가 장 가 까 운 1 0 0 쌍 찾 기

적벽대전

CSE244 Final Project

2018136153 최원홍
2015136133 최진우
2016136092 이상민
2017136008 권창덕

CONTENTS

01

문제 정의

01

문제 소개

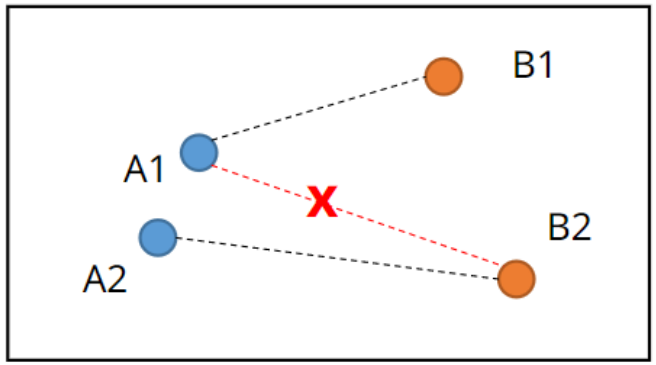
입력 : A부대, B부대 (부대별 병사 수, x,y,z 좌표)

출력 : 피아식별하며, 가장 가까운100쌍 목록

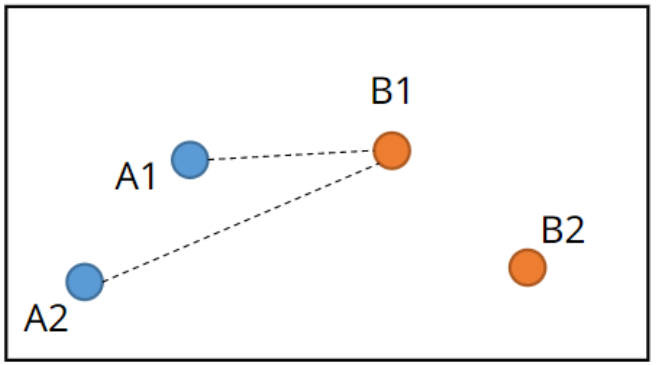
[IMG] https://koreatechackr-my.sharepoint.com/personal/bluekds_koreatech_ac_kr/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fbluekds%5Fkoreatech%5Fac%5Fkr%2FDocuments%2F%EA%B0%95%EC%9D%98%2F%EB%A9%80%ED%8B%B0%EC%BD%94%EC%96%B4%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%2F2022%5Fspring%2F%5BMP%5D%20Project02%5FCUDA%2Epdf&parent=%2Fpersonal%2Fbluekds%5Fkoreatech%5Fac%5Fkr%2FDocuments%2F%EA%B0%95%EC%9D%98%2F%EB%A9%80%ED%8B%B0%EC%BD%94%EC%96%B4%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%2F2022%5Fspring&ga=1

01

문제 소개



(X)



(O)

단, **A부대 기준**으로 검색
(Euclidean distance 이용)

[IMG] https://koreatechackr-my.sharepoint.com/personal/bluekds_koreatech_ac_kr/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fbluekds%5Fkoreatech%5Fac%5Fkr%2FDocuments%2F%EA%B0%95%EC%9D%98%2F%EB%A9%80%ED%8B%B0%EC%BD%94%EC%96%B4%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%2F2022%5Fspring%2F%5BMP%5D%20Project02%5FCUDA%2Epdf&parent=%2Fpersonal%2Fbluekds%5Fkoreatech%5Fac%5Fkr%2FDocuments%2F%EA%B0%95%EC%9D%98%2F%EB%A9%80%ED%8B%B0%EC%BD%94%EC%96%B4%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D%2F2022%5Fspring&ga=1

02

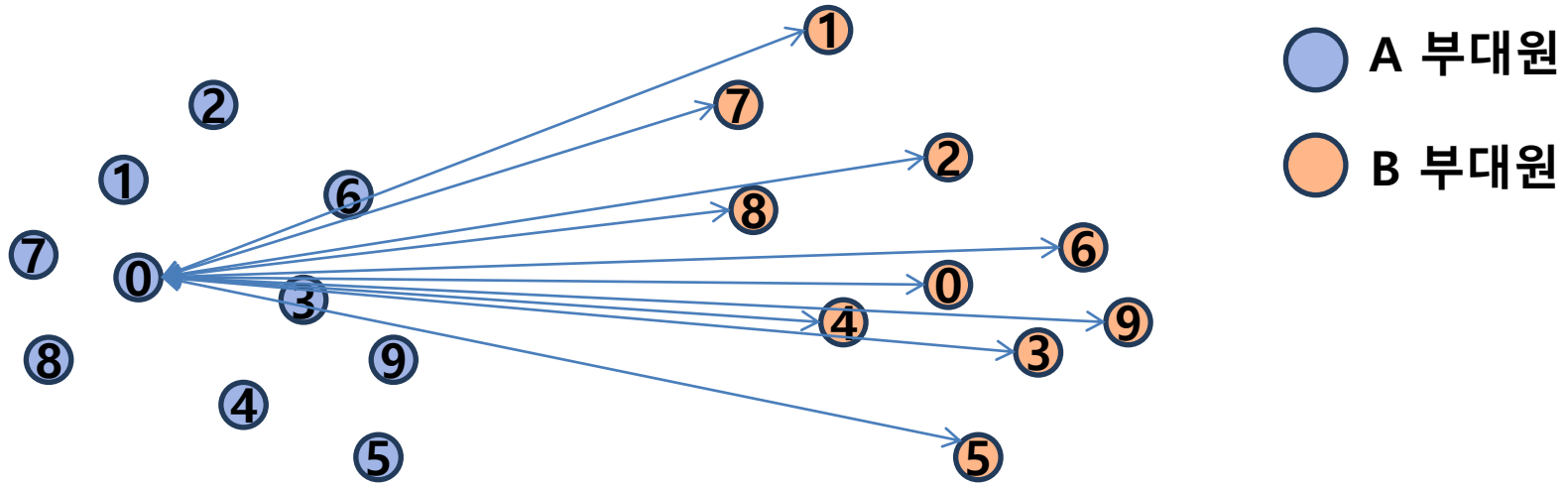
설계 및 구현

Parallel ver.1

Brute-force

02

ver1. Idea

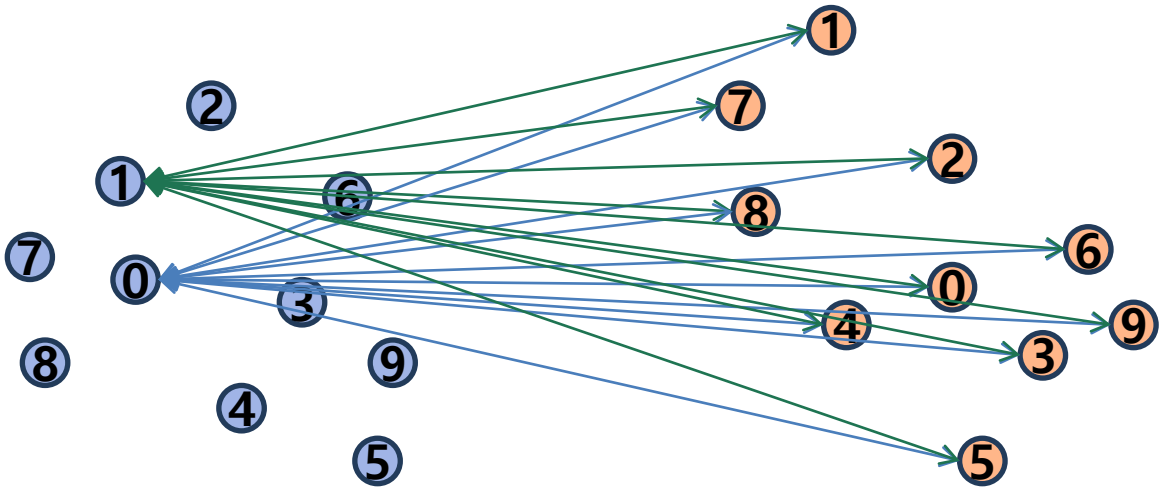


각 A부대원별, 모든 B부대원과의 거리를 계산하여, 최소거리인 B부대원을 찾음

한 A부대원당 총 B부대원 수만큼 매칭

02

ver1. Idea



A 부대원

B 부대원

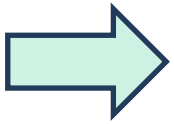
A 수 = M
B 수 = N

O(M*N)
searching
*single thread

02

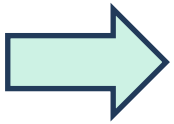
ver1. 구현

A부대원 수만큼
GPU Thread 생성



1D Grid & 1D Block
Block layout : (1024, 1, 1)
Grid layout : (ceil(M / 1024), 1, 1)

