x86 과 RISC-V 아키텍처에서의 SIMD 벡터 연산 성능 효율성 분석에 관한 연구

천혜수 [1], 김성민 [2]
[1] 성신여자대학교 융합보안공학과 석사과정
[2] 성신여자대학교 융합보안공학과 교수

220246058@sungshin.ac.kr, sm.kim@sungshin.ac.kr

A Comparative Study on the Performance of SIMD Vectorization in x86 and RISC-V Architectures

Hyesu Cheon[1], Seongmin Kim[2]
[1]Dept. of Convergence Security Engineering, Sungshin Women's University
[2]Dept. of Convergence Security Engineering, Sungshin Women's University

요 약

고성능 컴퓨팅을 위한 기술로 병렬 컴퓨팅의 중요성이 대두되고 있다. 이에 이 논문에서는 널리 사용되고 있는 아키텍처인 x86 아키텍처와, 오픈 소스와 전력 효율의 장점을 가지고 최근 각광받는 RISC-V 아키텍처에서 병렬 연산을 이용한 4x4 행렬 곱셈의 성능을 비교 분석하였다. 그 결과 RISC-V 의 최적화가 추가적으로 필요함을 확인하였다. 본 연구는 RISC-V 의 벡터 연산 성능을 향상시키기 위해 향후 연구 방향을 제시한다.

1. 서론

컴퓨터 아키텍처는 늘어나는 고성능 컴퓨팅 수요와 함께 지속적으로 발전해 왔으며, 병렬 컴퓨팅 기술은 처리해야 하는 데이터의 양이 방대해진 현대의 정보처리 시스템에서 고성능 컴퓨팅을 위한 핵심적인 기반 기술로 활용되어 왔다. 병렬 컴퓨팅은 동시에 다수의 연산을 병렬화하여 처리를 가능케 하여 어플리케이션의 성능 향상 및 효율적 전력 소모 등의 이점을 제공한다. 소프트웨어 레벨에서의 병렬 컴퓨팅 발전뿐만 아니라, 하드웨어 상에서의 효율적인 병렬 처리를 위해 컴퓨터 아키텍처 기술 또한 꾸준히 발전되어 왔으며, 대표적인 기술로는 SIMD (Single Instruction Multiple Data) 연산이 있다.

한편, 소프트웨어의 오픈소스화 트렌드와 더불어 하드웨어 레벨에서도 RISC-V 칩으로 대표되는 개방형 명령어 집합 (Instruction Set Architecture, ISA)이 이 등장함에 따라, 고성능 컴퓨팅 환경부터 경량 임베디드 시스템을 아울러 활용될 수 있는 유연한 마이크로아키텍처 설계가 가능해졌다. 이를 바탕으로, 사물 인터넷 시대의 다양한 산술 장치들에 대해 경량화되었으면서도 고성능 및 저전력을 보장하기 위한 시스템 설계를 위한 연구들이 활발히 이루어지고 있다.

최근 RISC-V 와 같은 개방형 명령어 집합 내에서도 병렬 컴퓨팅 관점에서의 기능적 확장성을 지원하고 있다. 전통적인 서버 및 데스크톱 환경인 x86 아키텍처에서 SIMD 구현을 위해 MMX, SSE, AVX 등을 지원하는 것과 마찬가지로, RISC-V 환경에서는 v-type 의 벡터 확장을 지원한다. 그러나 이러한 RISC-V 환경에서의 벡터 연산 병렬화의 효율성 및 이를 기반으로 한 어플리케이션 구현의 효율화와 관련된 연구는 심층적으로 이루어지지 않았으며, 기술적 성숙도가 높은 x86 아키텍처 대비 최적화가 충분히 이루어지지 않은 실정이다.

이에, 본 논문에서는 x86 과 RISC-V 환경에서 구현된 벡터 연산 프로그램과 스칼라 연산 프로그램의 성능을 비교 분석하고자 한다. 이에 대한 기초 연구로, 벡터 연산을 수행하는 마이크로벤치마크에 대해, 두 아키텍처에서 벡터 확장을 적용한 경우와 그렇지 않은 경우를 시뮬레이션을 통해 비교 분석하였다. 명령어 레벨에서의 메트릭들을 측정하여 RISC-V 아키텍처에서의 벡터 확장이 어플리케이션의 성능 향상에 효율화를 x86 아키텍처 대비 충분히 보장하는지 분석한 결과, RISC-V 아키텍처에 추가적인 병렬 프로그래밍 최적화가 필요함을 확인하였다.

2. 배경 지식

ISA 는 명령어 집합 아키텍처를 뜻하며 명령어와 데이터의 형식, 레지스터와 메모리에 접근하는 방식 등을 정의한다. ISA 는 크게 CISC, RISC, VLIW 로 분류되며 본 논문에서 비교 분석하는 x86 과 RISC-V 는

각각 CISC, RISC 에 해당한다.

