Efficient Lists Intersection by CPU-GPU Cooperative Computing

Yu Hai Zhang, Naiyong Ao, Gang Wang, Xiaoguang Liu, Jing Liu

Nankai-Baidu Joint Lab, Nankai University

Microsoft Research

Outline

Introduction

Cooperative Model

GPU Batching Algorithm

Experimental results

Related Work

Conclusion

Microsoft Research

Outline

Introduction

Cooperative Model

GPU Batching Algorithm

Experimental results

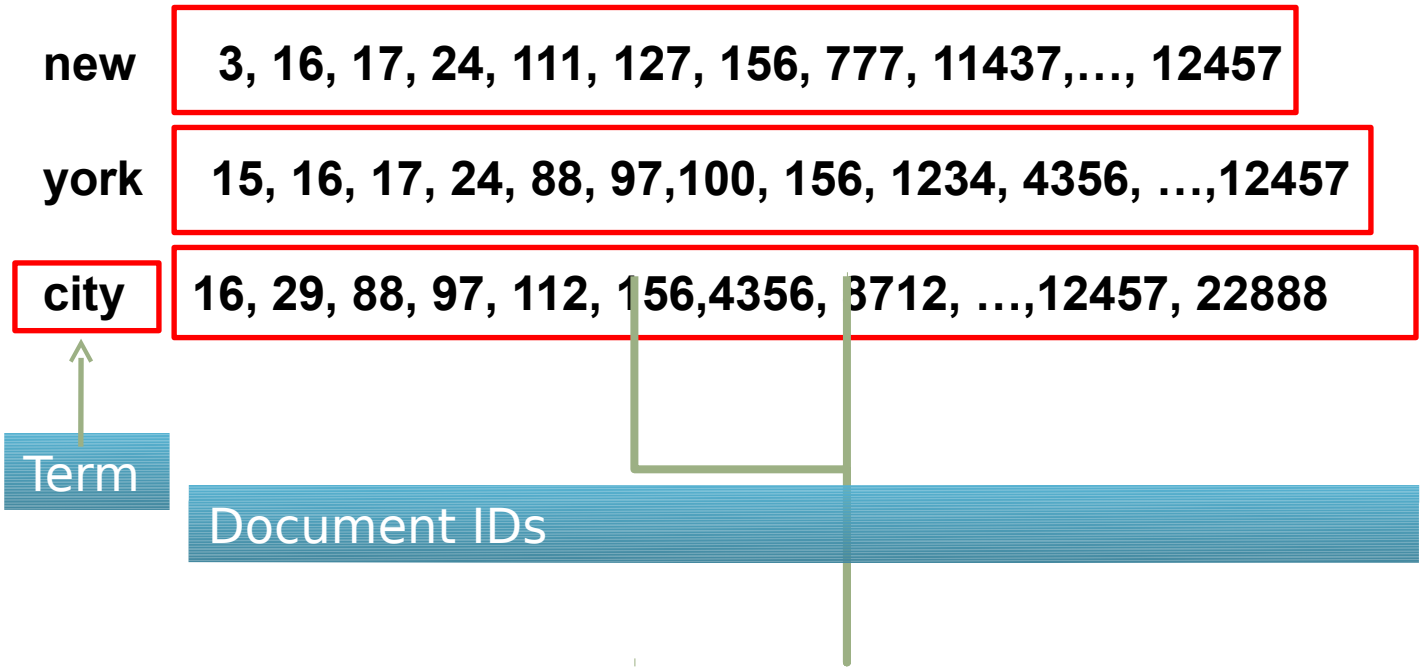
Related Work

Conclusion

Microsoft Research

Standard Query Processing

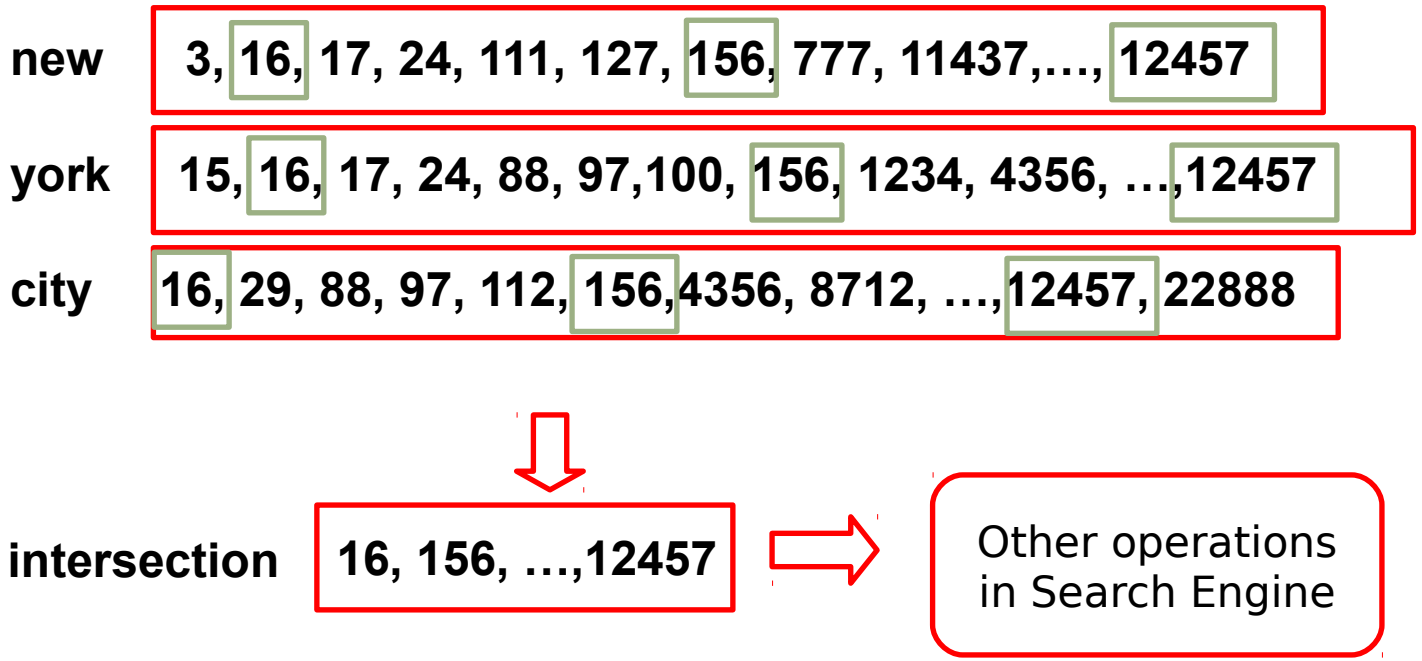
- What are inverted index and inverted lists?



Microsoft Research

Standard Query Processing(cont.)

When a query "*new york city*" submitted to the search engine, these 3 inverted lists will be loaded from the inverted index, and intersection operation will be applied.



Microsoft Research

Problem

Lists intersection operation occupies a significant part of CPU time in the modern web search engine

The query traffic could be quite heavy

Tens of thousands queries could arrive to one server in just one second

Response time: the less the better

Could the new GPU technology solve these problem?

Microsoft Research

Graphical Processing Units (GPUs)

Special purposes processors to accelerate applications

Driven by gaming industry

Powerful parallel computing ability

Nvidia's Compute Unified Device Architecture (CUDA)

A well-formed programming interface to the parallel architecture of Nvidia GPUs for general purpose computing

Our Goal

Improve the performance of lists intersection in real web search engines with the aid of GPU.

Microsoft Research

Outline

Introduction

Cooperative Model

GPU Batching Algorithm

Experimental results

Related Work

Conclusion

Microsoft Research

Cooperative Model

In practice, the load of a web search engine is changing every time

The system throughput and response time could be impacted seriously when system load fluctuates violently.

Traditional asynchronous mode

Newly arriving query is serviced by an independent thread

Some queries will be blocked by previous queries under heavy load

CPU-GPU cooperative model

Asynchronous mode

Under light load, system works in asynchronous mode

Every newly arriving query will be processed immediately

Before processing the query, we determine the query should be processed in which processor - CPU or GPU

Trade off: Long lists or short lists

Trade off: GPU kernel time and transferring time between CPU and GPU

Synchronous mode

Under heavy load, the system works in synchronous mode, queries are grouped in batches and processed by GPU

Firstly, Queries are blocked at CPU end and sent to GPU by group

Group size is decided according to the query load and response time limitation

Problem

How to design an efficient GPU batching algorithm?

