GPU TECHNOLOGY CONFERENCE

You Might Also Like: A Multi-GPU Recommendation System

Kimikazu Kato

Nihon Unisys, Ltd.

NVIDIA.

Summary

- We have implemented a "recommendation system" on multiple GPUs
- x20–300 speed-up compared to a latest CPU (single core implementation)

GPU TECHNOLOGY CONFERENCE

Table of Contents

- Introduction
  - Principle of recommendation system
  - Known serial algorithm
- Algorithm on CUDA
  - Singular value decomposition
  - k-nearest neighbor problem
- Benchmark
- Conclusion

NVIDIA.

Recommendation System

- Predicts unknown preferences from customers' known preferences
- Commonly used by web-based shops
- Famous examples:
  - Amazon
    - "Customers who bought this item also bought"
  - Netflix
    - Sends DVDs you are likely to prefer
    - Netflix Prize: competition about the predict of users' preference

Example

amazon.com

Hello. Sign in to get personalized recommendations. New customer? Start here.

Your Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards

FREE 2

Shop All Departments | Search Electronics

All Electronics | Brands | Bestsellers | Audio & Home Theater | Camera & Photo | Car Electronics & GPS | Cell Phones & Service | Computers | MI

This item is not eligible for Amazon Prime when purchased from Mega Micro Devices Inc.. See more buying choices

eVGA e-GeForce GTX280 1GB DDR3 PCI-Express 2.0 Graphics Card-Lifetime Warranty with Free Special Edition EVGA Precision Overclocking Utility

Other products by EVGA

(4 customer reviews) | More about this product

Price: $698.95

In Stock.

Ships from and sold by Mega Micro Devices Inc..

Only 1 left in stock--order soon.

2 used from $279.99

Customers Who Bought This Item Also Bought

Page 1 of 14

EVGA 132-BL-E758-A1 X58 3-Way SLI Core i7 Motherboard with Tri-C... (21) $296.93

Corsair TR3X6G1600C8D Dominator 6 GB 3 x 2 GB PC3-12800 1600MHz... (14) $193.00

Western Digital VelociRaptor 300 GB Bulk / OEM Hard Drive 2.5 In... (30) $419.95

Corsair TR3X6G1333C9 XMS3 6 GB 3 x 2 GB PC3-10666 1333MHz 240-Pi... (13) $129.99

Corsair CMPSU-750TX 750-Watt TX Series 80 Plus Certified Power S... (77) $109.99

Intel Core 2 Quad Q9550 Quad-Core Processor, 2.83 GHz, 12M L2 Ca... (23) $219.99

Intel Core i7 920 2.66GHz 8M L3 Cache 4.8GT / sec QPI Hyper-Thre... (27) $279.99

NVIDIA.

Why recommendation system?

- Computationally heavy when data size is large
  - Sometimes it takes days or weeks
- Directly useful for business

How it works

Each person gives score to movies

person \ movie	W	X	Y	Z
A	5	4		
B	4	3		3
C				3
D			4	
E		1	5	
	v_W	v_X	v_Y	v_Z

A person who like movie “X” might also like…

Find a nearest vector to v_X regarding a blank as 0

➡ The answer is v_W

Answer: “W”

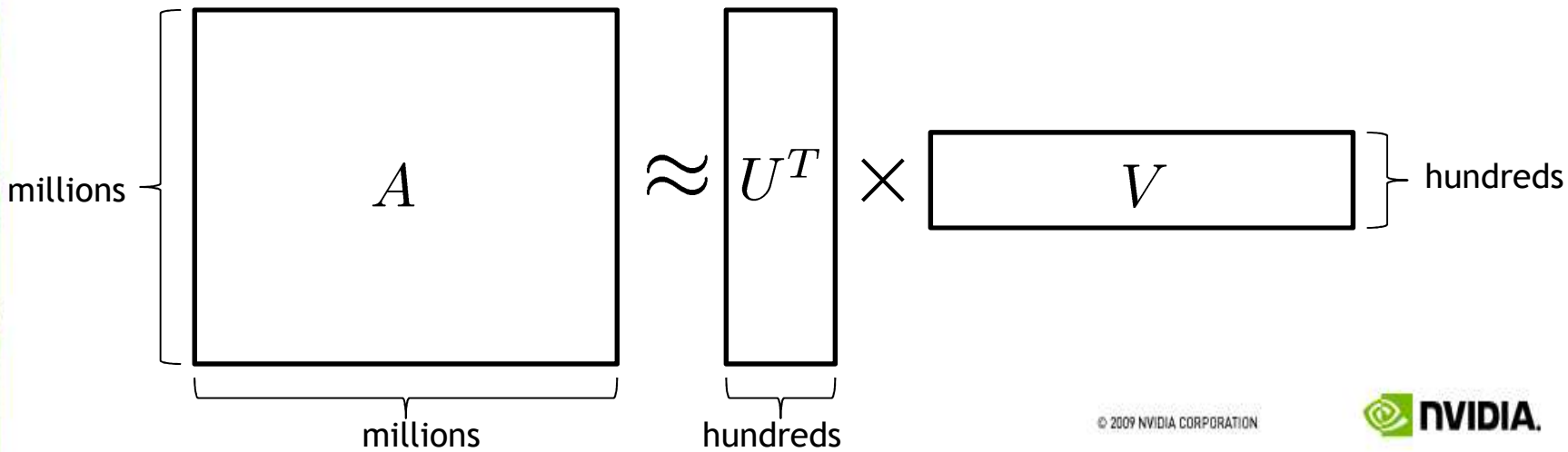
However…

- Usually the size of the matrix is very large
  - Imagine how many customs and items Amazon has
- Direct computation is too heavy
  - Millions of vectors and vectors of dimension of millions

