지배되지 않는 점을 찾는 GPU기반 병렬 알고리즘

황윤호 손완빈 안희갑
POSTECH 컴퓨터공학과 cypher@postech.ac.kr, mnbiny@postech.ac.kr, heekap@postech.ac.kr

Computing Non-Dominated Points using a GPU-Based Parallel Algorithm

Yoonho Hwang, Wanbin Son, Hee-Kap Ahn
Dept. of Computer Science and Engineering, POSTECH

요 약

본 논문은 평면 위의 점 집합에서 지배되지 않는 점 집합을 찾아내는 병렬 알고리즘을 제안한다. 우리는 먼저 기존의 지배되지 않는 점을 계산하는 문제가 SIMD형 병렬계산 장치인 GPU에서 병렬화가 가능하다는 사실을 보이고, 실제 GPU를 이용한 병렬 알고리즘을 설계·구현하였다. 또한 실험 결과 직렬 알고리즘에 비해 2배 이상의 성능 향상을 얻을 수 있었다.

1. 서 론

평면 위에 두 개의 점 집합 P(데이터 점 집합), Q(질의 점 집합) 가 있을 때, P의 한 점 p_1과 Q의 어떤 점 q_i와의 거리가 p_2와 q_i와의 거리보다 멀고, 이러한 p_1,p_2의 관계가 Q안의 모든 질의점 q_i에 대해서 성립할 때 p_1은 p_2에 의해 *지배되는 점*이라고 한다.

p_1이 P의 어떤 점에 대해서도 지배되지 않을 때, 우리는 p_1을 *지배되지 않는 점*이라고 한다..

지배되지 않는 점 집합 문제는 데이터베이스 분야에서 스카이라인 질의[*3, 4, 5, 6, 7*] 라고도 부르며, 질의에 대한 '좋은' 후보 군의 집합을 빠른 시간 내에 찾아내는 용도로 사용된다.

지배되지 않는 점 문제에 대해서 *Borzsonyi*등은[*4*] n이 P 집합의 점의 수이고, d가 차원일때 $O(n\log^{d-2}n + n\log n)$의 시간복잡도를 가지는 알고리즘을 제시하였으며, 공간상의 *지배되지 않는 점* 질의에 대한 개념은 처음 *Sharifzadeh*와 *Shahabi*[*E9*]에 의해 처음으로 제시되었다. 이 알고리즘은 *Son* 등에[2] 의해 $O(n|S|\log|CH(Q)| + n\log n)$ 으로 개선되었으며, 이때 $|CH(Q)|$는 Q를 포함하는 최소 볼록 다각형(convex hull)의 변 위에 존재하는 점의 수를 의미한다. 우리는 이 논문에서 [2]의 알고리즘을 바탕으로 병렬 알고리즘을 제시한다.

우리가 제시하는 병렬 알고리즘은 *GPU*와 같은 *SIMD*(*Single Instruction, Multiple Data*) 형태의 병렬 처리 장치를 기반으로 하고 있다. *GPU*기반으로 병렬 알고리즘을 고안하는 경우에는 여러 개의 코어에서

이 논문은 정보통신산업진흥원의 SW공학 요소기술 연구개발사업에 의해 지원 되었음을 밝힙니다.

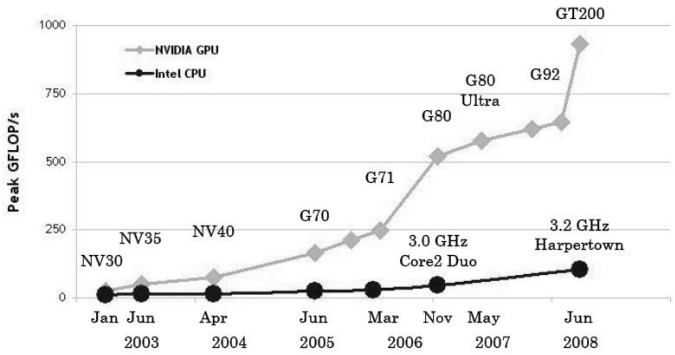

그림 1 최신형 *GPU*와 *CPU*의 계산성능차이

같은 명령어를 실행해야 하기 때문에 *CPU*기반 병렬 알고리즘보다 제약사항이 더 많지만, 일반적으로 *CPU*보다 가격 면에서 효율적이고, 계산성능 면에서도 *CPU*보다 높은 계산성능을 얻을 수 있어서[*1*] 최근에 각광받고 있는 기술이다.

