グリッドコンピューティング

2012/11/26

徐 駿剣(12M54060 脇田研究室)

紹介論文

Efficient Parallel Graph Exploration on Multi-Core CPU and GPU

[PACT 11]

Sungpack Hong, Tayo Oguntebi, Kunle Olukotun

Pervasive Parallelism Laboratory
Standford University

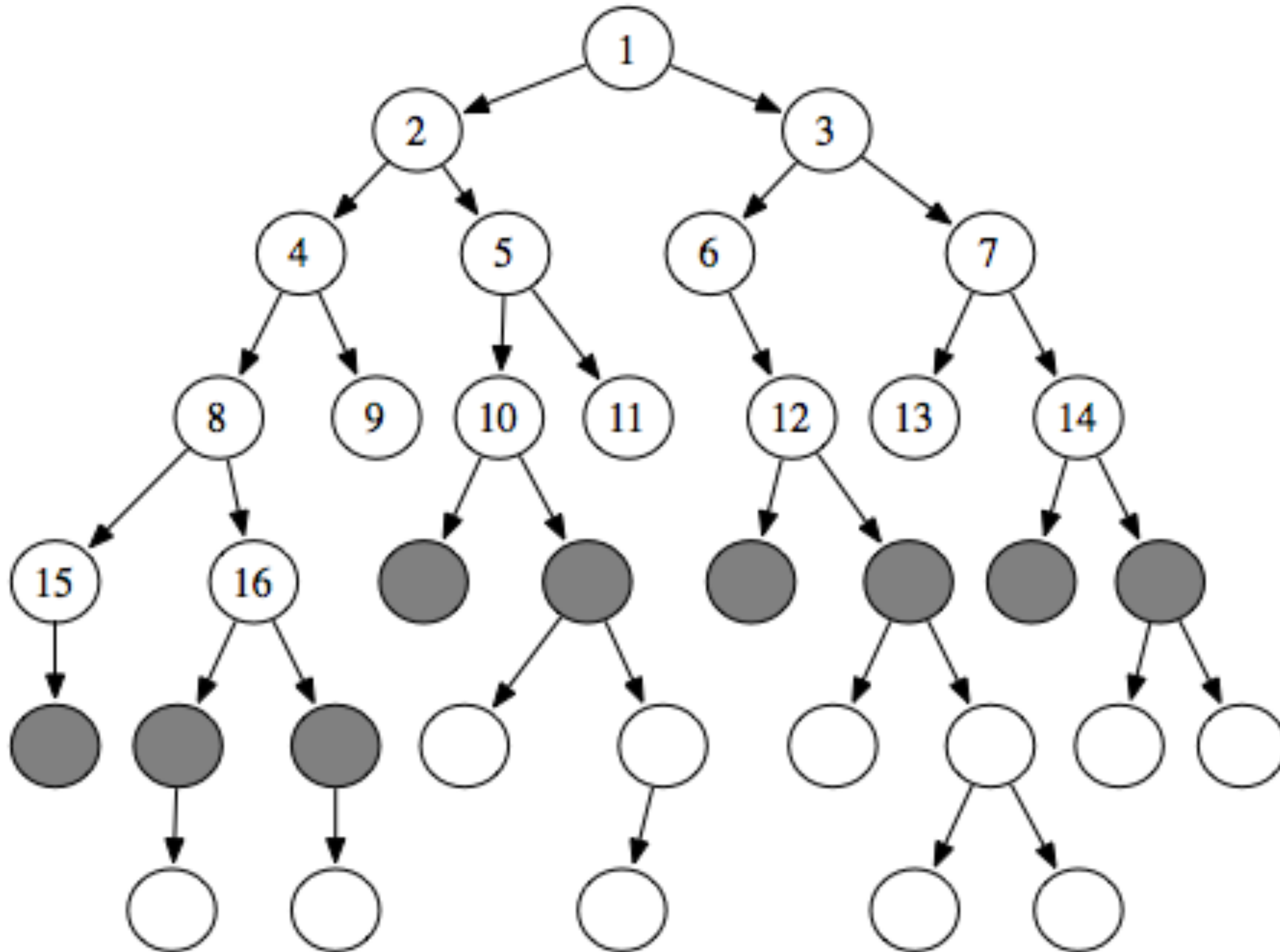
Outline

- Motivation
- Natural Parallel BFS Algorithm
- New Method for Multi-Core CPU
- Hybrid Methods
- Experiments

Outline

- Motivation
- Natural Parallel BFS Algorithm
- New Method for Multi-Core CPU
- Hybrid Methods
- Experiments

Breadth First Search (BFS)



Motivation

- Proliferation of parallelism and heterogeneity (simultaneous use of CPU and GPU)
- BFS serves as a building block for many graph algorithms
    - centrality calculation
    - connected component identification
    - community structure detection
    - max-flow computation

Related Research

- state-of-the-art BFS implemention
  - multi-core systems
  - reduce cache coherence traffic

- BFS implmentation for GPUs
  - solved the workload imbalance issue
  - good performance compared to multi-core CPU implementions

Outline

- Motivation
- Natural Parallel BFS Algorithm
- New Method for Multi-Core CPU
- Hybrid Methods
- Experiments

Level Synchronous BFS Algorithm

Algorithm 1 Level Synchronous Parallel BFS

```
1:  procedure BFS(r:Node)
2:      V = C = ∅; N = {r}                    ▷ Visited, Current, and Next set
3:      r.lev = level = 0
4:      repeat
5:          C = N
6:          for Node c ∈ C do                 ▷ in parallel
7:              for Node n ∈ Nbr(c) do        ▷ in parallel
8:                  if n ∉ V then
9:                      N = N ∪ {n}; V = V ∪ {n}
10:                     n.lev = level + 1
11:         level++
12:     until N = ∅
```