Some kind of data compression is necessary

Singular Value Decomposition

- Approximates the given matrix with a product of smaller size matrices preserving “which is near” relationship
- Only a rough approximation is sufficient
- It is under the assumption that the phenomenon is explained by small number of features

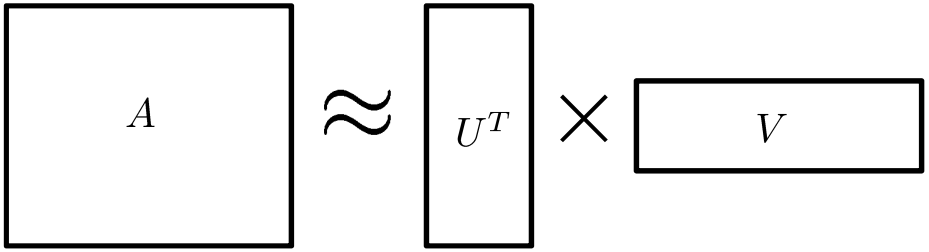


k-Nearest Neighbor Problem

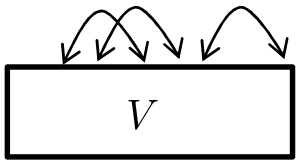
- For each vector, find k nearest vectors to it
- k is small enough compared to the size of matrix
  - Who cares the "10000th most likely movie you might like?"

Outline of the whole process

Singular Value Decomposition



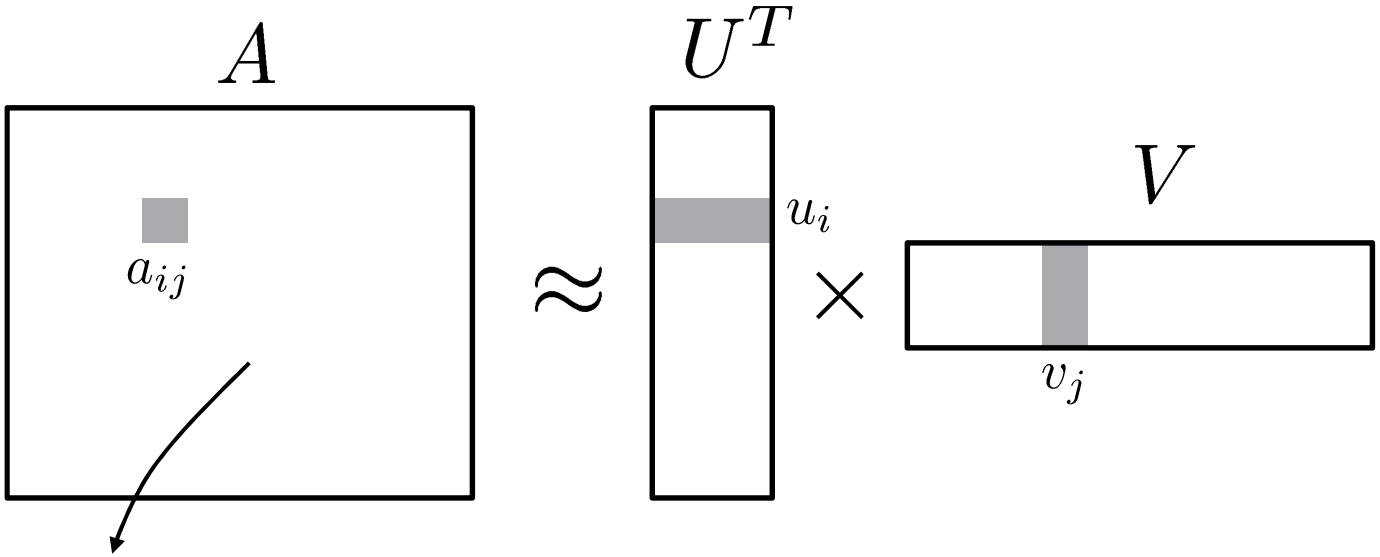
K-Nearest Neighbor



CUDA

- Programming environment with massive parallelism
  - Works on a GPU
  - Optimal for thousands of threads
- Hierarchical management of very light weighted threads
  - Each thread belong to a thread block
- Hierarchical memory system
  - Register, Shared memory, Global memory
- Locality of memory access is important
  - Memory access is coalesced if a certain rule is satisfied
  - Coalesced memory access gives a big performance gain

Computation of SVD



Sparse (expressed by index-value pair)

Error for each (i,j) is estimated by $\left(a_{ij} - u_i^T v_j\right)^2$

To diminish freedom of the variables and attain computational stability, smaller $|u_i|$ and $|v_j|$ are desirable