A 부대 정보
&
B 부대 정보



Global Memory

각 부대원 정보를
[XXX..., YYY..., ZZZ...]
형태로 저장

02

ver1. 구현

Thread 0 Thread 1 ... Thread 30 Thread 31

A부대원 0 A부대원 1 A부대원 30 A부대원 31

각 thread별 global index를 이용,
한 warp이 A부대 정보를
coalesced Mem. access

각 thread별 자신이 계산할
A부대원 정보 획득

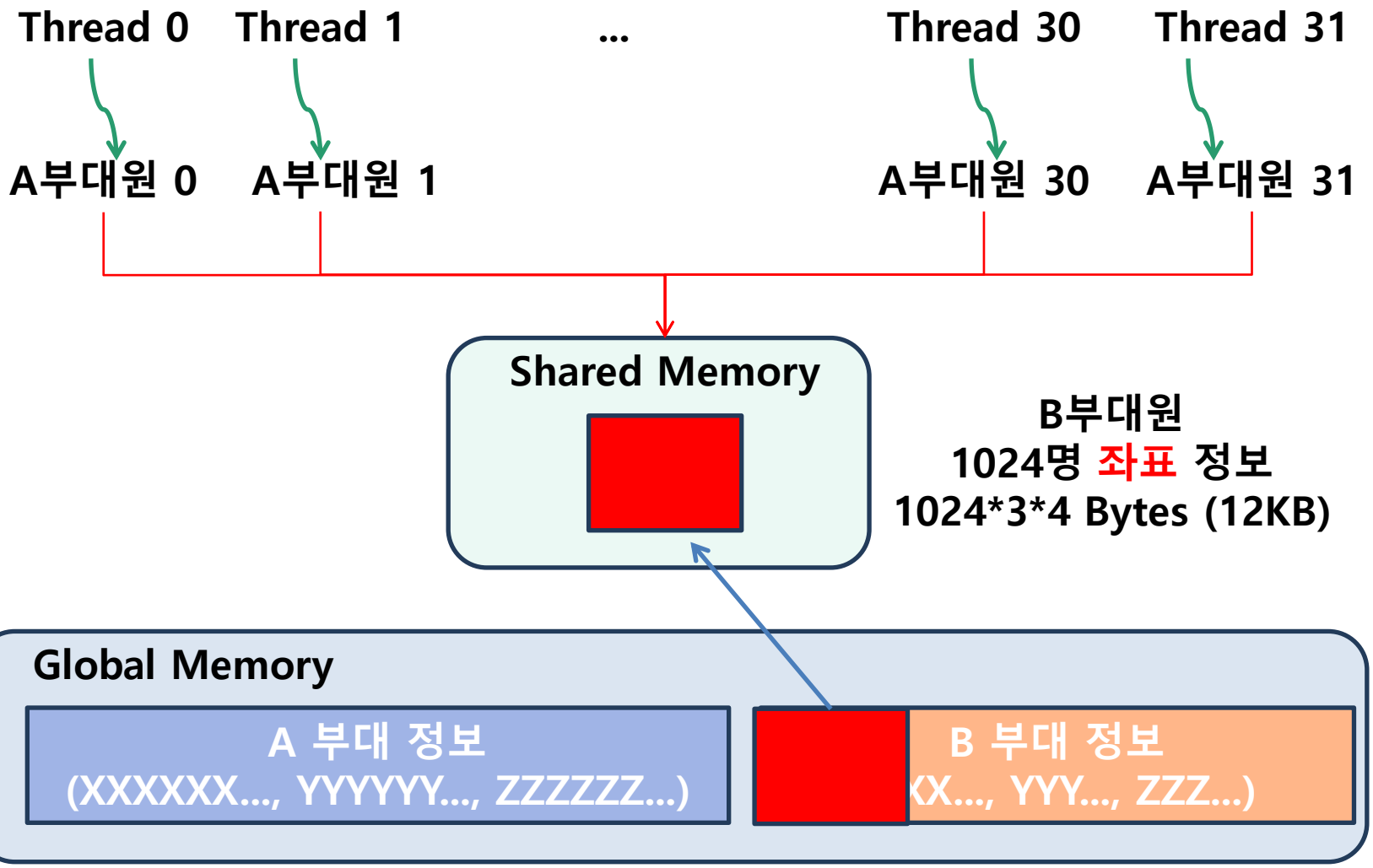
Global Memory

A 부대 정보
(XXXXXX..., YYYYYY..., ZZZZZZ...)

B 부대 정보
(XXX..., YYY..., ZZZ...)

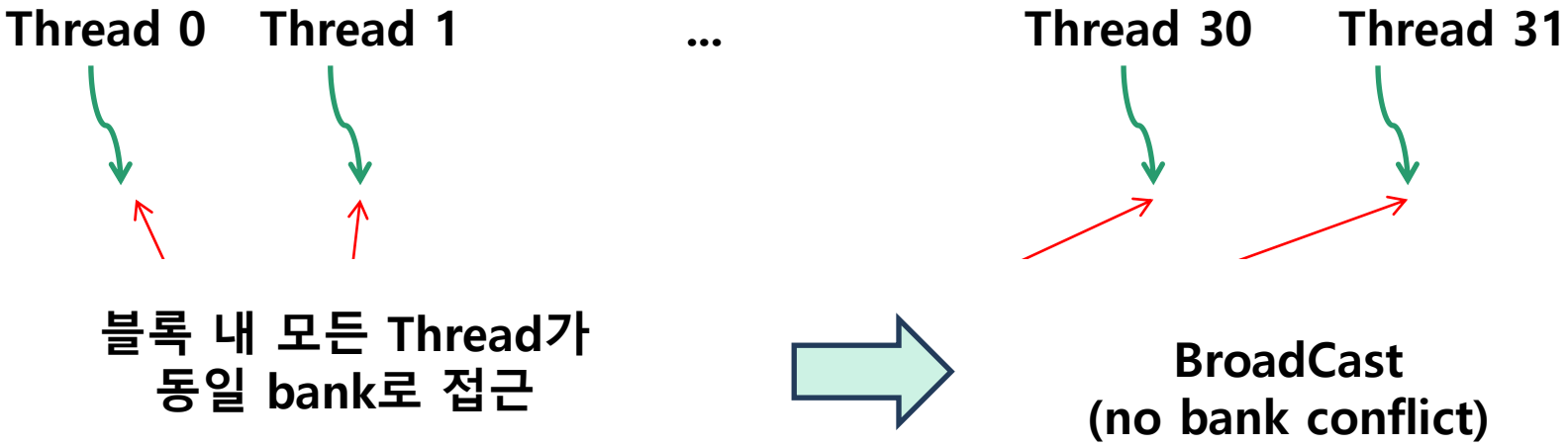
02

ver1. 구현


Thread 0
Thread 1
...
Thread 30
Thread 31
A부대원 0
A부대원 1
A부대원 30
A부대원 31
Shared Memory
B부대원
1024명 좌표 정보
1024*3*4 Bytes (12KB)
Global Memory
A 부대 정보
(XXXXXX..., YYYYYY..., ZZZZZZ...)
B 부대 정보
XX..., YYY..., ZZZ...)

