UNIVERSITY OF CAMBRIDGE

Efficient Large-Scale Graph Processing on Hybrid CPU and GPU Systems

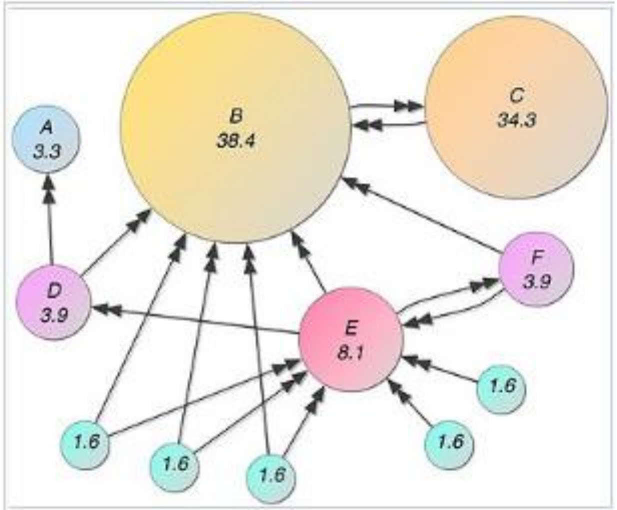
A. Gharaibeh, E. Santos-Neto, L. Costa, M. Ripeanu. IEEE TPC, 2014

Sami (sa894) - R244: Large-scale data processing and optimization

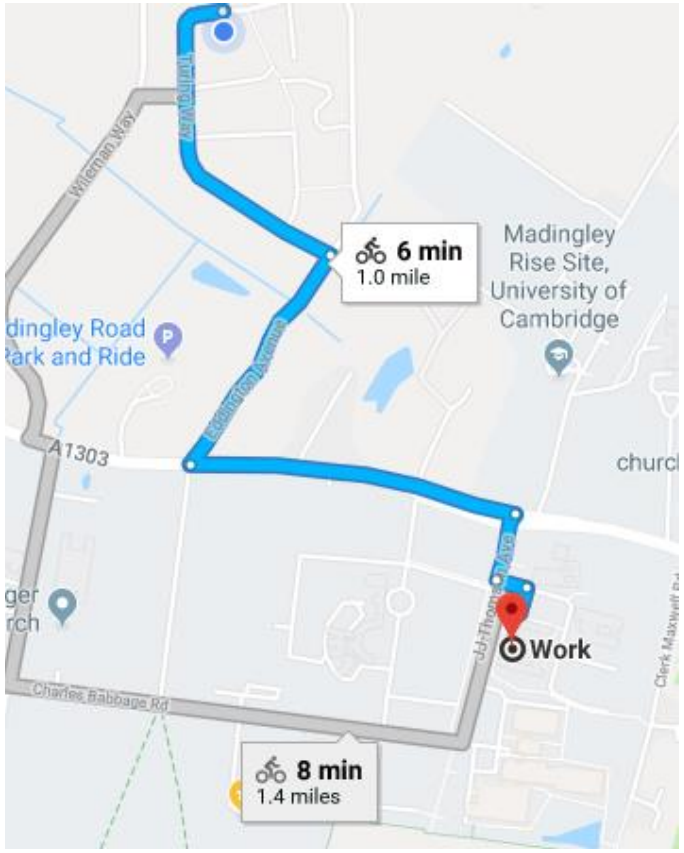
Efficient Large-Scale Graph Processing on Hybrid CPU and GPU Systems – The paper in a nutshell

- Built a processing engine – Totem, that provides a framework to implement graph algorithms on hybrid platform.
- Demonstrated various partitioning strategy to optimize graph problems on parallel systems.
- Benchmarked and evaluated the system to demonstrate a hybrid system can offer x2 on Graph500 challenge.
- At the time of publish, it was the only that did CPU processing with GPU offloading. Closet to it work [HONG 2011] did CPU first round then GPU, memory is an issue there.