Shortcomings

1. Synchronization overhead needs to be paid at every level

2. Amount of available parallelism is limited by the number of nodes in a given level

Small World Phenomenon

- Diameters of real-world graphs are small even for large graph instances

Level	Num. Nodes	Fraction (%)[+]
0	1	$3.1*10^{-6}$
1	4	$1.3*10^{-5}$
2	749	$2.0*10^{-3}$
3	109,239	0.34
4	7,103,690	22.20
5	9,088,766	28.40
6	130,298	0.41
7	172	$5.3*10^{-4}$
total visited nodes	16,432,919	51.35
total visited edges	255,962,977	99.99

(+) Fraction of the total number of nodes (edges) in the graph

Outline

- Motivation
- Natural Parallel BFS Algorithm
- New Method for Multi-Core CPU
- Hybrid Methods
- Experiments

Queue-based Method

1. Use bitmap to represent the visited set
2. Use 'test and test-and-set' operation when updating bitmap
3. Use local next-level queues
4. Maintain next-level implemented with ticket-locks and fast-forwarding algorithm

Pseudo Code

```
1   BFS_Queue(G: Graph, r: Node) {
2     Queue N, C, LQ[threads];
3     Bitmap V;
4     N.push(r); V.set(r.id);
5     int level = 0; r.lev = level;
6     while (N.size() > 0) {
7       swap(N,C); N.clear(); // swap Curr and Next
8       fork;
9         foreach(c: C.partition(tid)) {
10          foreach(n: c.nbrs) {
11            if (!V.isSet(n.id)) { // test and test-and-set
12              if (V.atomicSet(n.id)) {
13                n.lev = level+1;
14                LQ[tid].push(n);  // local queue
15                if (LQ[tid].size()==THRESHOLD){
16                  N.safeBulkPush(LQ[tid]); // global queue
17                  LQ[tid].clear();
18        } } } } }
19        if (LQ[tid].size() > 0) {
20          N.safeBulkPush(LQ[tid]);
21          LQ.clear();
22        }
23      join;
24      level++;
25  } }
```

Improvement

- The final optimization technique dont works when the size of input becomes very large
- This paper take a different approach
  - efficient use of **memory bandwidth**

Read-based Method

- Instead of a shared queue, read-based method manages a single O(N) array that tells if a node belongs to the current-level set, next-level set, or visited set

Pseudo Code

```
26  BFS_Read(G: Graph, r: Node) {
27    Bitmap V;
28    Bool fin[threads];
29    V.set(r.id);
30    int level = 0; r.lev = level;
31    bool finished = false;
32    while (!finished) {
33      fork;
34        fin[tid] = true;
35        foreach(c: G.Nodes.partition(tid)) {
36          if (c.lev != level) continue;
37          foreach(n: c.nbrs) {
38            if (!V.isSet(n.id)) { // test and test-and-set
39              if (V.atomicSet(n.id)) {
40                n.lev = level+1;
41                fin[tid] = false;
42        } } } }
43      join;
44      finished = logicalAnd(fin, threads);
45      level++;
46  } }
```

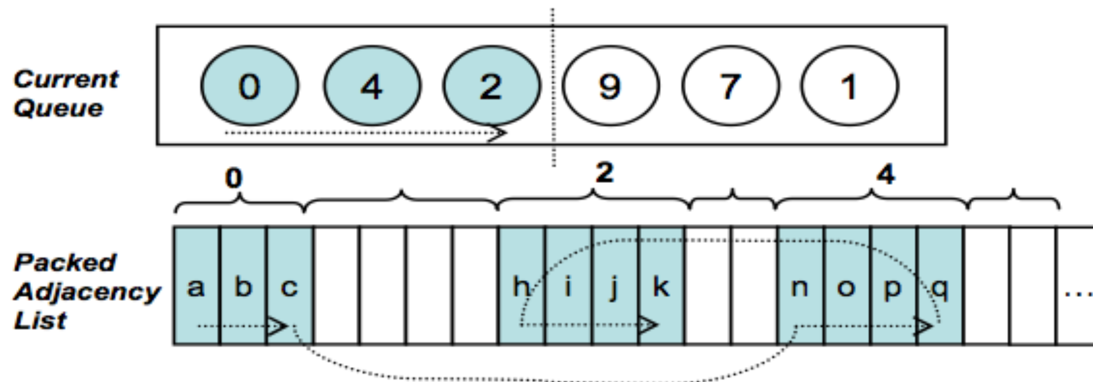
Advantages

1. Complete free from queue overhead
   a. remove atomic instructions used for the queue operations
   b. save on cache and memory bandwidth

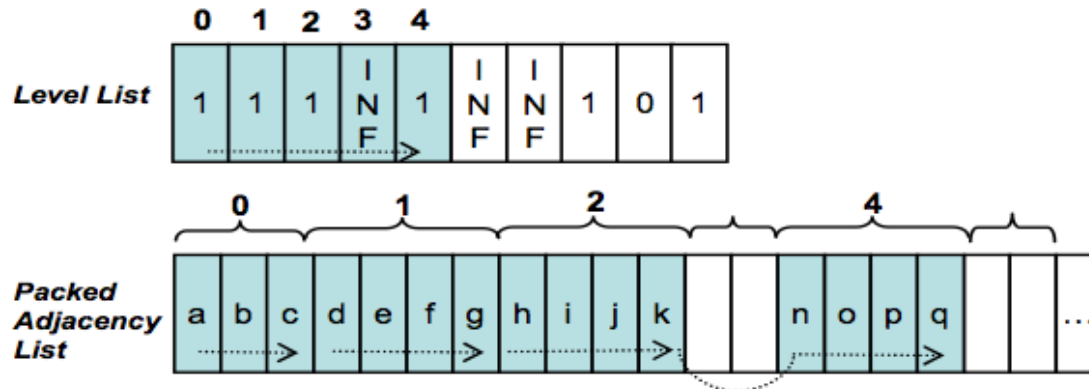
2. Memory access pattern is more sequential