02

ver1. 구현

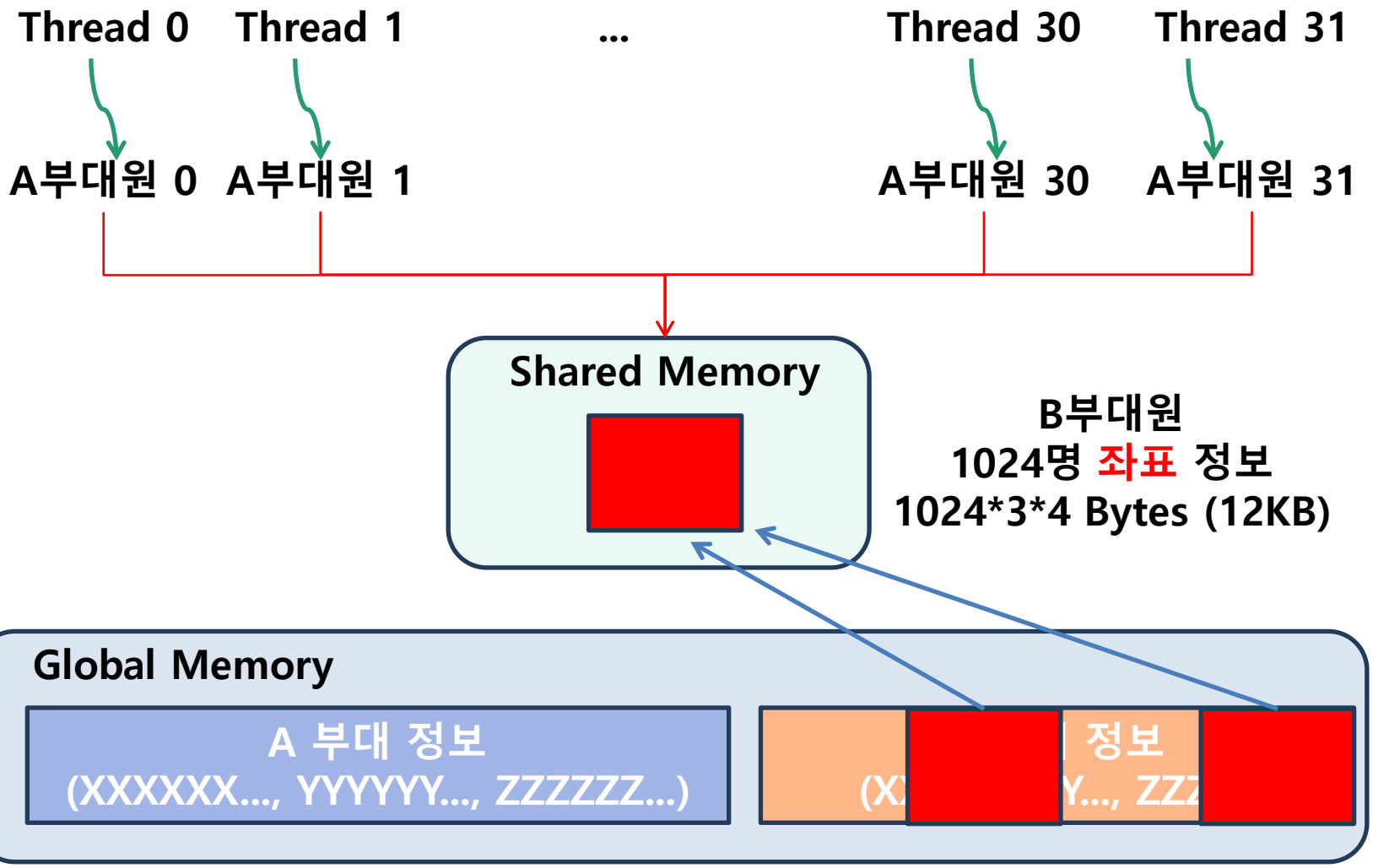


Bank	0	1	2	3	...	...	...	29	30	31
Data	x z y	y x z	z y x	x z y	...			z y x	x z y	y x z

Shared Memory

02

ver1. 구현


Thread 0
Thread 1
...
Thread 30
Thread 31
A부대원 0
A부대원 1
A부대원 30
A부대원 31
Shared Memory
B부대원
1024명 좌표 정보
1024*3*4 Bytes (12KB)
Global Memory
A 부대 정보
(XXXXXX..., YYYYYY..., ZZZZZZ...)
정보

Parallel ver.2

Spatial Partitioning

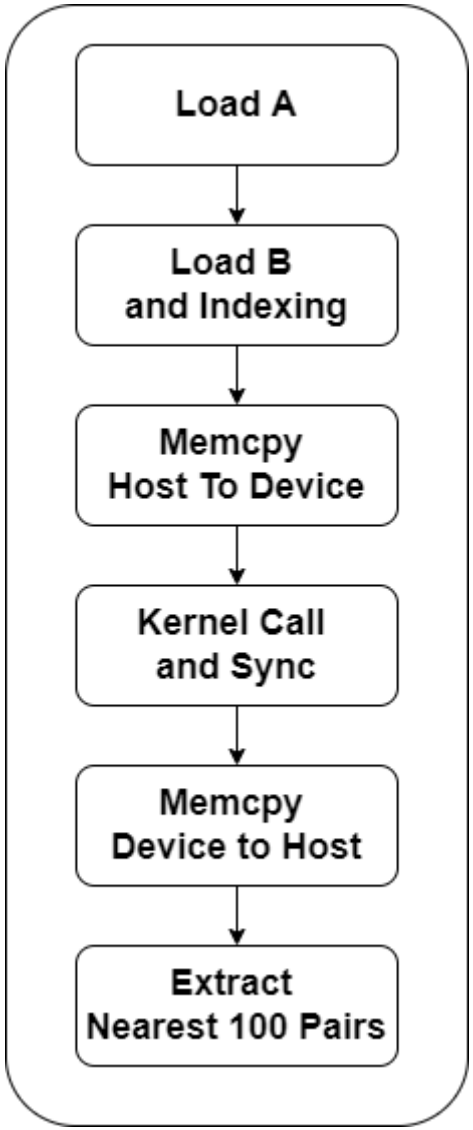
02

ver2. Motivation

[IMG] http://kkoma.net/product/%ED%98%84%EC%88%98%EB%A7%89-%EB%B0%B0%EA%B2%BD3-%EB%B0%A4%ED%95%98%EB%8A%98-%EB%B3%84-%EC%97%B0%EA%B7%B9-%EA%B3%B5%EC%97%B0-%EB%AC%B4%EB%8C%80-%ED%8F%AC%ED%86%A0%EC%A1%B4-%EB%B0%B0%EA%B2%BD/6305/

