한국컴퓨터정보학회 동계학술대회 논문집 제20권 제1호 (2012. 1)

임베디드 시스템에서의 성능 향상을 위한 루프 펼침과 형변환

성운°, 신동영*, 박준석*
°*인하대학교 컴퓨터정보공학과
e-mail:{wo2n, nuclear6}@naver.com, joonseok@inha.ac.kr

Loop unrolling and type casting operation for performance improvement in embedded system

Woon Sung°, Dong Young Shin*, Joon Seok Park*
°*Dept. of Computer and Information Engineering, Inha University

● 요 약 ●

임베디드 시스템에서 최적화 기술의 성능은 크로스 컴파일러의 성능과 실행상황, 대상 하드웨어의 특징등에 따라 좌우된다. 본 논문에서는 최적화 기술 중 루프 펼침과 형 변환을 이미지 처리 코드에 적용하여 성능을 측정하였다. 그 결과 기술을 적용하지 않은 성능을 기준으로 55%의 성능향상이 이루어졌다.

키워드: 루프 펼침(loop unrolling), 부동 소수점 연산(floating point operation), 고정 소수점 연산(fixed point operation)

Ⅰ. 서론

최초의 전자식 컴퓨터가 발명된 이후 컴퓨터의 성능 은 극적으로 발전하여 현재에 이르러서는 컴퓨터 사용자에게 제공되는 능력이 매우 증가하였다. 직접 회로 기술의 발달로 인해 CPU의 속도 및 처리능력의 향상과 메모리 서브시스템의 속도, 용량의 상승이 이루어졌다. 이로 인해 현재 소프트웨어 개발자와 사용자에게 과거와는 완전히 다른 컴퓨터 사용 환경을 제공한다. 불과 20년전에 슈퍼 컴퓨터에서나 가능했던 연산 능력의 CPU와 메모리사이즈를 가진 범용 컴퓨팅 시스템을 현재는 개인이 사용한다[1].

반면 현재 급격히 성장중인 임베디드 시스템 소프트웨어의 경우, 이동성, 저전력성 등에 중점을 둔 임베디드 시스템 하드웨어의 특징으로 인해 소프트웨어 개발 시 범용 시스템보다 더 많은 제약사항이 요구된다. 그 예로 임베디드 시스템의 제한된 하드웨어를 효과적으로 이용할 수 있는 최적화된 코드의 생성이 있다. 범용 시스템에서 주로 개발된 컴파일러와 각종 최적화 기술들과 달리 임베디드 시스템의 제한된 하드웨어에서 최적화 기술들의 유효성은 시스템 개발과 함께 사용되는 크로스 컴파일러의 성능에 좌우된다. 임베디드 시스템 개발에 사용되는 컴파일러의 경우 범용 시스템의 컴파일러에서 기본적으로 제공하는 최적화 기술들을 제공하지 않는 경우가 많다. 또한 임베디드 시스템의 특성 상 실행상황, 대상 하드웨어의 특징 등의 고려 여부에 따라 큰 성능차가 발생한다.

본 논문에서는 널리 사용되는 최적화 기술인 루프 펼침(loop unrolling)[2]과 응용에 따라 적용 가능한 부동 소수점 연산(floating point operation)을 고정 소수점(fixed point operation)으로 바꾸는 형변환 기법이 부동 소수점연산 처리능력이 상대적으로 부족한 임베디드 시스템 환경에서 성능에 미치는 영향에 대해 실험 및 분석을 수행하였다. 루프 펼침과 형변환 기법을 적용한 loop기반 이미지 처리 코드를 통하여 ARM기반 임베디드 시스템 환경에서의 성능 변화를 실험하였다.

본 논문의 구성은 다음과 같다. 2장은 연구에 적용된 최적화 기술에 대해 소개한다. 3장에서는 본 논문에서 사용되는 이미지 처리 코드에 대해 각각의 최적화 기술을 적용하는 과정과 성능 변화를 분석하기 위해 실험 내용 및 실험 결과를 기술한다. 4장과 5장에서는 기존 관련 연구에 대한 소개와 결론 및 향후 연구 방향에 대해 논한다.

Ⅱ. 관련연구

최근 들어 임베디드 시스템 환경에서의 전통적 최적화 기술에 대한 유효성에 대해 많은 연구가 진행되고 있다. J. Bungo는 전통적인 각종 컴파일러 최적화 기법 가운데 임베디드 시스템 소프트웨어에 영향을 줄 수 있는 요소 기법에 대해 설명하였다[3]. Yunyang Dai 등은 임베디드 시스템 환경에서 SIMD에 효과적인 루프 펼침 스키마를 제안한다[4].