Machine	Seq. Read	Random Read
Nehalem CPU	8.6 GB/s	0.98 GB/s
Core CPU	3.0 GB/s	0.25 GB/s
Fermi GPU	76.8 GB/s	2.71 GB/s
Tesla GPU	72.5 GB/s	3.15 GB/s

Data Access Pattern



(a) Data-Access Pattern of Queue-Based Method



(b) Data-Access Pattern of Read-Based Method

Discussion

- The primary disadvantage that it reads out the entire array every level iteration seldom affects the overall performance
  - graph diameter is small
  - sequential reading works well at critial level

- Undesirable graph
  - small (sub-)graph
  - long diameter graphs such as meshes

Outline

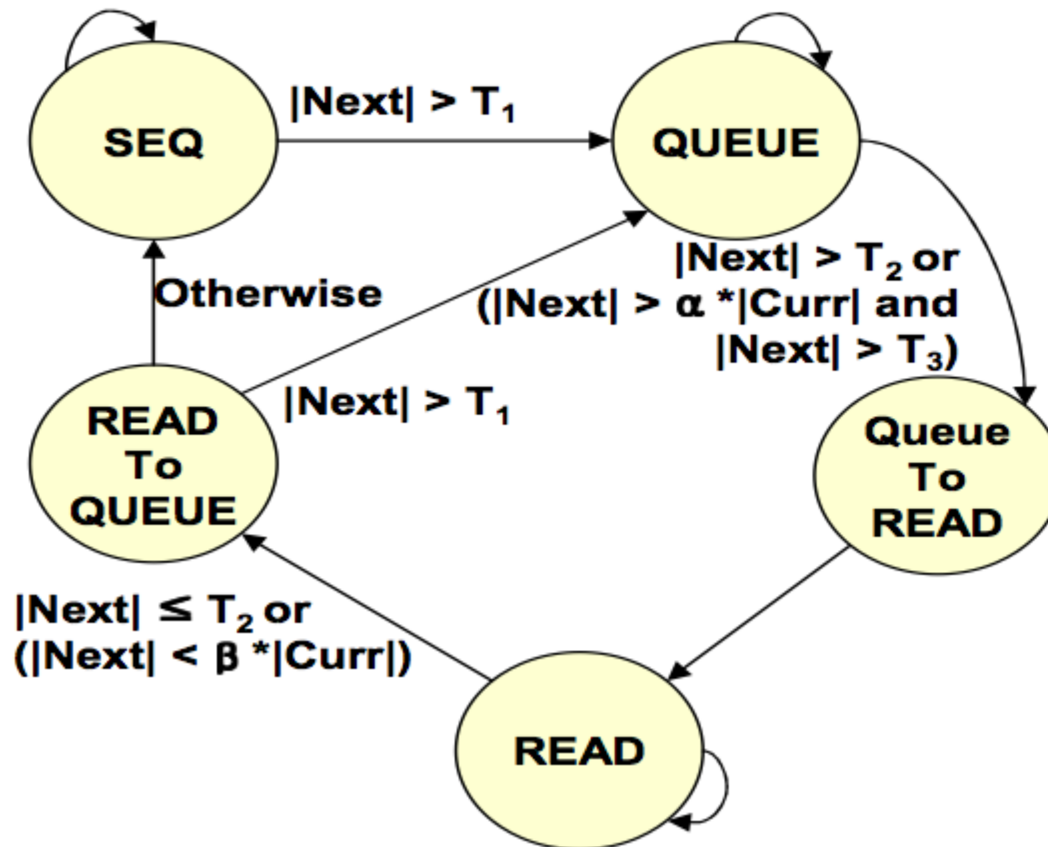
- Motivation
- Natural Parallel BFS Algorithm
- New Method for Multi-Core CPU
- Hybrid Methods
- Experiments

Hybrid Methods

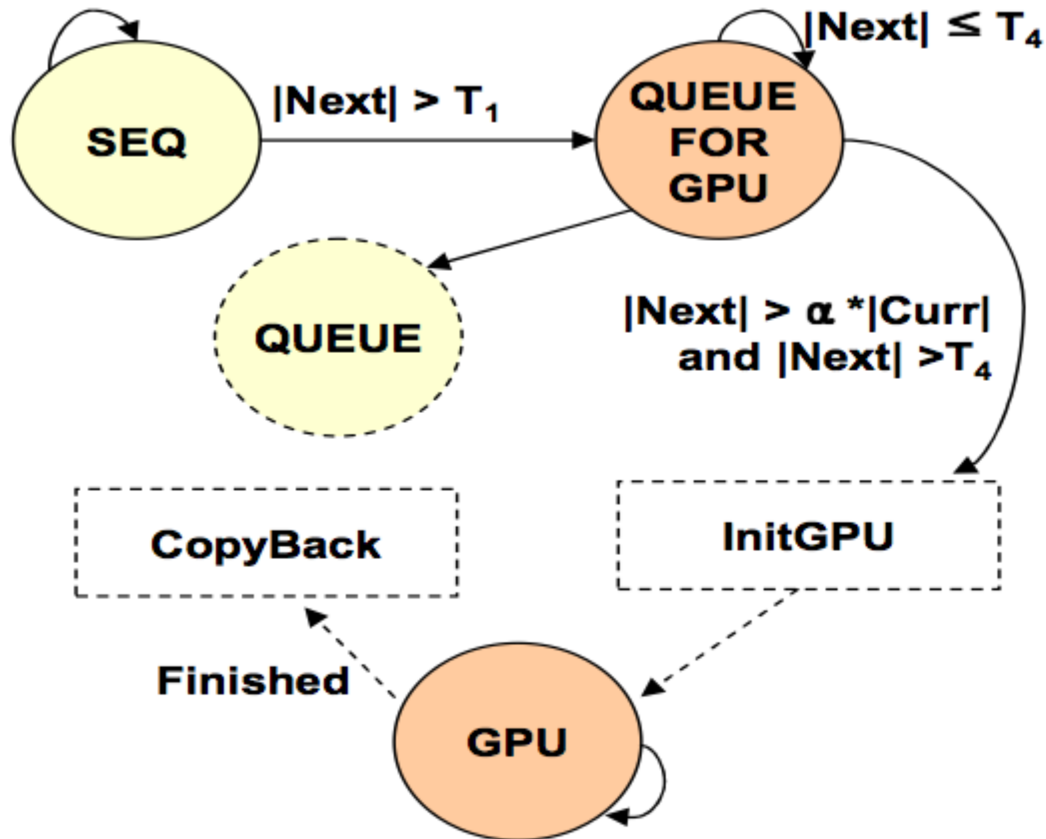
- Dynamically determines which method to apply each level to prevent worst-case execution