02

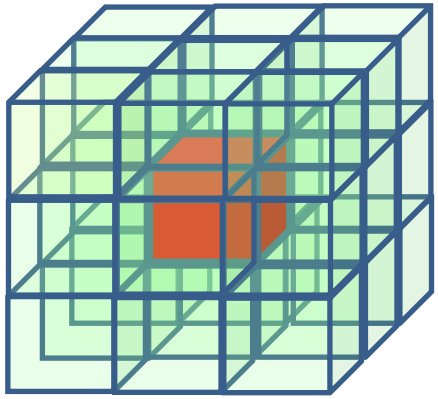
전체 흐름



** kernel 외 전부 serial

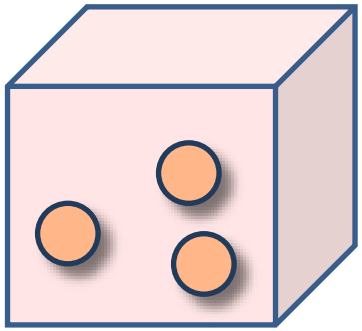
02

ver2. Idea



큐브 구조체 배열

< 전체 공간 >



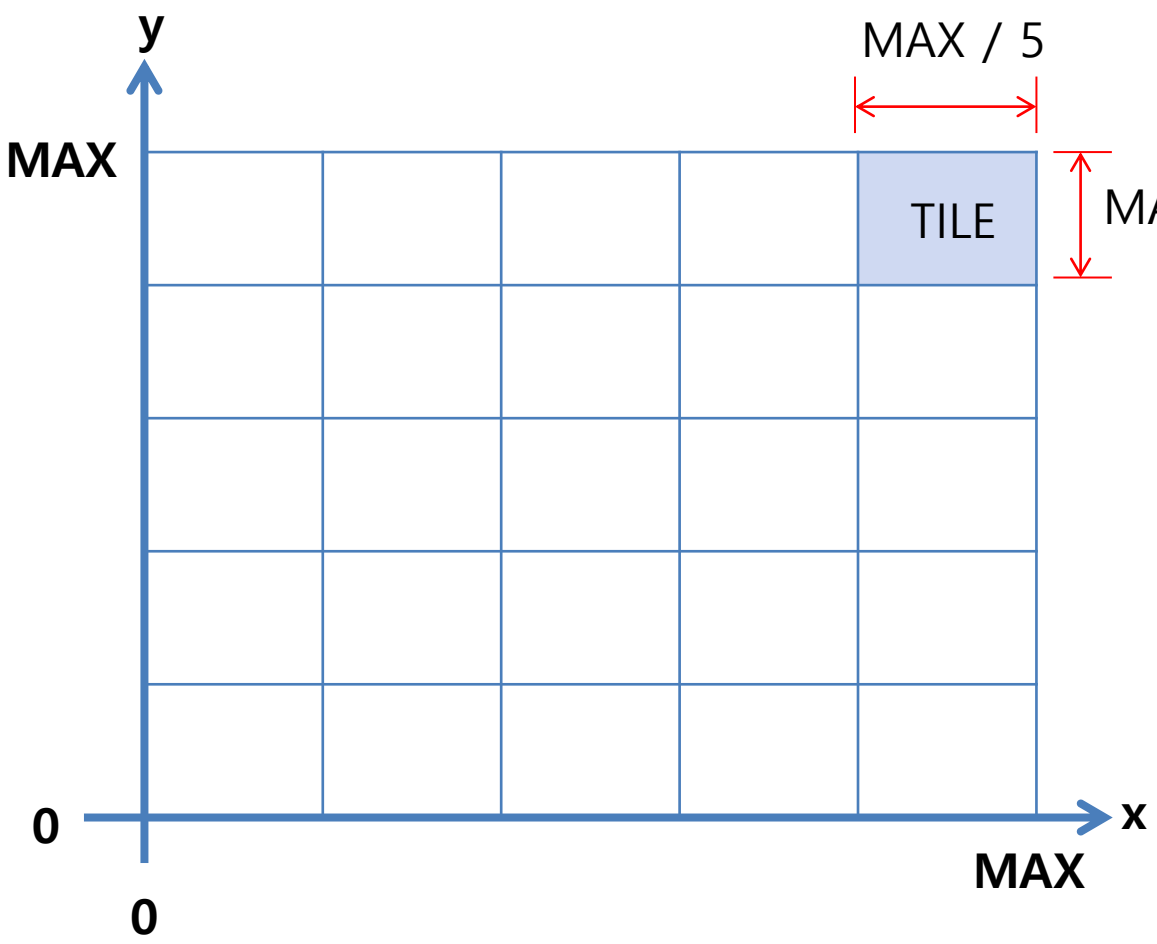
큐브 구조체
- B의 갯수
- B 부대원(ID, 좌표) 배열

< 분할된 공간 >

각 B의 (x,y,z) 좌료를 이용해
큐브 구조체 배열(큐브 배열)에서 저장 위치(큐브)를 결정

02

ver2. Idea



A 부대원

B 부대원

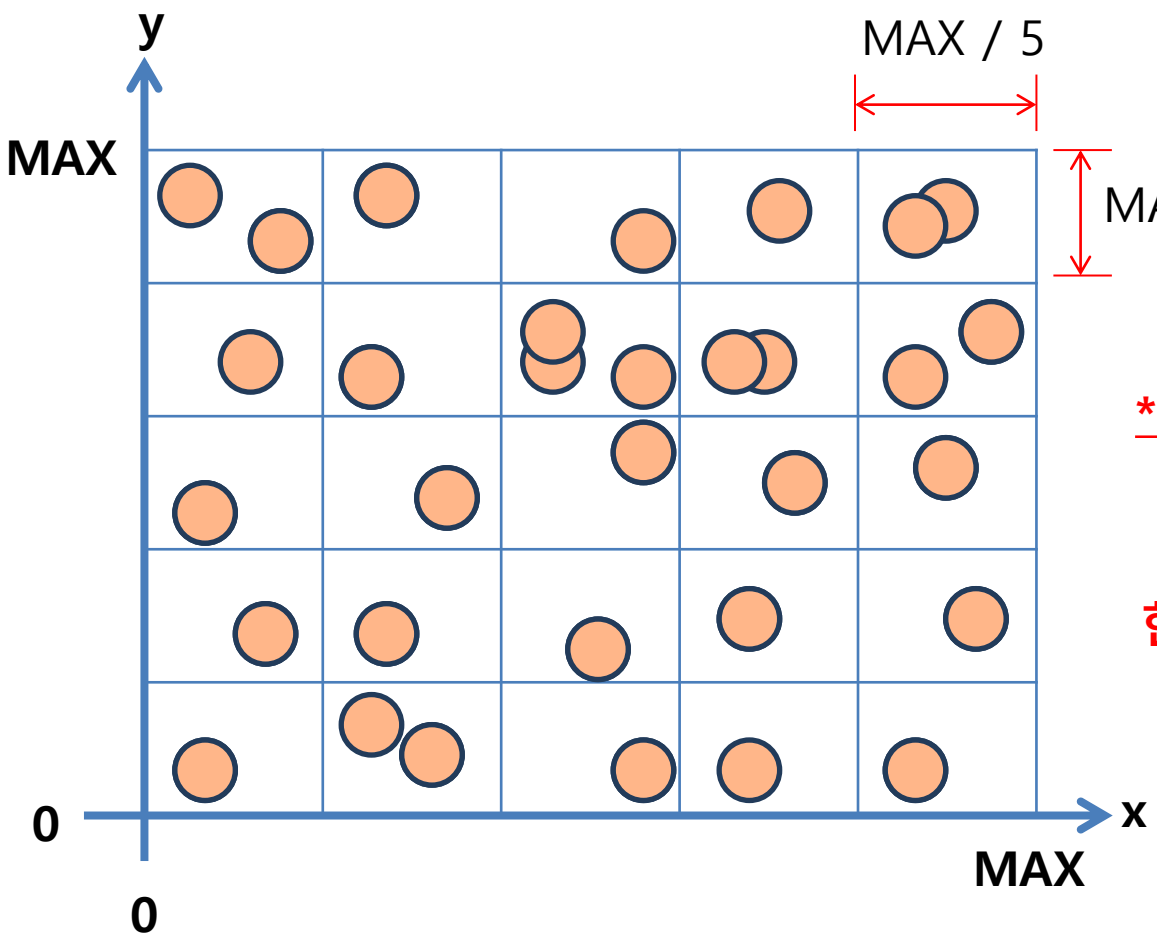
만약, B의 수= 25,

