비교
   1.2 나눗셈 그리고 나머지
      나눗셈 과 곱셈의 연산 속도 비교
   1.3 2의 배수로 나누기
      -1. UINT와 INT의 나눗셈 연산 속도 비교
      -2. 2의 배수 나눗셈연산과 Shift연산의 비교
   1.4 Binary Breakdown
      2중 If문을 사용한 조건검사 횟수 줄이기
   1.5 배열을 이용한 index 생성
      Switch, If~else, 배열인덱스 간에 접근시간 비교
   1.6 나머지 연산의 대체
      나머지연산의 값이 0 인 경우에 %연산과 비교연산과의 속도 차이 비교
   1.7 Using Aliases
      변하지 않는 값의 메모리 호출빈도 줄이기
2. 데이터 타입
   2.1 전역 변수
      전역변수와 지역변수의 접근시간 비교
   2.2 지역 변수
      지역변수인 int, char, short의 접근속도 비교
   2.3 포인터
      구조체를 포인터로 넘길때와 값으로 넘길때의 접근시간 비교
   2.4 Switch 대신 lookup table의 사용
      Switch 연산과 lookup table의 효율성 비교
3. 루프
   3.1 Loop termination
      증가 루프와 감소 루프의 실행시간 비교
   3.2 더욱 빠른 for문
      서로 다른 감소루프 간 실행시간 비교
   3.3 함수 루프
      루프 속 함수와 함수 속 로프의 수행시간 비교
   3.4 Population count - 비트 계수하기
      주어진 값에서 1씩 Shift와 4씩 Shift시에 참,거짓연산 시간 비교
   3.5 Early loop brea
      조건만족시 바로 루프를 나오는 것과 그냥 루프를 끝까지 실행시의 낭비시간 비교
   3.6 Loop 사용하지 않기
      몇 번만 순환하는 루프를 풀어서 코딩한 경우와 루프를 사용한 것과의 실행시간 비교
4. 함수 디자인
   4.1 함수 호출 Overhead
      함수 호출시 인자가 4개인 경우(G1)와 그 이상인 경우(G2)의 호출시간 비교
   4.2 가능한 인자의 수를 줄여라
      6인자 또는 4인자의 경우 구조체를 보낼 때와 값을 보낼 때의 실행시간 비교
   4.3 인라인 함수
      인라인함수와 일반함수의 호출시간 비교

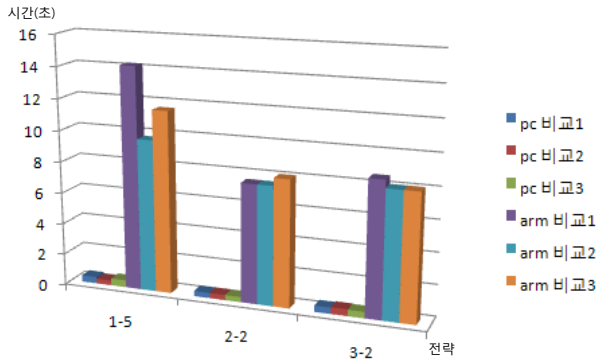
위의 표와 같이 총 20가지 코드 최적화 룰[1]이 적용된 경우와 적용되지 않은 코드의 성능을 비하 평가하였다.

3.3 실험 결과

전체 실험 결과를 보여주는 표는 지면의 제약 관계상 생략하였다. 평균 실행시간 결과표는 아래 <표 3> 과 같다. <표 3>을 통해 어떠한 최적화 전략이 어떤 플랫폼(PC 또는 ARM)에서 보다 효과적인 성능 개선을 보이는지 살펴볼 수 있다.

<표 3 평균 실행시간 결과표>





비교 1,2,3 은 비교대상을 <표 2>의 최적화 전략을 앞에서부터 순서대로 대입하면 된다. <표 3>의 실험결과에서 보이듯, ARM 플랫폼에서는 1-1 / 1-2 / 1-5 / 2-3 / 3-6 실험에 대해서는 확연한 차이를 보여주었으며, 따라서, ARM code 생성 시에 이와 관련된 최적화를 적절히 적용하는 것이 유익할 것이다. PC 플랫폼에서는 대체적으로 고른 실험 결과를 보여줬다. 보다 상세한 최적화 전략의 민감도 분석(sensitivity analysis) 및 이에 대한 프로세서의 구조적 해석은 논문의 지면 제약 관계로 생략한다.

4. 평가 및 결론

위의 실험표를 통하여 ARM 프로세서를 포함하는 임베디드 시스템의 경우 효과적인 최적화 전략을 알수 있다. 예를 들어, 전략 1-1의 경우 일반 x86계열의 경우 성능차가 미비하였으나, ARM계열의 임베디드 시스템의 경우 큰 성능변화가 있음을 알 수 있다. 테이블에 대한 보다 상세한 분석은 지면의 제한으로 생략한다. 본 연구는 기초적인 실험에 기인하지만, 향후 코드 최적화의 자동화 시에 최적화 룰을 탐색하는 핵심 자료로 사용될 수 있으리라 판단된다. 향후 과제로 임베디드 시스템에서 주요시 되는 전력소모 최적화 전략에 대한 분석을 수행하고자 한다.

참고문헌

[1] http://www.codeproject.com/cpp/C___Code_Optimization.asp