실제 실험을 위한 구현은 *NVIDIA*에서 제공하는 *CUDA*를 사용하였다.

2. 본론

2.1 정의

이 논문 전반에 걸쳐서 우리는 다음과 같은 기호와 정의를 사용하겠다.

표 1 본 논문에서 사용하는 기호와 의미

기호	의미

P	데이터 점 집합, $p_i \in P, R^2$
Q	쿼리 점 집합, $q_i \in Q, R^2$
S	지배되지 않는 점 집합 $Q \subset P, s_i \in S, R^2$
R	지배되지 않는 점인지 확정되지 않은 점 집합. $r_i \in R, R \subset P$
$CH(Q)$	Q의 *convex hull*
$\mid A \mid$	집합 A 의 원소의 수
$d(p_1,p_2)$	두 점 p1과 p2사이의 거리
n	|P|

정의1: *지배하는 점*
모든 $q_i \in Q$에 대해서 $d(p_1,q_i) \leq d(p_2,q_i)$가 성립하고, 최소한 한 점 $q_i' \in Q$ 에 대해서 $d(p_1,q_i') < d(p_2,q_i')$가 성립하면 p_1은 p_2를 지배하는 점이라 한다.

정의2: *지배되지 않는 점*
$p_i \in P$를 지배하는 점이 존재하지 않을 때, 우리는 p_i를 *지배되지 않는 점*이라고 한다.

2.2 기존의 알고리즘과 문제점

*Son*등에 의해 제시된 알고리즘[2]은 다음과 같은 세 단계를 거친다.

A. P의 *Voronoi Diagram*과 Q의 Convex Hull $CH(Q)$을 구한 후, $CH(Q)$과 교차하는 *Voronoi* 셀들을 정의하는 P의 점들은 지배되지 않는 점이라는 기존의 결과 [4]를 이용해서 지배되지 않는 점들의 부분집합 S를 구한다.
B. *R-Tree*를 이용해서 P의 점들 가운데 다른 점에 의해 지배되는 점들의 부분집합을 구한다.
C. A와 B의 과정을 거친 후에 나오는 지배되지 않는/지배되는 점인지 확정되지 않은 나머지 점들의 집합을 R이라고 할 때, R의 점 각각에 대해서 특정 순서에 따라 S에 의해서 지배되는 점인지 테스트를 하고 지배되지 않는 경우 S에 추가한다.

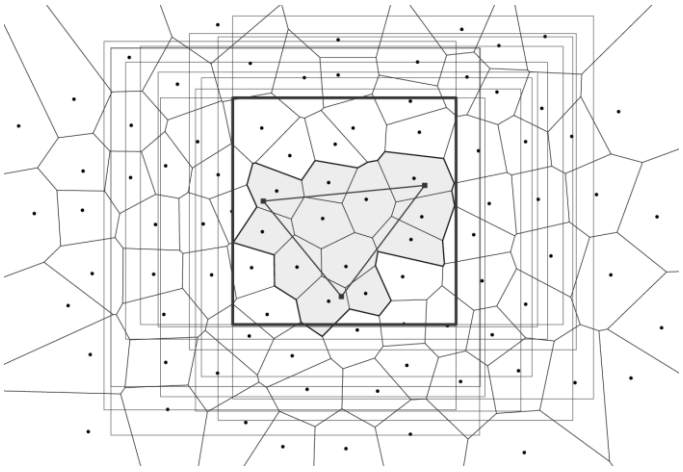

그림 2 진한 사각형 바깥쪽의 점은 *지배되는 점*이다.

이 과정의 총 시간복잡도는 $O(n|S| \log |CH(Q)| + n\log n)$이지만, *지배되지 않는 점*의 수 $|S|$가 많아지면, 알고리즘의 복잡도가 n에 대한 제곱함수에 비례하기 때문에 현저한 성능 저하를 보인다. 이러한 경우는 집합 $CH(Q)$가 집합 P의 많은 수의 점을 포함하게 될 때, 해당하는 점들이 모두 S에 속하게 되면서 발생한다.
하지만 $|S|$가 커지더라도 과정A와 과정B에서의 시간복잡도는 $O(n\log n + n\log|CH(Q)|)$ 으로 유지 된다.[2] 우리는 이 방법대로 구현한 알고리즘에서 $|S|$가 증가할 때 각각의 과정(A,B,C)에 소모되는 시간을 측정하는 실험을 수행했는데, 그림1에서 보는 것과 같이 $|S|$가 증가 했을 때는 대부분의 시간이 과정 C에서 소요된다.