$\sqrt{25}$ = 5,

각 축을 5등분
→ 전체 평면이
25등분됨

02

ver2. Idea



A 부대원

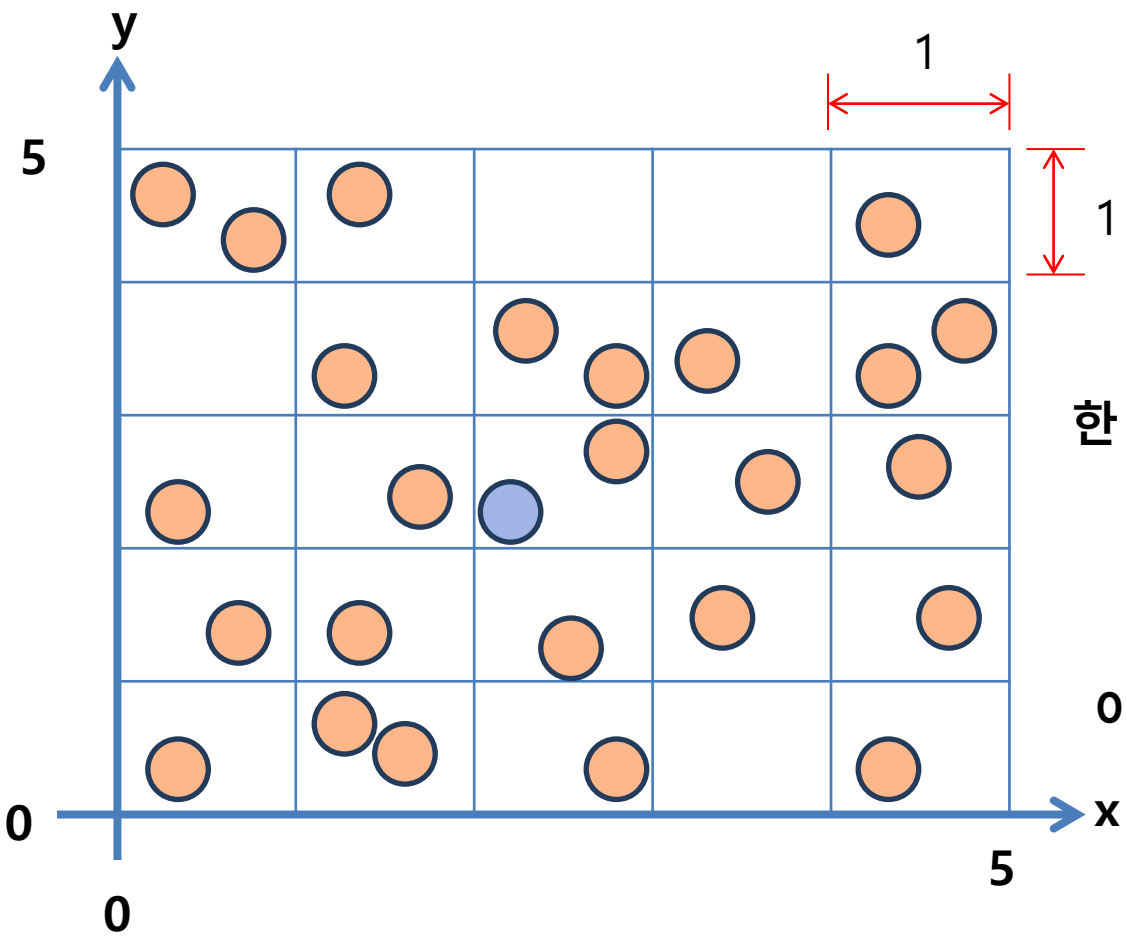
B 부대원

Army Generator →
* uniform_int_distribution
* RANGE_MAX

이상적인 상황에서,
한 Tile 당 존재할 B의 수
의 기대값은 1개

02

ver2. Idea



A 부대원

B 부대원

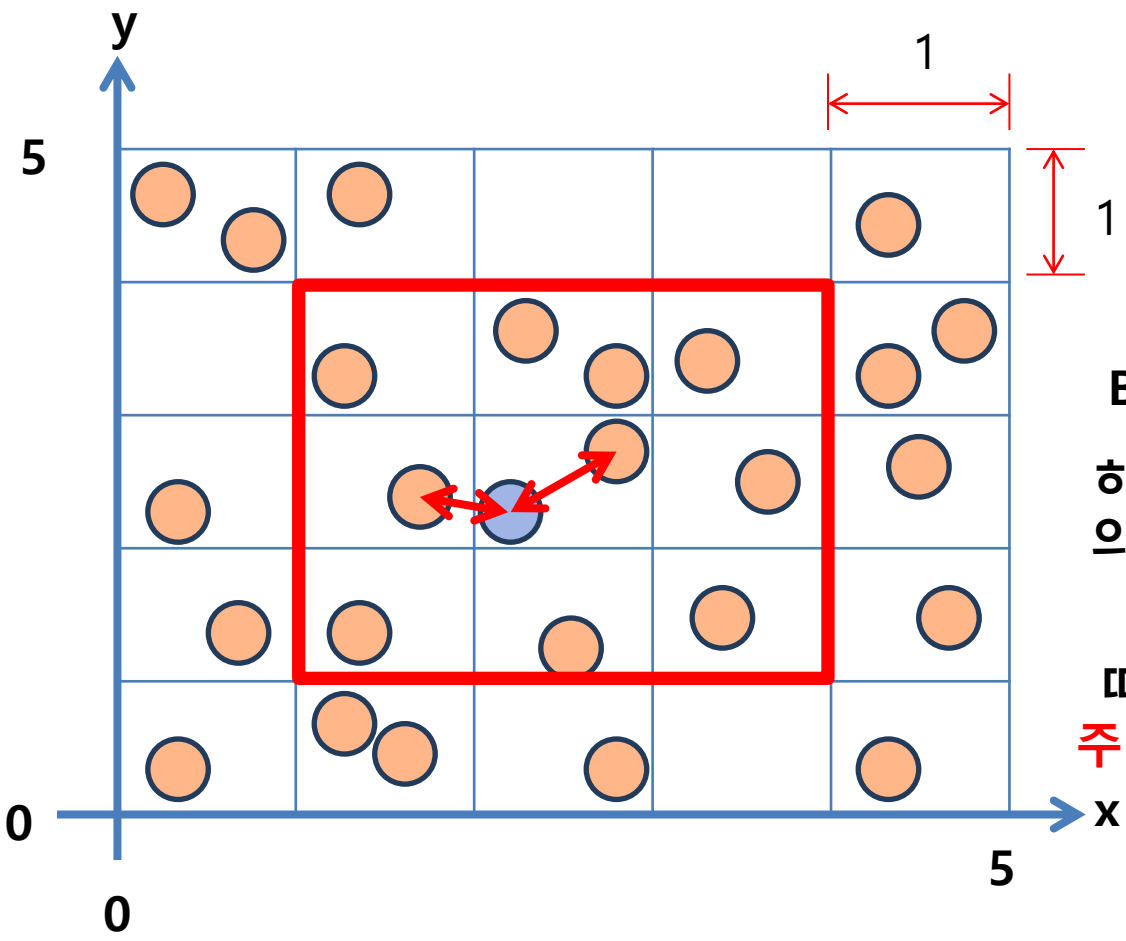
만약, MAX=5이고,

한 A의 좌표가 (2.1, 2.1)이면,
X' = 2.1 / 1
Y' = 2.1 / 1
* 정수 나눗셈

이 A가 속한 타일의 인덱스
(X',Y') = (2, 2)