Minimize $E_{ij} = \left(a_{ij} - u_i^T v_j\right)^2 + \lambda\left(|u_i|^2 + |v_j|^2\right)$
for each (i,j) such that $a_{ij} \neq 0$

Known serial algorithm

Solve

Minimize $E_{ij} = \left(a_{ij} - u_i^T v_j\right)^2 + \lambda \left(|u_i|^2 + |v_j|^2\right)$

for each (i,j) such that $a_{ij} \neq 0$

by sequential steepest descent method

Algorithm by B.Webb (a.k.a. Simon Funk)

Fill u_i's and v_i's with random values

Repeat until convergence

For each (i,j) where $a_{ij} \neq 0$

$$u_i \leftarrow E_{ij} - \alpha \frac{\partial}{\partial u_i} E_{ij}$$

$$v_j \leftarrow E_{ij} - \alpha \frac{\partial}{\partial v_j} E_{ij}$$

It might look strange because…

The direction where

$$E_{ij} = \left(a_{ij} - u_i^T v_j\right)^2 + \lambda \left(|u_i|^2 + |v_j|^2\right)$$

gets smaller might make another $E_{i'j'}$ bigger

But anyway, practically Webb's algorithm works, and is time efficient

The steepest path for E_{ij} can make $E_{i'j'}$ bigger but is not steepest for $E_{i'j'}$

Analysis toward parallelization

Original algorithm by Webb

Fill u_i's and v_i's with random values
Repeat until convergence

For each (i,j) where $a_{ij} \neq 0$

$$u_i \leftarrow E_{ij} - \alpha \frac{\partial}{\partial u_i} E_{ij}$$

$$v_j \leftarrow E_{ij} - \alpha \frac{\partial}{\partial v_j} E_{ij}$$

Fill u_i's and v_i's with random values
Repeat until convergence

For each (i,j) where $a_{ij} \neq 0$

$$u_i \leftarrow E_{ij} - \alpha \frac{\partial}{\partial u_i} E_{ij}$$

For each (i,j) where $a_{ij} \neq 0$

$$v_j \leftarrow E_{ij} - \alpha \frac{\partial}{\partial v_j} E_{ij}$$

Update of u_i and v_j can be separated

u_{i_1} and u_{i_2} can be computed parallelly $(i_1 \neq i_2)$
v_{j_1} and v_{j_2} can be computed parallelly $(j_1 \neq j_2)$

Because E_{ij} only depends on u_i and v_j

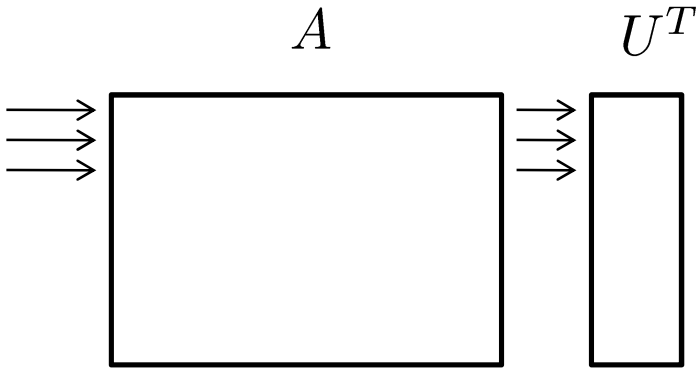
SVD in CUDA

Step1:

Each raw is assigned to a thread block

Each thread block computes u_i

$$u_i \leftarrow E_{ij} - \alpha \frac{\partial}{\partial u_i} E_{ij}$$

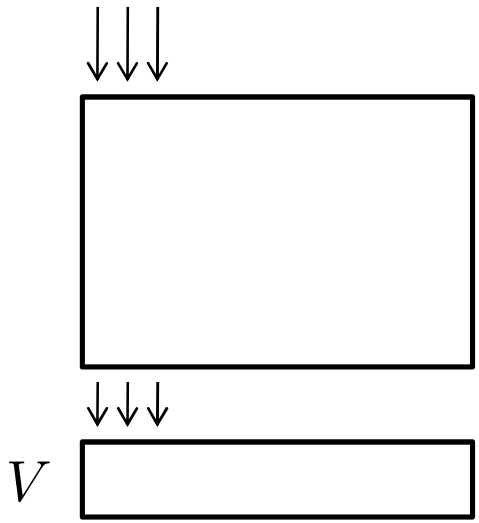


Step2:

Each column is assigned to a thread block

Each thread block computes v_j

$$v_j \leftarrow E_{ij} - \alpha \frac{\partial}{\partial v_j} E_{ij}$$



Detail of step 1

Each thread block calculates

$$u_i \leftarrow E_{ij} - \alpha \frac{\partial}{\partial u_i} E_{ij}$$

$$\frac{\partial}{\partial u_i} E_{ij} = -2(a_{ij} - u_i^T v_j) v_j + 2\lambda u_i$$