X86 아키텍처는 데스크탑에서 가장 널리 사용되는 아키텍처로 다양하고 복잡한 명령어를 사용해 하나의 작업을 수행하는 CISC ISA 에 해당한다. 다만 x86 아키텍처에서도 스칼라 연산뿐 아니라 하나의 명령어로 여러 연산을 수행하는 벡터 연산을 지원하여 더욱 효율적인 작업이 가능하다. 초기 벡터 연산을 위해서는 SSE(Streaming SIMD Extensions)를 도입하였으며 128 비트 레지스터를 사용해 총 8 개의 정수를 한 번에 처리할 수 있도록 하였다. 이후 AVX 를 도입하여 레지스터의 크기를 256 비트로 확장하였다. 현재는 AVX 의 후속 확장으로 AVX-512 를 도입하여 512 비트까지 처리 가능하다[1].

RISC-V 아키텍처[2]는 누구나 사용할 수 있는 오픈소스 아키텍처로 축소된 명령어 집합을 가지고 있어 불필요한 회로를 줄이고 효율적인 전력 소비가 가능하다는 장점이 있다. RISC-V 에서는 V-type 의 벡터 확장을 이용하여 벡터 연산을 수행하며 모듈러 연산을 통해 필요한 성능에 맞게 벡터 연산 유닛의 크기와 기능을 조절하는 것이 가능하다.

3. 실험 환경

x86 과 RISC-V 에서 벡터 연산의 CPU task-clock, context switches, cpu-migrations, page faults 시간 및 횟수를 측정 지표로 설정하였다. 해당 프로그램 컴파일 및 실행은 Ubuntu 22.04 가상 환경에서 진행하였다. 모든 측정은 perf[3] 6.10 버전 하에 이루어졌으며 RISC-V 환경의 시뮬레이터는 QEMU 9.1.0 을 이용하였다.

측정 지표 Task-clock 은 작업 완료까지 걸린 시간을 ms 단위로 나타내며 실행 시간은 짧을수록 효율적인 지표이다. 이와 함께 측정되는 CPUs utilized 지표는 프로세서의 활용도를 나타낸다. Context-switches 는 여러 프로세스 사이에서 CPU 자원을 교환한 횟수를 나타낸다. CPU-migrations 의 경우 작업의 코어간 이동 횟수를 나타내며 값이 작을수록 한 코어에서 집중적으로 수행되었음을 알 수 있다. Page-faults 지표의 경우 메모리 접근 중 페이지 폴트가 발생한 횟수를 나타내며 메모리 관리가 효율적으로 이루어지는지 확인할 수 있는 지표이다.

실험을 위해 RISC-V 와 x86 각각 벡터 연산을 수행하는 코드와 두 아키텍처 모두에서 컴파일 가능한 스칼라 연산 코드 총 2 개의 코드를 작성하였다.

비교 대상이 되는 4*4 행렬 곱셈 연산을 수행하는 스칼라 연산 의사 코드는 <표 1>과 같다. A 와 B 행렬의 곱셈 결과를 C 에 저장하는 함수이다.

<표 1> scalar pseudo code

```
function matrix_multiply_c(A, B, C):
    for i from 0 to 3:
        for j from 0 to 3:
            C[i, j] = 0
            for k from 0 to 3:
                C[i, j] += A[i, k] * B[k, j]
```

RISC-V 환경에서의 행렬 곱을 위한 함수의 의사코드는 <표 2>와 x86 환경에서의 행렬 곱을 위한 의사코드는 <표 3>과 같다. RISC-V 는 함수를 어셈블리어로 작성하여 v-type opcode, x86 의 경우 avx2 명령어를 이용하도록 하였다.

<표 2> RISC-V v-type pseudo code

```
matrix_multiply_4x4(matrixA, matrixB, resultMatrix):
    load_matrix_A(matrixA, v0, v1, v2, v3)
    initialize_result_matrix(v8, v9, v10, v11)
    for each row in matrix:
        load_row_of_matrix_B(row, ft0, ft1, ft2, ft3)
        calculate_dot_product(v0, ft0, v8)
        calculate_dot_product(v1, ft1, v8)
        calculate_dot_product(v2, ft2, v8)
        calculate_dot_product(v3, ft3, v8)
        move_to_next_row_of_matrix_B()
    store_result_matrix(resultMatrix, v8, v9, v10, v11)
```

<표 3> x86 AVX2 pseudo code

```
matrix_multiply_avx2(A, B, C)
  for each row i in A
    for each column j in B
      rowA = load_unaligned_256bit_float(A[i * 4])
      colB = construct_256bit_float(B[j], B[j + 4], B[j +
             8], B[j + 12], 0, 0, 0, 0)
      mul = multiply_256bit_float(rowA, colB)
      sum = reduce_sum_256bit_float(mul)
      C[j * 4 + i] = sum
```

위의 함수를 사용하여 작성된 행렬 곱셈 코드를 각각 RISC-V 환경과 x86 환경에 맞게 gcc 를 이용하여 컴파일 하였다. 성능 분석을 위하여는 perf 툴을 이용하였으며 RISC-V ELF 파일의 경우 perf 에서 QEMU[4]로 시뮬레이션 후 성능을 측정하는 방법을 사용하였다. 시뮬레이터의 영향을 고려하여 두 환경에서 각각 벡터 연산을 사용한 경우와 그렇지 않은 경우를 비교하여 분석한다.