Graph Processing - Motivation



Mathematical PageRanks for a simple network, expressed as



Graph Processing – Challenges

- Irregular and data dependent memory access pattern – poor locality
- Data-dependent memory access patterns – process parents before children
- Low compute to memory access ratio – updating and fetch state of vertices major overhead
- Large memory footprint – Requires a whole graph to be present in memory
- Heterogenous node degree distribution – difficult to parallelize
    - Beginning of BFS one vertex, middle of it many vertex to parallelize, end one vertex

Hybrid system – processing on CPU and GPU

CPU

Graph Challenge	CPU’s Answer
Large memory footprint	Have a large memory capacity
Data-dependent memory access pattern	Using BitMap can fit in CPUs caches
Low compute to memory access	☹ Limited Hardware threading capacity

GPU

Graph Challenge	GPU’s Answer
Large memory footprint	☹ (Limited memory capacity)
Data-dependent memory access	BitMap + caches (much smaller than CPU)
Low compute to memory access	☺ Can launch many threads to get around IO block

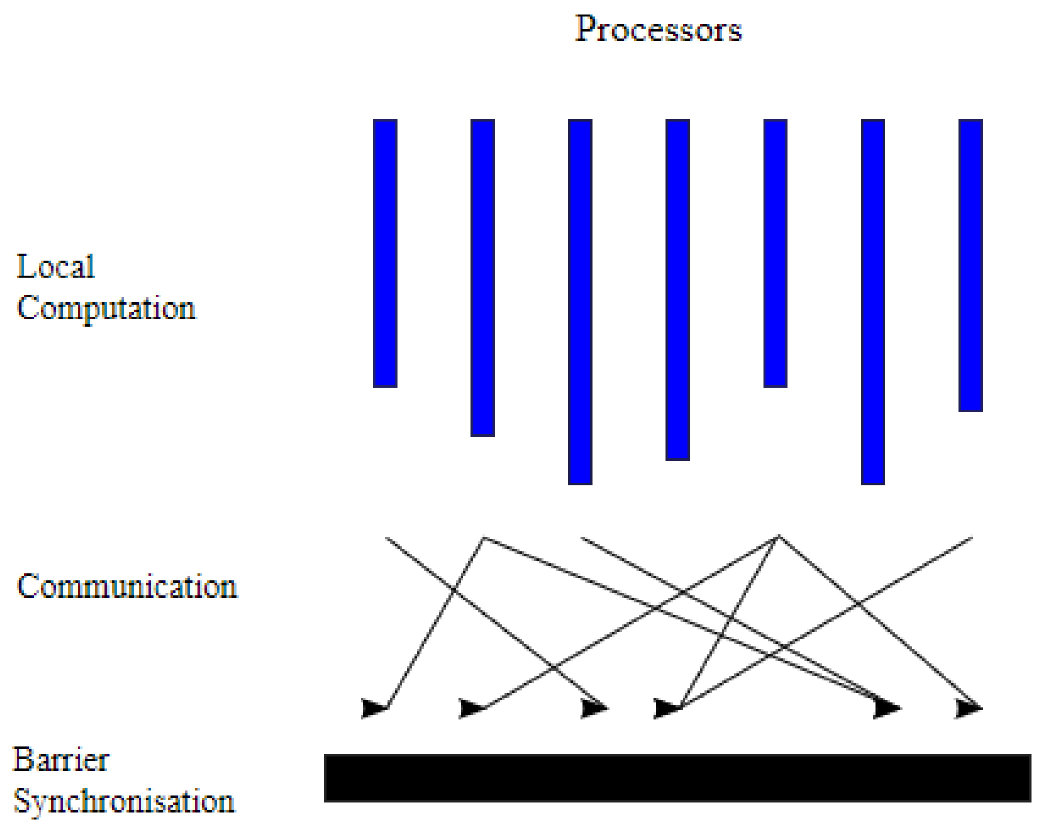
Hybrid system – processing on CPU and GPU

The Internals of Totems – Computation Model

- Bulk Synchronous Parallel (BSP) computation model. Where computations happen in rounds (*supsersteps*) in three phases:

  1. *Computation phase:* Totem assign partitions of the graphs to processes and they execute asynchronously.
  2. *Communication phase:* each process (remote vertices) exchange messages.
  3. *Synchronization phase*: Guarantees the delivery of messages and performed as part of the communication phase
  4. *Termination*: Partitions vote to terminate execution using a callback

Bulk Synchronous Parallel Compute Model (BSP)



Source: Wikipedia Bulk Synchronous Parallel

Internal of Totem – Graph Representation

- Graph partitions are represented as Compressed Sparse Rows (CSR) in memory [Barrett et al. 1994], a space-efficient graph representation that uses O(|V| + |E|) space.
- Each vertex access its edge using its vertex id to find neighboring edges
- Edges stores the partition id
- Improves communication between Edges and partition
- Improves data locality
- Allows storing varying number of edges on CPU and GPUs