Tradeoff between *throughout* and *response time*

Outline

Introduction

Cooperative Model

GPU Batching Algorithm

Experimental results

Related Work

Conclusion

Microsoft Research

GPU Intersection algorithm

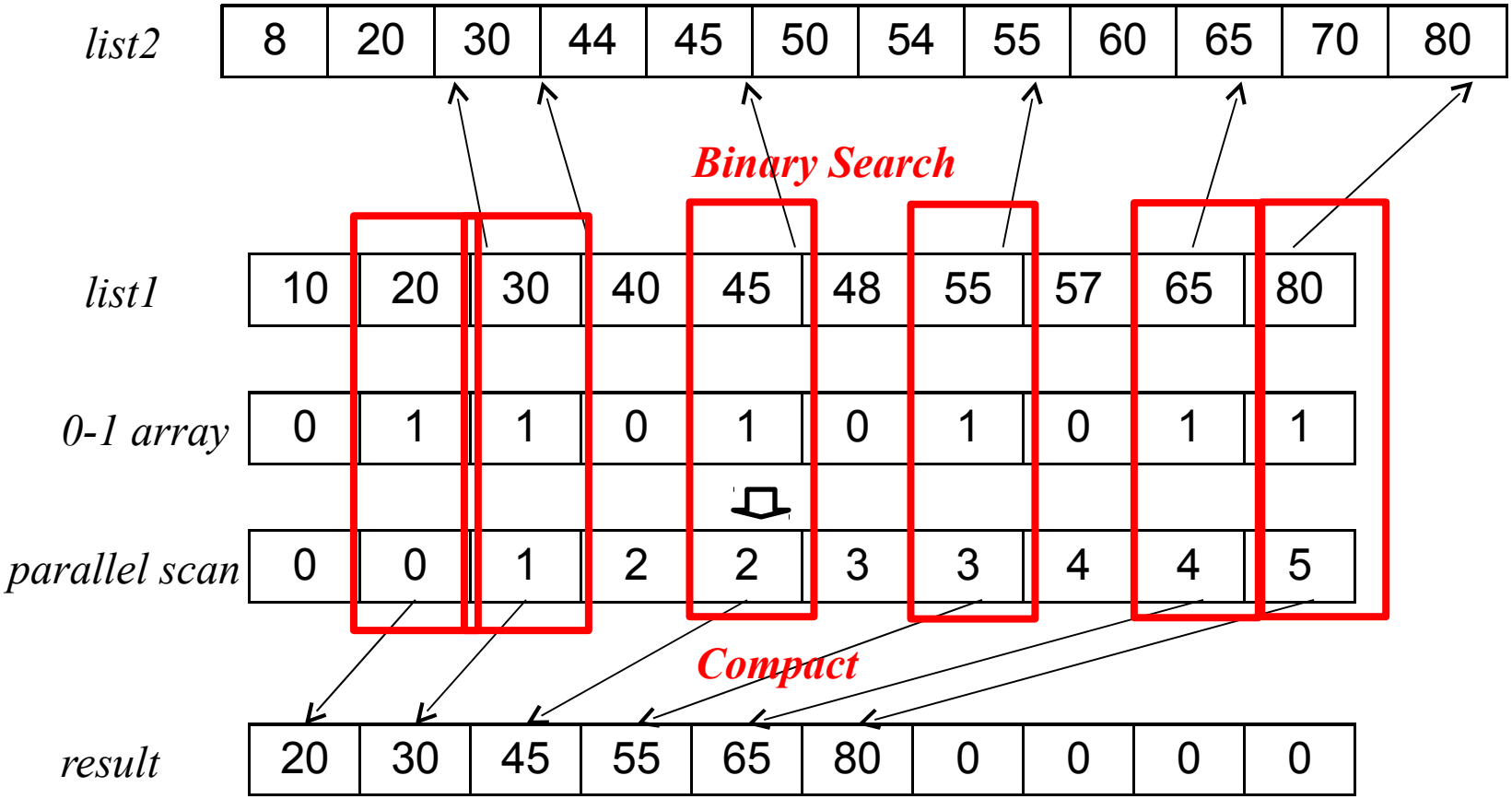
The basic idea for intersecting two lists intersection on GPU is *parallel binary search*

Assign each element of *list1* to a GPU thread

Do binary search in the *list2* to check whether the element is in *list2*

Use *scan* and *compact* operation to generate the final result

GPU Intersection algorithm(cont.)



Microsoft Research

GPU Batching Algorithms

Pump enough queries to CPU at a time to make full use of SPs in GPU

Problem

How should change the original GPU intersection algorithm

How should we partition the work to balance the load for each GPU thread?

How to decide the number of queries in each batch

Two GPU batching algorithms

Query-Partition algorithm (PART)

Query-Parallel Algorithm (PARA)

PART

In CUDA platform, threads are grouped in thread blocks

Synchronization between threads in different blocks is expensive

An intuitive idea to partition is assigning each query in the batch to a unique thread block

Queries may be quite different in lists' lengths, this lead to huge diversity of computation complexity

some multiprocessors idling while the other multiprocessors still busy on their (big) queries

PARA

Process a query by several blocks cooperatively according to its size instead of assigning each query to a single block

Every block will have similar amount of load

We will compare PARA and PART in 3 aspects next

CPU preprocessing

GPU processing

Data transferring

Microsoft Research

CPU preprocessing

When a batch of *N* queries are ready, CPU will first sort lists in each query by increasing length, and send the batch to GPU

N is determined by total computation load of queries in the batch

- Total computation load is estimated by a function of each query's shortest list's length (See in experiment section)

Compared with PART, PARA can control the total computation load delivered to GPU and load assigned to each block more precisely