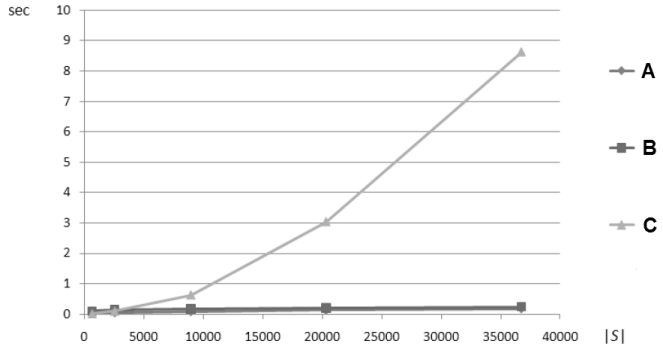

그림 3 $|S|$의 증가에 따른 *과정* A,B,C의 소요시간 변화

따라서 우리는 *GPU* 기반의 병렬 알고리즘을 고안할 때, 과정A와 B는 기존의 방법을 사용하고 과정C에 대해 알고리즘을 병렬화하였다.

2.2 *GPU*기반 병렬 알고리즘 구현

기존의 알고리즘은 과정A,B를 수행하고 남은 지배되지 않는/지배되는 점인지 확정되지 않은 점들 각각에 대해서 지배되지 않는 점 테스트를 수행한다.
이때, 우리는 기존 알고리즘이 사용하던 다음과 같은 소정리를 사용하여 테스트의 비교대상을 n이 아니라 $|S|$로 줄일 수 있다.

소정리1: P의 지배되지 않는 점집합인 S에 의해서 지배되지 않으면 P에 의해서도 지배되지 않는다.

우리는 *소정리1*을 이용해서 R의 각각의 원소 r에 대해서 S의 각의 원소 중에 어떤 하나라도 모든 $CH(Q)$에 대해서 r보다 가까운지를 테스트 해본다.

2.2.1 이진탐색의 대체

기존의 알고리즘은 $CH(Q)$에 대한 이진탐색을 사용함으로써 시간을 단축한다. 하지만 이진탐색은 *CPU*에서와는 달리 *GPU*에서는 그 효율이 떨어진다. 왜냐하면 *GPU*같은 *SIMD* 구조의 계산기에서는 동시에 여러 개의 데이터에 하나의 명령어를 수행하게 되는데, 이진탐색은 매번 하나의 데이터만 읽고 대소를 판별한

다음에 그 결과에 따라서 분기해서 다시 하나의 데이터를 읽는 방식으로 진행되기 때문에 효율적으로 *GPU*의 병렬 코어를 활용할 수 없기 때문이다.

물론 탐색을 수행할 데이터가 대단히 많아진다면 *GPU*에서도 이진탐색이 효율적이겠지만, 실용적인 관점에서는 *2*차원에서 *CH(Q)*의 크기가 그렇게 크지 않기 때문에, 우리는 *CH(Q)*를 *CUDA*의 *constant cache*에 두고 동시에 *CH(Q)*의 여러 점들에 대해서 같은 비교구문을 실행한 후 결과를 보는 방식을 선택했다.

2.2.2 루프의 변경

기존의 알고리즘은 하나의 $r \in R$에 대해서 모든 $s \in S$를 순차적으로 비교해봐서 *r*이 지배되는 점인지를 확인한다.

이러한 기존의 직렬 알고리즘을 그대로 *GPU*기반의 병렬 알고리즘으로 바꾸어, *r*이 다수개의 *S*의 점에 대해서 지배되는지를 테스트 하면 다음과 같은 두 가지 문제가 있다.

