SIMD 벡터 명령어를 이용한 다차원 레코드 스캔

(Multi-Dimensional Record Scan with SIMD Vector Instructions)

조 성 룡 † 한 환 수 ††
(Sung-Ryong Cho) (Hwansoo Han)

이 상 원 †††
(Sang-Won Lee)

요 약 대량의 데이터 처리 영역에 대한 중요성이 증가하는 가운데 다차원의 속성을 갖는 레코드에 대한 스캔을 필요로 하는 질의처리에 있어 SIMD 명령어 셋을 이용하여 보다 효율적인 스캔성능을 얻을 수 있다. 이러한 배경하에서 이 논문에서 제시하는 기법인 'SIMD 레코드 스캔'은 행-기반의 스캔으로 열-기반의 저장구조를 갖는 기존의 메모리 기반 데이터베이스 시스템에서 조건식 처리나 집계연산 등에서의 연산성능을 높이기 위해 열에 종속적으로 SIMD 명령어를 이용하던 것과는 달리 다차원 속성들의 비교가 요구되는 레코드 스캔에서의 효율을 높일 수 있다. 이는 레지스터 및 시스템 메모리의 크기가 증가함에 따라 더 큰 성능향상을 가져올 수 있으며, 멀티코어 기반의 병렬화 기법과 독립적이므로 SIMD를 지원하는 단일 프로세서뿐 아니라 이들로 구성된 멀티코어 프로세서에도 기존 시스템이나 아키텍처를 변경하지 않고도 적용이 가능하다.

키워드 : 다차원 레코드 스캔, SIMD 명령어, 메모리 기반 데이터베이스 질의 처리

Abstract Processing a large amount of data becomes more important than ever. Particularly, the information queries which require multi-dimensional record scan can be efficiently implemented with SIMD instruction sets. In this article, we present a SIMD record scan technique which employs row-based scanning. Our technique is different from existing SIMD techniques for predicate processes and aggregate operations. Those techniques apply SIMD instructions to the attributes in the same column of the database, exploiting the column-based record organization of the in-memory database systems. Whereas, our SIMD technique is useful for multi-dimensional record scanning. As the sizes of registers and the memory become larger, our row-based SIMD scan can have bigger impact on the performance. Moreover, since our technique is orthogonal to the parallelization techniques for multi-core processors, it can be applied to both uni-processors and multi-core processors without too many changes in the software architectures.

Key words : multi-dimensional record scan, SIMD instruction, in-memory database query processing

1. 서 론

대량의 데이터에 대해 복잡한 처리를 필요로 하는 근래의 컴퓨터 환경에서 여러 속성, 즉 다양한 선택요소들을 종합적으로 비교하여 보다 우월한 선택을 하고자 하는 경우라면 레코드의 차수, 즉 비교 속성의 개수(dimension)가 증가함에 따라 연산속도가 크게 달라질 수 있다. 최근 멀티코어프로세서 환경에서 이런 유형의 데이터 처리에 있어 성능을 향상시키기 위한 병렬화를 어떻게 구현할 것인가에 대해 여러 기법들이 고안되고 있다. GPUs(Graphic Processing Units)를 이용한 데이터베이스 연산수행속도의 가속화나[1], 병합-정렬 등의 기본 알고리즘에 SIMD 명령어를 적용하여 조인 알고리즘의 성능을 높이는 방법[2,3], 다차원 속성을 고려한 경우로는 병렬 스카이라인(parallel skyline)을 그 예로 들 수 있다[4].

특히 병렬 스카이라인은 다차원의 속성에 대한 스캔 효율을 높이기 위하여 스카이라인[5] 알고리즘을 멀티코어환경에 확장시켜 적용한 예로써 대량의 데이터 집합을 멀티코어를 구성하는 각 프로세서에 분산시켜 같은 CPU 시간 동안 최대한 많은 처리를 하도록 함으로써 수퍼스칼라 단일 프로세서에서 명령어 수준의 병렬화(ILP: instruction level parallelism)를 이용하는 것만으로는 극복하기 힘들었던 문제를 해결하고 있다. 그러나 다차원의 속성들로 구성된 레코드를 다루는 경우라면 멀티코어의 프로세스 수만큼 병렬화된 하드웨어 쓰레드를 이용하는 것 외에도 SIMD 명령어를 이용한 데이터