k-th thread calculates the k-th coordinate value of

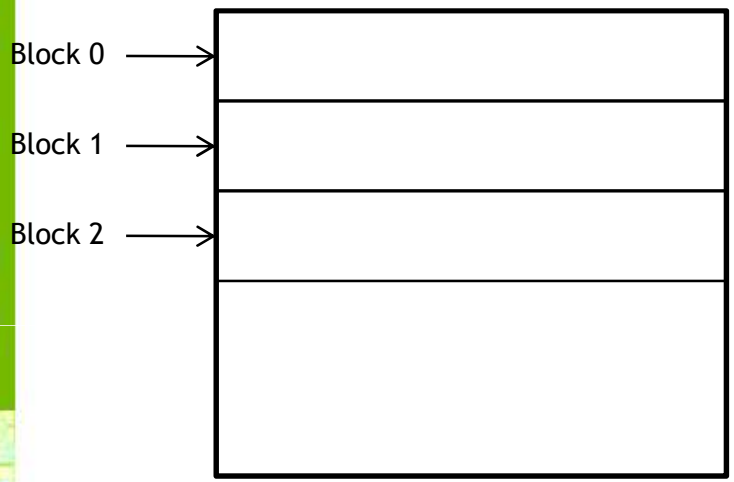
Compute $p = u_i^T v_j$ in parallel

(Reduction framework, see SDK sample of CUDA)

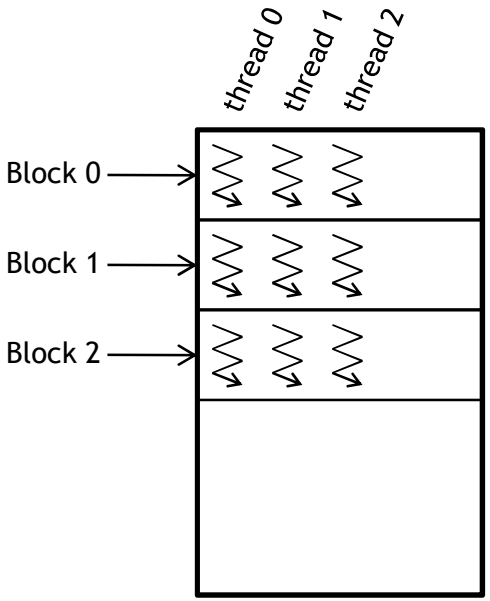
$$u_i \leftarrow E_{ij} - \alpha \left[-2(a_{ij} - p) v_j + 2\lambda u_i \right]$$

Same process for Step2

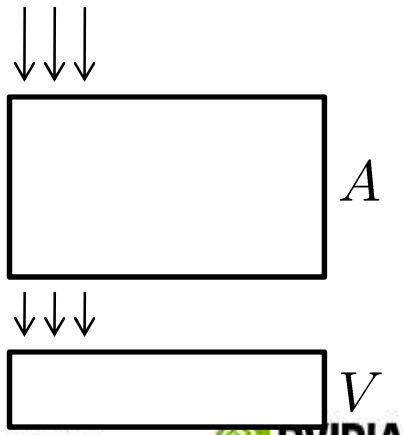
Outline of SVD



A



U^T



Problem of load balancing

- The workload among thread blocks are very unbalanced because the number of non-zero elements in each raw (column) differs
  - For example, some items are very popular and some are rarely bought.
- Basically thread blocks are executed in pipeline, so the unbalanced thread blocks don't affect the performance
- However, in fact, the order of thread blocks affect the performance
  - I don't know why

k-Nearest Neighbor Problem

Description of problem:

For a given k and set of vectors $\{v_1, v_2, \ldots, v_n\}$,
list up nearest k vectors to each v_i

In other words,

For each i ,
Find $i_1, i_2, \ldots, i_k$ such that

$$|v_i - v_{i_1}| \leq |v_i - v_{i_2}| \leq \cdots \leq |v_i - v_{i_k}| \quad (i_1, i_2, \cdots, i_k \neq i)$$

$$|v_i - v_{i_k}| \leq |v_i - v_{\bar{i}}| \quad \text{for} \forall \bar{i} \neq i, i_1, i_2, \cdots, i_k$$

Bruteforce vs Clustering

- For a performance reason, clustering algorithms such as k-means method are often used.
- Practically, it works well especially the data actually has some clusters
- But a bruteforce algorithm which calculates all the pair of vectors is more accurate