02

ver2. Idea



A 부대원

B 부대원

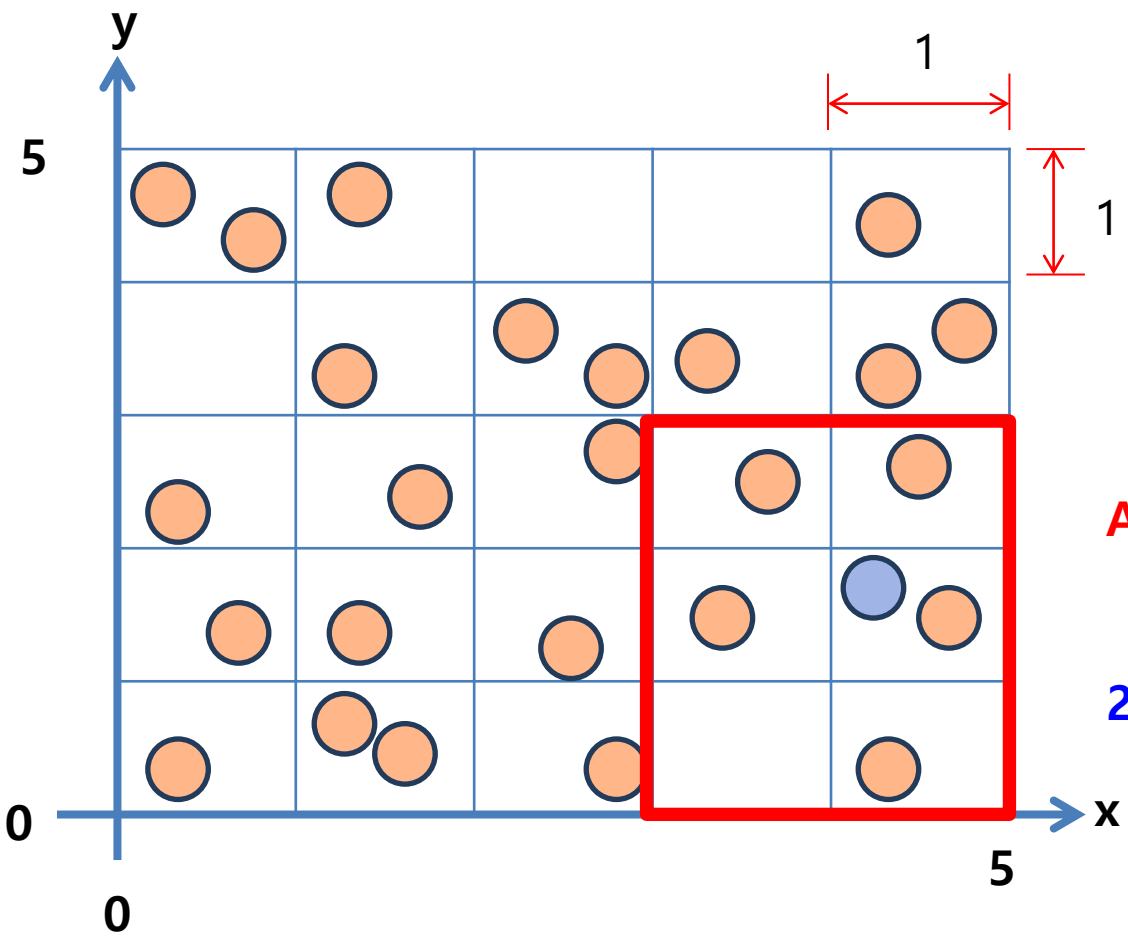
A가 속한 타일 내
B의 점과의 거리가 최소

하지만, 근처에 속한 타일의 B의 점과 최소가 될 수 있음

따라서, A가 속한 타일과 주변의 타일을 검사할 필요가 있음

02

ver2. Idea



A 부대원

B 부대원

다른 A부대원 역시
A가 속한 타일과 주변의
타일을 검사

25개의 B 중 5개만 검색

02

ver2. Idea

B의 수= N

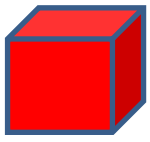
각 축을 $\sqrt[3]{N}$ 등분

→ 전체 공간 N등분
(=N개의 큐브)

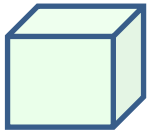
큐브 모서리 길이
= RANGE_MAX / $\sqrt[3]{N}$

부대원 좌표 (x,y,z) 각각
큐브 모서리 길이로 정수 나눗셈

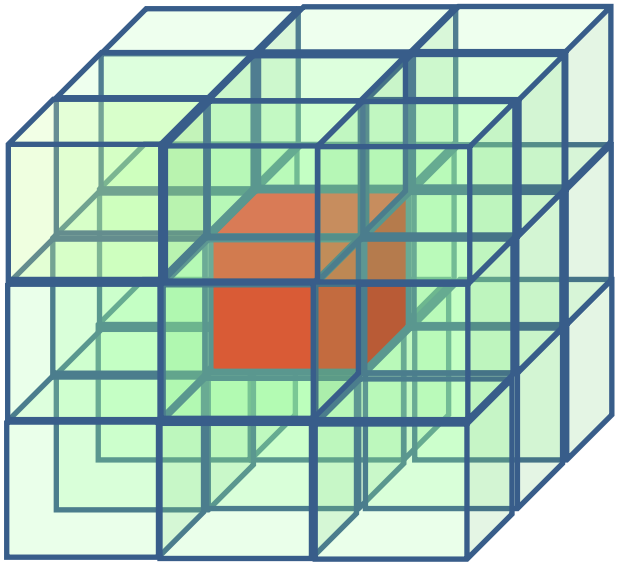
몫 → 해당 부대원이 존재하는
큐브 위치를 알 수 있음



A가 속한 큐브



주변 큐브



인접한 큐브를 포함
총 27개(3x3x3)의 큐브
조사

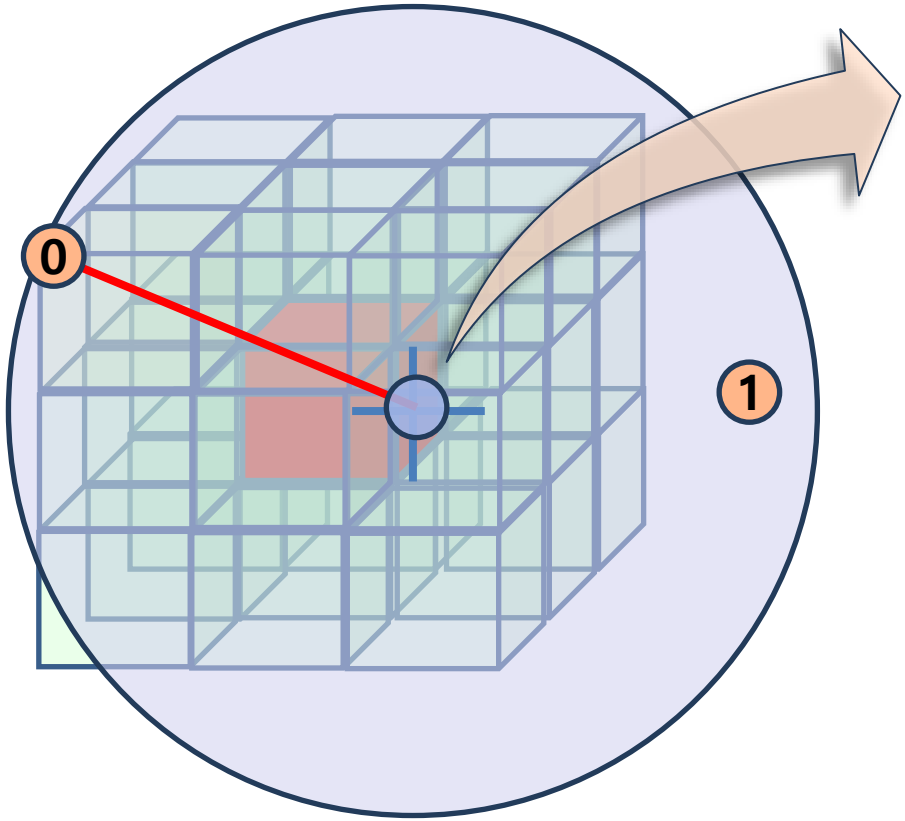
ver2. Idea

총 27개(3x3x3) 큐브 내
B가 하나라도 존재하지 않으면,

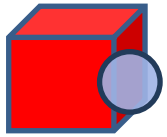
가장 가까운 100쌍 내 속하지 않을 것으로 추정

02

ver2. Idea



검색 영역 내 B가 하나 존재하고,
가장 멀리 떨어진 경우



A 부대원

B 부대원

0번은 검색 영역에 속하지만,
1번은 검색 영역에 속하지 않음

하지만, 1번 점과의 거리가 더 짧음

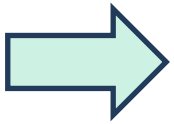
100 << A의 수

* A: 100만, B: 40만 기준
큐브 모서리 길이 = 약 14,170
GT 100쌍 중 가장 큰 거리 = 395.30

02

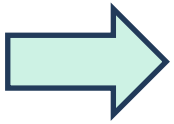
ver2. Idea

A부대원 수만큼
GPU Thread 생성



1D Grid & 1D Block
Block layout : (1024, 1, 1)
Grid layout : (ceil(M / 1024), 1, 1)