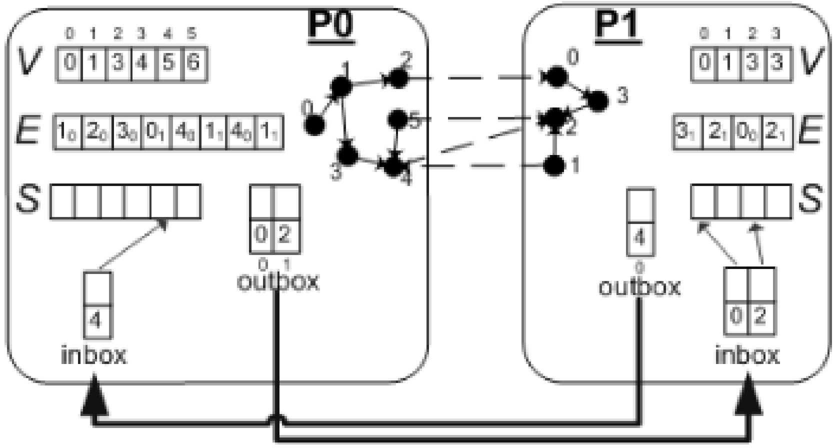


Figure 6: An illustration of the graph data structure and the communication infrastructure in a two-way partitioning setup.

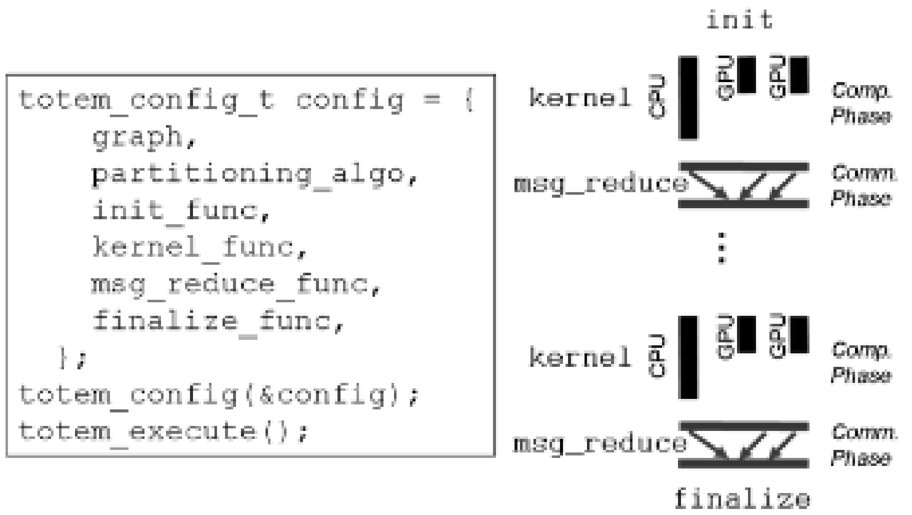
Totem – API Abstraction

Inspired by success of Pregel.

Allows user to define the function to run simultaneously on each partition.

Totem will take care of BSP and spreading workload on CPU and GPU.

Allows defining an aggregation function (similar to combiners in MapReduce)



```
totem_config_t config = {
    graph,
    partitioning_algo,
    init_func,
    kernel_func,
    msg_reduce_func,
    finalize_func,
  };
totem_config(&config);
totem_execute();
```

Figure 5: A simplified TOTEM configuration and how an algorithm callbacks map to the BSP phases.

Evaluation Platform

Characteristic	Sandy-Bridge (Xeon 2650) (x2)	**Kelper Titan (x2)**
Number of processors	2	2
Cores / Proc	8	14
Core frequency (MHz)	2000	800
Hardware Threads / Core	2	192
Hardware Thread / Proc	16	2688
Last Level Cache (MB)	20	2
Memory / Proc (GB)	128	6
Mem. Bandwidth / Proc (GB/s)	52	288

Evaluation workload – Graph500

Workload	|V|	|E|
Twitter [Cha et al. 2010]	52M	1.9B
UK-Web [Boldi et al. 2008]	105M	3.7B
RMAT27	128M	2.0B
RMAT28	256M	4.0B
RMAT29	512M	8.0B
RMAT30	1,024M	16.0B

Partitioning – Assignment strategies

System \ Strategy	HIGH	LOW	RAND
CPU	Highest degree vertices	Lowest degree vertices	Random
GPU	Lowest degree vertices	Highest degree vertices	Random

* Partitioning isn't to reduce communication, aggregation is used to reduce communication