첫째, 어느 한 *s*라도 *r*을 지배하게 되면 *정의2*에 의해서 *r*은 그 즉시 지배되는 점이 되고, *s* 이외의 *S*의 점들과 *r*에 대한 비교를 더 이상 수행할 필요가 없어진다. 이러한 성질은 직렬 알고리즘의 시간복잡도에는 영향을 미치지 않지만, 상수 배 만큼 성능을 끌어올릴 수 있는 좋은 성질이다. 하지만 *GPU*기반의 병렬 알고리즘에서는 특정 *s*에 대해서 *r*이 지배되는 것이 밝혀지더라도 또 다른 점 $s' \in S$에 대한 테스트를 취소하도록 구현하는 것이 어렵기 때문에 직렬 알고리즘과 같이 *정의2*로 인한 성능 향상을 기대하기 어렵다.

둘째, 경우에 따라서 $|S|$가 굉장히 커질 수 있기 때문에 *GPU*메모리에 한번에 다 수용할 수 없을 경우에는 매 루프마다 *S*를 일부씩 *GPU*메모리에 저장했다가 지우는 것을 반복해야 한다. 하지만 *GPU*와 *CPU*간의 데이터 전송은 시간이 많이 소요되는 작업이기 때문에 이러한 경우에는 성능저하가 생기게 된다[1].

위의 두 가지 이유로 인해서 본 알고리즘에서는 루프를 반복하는 방법을 바꾸었다. 우선은 *k*라는 윈도우 크기를 정해놓고, *S*에서 *k*개씩만 점을 선택해서 *GPU* 메모리로 읽는다. 이렇게 해서 읽은 *S*의 *k*개의 점들은 *그림4*와 같이 각각의 *GPU* 쓰레드 들에서 똑같이 사용하며, 테스트 대상이 되는 *R*만 각각의 쓰레드마다 다르게 비교한다.

이렇게 함으로써 우리는 $|S|$가 굉장히 커지더라도 각각의 $|S|$의 원소들을 한번씩만 *GPU*메모리에 올리면 되어 메모리 대역폭 문제를 해결할 수 있다. 그리고 *S*를 윈도우 크기인 *k*개 단위로 비교함으로써, 비교대상인 *r*이 *정의2*에 의해서 지배되는 점이라고 확정되었을 경우에도 남은 *S*의 점들에 대해서는 비교를 하지 않을 수 있어서 좀 더 효율적으로 병렬 알고리즘을 구현할 수 있게 된다.

추가적으로, *그림4*에서 검은색으로 표시된 지배되는 것으로 확정된 점에 대해서는 더 이상 지배되는지 테스트를 수행할 필요가 없는데, 이 점들을 그냥 놔두면 *R*의 밀도가 점점 더 희박해 지므로, 특정 횟수만큼 반복 후에는 검은 점들을 제외하고 *packing*을 수행하도록 하였다.

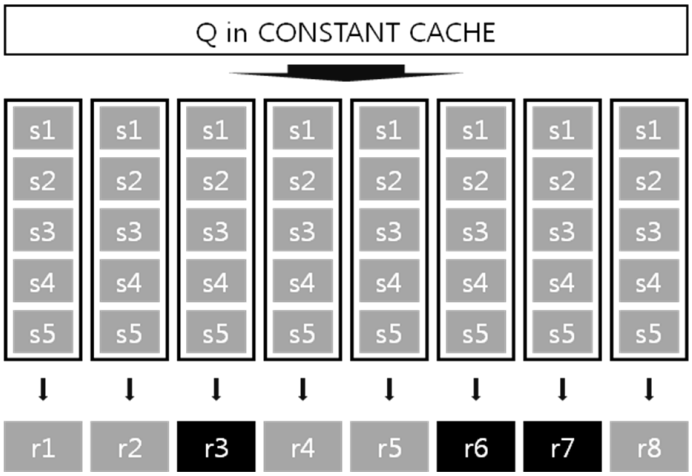

그림 4 *CUDA constant cache*에 *Q*가 존재하고, 각각의 *CUDA* 쓰레드는 동일한 *k*개의 $s \in S$를 가지고 각각의 *r*에 대해서 테스트를 한다. 검은색은 이미 지배되는 점으로 확정된 점이다.

3. 결과

실험 환경은 다음과 같다.

CPU: Q6600 2.4GHz, 2GB RAM
GPU: Geforce 8600GTS, 512MB Video RAM

이 환경에서 각각 *CPU*와 *GPU*버전의 코드를 구현해서 실험해본 결과 다음과 같은 결과를 얻을 수 있었다.