GPU processing

Unlike PART's query-block mapping, PARA adopts element-thread mapping

PARA assigns each element in the shortest list to a unique thread

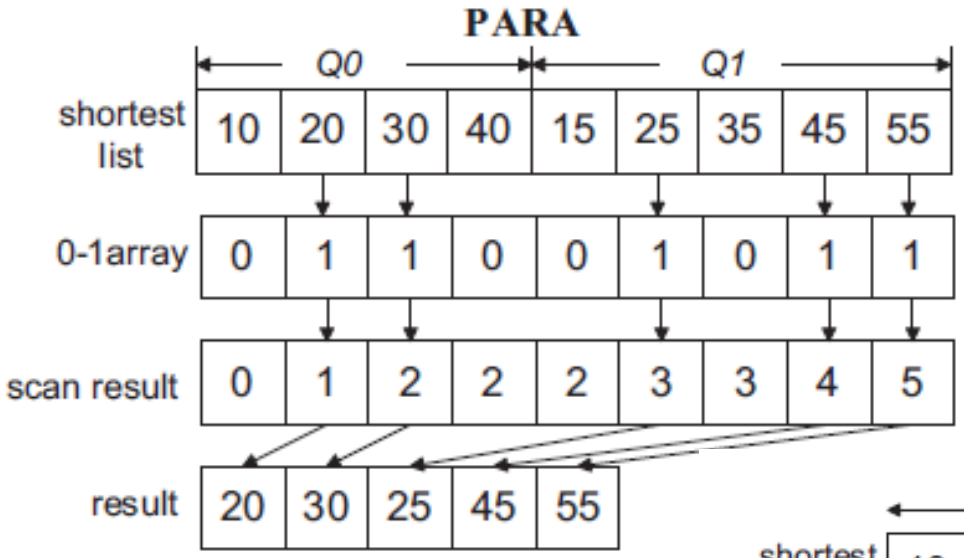
PARA is more likely to distribute computation load evenly

PART and PARA both use binary search to check element, but there are some differences in the compact phase

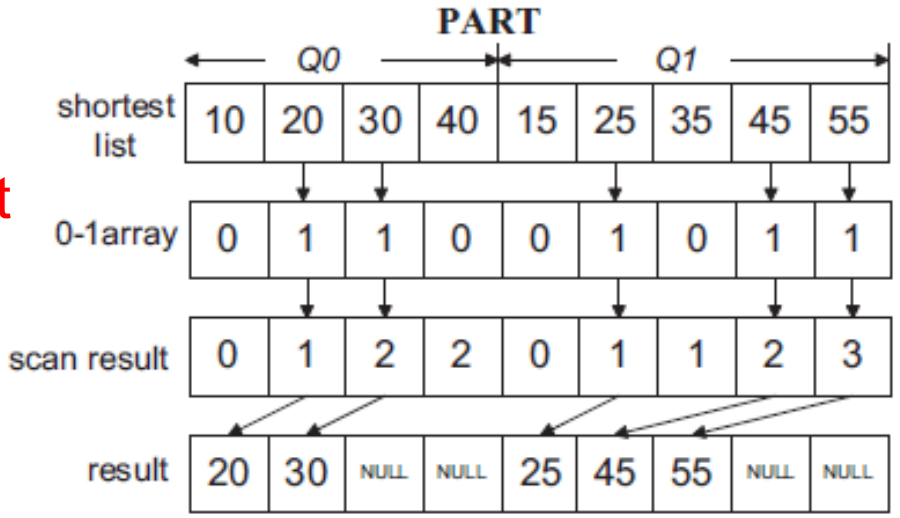
For PARA: each thread is responsible for an element, a global scan is used.

For PART: each query is processed by a single block, so each block executes a *sectionalized* scan algorithm

GPU processing(cont.)



PARA will transfer less result data back to CPU!!



Microsoft Research

Data Transferring

The GPU(4GB global memory) we use could hold the two data sets, we upload the whole data set to GPU when initialization.

In a large-scale search engine, we could put those inverted lists which are most frequently accessed in GPU memory

For each batch, necessary information, such as terms of each query are uploaded to GPU before processing

