GPU-Accelerated Algorithms for Allocating Virtual Infrastructure in Cloud Data Centers

Lucas Leandro Nesi[◇], Mauricio Aronne Pillon[◇], Marcos Dias de Assunção[⊕], Guilherme Piegas Koslovski[◇]
Graduate Program in Applied Computing – Santa Catarina State University – Joinville – Brazil[◇]
Inria Avalon, LIP Laboratory, ENS Lyon, University of Lyon – France[⊕]
lucas.nesi@edu.udesc.br, mauricio.pillon@udesc.br, assuncao@acm.org, guilherme.koslovski@udesc.br

***Abstract*—Allocating IT resources to Virtual Infrastructures (VIs) (*i.e.* groups of VMs, virtual switches, and their network interconnections) is an NP-hard problem. Most allocation algorithms designed to run on CPUs face scalability issues when considering current cloud data centers comprising thousands of servers. This work offers and evaluates a set of allocation algorithms refactored for Graphic Processing Units (GPUs). Experimental results demonstrate their ability to handle three large-scale data center topologies.**

1. Introduction

Under the Cloud computing model, Infrastructure-as-a-Service (IaaS) providers deliver on-demand access to VIs comprising virtual resources that are deployed onto Data Center (DC) infrastructure. Each tenant can customize a VI according to the requirements of an application whose performance can be influenced by the network configuration and assigned resources. An allocation solution can optimize a single or multiple criteria such as decreasing DC load and VI latency, reducing energy consumption, among others [1].

Such problem is NP-hard [2] as the number of possible solutions for a VI request grows with its size and the size of DC graphs, which is an issue as modern DC architectures, such as Fat-Tree, BCube, and DCell can have thousands of servers and links.

The high-performance of GPUs make them potential candidates to overcome CPU limitations and support the allocation of VIs. GPUs offer a high-degree of parallelism, high throughput memory [3], and general purpose programming tools, such as Compute Unified Device Architecture (CUDA). To run efficiently on GPUs, well-known graph algorithms [4], [5] must be refactored.

Therefore, this refactors a set of graph algorithms that are then used to compose allocation strategies as part of a GPU-accelerated framework for VI allocation. Microbenchmarks are used to compare the speedup of each individual algorithm against CPU-based approaches. The results show the applicability of GPU algorithms to large-scale cloud data centers. The framework, examples, and documentation are publicly available[1].

This research was carried out at the LabP2D, and supported by NVIDIA, FAPESC, and UDESC.

1. Available at https://bitbucket.org/lucasnesi/vnegpu.

2. CPU-based Algorithms for VI Allocation

The state-of-the-art provides essentially Virtual Network Embedding (VNE) and cloud solutions for problem of mapping VIs to physical infrastructure. Mixed Integer Programming (MIP) or Linear Programming (LP) offer optimal solutions generally used as baseline for comparisons [6], [7], but the problem complexity and search space often create opportunities for heuristic-based solutions [1] and grouping techniques [5]. Their scalability and applicability to real DCs, however, remain a challenge as aggressive pruning of physical and virtual candidates may lead to partial and inefficient solutions under certain topologies [8].

GPU-accelerated algorithms can be successfully applied to speed up heuristics and simulators, hence allowing for investigating and evaluating large-scale scenarios that more closely match the requirements of real cloud DCs. The microbenchmark analysis (Sec. 4) highlight the application of GPU-accelerated algorithms.

3. GPU-Based Algorithms for VI Allocation

The allocation of resources to VIs is commonly supported by well-known graph algorithms. We select and refactor for GPUs a set of algorithms for **comparing edges and vertices** (Local Resource Capacity (LRC), Worst-Fit (WF), Best-Fit (BF), and Page Rank (PR)), **shortest path algorithms** (Dijkstra, R-Kleene), **data clustering** (K-Means and MCL), and a **custom graph allocation algorithm**.