Bruteforce is better if it works fast

Our algorithm is bruteforce

k-Nearest Neighbor Algorithm

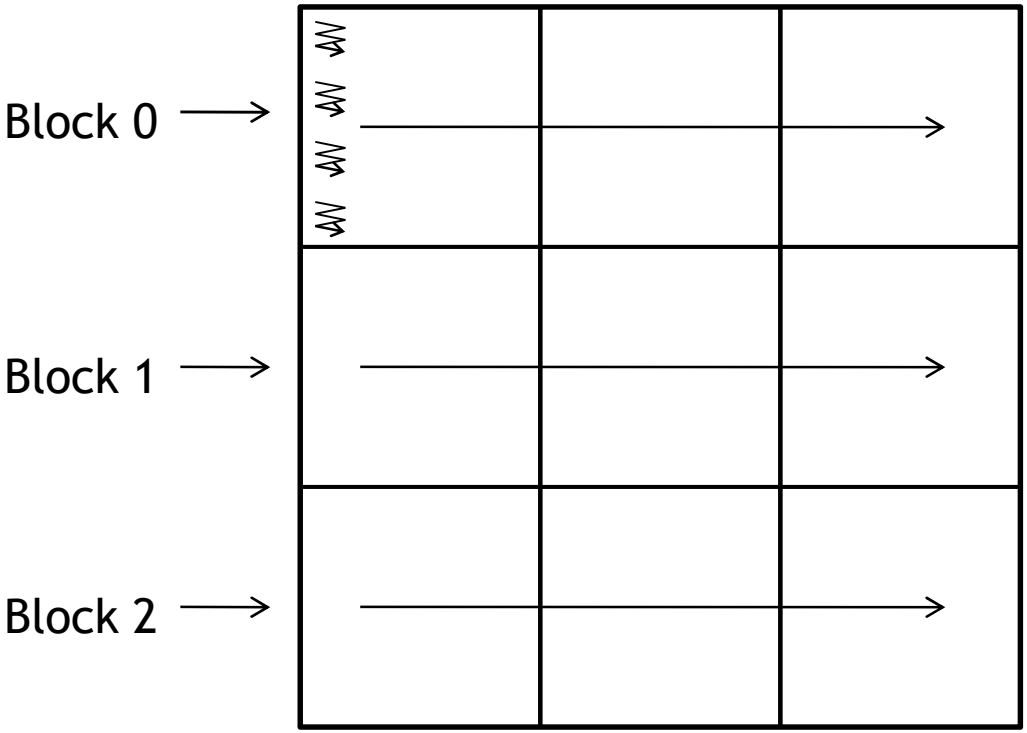
The algorithm consists of two parts:

Computation of distances
and
k-partial sort

Computation of distances

- Basically same as n-body algorithm
- But need another trick to deal with relatively high dimension

N-Body Algorithm [Nyland et al. 2008]

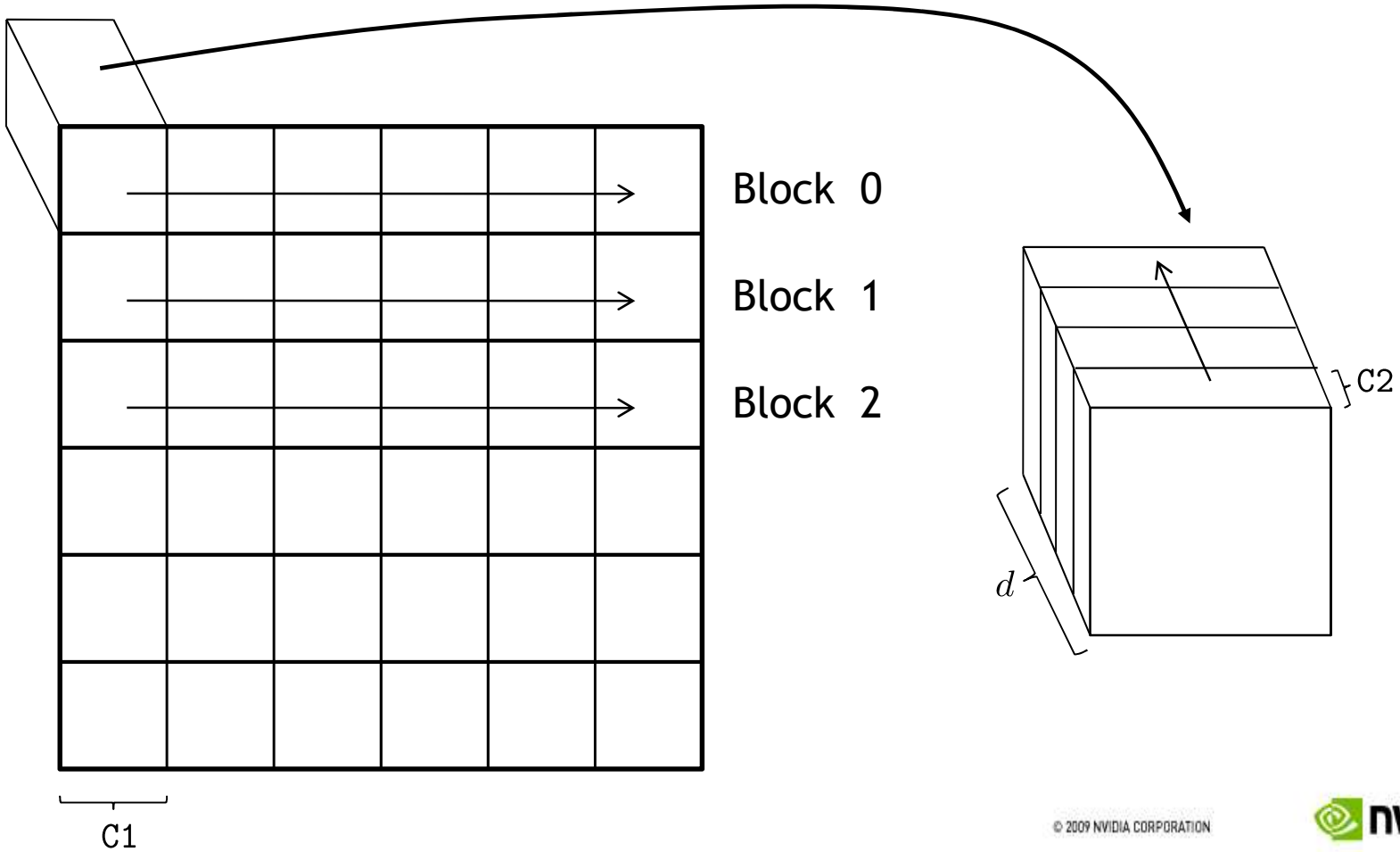


Because of limited size of shared memory

Nyland et al. “Fast N-Body Simulation with CUDA”, in GPU Gems III, pp 677—695, 2008