Evaluation (Low compute) – Breadth First Search

- Traversal algorithm with little computation per vertex
- Bitmap optimisation helps improve cache utilization

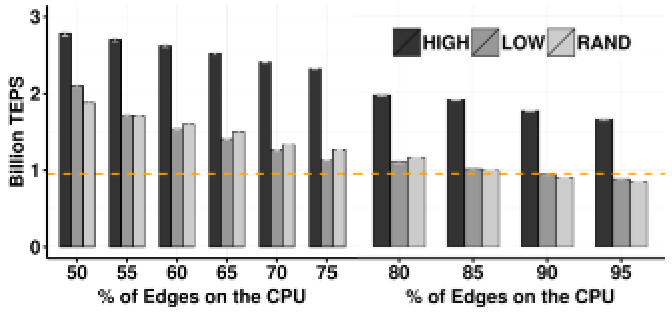


Figure 9: BFS traversal rate (in billions of traversed edges per second - TEPS) for the RMAT28 graph and different partitioning algorithms while varying the percentage of edges placed on the CPU. *Left*: two GPUs (2S2G); *Right*: one GPU (2S1G). The performance of processing the whole graph on the host only (2S) is shown as a straight line.

Observation – CPU is the bottleneck

- GPU has higher processing rate
- Communication overhead is negligible compared to computation

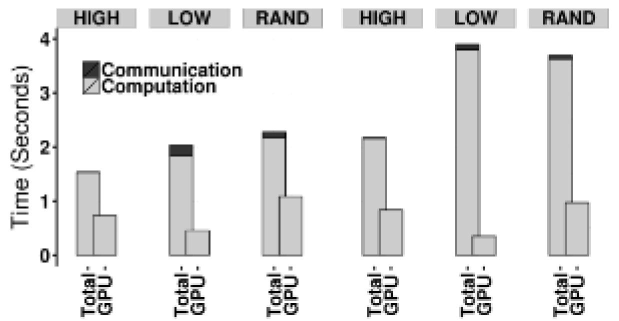


Figure 10: Breakdown of execution time for the RMAT28 graph. *Left*: using two GPUs and 50% of the edges are assigned to the CPU. *Right*: using one GPU and 80% of the edges are assigned to the CPU. The "Computation" bar refers to the computation time of the bottleneck processor (the CPU in this case).

Evaluation (High compute) – PageRank

- No summary table (BitMap), therefore cache isn't utilized as much.
- Higher compute-to-memory access

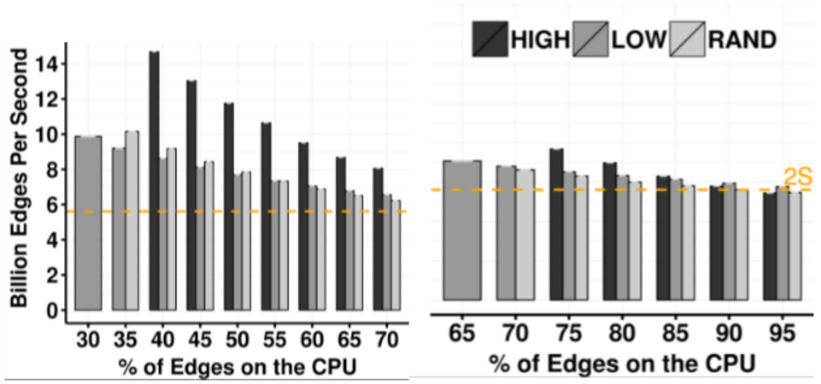


Figure 15: PageRank traversal rate for the UK-WEB graph. *Left*: using two GPUs. *Right*: using one GPU. Missing bars represent cases where the GPU memory space is not enough to fit the GPU partition. The performance of processing the whole graph on two CPU sockets (2S) is shown as a straight line.

PageRank – Breakdown execution

- Still the computation of CPU is the bottleneck!