- Represented as a **state machine**

Hybrid Read and Queue Method (CPU)



Hybrid CPU and GPU Method



Outline

- Motivation
- Natural Parallel BFS Algorithm
- New Method for Multi-Core CPU
- Hybrid Methods
- Experiments

Methodology

- Measure performance by various machines and different graph instances
  - execute 10 times from 10 different root nodes and take average
- Graph generators
  - uniformly random model
  - RMAT model: small world property

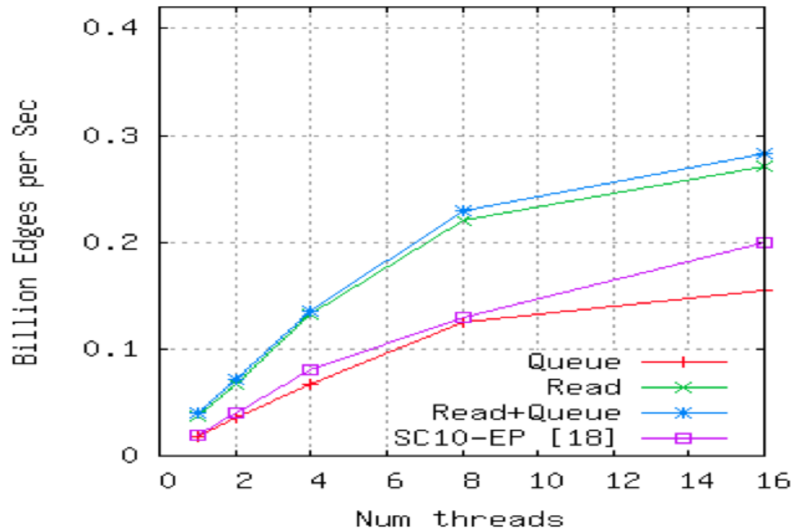
Specification of Machines

	Nehalem CPU	Fermi GPU	Core CPU	Tesla GPU	SC10-EP	SC10-EX
Core Architecture	Intel Nehalem	Nvidia Fermi	Intel Core	Nvidia Tesla	Intel Nehalem	Intel Nehalem
Model No.	Xeon X5550	Tesla C2050	Xeon E5345	GeForce GTX275	Xeon X5570	Xeon X7500
Core Frequency	2.67 GHz	1.15 GHz	2.33 GHZ	1.40 GHz	2.93 GHz	2.26 GHz
Num Socket	2	1	2	1	2	4
Num Core/Socket	4	14*2[a]	4	30	4	8
HW-thread/Core	2	~32	1	~32	2	2
SIMD/SIMT width	- (not used)	32	-	32	-	-
Total Last Level Cache	16 MB	2MB	8 MB	-	16 MB	96 MB
Main Memory	24 GB	3GB	32 GB	896MB	48 GB	256 GB
Memory Bandwidth[b]	100 GB/s	128 GB/s	10.4 GB/s	127 GB/s	100 GB/s	266 GB/s
Total Num Transistors	1.4 Billion	3.0 Billion	1.1 Billion	1.4 Billion	1.4 Billion	9.2 Billion
Total Power (TDP)	190 W	238 W	160 W	210 W	190 W	520 W

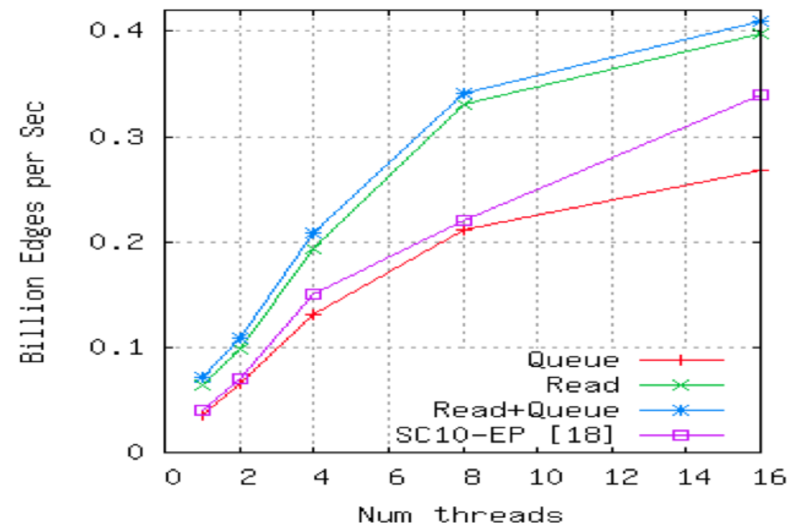
[a] Each core processes two warps at a time.
[b] Theoretical maximum: For GPU and Nehalem CPU, this is (num channels) x (dram bandwidth). For Core CPU, this is FSB bandwidth. See Table II for bandwidths actually measured on the systems.