A 부대 정보
&
B가 할당된 큐브 배열



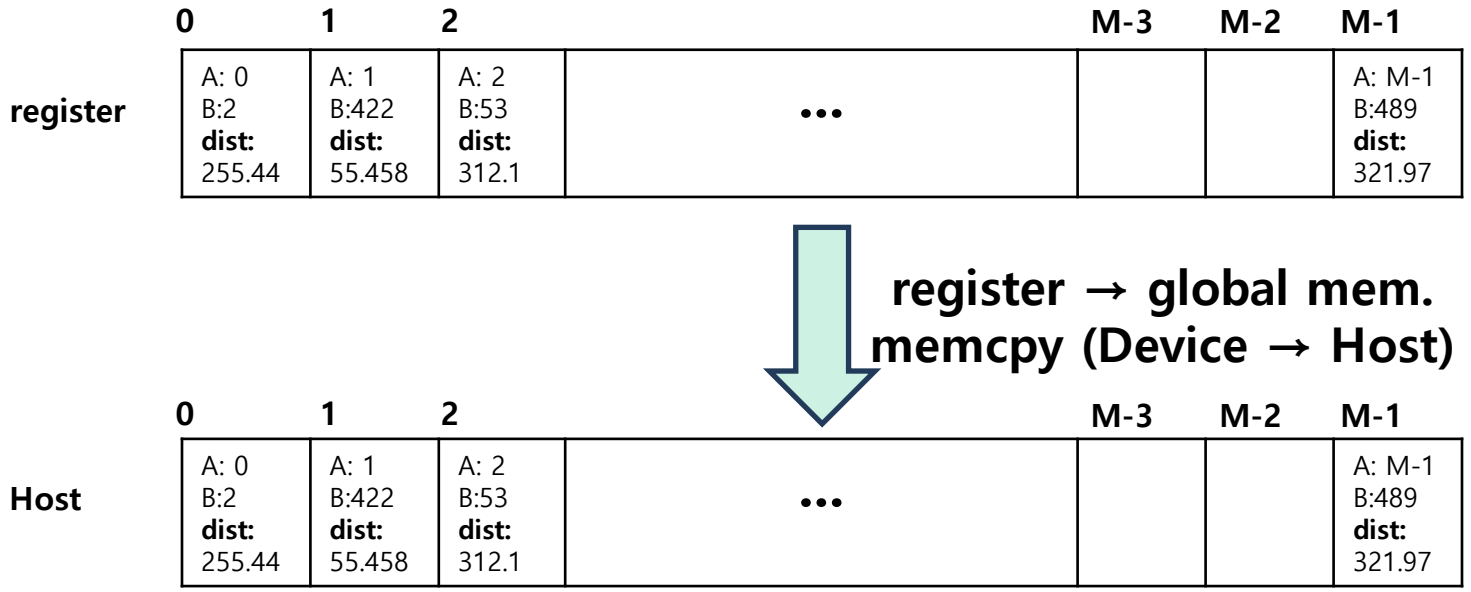
Global Memory

각 Thread별
총 27개(3x3x3)의 큐브 search

가장 가까운 B ID, 거리를
자신의 register에 기록

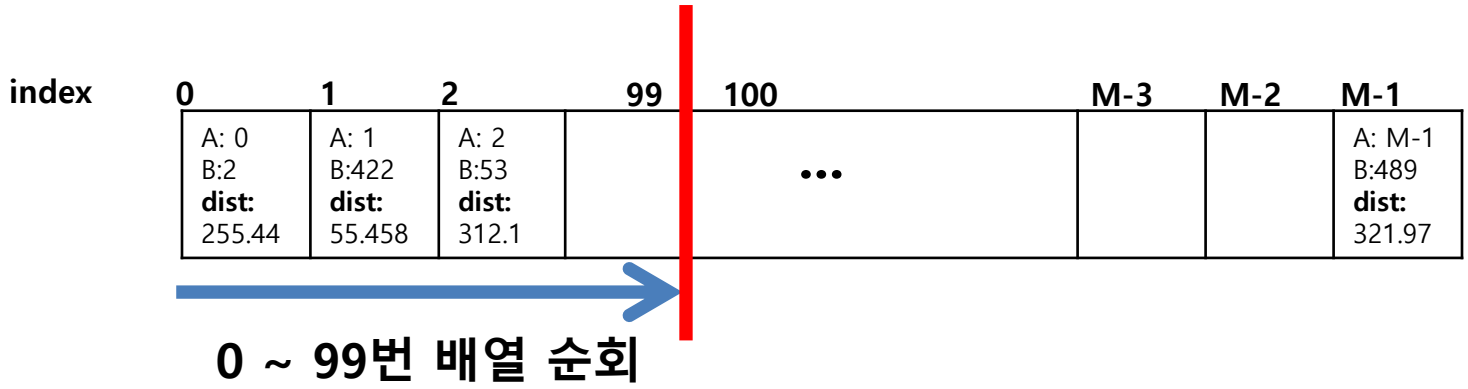
02

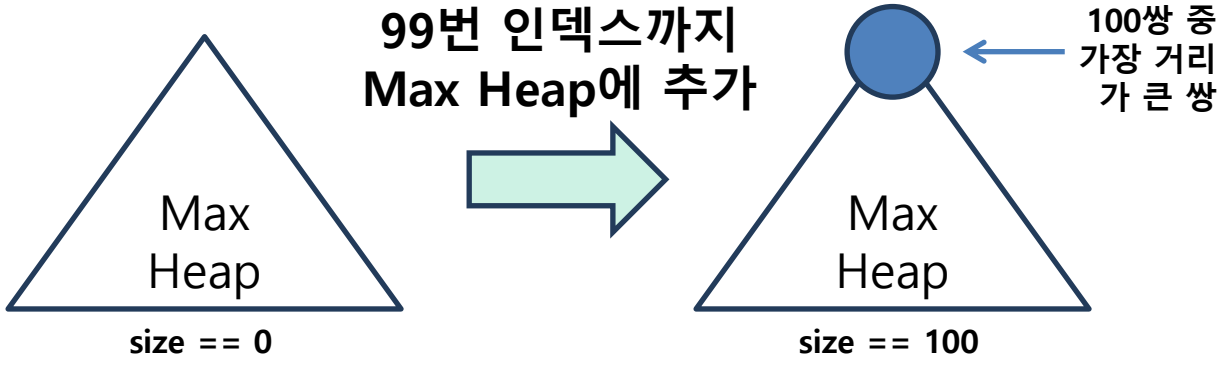
ver2. Idea


0
1
2
M-3
M-2
M-1
register
A: 0
B:2
dist:
255.44
A: 1
B:422
dist:
55.458
A: 2
B:53
dist:
312.1
•••
A: M-1
B:489
dist:
321.97
register → global mem.
memcpy (Device → Host)
0
1
2
M-3
M-2
M-1
Host
A: 0
B:2
dist:
255.44
A: 1
B:422
dist:
55.458
A: 2
B:53
dist:
312.1
•••
A: M-1
B:489
dist:
321.97