4. 실험 결과

<표 4> perf result

		task-clock msec	CPUs utilized	context-switches	K/sec
RISC-V	std	6.6508	0.87586	28.54	70.67244
	vec	8.8918	0.88596	31.12	38.9172
x86	std	0.4558	0.54564	23.14	50.45398
	SIMD	0.422	0.49912	19.68	43.275026
		cpu-migrations	/sec	page-faults	K/sec
RISC-V	std	0.24	35.17674	414.8	62.704026
	vec	0.46	51.90416	425.02	47.9161
x86	std	0.02	0.04324	60.68	137.33034
	SIMD	0	0	61	147.97906

Perf 를 이용하여 각각의 지표를 측정한 결과는 <표 4>와 같다.

먼저 RISC-V 에서는 CPUs Utilized 의 경우에서만 큰 차이를 보이지 않았으며 이외의 다른 지표 상으로는 스칼라 연산의 성능 이점이 더욱 큰 것으로 측정되었다. 이 경우 벡터 연산의 최적화가 충분하지 않았거나, 벡터 연산에서 발생한 오버헤드로 인해 이런 결과가 도출된 것으로 예상된다.

반면에 x86 의 경우 스칼라 연산보다 벡터 연산에서 task-clock 단축되었고 CPU 활용도 또한 낮게 측정되었다. Context-switch 횟수와 cpu-migration, page-faults 횟수의 경우에도 벡터 연산이 더 효율적인 지표를 보여주고 있음을 확인할 수 있다.

지표 측정 결과를 분석하면 RISC-V 에서 page fault 가 더 높게 나오는 이유는 x86 아키텍처에서는 많은 다수의 캐시 계층을 통해 메모리 접근 지연 시간을 줄이는 데 중점을 두지만 RISC-V 의 경우 표준화된 캐시 설계가 아닌 맞춤형 캐시 구조를 설계하기 때문에 캐시 최적화가 부족하기 때문인 것으로 판단된다.

RISC-V 의 경우 실험 시 시뮬레이터를 사용하였기 때문에 task-clock 지표에 있어서 높은 지표가 측정되었다. 그와 더불어 x86 에서는 복잡한 명령어 세트를 통해 효율을 높이는 반면 RISC-V 에서는 단순한 명령어 집합 구조를 통해 연산 지연은 줄일 수 있으나 복잡한 병렬 연산을 처리하는 데 있어 효율성이 떨어지는 것으로 판단된다.

측정 결과를 최종적으로 종합하였을 때 오픈소스의 특성으로 인하여 유연성의 장점을 갖는 RISC-V 환경에서의 병렬 연산 최적화가 x86 최적화에 비해 부족하기에 비효율적임을 확인하였다.

5. 결론 및 향후 연구

이 연구에서는 두 아키텍처 상에서의 병렬 연산 성능을 비교 분석하였다. 성능 측정 결과, x86 아키텍처는 벡터 연산을 통해 성능을 효과적으로 향상시켰으며 작업 시간 단축 및 CPU 자원 활용 측면에서 최적화가 이루어졌으나 RISC-V 는 벡터 연산 도입에도 불구하고 성능 향상이 거의 이루어지지 않았으며 오히려 스칼라 연산보다 비효율적인 결과가 측정되었다. 다만 RISC-V 는 x86 에 비해 단일 표준화된 설계가 아닌 프로세서의 사용 목적에 맞는 맞춤형 캐시, 메모리 설계를 통해 보다 효율적으로 활용할 수 있기에 향후에 RISC-V 최적화 관련 연구가 더 이루어지기를 기대한다.

이번 실험의 한계로는 4*4 사이즈의 비교적 단순한 벡터 연산이기 때문에 병렬 연산보다 스칼라 연산이 효율적으로 작동했을 수 있다는 점과 C 언어로 작성된 x86 프로그램과 어셈블리 사이 포팅이 제대로 이루어지는지 검증 과정이 미흡했다는 점이 있다.

향후 연구에서는 RISC-V 아키텍처에서의 향상된 보안 어플리케이션 구현을 RISC-V 아키텍처에서의 벡터 연산 최적화, 메모리 관리 최적화에 관한 연구를 진행할 예정이다.

Acknowledgement

본 논문은 2024 년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(No. RS-2024-00351898)과 정보통신기획평가원의 지원 (No. IITP-2022-RS-2022-00156310), 산업통상자원부 및 한국산업기술진흥원의 지원(No. RS-2024-00415520)을 받은 연구결과로 수행되었음

참고문헌

[1] Intel, 12th Generation Intel® Core™ Processors, Datasheet, Volume 1 of 2, 47, 2023

[2] RISC-V International – RISC-V : The Open Standard RISC Instruction Set Architecture, https://riscv.org, Sep 18. 2024

[3] brendangregg/perf-tools: Performance analysis tools based on Linux perf_events (aka perf) and ftrace, https://github.com/brendangregg/perf-tools, Sep 18. 2024

[4] QEMU, https://www.qemu.org, Sep 18. 2024