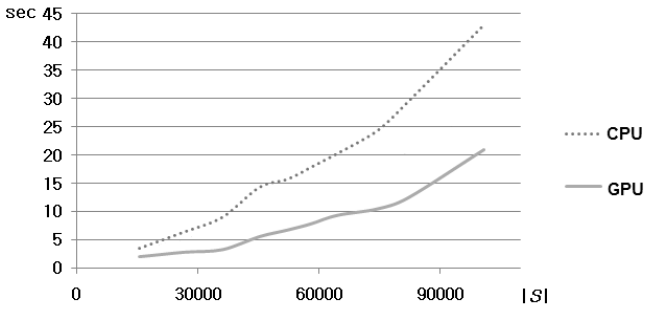

그림 5 CPU와 GPU 구현의 성능비교

*그림5*를 보면 *GPU*구현에서 전반적으로 *2*배 가량의 성능 향상이 있었다. 하지만 이 정도의 성능 향상은

*GPU*에 친화적인 알고리즘을 *GPU*로 병렬화 했을 때 얻어지는 수십 배[1]의 성능향상에는 미치지 못하는데, 이는 기본적으로 지배되지 않는 점 문제가 이진탐색을 사용하고 정의2에 의해서 임의의 시점에 *R*에 대한 지배 테스트가 종료되기 때문에 *SIMD*형의 병렬계산 장치상에서 병렬화하기에 좋은 형태가 아니기 때문이다.

4. 결론

우리는 본 논문에서 기존의 지배되지 않는 점 문제가 *SIMD*형 병렬계산 장치인 *GPU*에서 병렬화가 가능하다는 것을 보였다. 그리고 실제 *GPU*를 통한 병렬 알고리즘을 구현함으로써 실험 결과 기존의 결과보다 2배 가량의 성능 향상을 얻을 수 있었다.

이러한 결과는 *GPU* 친화적인 다른 알고리즘의 *GPU*구현을 통해 얻을 수 있는 결과인 수십배[1]의 성능 향상에는 미치지 못하는데, 이는 지배되지 않는 점 문제가 *GPU*와 같은 *SIMD*형의 병렬계산 장치로는 병렬화 하기 좋지 않은 다음과 같은 두 가지 특성을 가지고 있기 때문이다. 첫째로 이진탐색을 사용하고, 둘째로 임의의 시점에 *R*에 대한 지배 테스트가 종료되기 때문에 모든 쓰레드에서 같은 명령을 수행하도록 하는 형태의 병렬화가 어렵다.

본 연구팀이 지배되지 않는 점의 *GPU*기반 병렬 알고리즘을 구현함에 있어서 메모리 접근 패턴과 캐쉬를 고려하여 구현하였기 때문에, 기존의 [2] 알고리즘을 *GPU*기반으로 병렬화 시키는 방식으로 *GPU*구현을 해서 *2*배 이상의 큰 성능 향상은 기대하기 힘들다고 판단된다.

*2*배의 성능향상은 비록 시간복잡도의 향상은 아니지만, 지배되지 않는 점 문제에서 $|S|$가 커질 때에 속도가 크게 느려지는 기존 알고리즘의 약점을 *GPU* 하드웨어를 이용해서 어느 정도 보완할 수 있는 수준이라 할 수 있다.

5. Reference

[*1*] *Nvidia CUDA programming guide.*

[*2*] *W. Son, M. Lee, H. Ahn, and S. Hwang. Spatial Skyline Queries: An Efficient Geometric Algorithm. In Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases, pages 247–264. Springer-Verlag, 2009.*

[*3*] *H. T. Kung, F. Luccio, and F. Preparata. On finding the maxima of a set of vectors. In Journal of the Association for Computing Machinery.*

[*4*] *S. B¨orzs¨onyi, D. Kossmann, and K. Stocker. The skyline operator. In ICDE '01: Proc. of the 17th International Conference on Data Engineering, page 421, 2001.*

[*5*] *K. Tan, P. Eng, and B. C. Ooi. Efficient progressive skyline computation. In VLDB '01: Proc. of the 27th International Conference on Very Large Data Bases, pages 301–310, 2001.*

[*6*] *D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In SIGMOD '03: Proc. of the 2003 ACM SIGMOD International Conference on Management of Data, pages 467–478, 2003.*

[*7*] *J. Chomicki, P. Godfery, J. Gryz, and D. Liang. Skyline with presorting. In ICDE '07: Proc. of the 23rd International Conference on Data Engineering, 2007.*