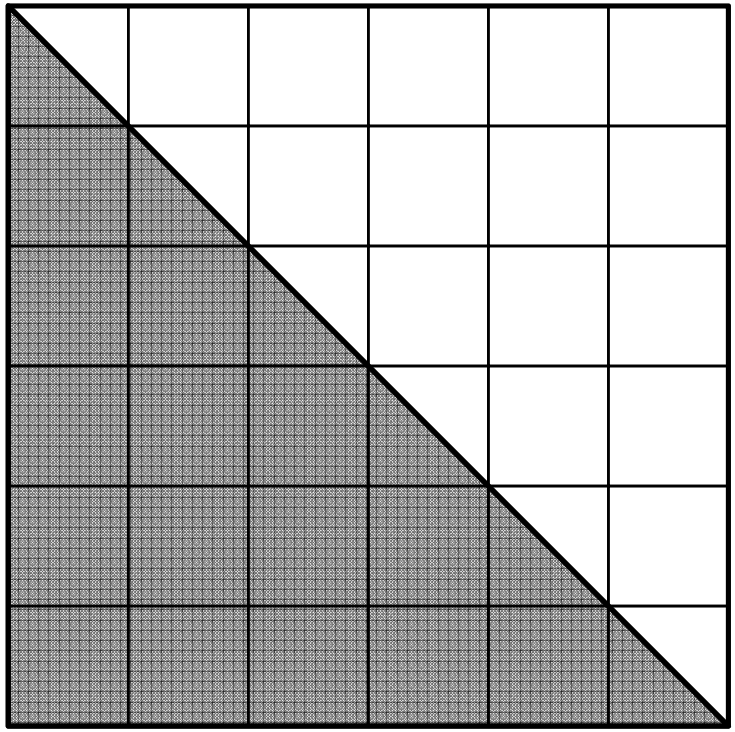
High dimensional case

"Slice" the dimension so that the data can be loaded in a shared memory



Note

- If the distance function is symmetric, it is enough to compute only the half of the distances.



Related research

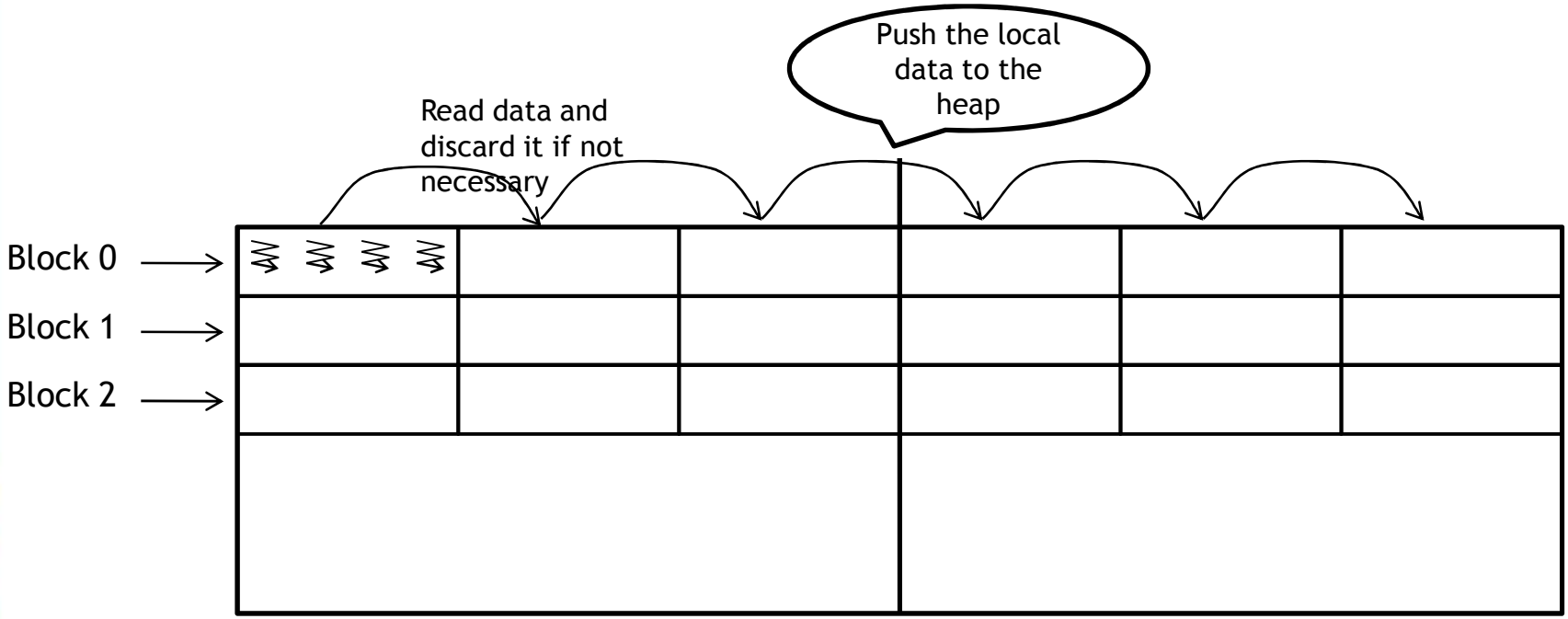
- For k-Nearest neighbor problem, Garcia et al. proposed an algorithm which uses texture memory.
- So far, we can not tell our algorithm is better or not.