· 이 논문은 제36회 추계학술발표회에서 '효율적인 다차원 in-Memory Scan을 위한 SIMD instruction의 활용'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 성균관대학교 정보통신공학부
applys@skku.edu
†† 종신회원 : 성균관대학교 정보통신공학부 교수
hhan@skku.edu
††† 정 회 원 : 성균관대학교 정보통신공학부 교수
wonlee@ece.skku.ac.kr

논문접수 : 2009년 12월 24일
심사완료 : 2010년 3월 5일

수준의 병렬화(DLP: data level parallelism)를 통해 성능을 더욱 향상시킬 수 있다. SIMD(Single Instruction, Multiple Data)는 하나의 명령어로 레지스터에 담긴 여러 데이터를 벡터처럼 취급하여 병렬적으로 처리하는 명령어 셋을 갖는 아키텍처를 말한다.

본 논문은 인텔, AMD, 썬, 컴팩 등 여러 회사들이 지원하고 있는 SIMD 명령어 셋을 이용하여 다수의 속성을 갖는 레코드로 구성된 릴레이션에 대해 열-기반(column-based)[6]이 아닌 행-기반(row-based)의 DLP를 구현하고, 레코드에 대해 적용함으로써 효율적인 다차원 스캔이 가능함을 보인다. 논문에서 제시하는 스캔기법은 SIMD 명령어를 지원하는 대부분의 단일 프로세서에 적용가능하며 이들 프로세서 여러 개로 구성된 멀티코어 시스템에도 적용이 가능하다. 본 논문에서는 x86 아키텍처의 SIMD 명령어를 활용하여 성능평가를 수행하였으며 이는 인텔과 AMD사의 프로세서가 공통적으로 지원하고 있는 SSE/SSE2/SSE3 명령어들로 구성되어 있다.

이 논문에서의 결과는 앞으로 SIMD 레지스터의 크기가 128비트에서 512, 1024 등으로 더욱 커질 경우 적용여하에 따라 더욱 큰 성능 향상을 가져올 수 있다. 더욱이 기존 아키텍처를 따로 변경하지 않고도 적용가능하고, 병렬 스카이라인과 같은 멀티코어 환경에서의 최적화 알고리즘과 독립적인 기술이므로 이에 대한 적용도 가능하다.

이후의 기술 내용을 정리하면, 2장에서는 SIMD 연산에 대한 배경지식에 대해 간략히 논의하고 3장에서는 이 논문의 주제인 SIMD 명령어들을 이용한 레코드 스캔을 설명하고, 4장에서는 성능평가와 실험결과를 제시하고, 마지막으로 5장에서 결론을 맺는다.

2. SIMD 연산

SIMD 연산은 피연산자로 데이터를 저장할 레지스터(이하 SIMD 레지스터)와 SIMD 명령어들을 이용하여 수행된다. SIMD 레지스터에는 레지스터의 크기에 해당하는 만큼의 데이터들이 벡터화되어 저장될 수 있으며 저장의 오류를 방지하고 이후의 명령어적용시의 속도가 저하되는 것을 막기 위해 사전에 경우에 따라 바이트를 정렬(align)해주는 작업이 필요하다. 따라서 기본적인 연산 및 마스크(mask), 셔플(shuffle)등 다양한 연산들에 대해 해당 명령어가 어떤 레지스터를 사용하는지, 그 레지스터에 어떻게 정렬되어 저장되는지 고려해야 한다.

SIMD 명령어들 대부분은 레지스터 간(register to register) 혹은 레지스터 셔플링(register shuffling)을 통해 목적 레지스터에 해당 연산의 결과를 저장한다. 따라서 연산의 결과를 원하는 값으로 추출하기 위해서는 해당 명령어의 수행 결과가 어떻게 반영되는지를 고려해야 한다. 특히 마스킹(masking)을 통해 이후 결과에 대한 조건분기 등을 수행하는 경우는 연산을 통해 나온 결과로 어떤 타입의 값들이 반환되며 반환값으로는 몇 비트가 사용되고 있는지를 고려하여 SIMD 명령어를 사용하여야 한다.