Ⅲ. 본론

1. 최적화

1.1 loop unrolling

루프 펼침(loop unrolling)은 루프 내의 명령어를 펼침 인자(unrolling factor)만큼 복제하고, 펼침인자 단위로 루프의 회전이

일어나도록 하는 변환이다[2]. 루프 펼침은 크게 두 개의 성능향상 효과를 가진다. 첫째는 루프 종료 검사(end of loop test)가 가지는 오버헤드 감소다. 루프의 경우 바디 부분의 명령어를 수행 뒤 루프 종료 검사를 위해 비교연산과 분기 명령을 실행한다. 루프 펼침은 펼침인자 단위로 루프의 회전이 일어남으로 루프 종료 검사의 실행 빈도를 줄여 오버헤드를 줄인다. 둘째, 루프 내의 명령어를 복제함으로 루프의 바디가 커진다는 점이다. 루프 바디가 커짐에 따라 내부에 복제된 명령어가 각각이 독립적(independent)이라면 명령어 단위 병렬성(Instruction Level Parallelism, ILP)의 확보가 가능해지며, 각종 컴파일러 최적화 기술의 적용이 더욱더 용이하다.

그러나 루프 펼침은 루프 내의 명령어를 복제하여 사용함으로 코드의 사이즈가 증가하며, 명령어 캐시 미스(instruction cache miss)가 증가 할 수 있다. 또한 루프 내에서 특정 함수 호출(function call)이 있을 경우에는 루프 펼침이 코드 사이즈를 극단적으로 증가시킨다. 그리고 변수증가로 인한 레지스터 과도 사용(register pressure)이 발생할 수 있다. 이러한 단점들은 특히 임베디드 시스템용 소프트웨어의 성능에 많은 영향을 준다.

1.2 형변환

부동 소수점(floating point)연산을 고정 소수점(fixed point)연산으로 변환하는 방법이다. C 코드 내에서 float형 변수를 사용하는 연산을 int형 변수를 사용하는 연산으로 변환한다. 부동 소수점 연산은 CPU의 산술 논리 장치(Arithmetic logic unit,ALU)에서 수행 시 FPU(Floating point unit)의 기능을 모방(emulate)하여 수행되기 때문에 별도의 FPU가 있지 않은 컴퓨터에서는 연산속도가 매우 느리다[5]. 현재 대부분의 컴퓨터들이 FPU를 가지고 있지만 몇몇 임베디드 시스템의 경우 FPU의 성능이 범용 시스템의 FPU에 비해 떨어지거나 FPU를 가지고 있지 않음으로 임베디드 시스템용 소프트웨어 성능에 많은 영향을 준다. 최적화 기술은 컴파일러가 수행할 수 없는 기술로 최적화를 수행하는 시스템의 하드웨어적인 특성에 따라 사용 여부를 결정한다.

2. 코드 변환 및 실험

2.1 코드 변환

본 논문에서 최적화 기술을 적용하기 위해 사용된 코드는 2-D 이미지 처리 소프트웨어다.

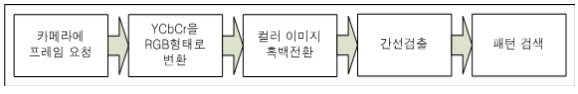

그림 1. 2-D 이미지 처리 소프트웨어 알고리즘
Fig. 1. Image processing software algorithm

그림 1과 같이 5개의 단계를 거치며 4개의 함수로 구현되어 있다.

첫째 함수는 카메라에 프레임을 요청하여 이미지를 받는다. 둘째 함수는 첫 번째 함수에서 받아온 프레임을 YCbCr 형태의 이미지에서 RGB형태의 이미지로 변환한다. YCbCr과 RGB간의 변환

표 1. YCbCr과 RGB간 변환 공식 소스 코드 - 부동 소수점 연산
Table 1. Conversion between YCbCr and RGB formulas - floating point operation

```
pix_1 = (int) (Y11 + 1.402*(Cr-128));
pix_2 = (int) (Y12 + 1.402*(Cr-128));
pix_3 = (int) (Y21 + 1.402*(Cr-128));
pix_4 = (int) (Y22 + 1.402*(Cr-128));
pix_1 += (int) (Y11 - 0.344*(Cb-128) - 0.714*(Cr-128));
pix_2 += (int) (Y12 - 0.344*(Cb-128) - 0.714*(Cr-128));
pix_3 += (int) (Y21 - 0.344*(Cb-128) - 0.714*(Cr-128));
pix_4 += (int) (Y22 - 0.344*(Cb-128) - 0.714*(Cr-128));
pix_1 += (int) (Y11 + 1.772*(Cb-128));
pix_2 += (int) (Y12 + 1.772*(Cb-128));
pix_3 += (int) (Y21 + 1.772*(Cb-128));
pix_4 += (int) (Y22 + 1.772*(Cb-128));
```