V.Garcia et al. “Fast k-Nearest Neighbor Search Using GPU”, CVPRW ’08, 2008

k-partial sort in parallel

- Sort multiple arrays in parallel
- Use a heap so that the k-th smallest number can be found quickly
- Since k<<n, most of the data are discarded

k-partial sort in parallel



Each thread prepare a local array

Fetch a data and compare it to a current k-th smallest number

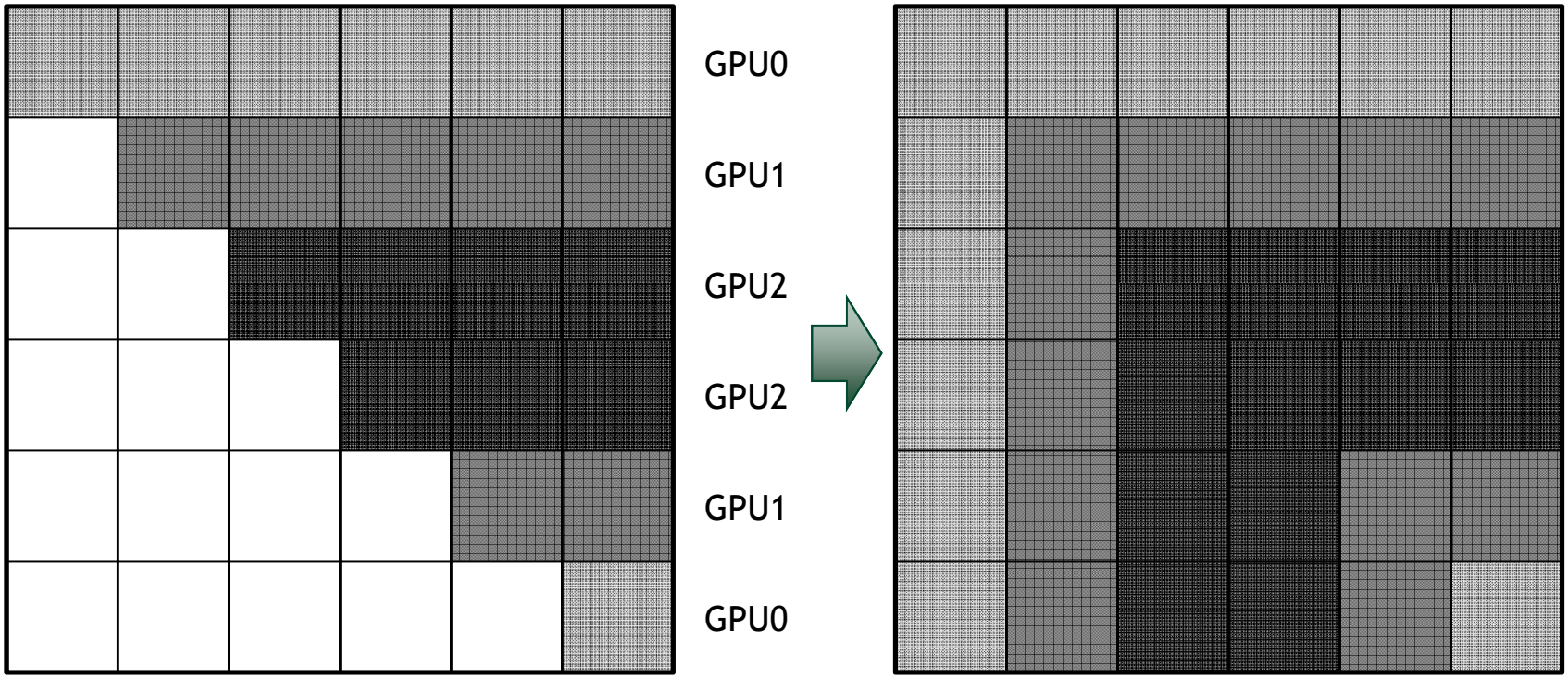
- Discard it if it is larger
- Store it to a local array if it is smaller