The result data is sent back to CPU when a batch queries processed

Outline

Introduction

Cooperative Model

GPU Batching Algorithm

Experimental results

Related Work

Conclusion

Microsoft Research

Environment

CPU
- PhenomIIX
- AMD

Memory
- 2GB*2
- DDR3 1333 memory

GPU Card
- C1060
- NVidia

Microsoft Research

PARA on GOV data set

Computation threshold is used to control how many query a batch contains

We set the computation threshold according to the factors below

The computing power of GPU

Required system throughput

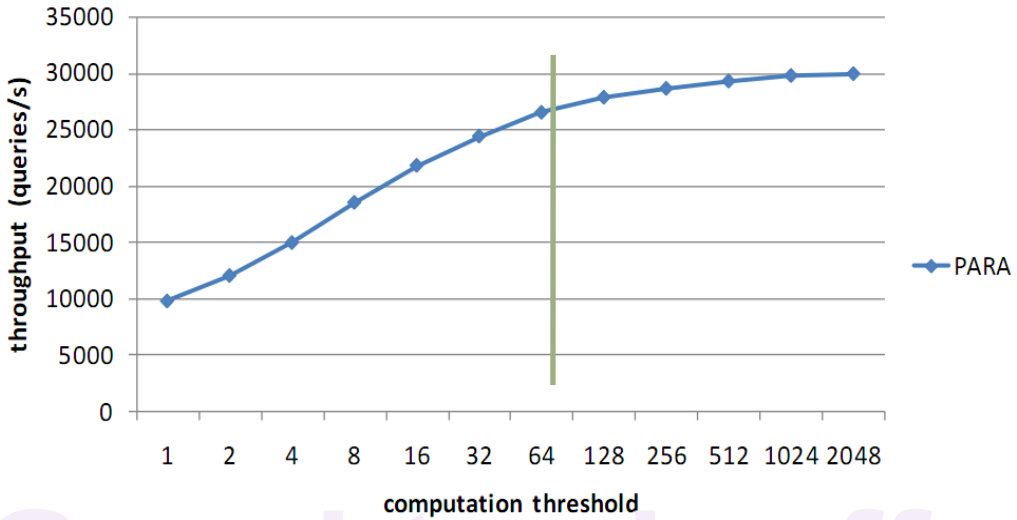