표 2. YCbCr과 RGB간 상호 변환 공식 소스 코드 - 고정 소수점 연산
Table 2. Conversion between YCbCr and RGB formulas - fixed point operation

```
pix_1 = (int) (Y11*1000 + 1402*(Cr-128))/1000;
pix_2 = (int) (Y12*1000 + 1402*(Cr-128))/1000;
pix_3 = (int) (Y21*1000 + 1402*(Cr-128))/1000;
pix_4 = (int) (Y22*1000 + 1402*(Cr-128))/1000;
pix_1 += (int) (Y11*1000 - 344*(Cb-128) - 714*(Cr-128))/1000;
pix_2 += (int) (Y12*1000 - 344*(Cb-128) - 714*(Cr-128))/1000;
pix_3 += (int) (Y21*1000 - 344*(Cb-128) - 714*(Cr-128))/1000;
pix_4 += (int) (Y22*1000 - 344*(Cb-128) - 714*(Cr-128))/1000;
pix_1 += (int) (Y11*1000 + 1772*(Cb-128))/1000;
pix_2 += (int) (Y12*1000 + 1772*(Cb-128))/1000;
pix_3 += (int) (Y21*1000 + 1772*(Cb-128))/1000;
pix_4 += (int) (Y22*1000 + 1772*(Cb-128))/1000;
```

공식은 부동소수점인 float형 변수를 이용해 연산한다. 그러므로 형변환을 통하여 표 1의 코드의 부동 소수점인 float형 변수를 이용한 연산을 표 2의 고정 소수점인 int형 변수를 이용한 연산으로 변환한다. 셋째 함수는 RGB형태로 변환된 이미지를 흑백으로 전환시킨 후 간선을 검출한다. 이미지에서 간선을

표 3. 3x3 matrix 필터
Table 3. 3x3 matrix filter

```
float val=0;
....
for(m=-1;m<2;m++){
for(n=-1;n<2;n++){
    val+=kernel[m+1][n+1]*processImage[j+m][i+n];
    }
}
```

표 4. 3x3 matrix 필터 loop unrolling과 형변환
Table 4. 3x3 matrix filter loop unrolling and type casting

```
int val=0;
....
val+=kernel[0][0]*processImage[j-1][i-1];
val+=kernel[0][1]*processImage[j-1][i];
val+=kernel[0][2]*processImage[j-1][i+1];
val+=kernel[1][0]*processImage[j][i-1];
val+=kernel[1][1]*processImage[j][i];
val+=kernel[1][2]*processImage[j][i+1];
val+=kernel[2][0]*processImage[j+1][i-1];
val+=kernel[2][1]*processImage[j+1][i];
val+=kernel[2][2]*processImage[j+1][i+1];
```

검출하기 위해 3x3 행렬 필터를 사용하는데 3x3 행렬 필터는 표 3의 코드로 각각 3번씩 반복되는 2개의 중첩루프로 구성되어 있다. 이 2개의 중첩 루프 내의 명령어는 3x3 배열 2개를 참조하여 하나의 변수에 더하는 작업을 수행한다. 중첩 루프 내의 명령어는 반복에 상관없이 독립적임으로 표 4의 코드로 루프 펼침으로 변환이 가능하다. 각각의 루프에서 루프 펼침 인자는 3으로 하며 중첩 루프 내의 명령어가 9번 복제되어 나타나고 루프문은 사라진다. 이때명령어의 복제로 인한 레지스터 과도 사용은 발생하지 않으며 명령어 캐시 미스에 영향을 미치지 않는다. 또한 복제된 명령어의 결과값이 부동 소수점 인 float형 변수에 반복적으로 더해져 저장됨으로 이를 표 4의 코드처럼 고정 소수점인 int형 변수를 이용한 연산으로 변환하여 부동 소수점 연산의 실행을 줄인다. 넷째 함수는 각각의 이미지 픽셀에 대해 모든 주변 그룹 이미지에서 패턴이 나타날 수 있는 모든 조합을 탐색한다.

2.2 실험 내용

본 실험은 삼성의 모바일 프로세서인 S5PV210이 탑재된 hybus의 H-AndroSV210에서 실험하였다. S5PV210은 ARM Cortex-A8 기반의 CPU로 NEON[6]을 지원하며 Core speed는 833MHz이다. OS는 android 2.2 프로요를 사용한다.