TABLE III

Perfomance on Nehalem CPU

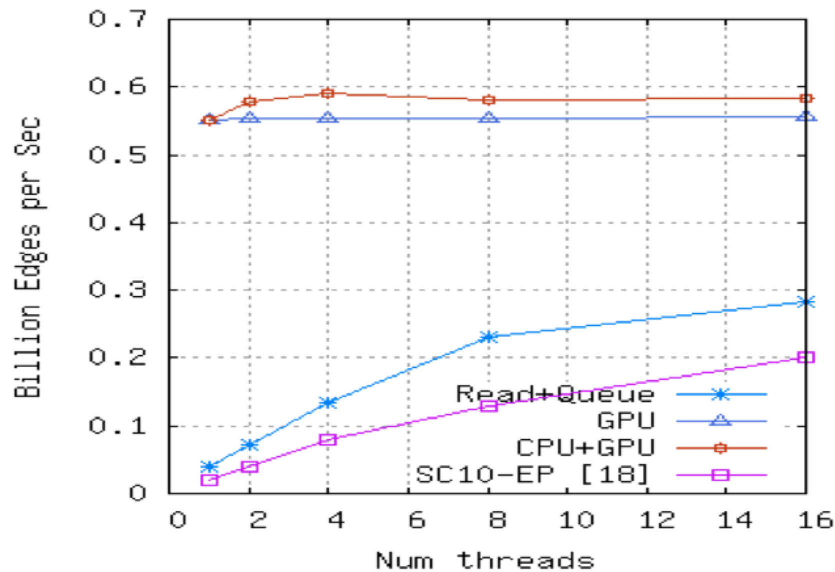


(a) Uniform

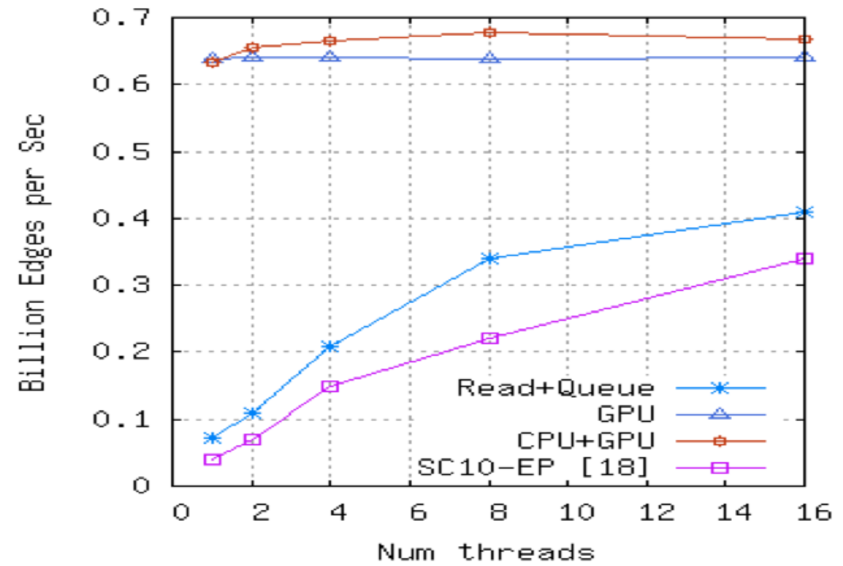


(b) RMAT

Performance on Fermi GPU

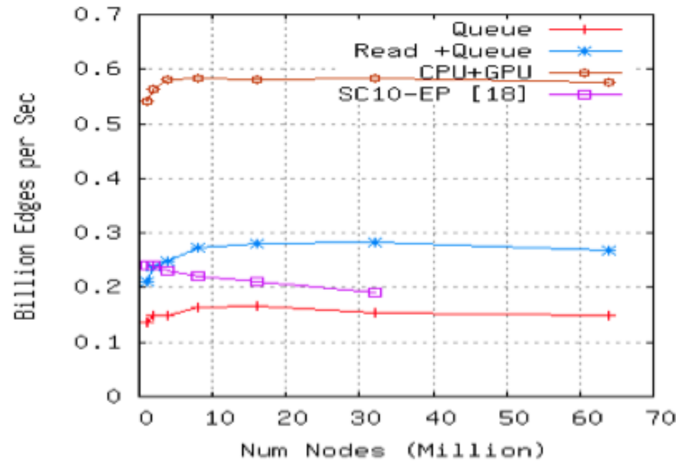


(a) Uniform