Required response time

We use “number of thread blocks on every SM” as the threshold

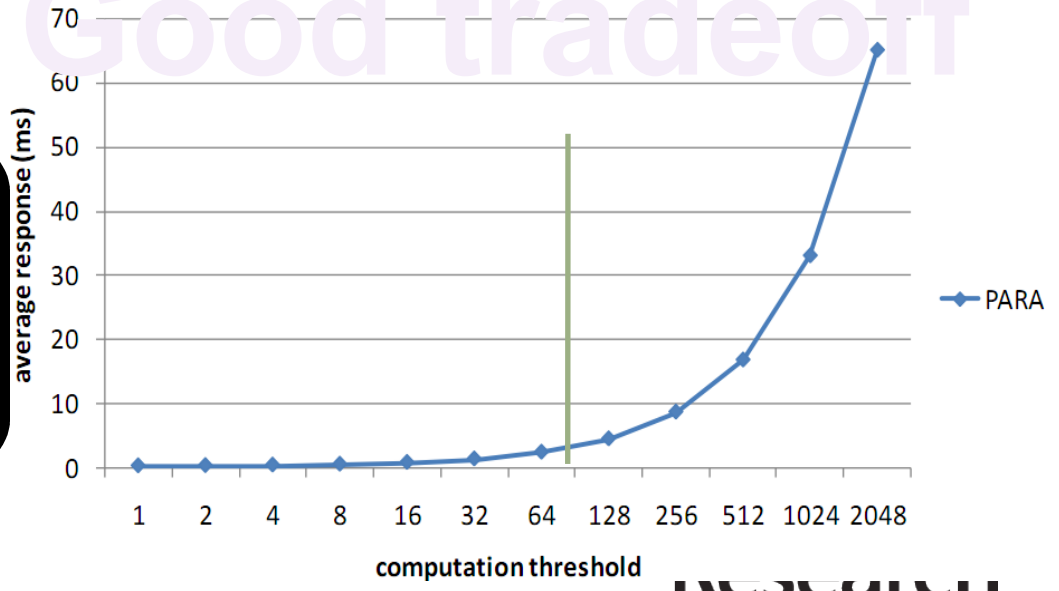
Microsoft Research

PARA on GOV data set

throughput



response

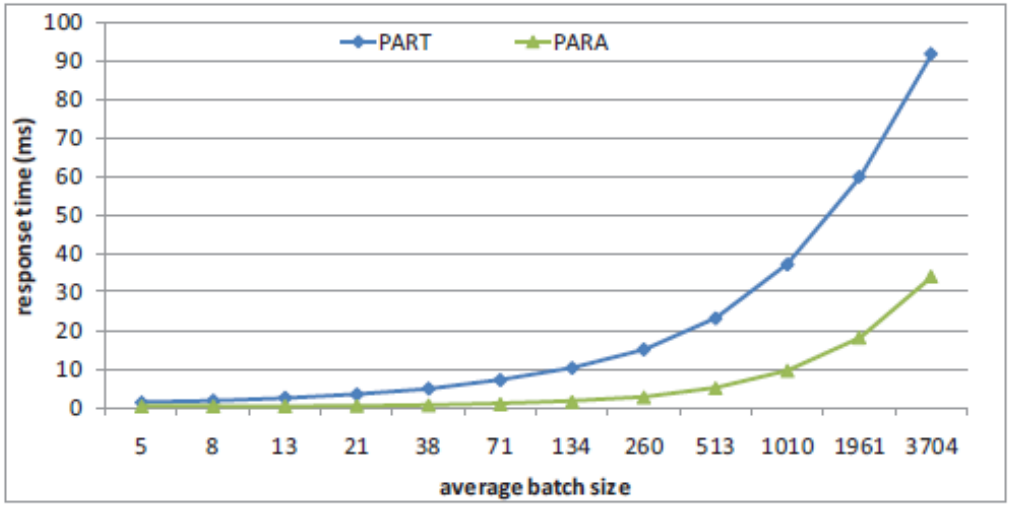


Good tradeoff

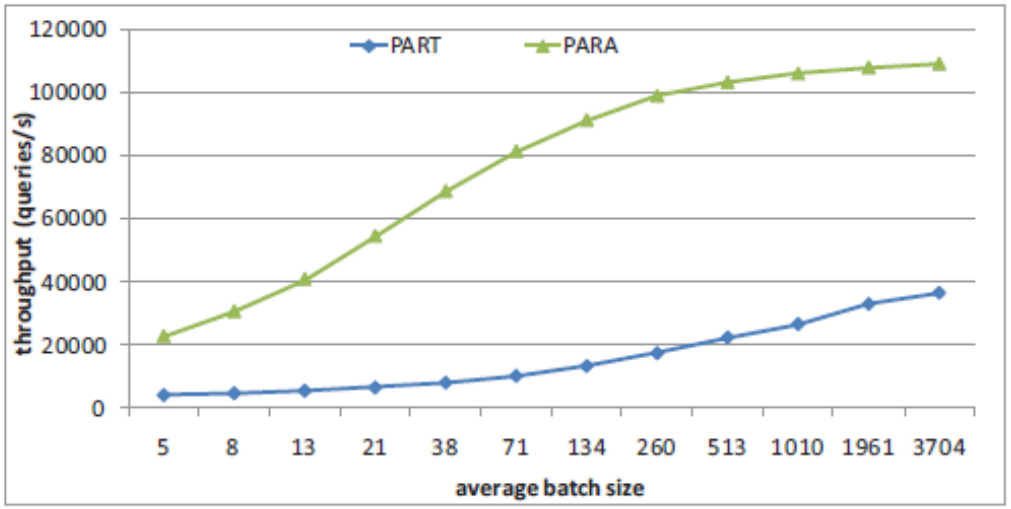
Research

PART VS PARA

Response time



Throughput



Microsoft Research

Response time fluctuation

Response time fluctuation is bad to search engine

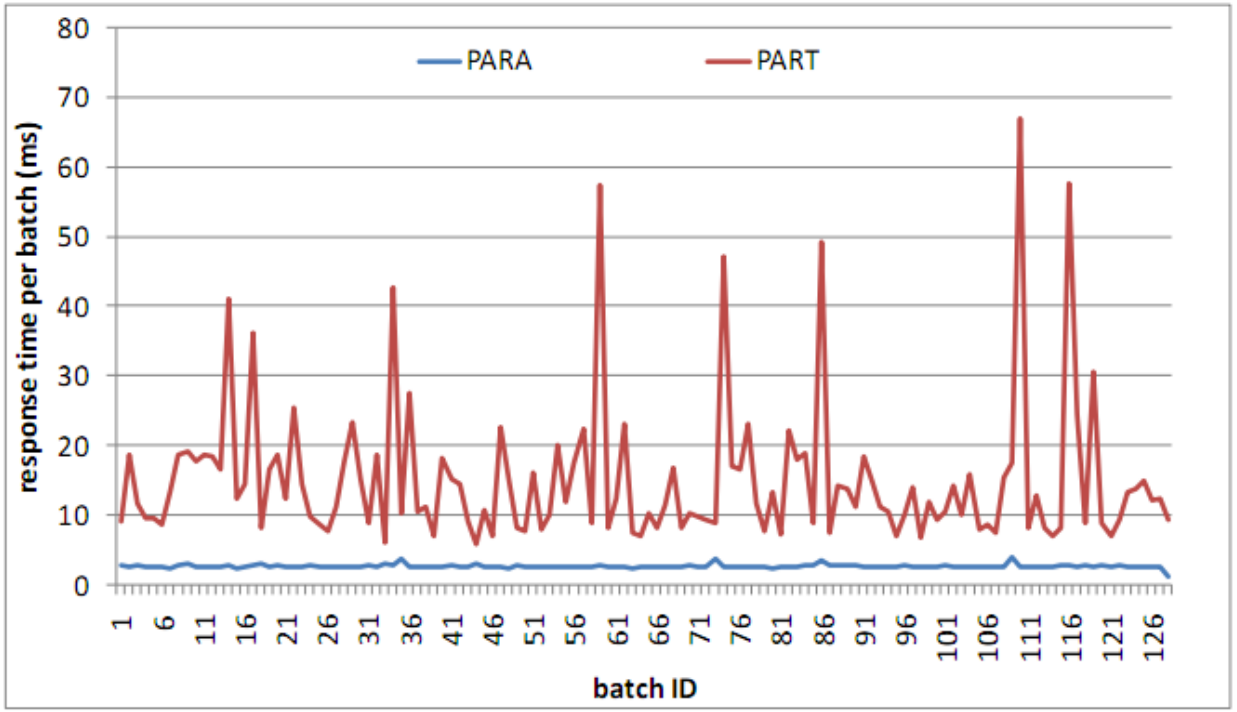
Violent fluctuations mean horrible user experience