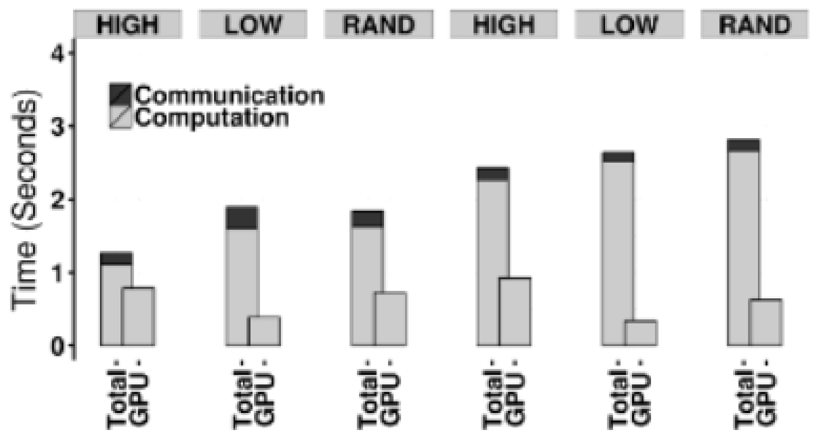


Figure 16: Breakdown of PageRank execution time (five iterations) for the UK-WEB graph when offloading the maximum size partition to two (left three bars) and one GPU (right three bars). The "Computation" bar refers to the compute time of the bottleneck processor (the CPU in this case).

PageRank - But why High is performing better?

- Number of memory read is proportional to number of edges in graph
- Number of writes is proportional to number of vertices (high less vertices)

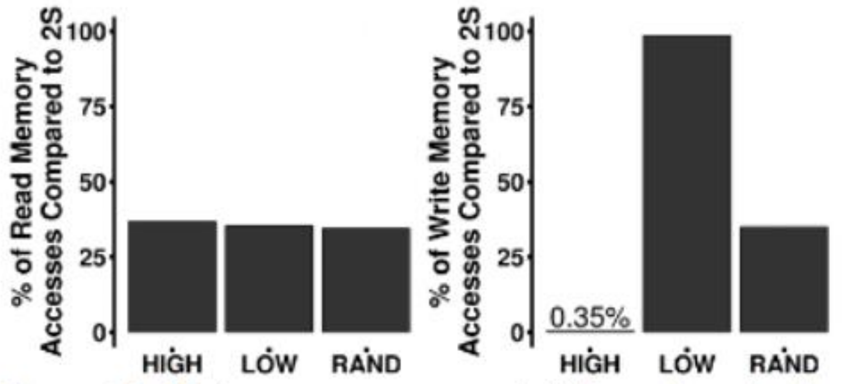


Figure 17: Host memory accesses statistics gathered when running PageRank on UK-WEB graph while when offloading the maximum size partition to two GPUs (2S2G). The performance counter used to collect these statistics is "*mem_uops_retired*". *Left*: read accesses; *right:* write accesses compared to processing the graph on the host only.

Betweenness Centrality (BC) – Complex & high compute

- Backward & Forward BFS.
- Expensive operation proportional to edges and vertices
- Performs more on edges than vertices than PageRank

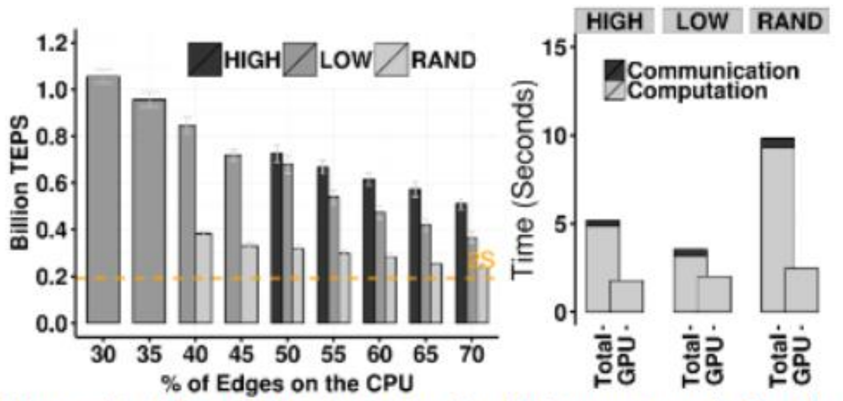


Figure 19: BC performance on the Twitter network for the 2S1G system. *Left*: traversal rate (in Billion TEPS) using one GPU. The horizontal line indicates the performance of a two socket system (2S). *Right*: Breakdown of execution time when offloading the maximum size partition to one GPU (i.e., the percentage of edges offloaded is 50%, 30% and 40% for HIGH, LOW and RAND, respectively).

More CPU? More GPU? Speedup Comparison!