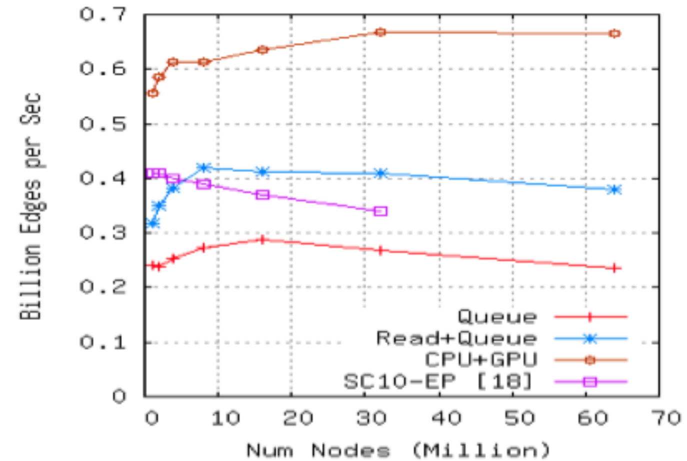


(b) RMAT

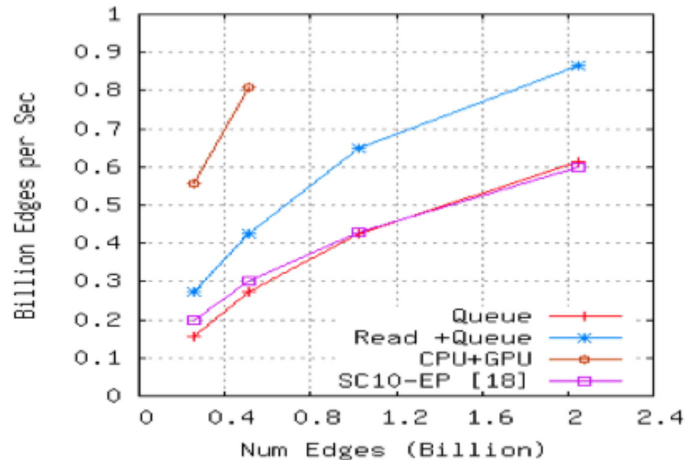
Effect of Graph Size Scaling



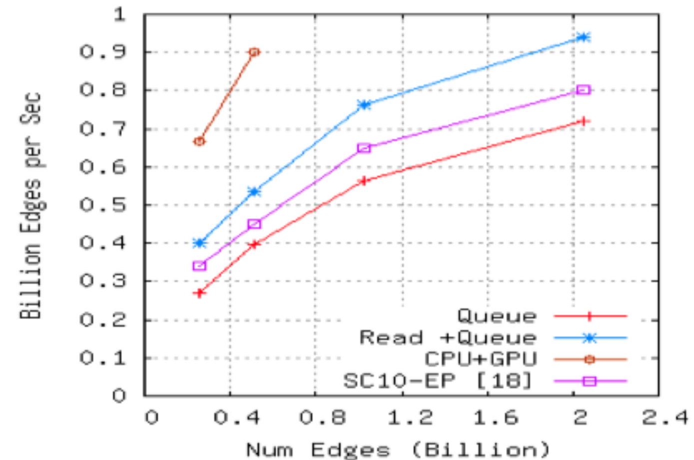
(a) Node (Uniform)



(b) Nodes (RMAT)

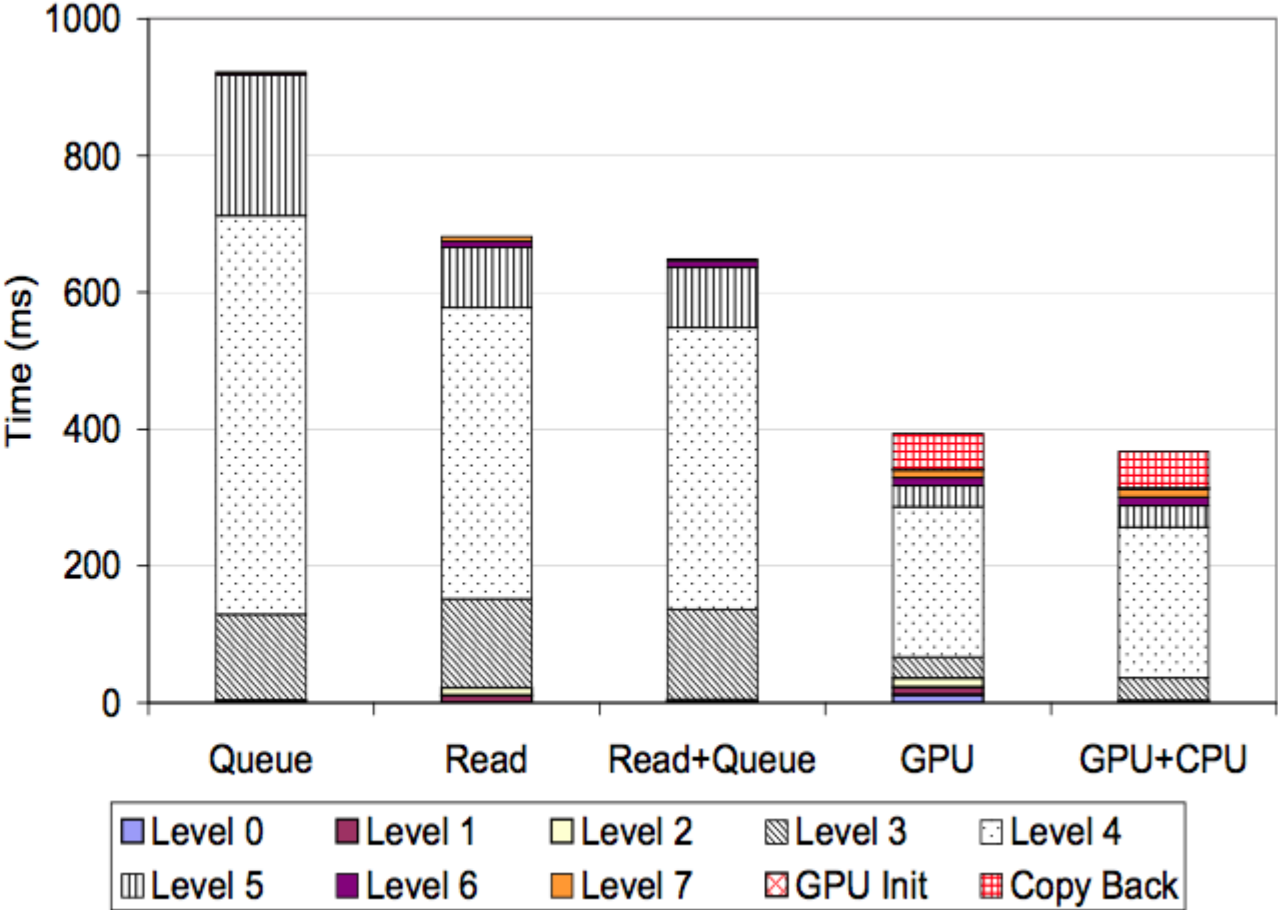


(c) Edges (Uniform)