본 논문에서 성능평가를 위해 128비트 크기의 SIMD 레지스터 16개를 지원하는 프로세서를 사용하였고 32비트 크기의 레코드 속성을 비교하므로 SIMD 연산 수행으로 얻어내는 마스크 값(mask value)은 4비트의 유의미한 값을 갖는다.

3. SIMD를 이용한 레코드 스캔

데이터베이스 시스템의 질의 처리과정에 있어 최적화 단계를 거쳐 만들어지는 관계대수(그래프 등)의 시작점(leaf node)은 보통의 경우 해당 레코드, 혹은 열(column)에 대한 조건검색을 통한 스캔으로 구성된다. 여기서 스캔은 범위스캔(range scan), 혹은 포인트 스캔(point scan)을 통해 대량의 데이터(릴레이션에 존재하는 레코드 집합 혹은 칼럼의 해당 값)에 대한 부분 집합을 추출하는 것을 의미한다.

이 논문에서 대상으로 하는 스캔은 두 비교 대상 레코드의 비교속성 값에 대해 비교연산을 수행하여 나온 결과를 바탕으로 결과집합에 포함시킬지 여부를 판단하는 일반적인 if then else 형태의 조건비교연산의 다차원적으로의 확장이다. 속성의 수가 적은 경우, SIMD 연산을 사용하지 않는 일반적인 스캔과 이 논문에서 제시하는 SIMD 레코드 스캔이 성능상 그리 큰 차이를 보이지 않는다. 이 경우, SIMD 스캔의 주된 효과는 다차원의 비교연산에 있어서 조건분기의 횟수를 감소시켜 분기예측오류(branch misprediction)가 야기하는 CPU 지연(latency)을 줄여 주는데 있기 때문이다[6].

3.1 일반적인 레코드 스캔

만약 어떤 릴레이션 R이 메모리에 존재하고, R을 구성하는 두 레코드 r1, r2에 대해 각각의 속성을 기준으로 단순비교를 수행한 뒤 해당 조건에 맞는 결과집합을 산출한다고 할 때, 레지스터에 데이터를 읽어 들이고 수행결과를 저장하는 것을 제외한 조건수행의 처리 자체만을 그림으로 나타내보면 그림 1과 같다. 즉 그림에서와 같이 각 차원에 해당하는 속성의 수만큼 루프를 돌면서 해당 차원의 조건에 해당하는지를 판단하게 되는 것이다.

이 경우 차원이 확장될수록, 즉 비교대상이 되는 속성의 개수가 증가할수록 비교레코드와 기준레코드의 각 속성끼리 비교연산을 모두 수행한 후 해당 결과가 최종 결과집합에 포함되는지 여부를 판단하는 경우는 차원의 수에 해당하는 만큼의 비교연산이 레코드 수만큼 반복하여 수행된다. 따라서 이를 SIMD 명령어를 이용하여 벡터화하게 되면 비교연산의 횟수가 벡터화된 요소의

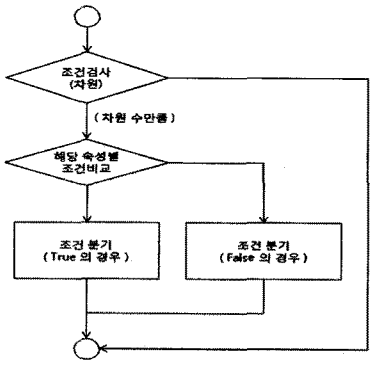


그림 1 다차원 속성에 대한 스칼라 조건분기 처리과정

수만큼 감소하게 되며 조건분기의 횟수를 줄여 성능의 향상을 가져올 수 있다.

3.2 SIMD 명령어를 이용한 스캔

본 논문에서는 SIMD 명령어를 이용하여 스캔 메소드(scanning method)를 구성할 때, 단순화를 위해 벡터화 되는 각 차원은 32비트 실수 값을 갖는다고 가정한다. 즉 해당 레코드의 각 속성이 갖는 도메인은 32비트 실수 값 범위내의 임의의 값을 갖는다. 따라서 용도의 제한이 발생하게 되나 다른 크기나 다른 타입으로의 확장이 쉽게 가능하므로 제한된 속성을 사용하는 본 논문의 결과도 여전히 적용될 수 있다.