Atomicly push to the heap after a certain number of steps

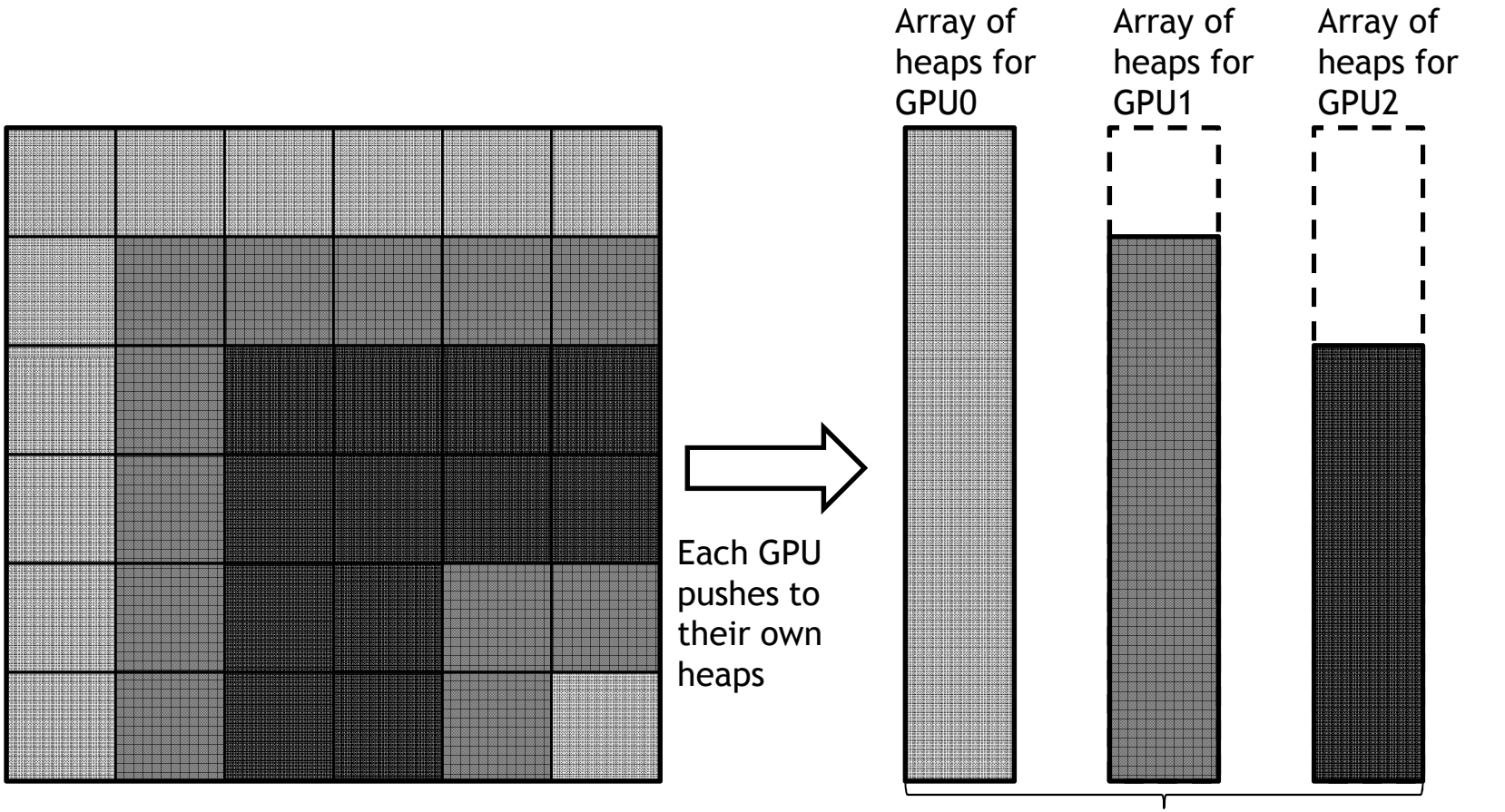
Research wanted: k-partial sort

- Useful for a general application
- Full sort is well studied (see [Cederman—Tsigas 2008] for example), but there might be a better implementation for a partial sort

For multiple GPUs



For multiple GPUs (cont.)



Bench mark (SVD)

Elapse time per iteration (sec)

n	10000	20000	40000
GTX280 (a)	0.2	1.0	4.1
Core i7 (b)	6.2	25.0	100.0
(b)/(a)	31.0	25.0	24.3

nxn matrix reduced dimesion=256 1% non-zero elements

CPU implementation is only on a single core

Bench mark (kNN)

Hellinger distance $\sum_i \left(\sqrt{v^{(i)}} - \sqrt{u^{(i)}}\right)^2$

n	10000	20000	40000	80000
1x GTX280 (a)	2.7	8.6	34.1	131.8
2x GTX280 (b)	1.8	5.7	17.7	68.6
Core i7 (c)	354.2	1419.0	5680.7	22756.9
(c)/(a)	131.1	173.3	166.5	172.6
(c)/(b)	196.7	248.9	320.9	331.7

Euclidian distance

n	10000	20000	40000	80000
1x GTX280 (a)	2.6	8.2	32.1	124.8
2x GTX280 (b)	1.7	5.5	16.9	65.2
Core i7 (c)	124.8	503.0	2010.0	8041.4
(c)/(a)	48.0	61.3	62.6	64.4
(c)/(b)	73.4	91,4	118.9	123.3

d=256, k=100

GPU TECHNOLOGY CONFERENCE

Conclusion

- A recommendation system can be faster on a GPU
- The k-nearest neighbor problem, which is the heaviest part of a recommendation system, has a scalable implementation on multiple GPUs
- GPU is also useful for marketing
  - CUDA is not only for scientists!

NVIDIA.