Also, it will be difficult for administrator to predict system performance

Therefore, it is an important metric for real time system

Microsoft Research

Response time fluctuation



Response time per batch

Blue line for PARA
Red line for PART

Response time in PARA is stable

PARA assembles batches according to computational complexity, so all batches have almost the same computation load

Microsoft Research

Query scheduling under asynchronous mode

If query load is light, system works in asynchronous mode

Both CPU and GPU can offer enough throughput

processing queries by CPU may lead to better response time

It is helpful to energy-saving by letting GPU idle

We need a routing-algorithm to decide which device to deal with the query, CPU or GPU

Microsoft Research

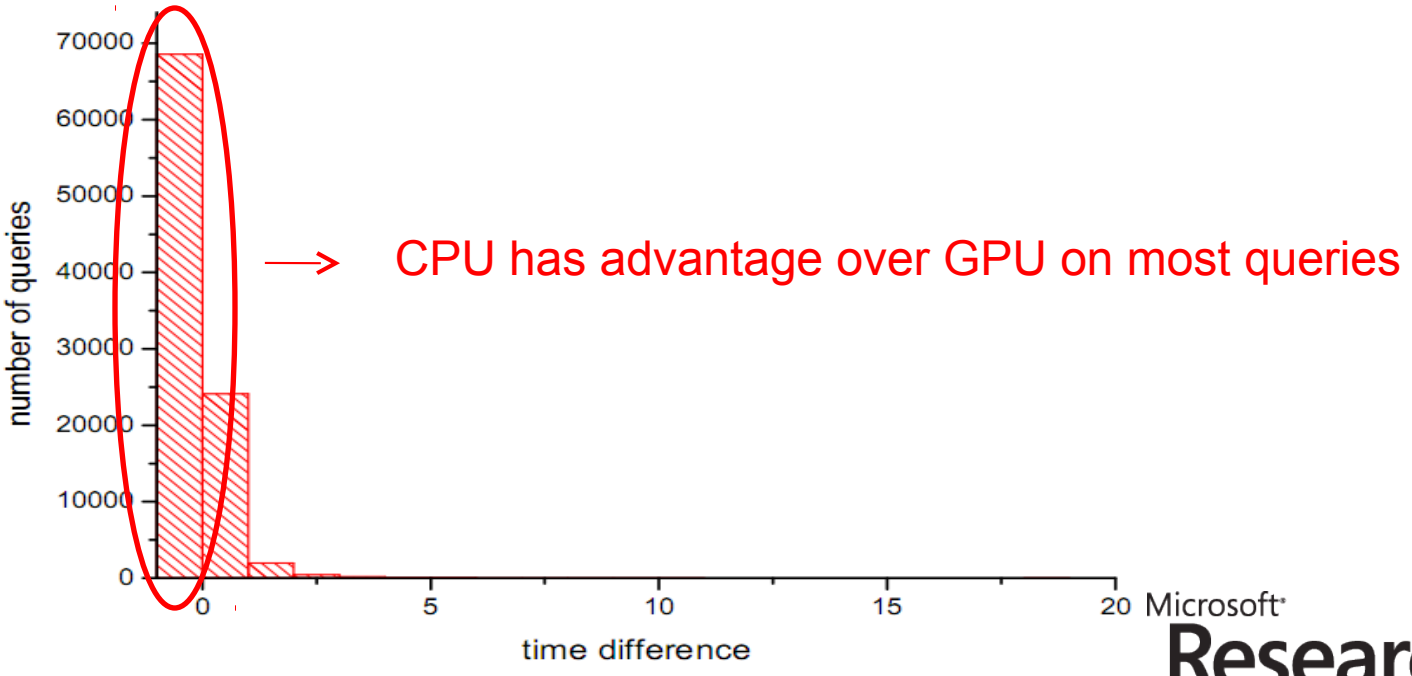
Route algorithm

Histogram

x-axis shows the time difference (CPU Time - GPU Time) per query

y-axis shows the number of queries

CPU has advantage over GPU on most queries, as these queries contains low computation complexity(short lists)



Microsoft Research

Route algorithm(cont.)