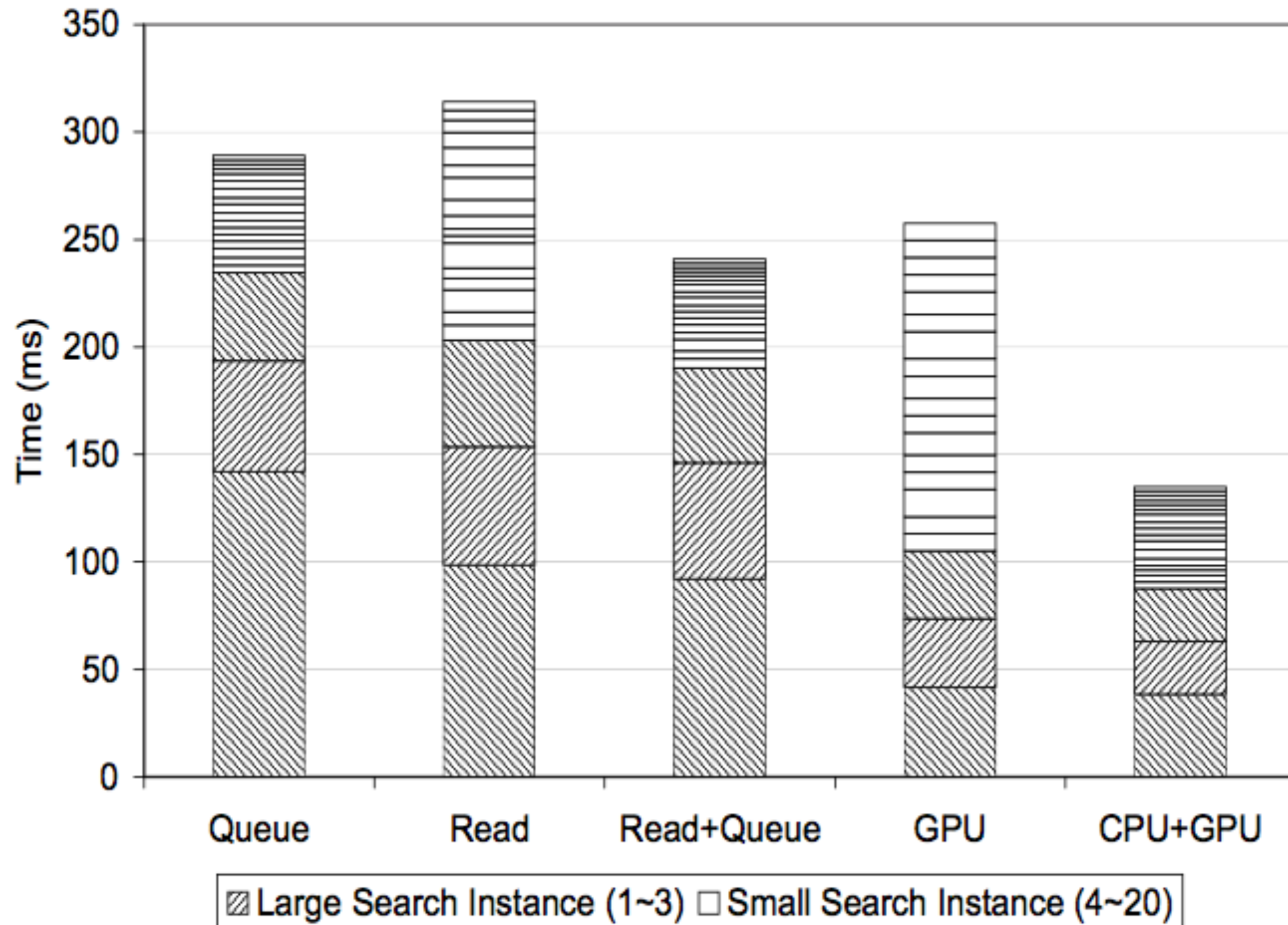


(d) Edges (RMAT)