그림 2는 SIMD 명령어를 이용하여 다차원의 속성을 갖는 레코드에 대한 스캔을 하는 경우를 나타낸 것이다.

그림 2에서 '조건검사 및 벡터로딩'의 횟수는 대상 레코드의 차원수를 해당 벡터요소의 수만큼 균등하게 분할한 값으로 예를 들어 차원이 16인 경우는 4, 32인 경우 8이 된다. 이것만으로도 컴파일 시점에서 최적화 옵션을 수행하지 않고도 일정부분 자동으로 루프가 언롤링(unrolling)되는 효과를 얻을 수 있다. 이는 SIMD 레지스터를 사용하기 위해 다차원의 속성을 벡터요소의 개수만큼씩 한번에 읽어 들여 처리하는 SIMD 명령어 사용시 얻어지는 효과이다.

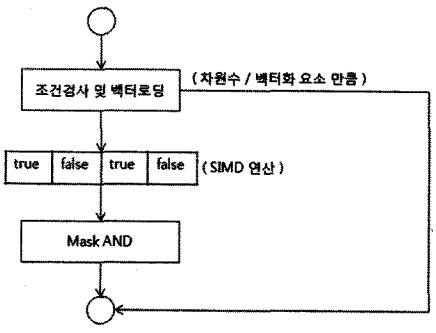


그림 2 다차원 속성에 대한 SIMD 조건분기처리 과정

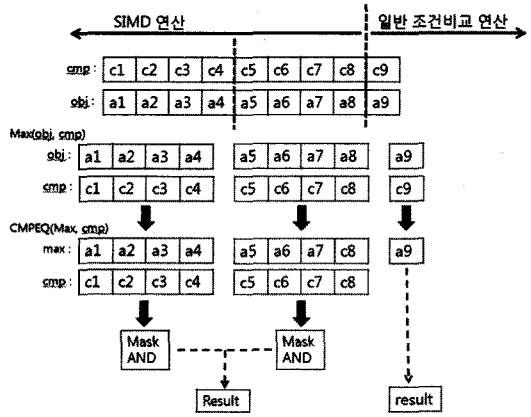


그림 3 SIMD 연산 단계의 처리과정

그림 3의 'SIMD 연산' 부분은 일반적인 조건비교와 동일한 결과를 만들어내는 SIMD 명령어들의 연산조합으로 하나의 마스크 값을 얻어내기까지의 과정을 요약하여 나타낸 것이다. 마지막 단계인 'Mask AND' 부분은 해당 마스크 값이 벡터 요소의 수에 따라 미리 정해진 상수 값과 같은지를 판단하는 부분으로 AND 연산을 수행하여 결과를 얻어낸다.

그림 3은 본 논문에서 제시하는 SIMD 연산(2단계)의 처리과정에 대한 예시이다. 9개의 속성을 갖는 레코드의 경우로 한번에 4개의 속성에 대한 비교가 가능하다고 할 때(레지스터 크기/데이터 크기), SIMD 루프가 2회, SIMD 연산을 하지 않는 나머지 속성 a9에 대한 비교연산이 1회 수행되게 된다. 이는 일반적인 비교수행시의 9회에 비해 3분의 1의 비교연산 수행만으로 최종 결과를 얻어낼 수 있음을 의미하며 조건분기의 횟수 또한 크게 줄어들게 된다.

이처럼 SIMD 스캔을 이용하여 일반적인 스캔과 동일한 결과를 산출하기 위해 수행되는 연산의 단계는 크게 3부분으로 나누어 요약해 볼 수 있다.

1단계 조건검사 및 벡터로딩: 입력으로 받아들인 레코드의 속성값들을 벡터화하여 SIMD 레지스터에 로딩하는 부분.

2단계 SIMD 연산: SIMD 명령어들의 조합을 이용, 해당 조건을 수행하고 최종 마스크 값을 생성하는 부분.

3단계 Mask AND 연산: 2단계에서 산출된 마스크 값과 기준이 되는 상수 값과의 AND 연산을 통해 해당 레코드를 결과집합에 포함시킬지 여부를 판단하는 부분.