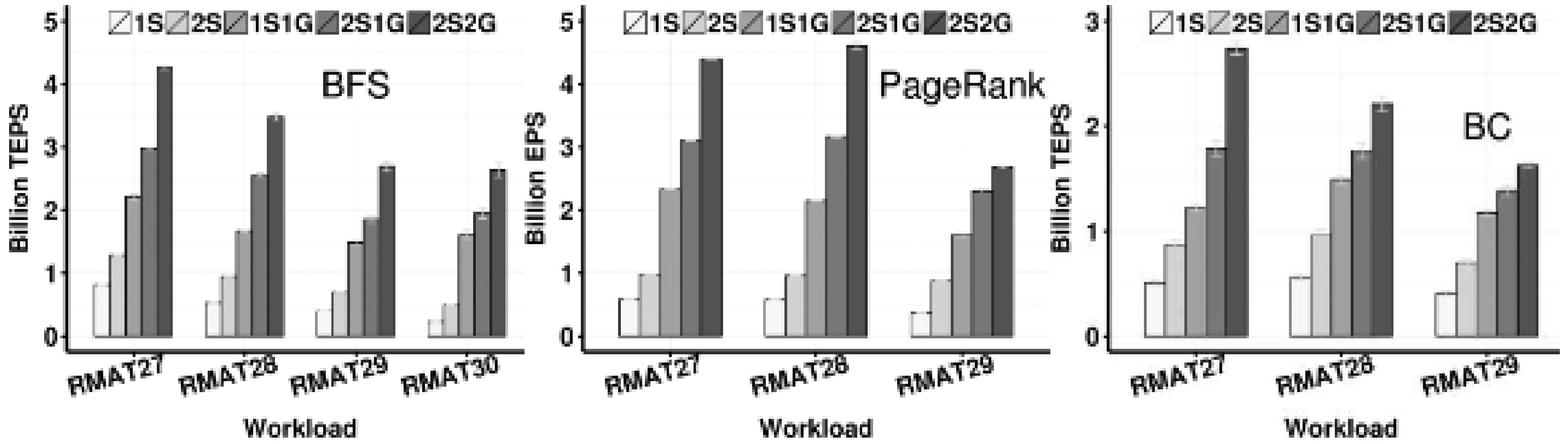


Figure 20: BFS, PageRank and BC processing rates for different hardware configurations and R-MAT graph sizes. When GPUs are used, the graph is partitioned to obtain best performance. Experiments on configurations with a single socket (i.e., *1S* and *1S1G*) were performed by binding the CPU threads to the cores of a single socket. The result for an RMAT30 graph is missing for PageRank and BC because of memory space constraints (the state required by PageRank and BC is larger than that for BFS).

Side Effects – Power Consumption

- Follow up research was done to investigate power consumption in [Gharaibeh et al. 2013b] .
- Concerns about high energy consumption were rejected with detailed discussion and evaluation were presented in that paper.
- GPUs in idle state are power-efficient.
- GPUs finishes much faster than CPU, therefore they reach the idle state faster. (known as ‘race-to-idle’)

Totem Today

- GitHub repository last active in 2015.
- Follow-up research shows efficient energy consumption [1].
- In [2], Offers numerous optimization technique for BFS problem making hybrid system attractive for large scale graph processing.
- New benchmarks were published no a newer system that still shows the linear speedup [Y GAU 2015][X PAN 2016]

[1] *The Energy Case for Graph Processing on Hybrid CPU and GPU Systems*, Abdullah Gharaibeh, Elizeu Santos-Neto, Lauro Beltrão Costa, Matei Ripeanu
[2] *Accelerating Direction-Optimized Breadth First Search on Hybrid Architectures*, Scott Sallinen, Abdullah Gharaibeh, Matei Ripeanu, 13th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms

Main Contributions

- Propose a novel way to process large-scale graphs utilising GPUs.
- Investigated the trade-off on offloading workload between CPU and GPU.
- Partitioning is important optimisation in graph processing.
- Built on findings in [HONG, TAYO, KUNLE 2011] that GPUs process faster for the case of BFS, and generalised it for other problems.

Presenter’s opinion

- The system is non-distributed, that fact is just brushed over, however it is a big concern it won’t scale for larger graphs, and a single point of failure. (future direction?)
- It would have been interesting to see benchmarks where the system was deployed into a system with more than 2 CPU, 2GPU. Especially if more GPUs than CPUs
- Cost comparison would have been nice, GPUs tend to be order of magnitude more expensive.
- I really do like the system ☺ paper is really wordy and hard to read ☹

References

- Every figure, equation, and picture unless stated otherwise, is referenced from the paper in review
  [Efficient Large-Scale Graph Processing on Hybrid CPU and GPU Systems, A. Gharaibeh, E. Santos-Neto, L. Costa, M. Ripeanu. IEEE TPC, 2014]