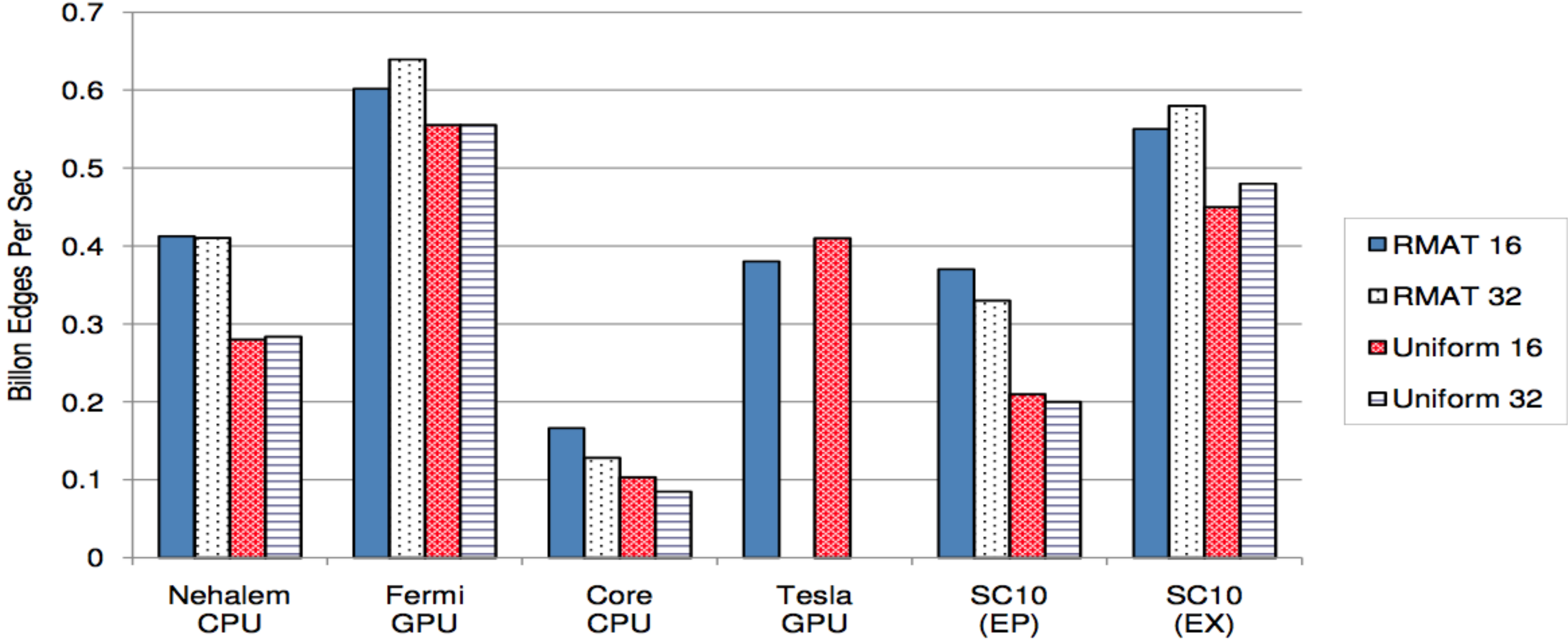
Breakdown Execution Time



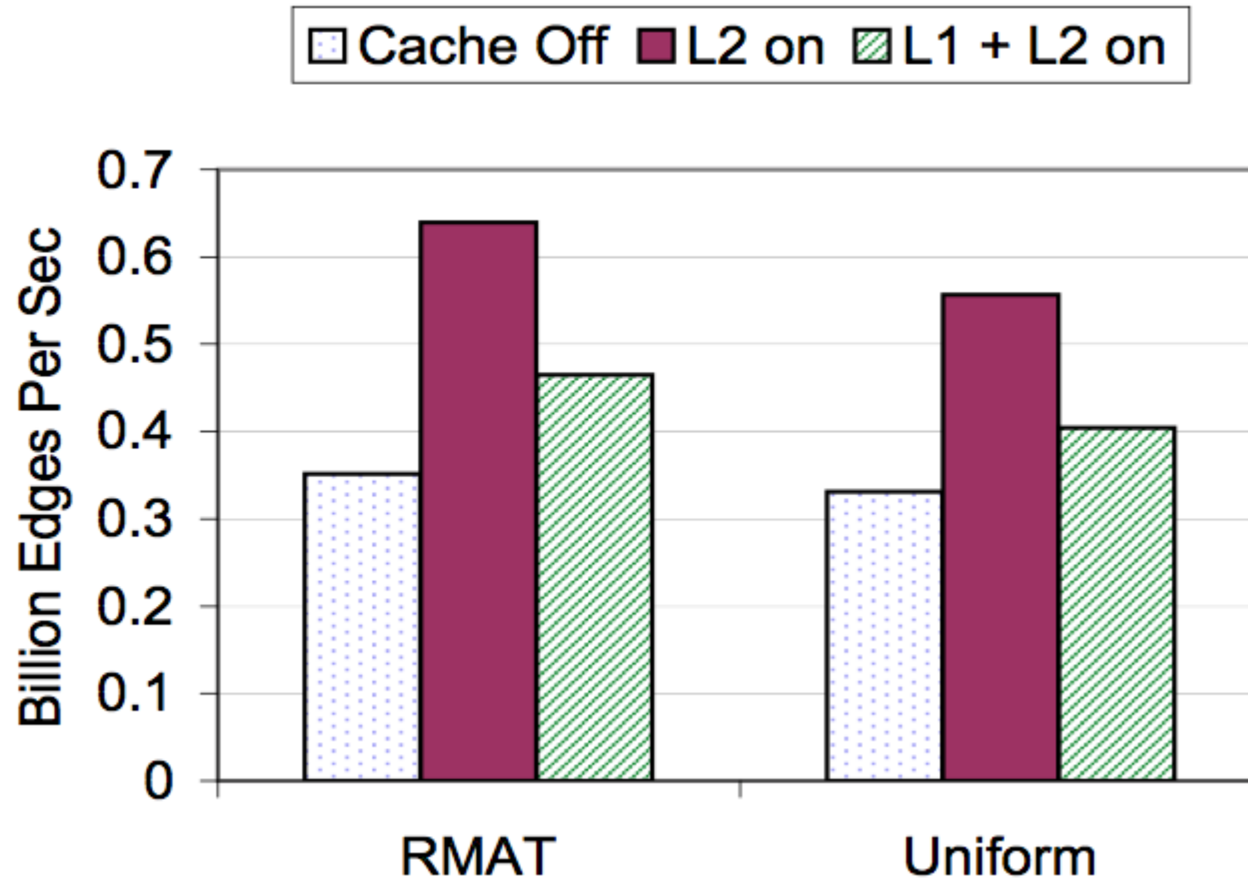
Accumulated Execution Time



Execution Performance on Various Machines



Effect of GPU Cache



Conclusion

- Read-based method
  - simple to apply yet efficient in utilizing memory bandwidth so that it works well on large-scale graph
- Hybrid method
  - choose the best implementation each level; such a method benefits both large and small graphs
- Experiment result
  - the governing factor for performance is primarily random memory access bandwidth