여기서 2단계는 조건이 어떠한가에 따라 각기 다른 연산이 수행될 수도 있으나 레코드간 다차원의 속성값에 대한 크기비교가 일반적인 기준이라고 보면 3~4개 이하의 유사한 명령어들의 구성만으로도 마스크 값을 생성해 낼 수 있을 것이다.

3.3 인트린직 코드를 이용한 SIMD 스캔의 구성

이해를 돕기 위해 위의 각 단계를 통해 어떻게 SIMD 스캔이 이루어지는가를 인트린직 코드(intrinsic code)를 이용하여 간단하게 구성해 보면 다음과 같다.

```
0: __128m obj, cmp, dst, msk ;
1: obj = _mm_load_ps (aligned input_obj) ;
2: cmp = _mm_load_ps (aligned input_cmp) ;
3: dst = _mm_max_ps(obj, cmp) ;
4: msk = _mm_cmpeq_ps(dst, cmp) ;
5: movemask = _mm_movemask_ps(msk) ;
6: constant_mask_value AND movemask ;
```

위에서 3~5라인은 2단계 'SIMD 연산'부분에 해당하는데, 이 예시는 레코드를 구성하는 모든 속성에 대해 obj가 비교대상인 cmp보다 우월하여 결과집합에 포함될 것인가를 결정하기 위해 하나의 마스크 값을 얻어내는 경우이다. 경우에 따라서 마스크 값은 하나가 아닐 수도 있고 비교의 방향에 따라 사용되는 명령어 조합의 구성에 약간의 차이가 있을 것이나 기본적인 구성방식이나 사용되는 명령어의 수는 위와 크게 달라지지 않는다.

본 논문에서 사용된 벡터요소의 수는 그림 2와 그림 3에서 보는 바와 같이 4개이다. 이는 32비트 실수 값이 128비트 레지스터에 4개까지 저장될 수 있기 때문이며 비교에서 사용되는 속성의 크기에 따라 다르게 설정된다. 256비트, 혹은 인텔 Larabee GPU처럼 512비트의 SIMD 레지스터를 갖는 경우는 32비트 데이터의 경우 16개, 64비트 데이터는 8개까지 벡터화할 수 있으므로 SIMD 레지스터의 크기가 커질수록 더욱 큰 성능의 향상을 얻을 수 있다.

4. 성능평가 및 실험결과

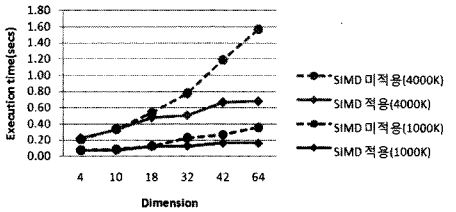
4.1 실험 환경

다차원 레코드 스캔의 성능평가를 수행하는데 사용된 시스템 환경은 다음과 같다. AMD 'AthlonX2 7750 Dual core processor'를 사용하며 이는 16개의 128비트 xmm 레지스터를 지원한다. 컴파일환경은 GNU 'gcc4.3.3'을 사용하였으며 인스트럭션 메소드는 gcc에서 제공하는 라이브러리를 이용하였다.

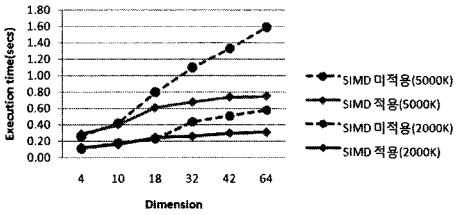
성능평가에서는 SIMD 명령어의 영향을 중점적으로 파악하기 위하여 프로세서간 데이터의 이동을 배제할 목적으로 단일 프로세서에서 수행되었으나, OpenMP등의 병렬화 라이브러리를 이용하여 멀티코어 환경에 확장시켜 적용하는 것도 가능하다.

4.2 성능평가

성능평가는 SIMD 명령어를 사용하지 않는 경우의 스캔(이하 SIMD 미적용)과 SIMD를 사용한 스캔(이하 SIMD)에 동일한 데이터 셋을 사용하여 수행한다. 데이터 셋은 4, 10, 18, 32, 42, 64개의 속성(dimension)을 갖는 레코드를 그 대상으로 하였으며 속성값으로는 32비트 부동소수점 실수형을 가정하고 있다. 성능은 측정된 수행시간을 기준으로 나타내며 결과 그래프들의 세로축은 모두 수행시간을 나타낸다.