Graph

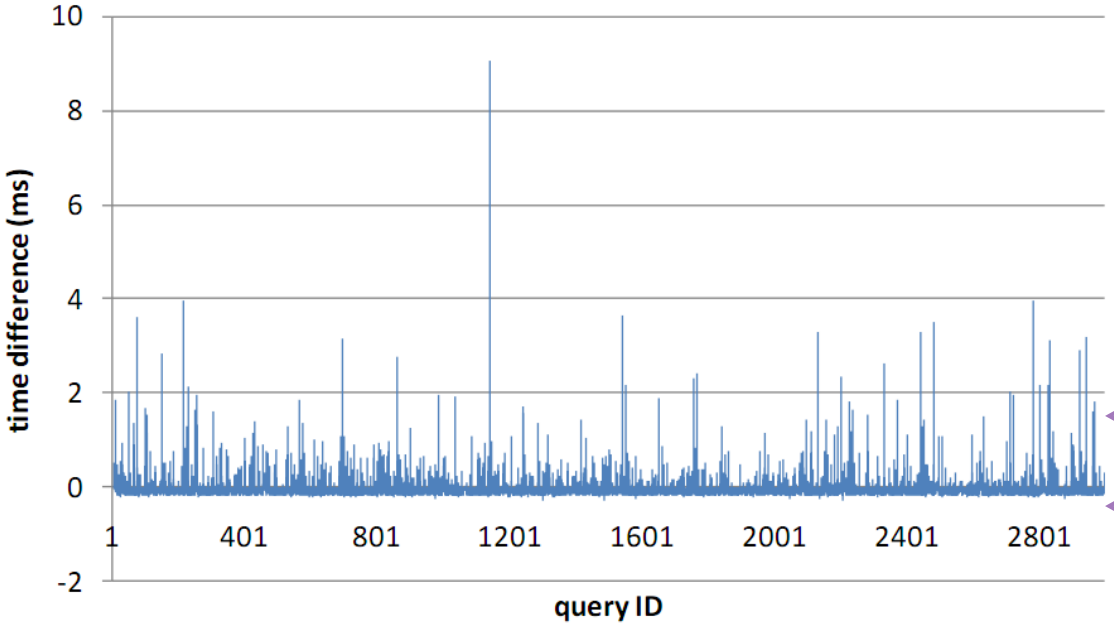
X-axis: query ID (we count 3000 queries, GOV data set)

Y-axis: time difference (CPU Time- GPU Time)

Compare

CPU's s advantage is not significant

GPU is far superior in the queries whose computation complexity is high



How can we measure the computation complexity in each query?

GPU advantage

CPU advantage

Microsoft Research

Route algorithm(cont.)

What we have

The number of lists in each query

The length of each list

That is all …

The information is not enough

We do not know:

- How many docIDs are common docIDs
- Number of comparisons of each docIDs

Microsoft Research

Route algorithm(cont.)

We use statistical methods

We run each query (training set) on CPU and GPU separately, record the time difference

We introduce three metric to estimate the computational complexity

The scheduling algorithm boils down to the relationship between the time difference and each metrics.

We adopt regression analysis

- to test the correlation between each metric and actual time difference

Microsoft Research

Route algorithm(cont.)

Metric to each query:

LOS: the length of the shortest list

UBOS: the upper bound of the number of comparisons

- UBOS = LOS * (logL1 + logL2 + …)

UBOCT: the upper bound of the number of comparisons per thread

- If one query is processed fully in parallel on GPU, UBOCT will be good metric

Route algorithm

Result

R-square is the coefficient of determination, which is the proportion of variability in dataset that is accounted for by the metric

Regression formula:

$$TimeDiff = -0.14164 + (3.7849E - 7) * UBOC$$

TABLE II

REGRESSION ANALYSIS ON TIME DIFFERENCE AND SCHEDULING METRICS

Winner!

metric	r-square	p-value	coefficient	Intercept
LOS	0.6312	< 0.0001	0.00001881	-0.13488
UBOC	0.7002	< 0.0001	3.784915E-7	-0.14164
UBOCT	0.0125	< 0.0001	0.00141	-0.04813

Microsoft Research

Route algorithm

How to use the formula

When CPU gets a query under light load, it calculates UBOC first

Then, timediff is got from the regression formula

If timediff is positive, the query will be routed to the GPU, which means GPU may process it faster.

- Otherwise, CPU will process the query by itself

Conclusion

We present a CPU-GPU cooperative model which can dynamically switch between the asynchronous mode and synchronous mode

Under light load, the system works in asynchronous mode. We minimize query response time in the aid of GPU. Heuristic strategies are designed to decide whether the current query should be processed by GPU or CPU.

Under heavy load, the system works in synchronous mode. We propose a query-parallel algorithm to balance the load between thread blocks, therefore process a batch efficiently.

Microsoft Research

Microsoft Research

Questions?

Microsoft Research