실험에 사용된 데이터는 특정 패턴이 포함되고 일정한 노이즈가 있는 이미지 데이터를 일정한 시간동안 받아 처리한다. 5분동안 처리되는 이미지 데이터들에 대하여 각각의 평균 실행 시간을 측정하였다. 전체 응용 프로그램에서 실제 코드변환이 이루어진 부분을 포함하여 실행 속도를 측정하였다. 크로스 컴파일러는 Sourcery G++ Lite 2009q3-68 for ARM EABI를 사용하였다. 컴파일 옵션은 모든 경우에 O2와 mfpu설정을 neon으로 하고 mfloat-abi옵션을 softfp로 하여 NEON기능을 사용할 수 있게 하였다.

2.3 실험 결과

루프 펼침을 한 후 루프 내의 명령어 결과 연산을 고정 소수점인 int형을 이용한 연산으로 형변환 한 후 실행 속도를 보면 변경 전보다 약 22%의 성능향상 효과가 있음을 알 수 있다. 그리고 YCbCr과 RGB간 상호변환 공식을 고정 소수점인 int형을 이용한 연산으로 형변환을 추가로 하였을 경우 코드의 변경전 보다 55%의 성능향상 효과가 있음을 알 수 있다. 루프 펼침이라는 최적화 기술은 임베디드 시스템 상에서 효과가 있다. 또한 ARM Cortex-A8기반의 CPU에서 NEON을 이용하여 부동 소수점 연산을 가속하였음에도 불구하고 형변환을 수행하면 성능이 개선됨을 볼 때 임베디드 시스템에서 부동 소수점 연산 가속기능을 사용하여 부동 소수점 연산을 수행 하는것보다 고정 소수점 연산으로 변환하여 수행하는 것이 성능향상에 더 효과적임을 알 수 있다.

표 5. H-AndroSV210(ARM Cortex-A8)상에서의 성능 실험 결과
Table 5. Experimental results over the performance of H-AndroSv210(ARM Cortex-A8)

	실행속도(usec)	
변경전	167960.8	100%
loop unrolling과 루프 내의 명령어 결과 형변환	131784.5	78%
loop unrolling 과 루프 내의 명령어 결과 형변환 적용 후 YCbCr과 RGB간 상호변환 공식 형변환	76474.9	45%

Ⅳ. 결론

본 연구를 통해 기존의 최적화 기술인 루프 펼침과 부동 소수점 연산을 고정 소수점 연산으로 바꾸는 형변환이 임베디드 시스템 상에서 성능향상에 어떤 영향을 미치는지에 대해 실험을 수행하였다. 범용 시스템과 많은 측면에서 다른 구조를 가진 임베디드 시스템에서도 루프 펼침으로 성능향상을 가질 수 있다는 것을 알 수 있다. 또한 실험을 통해 부동 소수점 연산을 고정 소수점 연산으로 형변환 하는 것이 임베디드 시스템에서 제공하는 부동 소수점 연산 가속 유닛을 사용하는 것보다 성능향상에 효과적이라는 것을 알 수 있다.

향후 연구를 통해 루프 펼침을 제외한 다른 범용 컴퓨팅 환경에서의 최적화 기술이 임베디드 시스템에서 성능향상에 어떤 영향을 미치는지 알고자 한다. 그리고 부동 소수점 연산이 임베디드 시스템에서 성능에 미치는 영향에 대해 추가적으로 연구하여 하드웨어에 적합한 최적화 시스템을 개발하는 것을 최종 목표로 한다.

acknowlegement

본 논문은 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발산업(정보통신)의 일환으로 수행하였음. [KI001824, 장애인 및 고령자를 위한 Digital Guardian 기술개발]

참고문헌

[1] John L. Hennessy, and David A. Patterson, “Computer Architecture: A Quantitative Approach” 4th Ed.Morgan Kaufmann, pp. 2-7, 2007.

[2] David F. Bacon, Susan. L. Graham, and Oliver J. Sharp, “Compiler Transformations for High-Performance Computing”

ACM Computing Surveys, Vol. 26. No. 4, Dec 1994.

[3] J. Bungo, “The Use of Compiler Optimizations for Embedded Systems Software” ACM crossroads Vol. 15, No. 1, Fall 2008.

[4] Yunyang Dai, Qing Li, Qi Zhang and C.-C. Jay Kuo, “SIMD - efficient loop unrolling design for embedded multimedia applications” ICME'04, June 2004.

[5] Floating point unit, http://en.wikipedia.org/wiki/Floating-point_unit

[6] ARM Limited “Cortex-A8 Technical Reference Manual” ARM limited, May 2010.