(a) Volume = 1000K, 4000K



(b) Volume = 2000K, 5000K

그림 4 차원 별 수행시간 비교

그림 4는 SIMD 미적용(점선으로 표시)과 SIMD 적용(실선으로 표시)의 경우에 대해 레코드 셋의 크기를 (a)는 1000K와 4000K, (b)는 2000K와 5000K로 고정시키고, 비교속성의 개수(Dimension)를 증가시켜가며 수행시간을 라인 그래프로 비교하여 나타낸 것이다.

먼저, SIMD 미적용(점선)의 경우 그림에서 나타나듯 'Dimension'의 수가 증가함에 따라 거의 선형적으로 수행시간이 증가함을 알 수 있다. 이는 차원이 증가함에 따라 비교횟수가 증가하게 되고 조건분기의 오버헤드가 커짐에 따라 나타나는 당연한 결과라고 할 수 있다.

SIMD 미적용시 스캔의 비용을 Cg, SIMD 스캔의 비용을 Cs라 하고, Nr은 비교기준으로 제시된 레코드의 수, Cb는 분기가 1회 발생할 때의 CPU 관점에서의 비용요소(1회 조건분기 수행시의 지연), Di는 레코드를 구성하는 차원의 수, Vf는 벡터요소의 개수, RDi는 전체 차원을 벡터화하고 남는 차원의 수, Tv는 SIMD 레지스터에 데이터를 읽어 들이는데 따른 지연(latency) 등으로 야기되는 비용요소를 나타낸다고 할 때, 두 경우에 있어 전체비용은 다음의 식 (1), (2)와 같이 나타낼 수 있다.

$$Cg = Nr * Cb * Di \quad (1)$$

$$Cs = Nr * Cb * \{ (Di / Vf) + RDi \} + Tv * (Di / Vf) \quad (2)$$

예를 들어 Nr = 5000K, Cb = 1, Di = 42, Vf = 4, Tv = 4

라고 한다면, $RDi = 42\%\ 4 = 2$가 되고, $Cg = 5M * 1 * 42 = 210M$, $Cs = 5M * 1 * \{(42/4) + 2\} + 4 * (42/4) = 100M$ 이 된다. 따라서 SIMD가 SIMD 미적용의 경우보다 약 2.1배 가량 빠를 것으로 예상할 수 있다. 실제 측정 결과도 다른 조건이 동일하고 42인 경우를 비교해보면 SIMD(0.74초)가 SIMD 미적용(1.33초)보다 약 1.8배 빠른 것으로 나타났다. 레코드수가 3000K, 5000K인 경우의 수행시간을 비교하여 나타내 보면 다음과 같다.

표 1 SIMD와 SIMD 미적용의 수행시간 비교

차원	SIMD 미적용(secs)		SIMD (secs)		Speedup	
	3000K	5000K	3000K	5000K	3000K	5000K
4	0.16	0.26	0.16	0.29	1.00	0.90
10	0.23	0.42	0.22	0.40	1.05	1.05
18	0.39	0.80	0.35	0.61	1.11	1.31
32	0.73	1.10	0.43	0.68	1.70	1.62
42	0.79	**1.33**	0.48	**0.74**	1.65	**1.80**
64	1.17	1.59	0.49	0.75	2.39	2.12