02

ver2. Idea

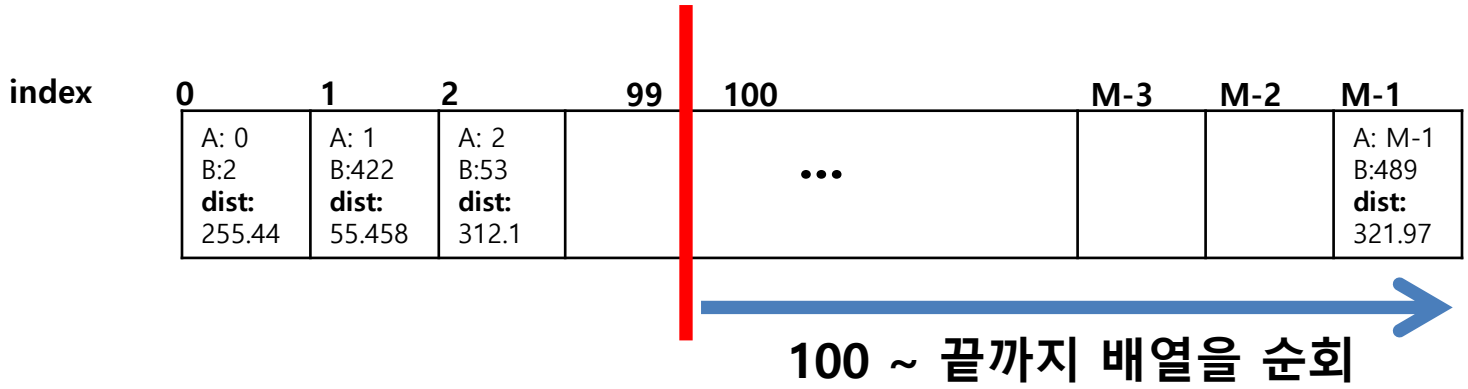




거리가 가까운 100쌍을
유지하기 위한
자료구조

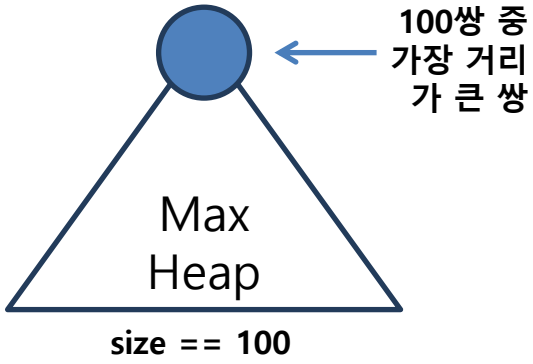
02

ver2. Idea



현재 순회 중인 원소의 거리가
Max Heap의 root 노드의
거리보다 작을 경우,

root 노드를 pop하고,
해당 원소를 push

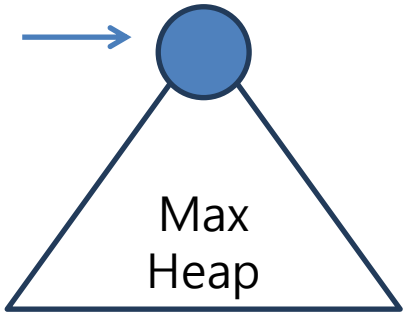


최종 Max Heap엔
거리가 가까운 100쌍만 남음

02

ver2. Idea

Max Heap에 저장된 쌍 중
가장 거리가 큰 쌍



**Max Heap을 pop하면서,
최종 결과 배열의 역순으로 저장**

index	0	1	2	...	97	98	99
RESULT	A: 573 B:2 **dist:** 2.54	A: 521 B:422 **dist:** 5.458	A: 12 B:53 **dist:** 12.1	...			A: 1 B:489 **dist:** 321.97

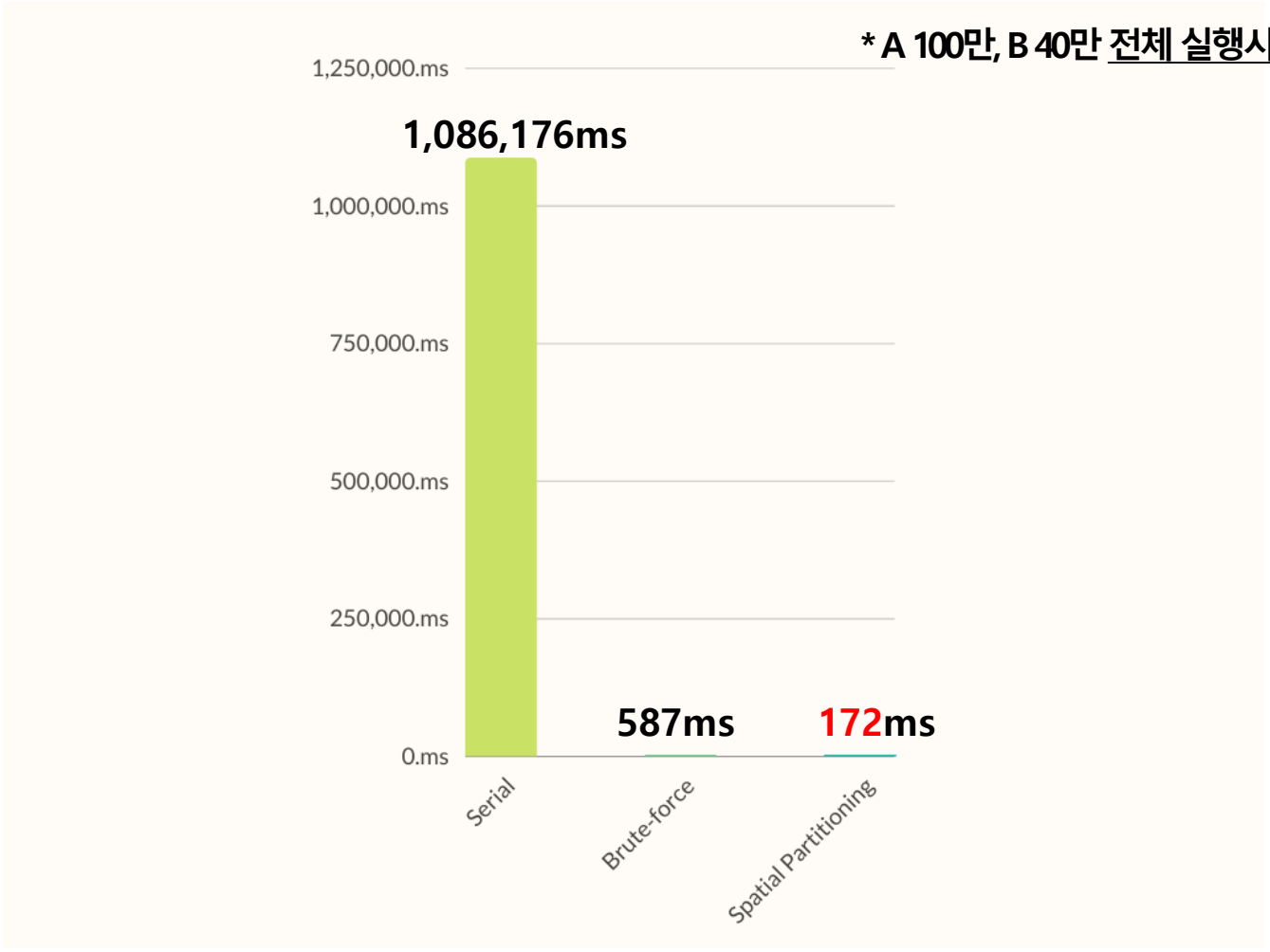
03

결과 및 분석

03

결과 분석

실습 환경 : Ryzen 7 5800 X / RTX 2080 TI

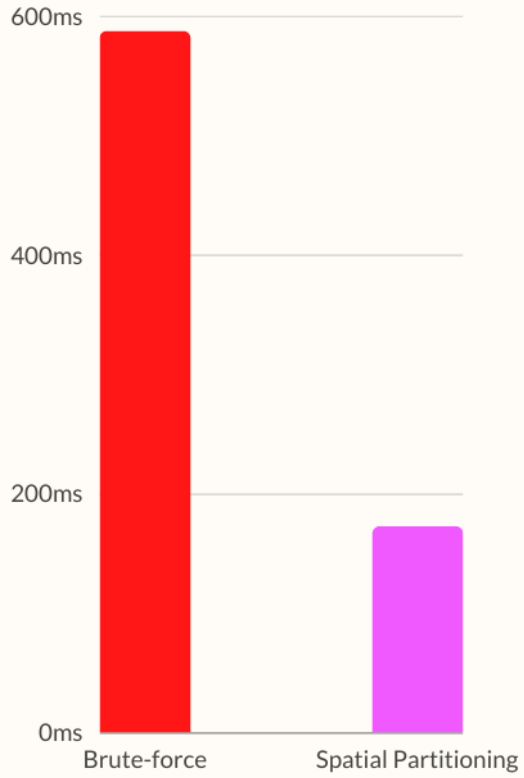


03

결과 분석

실습 환경 : Ryzen 7 5800 X / RTX 2080 TI

* A 100만, B 40만 전체 실행시간



Parallel ver.1 (Brute-force)
약 587ms
Serial 대비 성능향상 약 1,850배
정확도 : 100%

Parallel ver.2 (Spatial Partitioning)
약 172ms
Serial 대비 성능향상 약 6,314배
정확도 : 100%

03

결과 분석

실습 환경 : Ryzen 7 5800 X / RTX 2080 TI

* A 100만, B 40만 전체 실행시간

* 단위 : ms

	1회	2회	3회	4회	5회	평균
Parallel ver.1	587.64890	594.12080	564.26280	585.85450	594.34940	585.2473
Parallel ver.2	172.96660	177.01230	173.69550	197.92030	205.06940	185.3328

평균 성능 기준 약 3.16 배 성능 향상

THANK YOU

[IMG] https://www.nvidia.co.kr/docs/IO/57631/Volumel.pdf