4.3 원인분석

표 1의 결과에 나타난 것처럼(Speedup을 참조) 차원이 4인 경우 SIMD 미적용의 성능이 좋게 나오는 이유는 비교 속성의 수 자체가 작은 경우, 위 식에서 Tv 로 표현된 비용요소가 SIMD를 적용함으로써 얻을 수 있는 이점보다도 크기 때문으로 볼 수 있는데 데이터 셋의 크기에 따라 그 차이는 그리 크지 않음을 알 수 있다. 보다 주의 깊게 볼 부분은 차원이 18일 때와 32일 때, 그리고 42일 때와 64일 때 각각에 대해 SIMD 적용의 성능차이를 나타내는 그래프의 변화가 두드러지지 않는다는 점이다. 이러한 결과가 나타나게 된 이유는 벡터화되지 않은 속성(RDi)의 존재로 인해 일반조건비교를 수행하는 부분으로의 이행이 불가피하기 때문인데 이는 일반조건비교 부분을 제거하고 대신 SIMD 루프를 1회 더 증가시키면서 모자란 벡터부분에 대해서는 더미 값을 넣어 수행하거나 하는 방식으로 변환하는 것도 생각해 볼 수 있을 것이다. 그러나 실험결과와 같이 전체적으로 성능이 향상되는 것은 분기예측오류를 일으킬 가능성이 있는 조건분기의 수가 크게 감소한 것에 기인한 것이며, Tv 의 비용요소를 감안하더라도 그 효과를 상쇄시키진 못하기 때문인데 이러한 결과는 그림 5의 그래프를 보면 더욱 분명하게 알 수 있다.

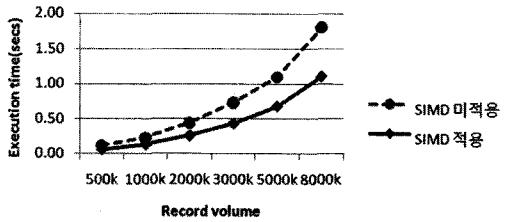


그림 5 데이터 셋의 크기 별 수행시간 비교(차원=32)

5. 결 론

지금까지 SIMD를 이용하여 다차원 속성을 갖는 레코드의 스캔에 보다 뛰어난 성능을 얻을 수 있음을 보이고 처리과정을 3단계로 구분하여, 'SIMD 연산' 단계에서 SIMD 레지스터와 명령어들의 조합을 통해 조건분기의 수를 줄여 분기예측오류를 감소시킴으로써 스캔성능의 향상을 가져올 수 있음을 보였다. 성능평가에서는 위에 언급한 성능향상의 요소들이 얼마나 구체화될 수 있는지를 레코드가 갖는 속성의 개수와 데이터 셋의 크기를 변화시켜가며 실험한 결과, 32차원 3000K의 데이터 셋의 경우 약 1.7배, 64차원 5000K의 경우 약 2.12배까지 스캔의 수행속도가 빨라졌다. 또한 벡터화되지 않는 속성의 존재가 SIMD 스캔의 성능에 저해요소가 됨을 원인분석을 통해 보였다.

결론적으로 시스템 메모리나 레지스터의 크기가 커짐에 따라, 단일 프로세서뿐 아니라 멀티코어 프로세서에 SIMD 스캔을 확장시켜 적용한다면 다차원의 속성을 갖는 레코드 스캔의 성능을 향상시킬 수 있을 것이다.

참 고 문 헌

[1] N. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha, "Fast Computation of Database Operations using Graphics Processors," *Proc. of ACM SIGMOD International Conference on Management of Data*, pp.215-226, 2004.

[2] J. Chhugani, W. Macy, A. Baransi, A. Nguyen, M. Hagog, S. Kumar, V.W. Lee, Y. K. Chen and P. Dubey, "Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture," *Proc. of the Very Large Data Base Endowment*, vol. 1 issue2, August 2008, pp.1313-1324, 2008.

[3] C. Kim, T. Kaldewey, V. W. Lee, E. Sedlar, A. D. Nguyen, N. Satish, J. Chhugani, A. D. Blas, P. Dubey, "Sort Vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs," *Proc. of the 35th International conference on Very Large Data Bases*, pp.1378-1389, 2009.

[4] S. Park, T. Kim, J. Park, J. Kim, H. Im, "Parallel Skyline Computation on Multicore Architectures," *Proc. of IEEE International Conference on Data Engineering*, pp.760-771, 2009.

[5] S. Borzsonyi, D. Kossmann, K. Stocker, "The Skyline Operator," *Proc. of the 17th International Conference on Data Engineering*, pp.421-430, 2001.

[6] J. Zhou, K. A. Loss, "Implementing database operations using SIMD instructions," *Proc. of ACM SIGMOD International Conference on Management of Data*, pp.145-156, 2002.