As a DC graph and a VI request have common characteristics such as vertex and edge attributes, and both are sparse graphs, we extend the Compressed Sparse Row (CSR) for representing both graphs. CSR stores the graph topology using a directed edge representation enabling random access on edges of any node. As a VI is commonly formulated as an undirected graph [1], we enhance CSR by using the Edge Map (EM) vector for identifying the source and destination of undirected graphs. The access of edges on directed or undirected graphs is hence performed with $O(1)$ time complexity. Our custom graph allocation algorithm is performed in two steps. The first step iterates over the VI nodes, matching each VI node to the most appropriated DC node using either BF or WF. If a node cannot be allocated, the algorithm aborts and the VI is rejected. The second step maps the edges of the VI to the DC paths finding the shortest valid path using a shortest path algorithm.

TABLE 1. RUNTIME FOR MICROBENCHMARKS.

Graph	Hardware	Page Rank	Dijkstra	R-Kleene	K-Means	Markov Clustering (MCL)	Graph Allocation (WF+$\|L^v\|\times$SSSP)
BCube $8,3$	GTX 1080	2.13 ± 0.03 ms	0.58 ± 0.05 ms	592.22 ± 1.77 ms	6.60 ± 0.02 ms	1.29 ± 0.01 s	111.06 ± 2.30 ms
	Titan XP	2.13 ± 0.04 ms	0.55 ± 0.04 ms	$\mathbf{378.68 \pm 1.70}$ ***ms***	5.43 ± 0.01 ms	923.87 ± 2.56 ms	112.11 ± 1.03 ms
	Intel i7	5.66 ± 0.03 ms	0.49 ± 0.06 ms	$\mathbf{32.02 \pm 0.41}$ ***m***	70.48 ± 1.09 ms	1.12 ± 0.01 h	200.95 ± 2.05 ms
BCube $7,4$	GTX 1080	2.01 ± 0.03 ms	0.92 ± 1.08 ms	80.21 ± 0.02 s	107.99 ± 0.52 ms	11.01 ± 0.08 m	161.09 ± 2.84 ms
	Titan XP	2.00 ± 0.07 ms	0.63 ± 0.04 ms	44.52 ± 0.03 s	65.18 ± 0.49 ms	6.78 ± 0.02 m	$\mathbf{161.08 \pm 0.99}$ ***ms***
	Intel i7	25.45 ± 0.02 ms	2.38 ± 0.01 ms	22.23 ± 0.26 h	1.67 ± 0.05 s	13.18 ± 0.01 h	$\mathbf{1.13 \pm 0.10}$ ***s***
Fat Tree 24	GTX 1080	2.49 ± 0.06 ms	0.59 ± 0.05 ms	185.83 ± 6.27 ms	5.17 ± 0.03 ms	3.72 ± 0.00 s	120.06 ± 2.75 ms
	Titan XP	2.38 ± 0.12 ms	0.60 ± 0.08 ms	122.17 ± 1.97 ms	5.06 ± 0.03 ms	$\mathbf{2.61 \pm 0.00}$ ***s***	126.22 ± 3.38 ms
	Intel i7	4.96 ± 0.07 ms	0.30 ± 0.01 ms	2.86 ± 0.01 m	116.90 ± 0.05 ms	$\mathbf{4.52 \pm 0.02}$ ***h***	123.04 ± 0.22 ms
Fat Tree 48	GTX 1080	3.64 ± 0.16 ms	0.72 ± 0.05 ms	93.90 ± 0.38 s	523.63 ± 1.23 ms	24.31 ± 0.00 m	209.05 ± 1.56 ms
	Titan XP	$\mathbf{3.60 \pm 0.05}$ ***ms***	$\mathbf{0.68 \pm 0.04}$ ***ms***	$\mathbf{51.60 \pm 0.10}$ ***s***	$\mathbf{164.98 \pm 0.64}$ ***ms***	$\mathbf{14.06 \pm 0.00}$ ***m***	208.89 ± 0.44 ms
	Intel i7	$\mathbf{32.80 \pm 0.07}$ ***ms***	$\mathbf{2.38 \pm 0.10}$ ***ms***	$\mathbf{26.13 \pm 0.18}$ ***h***	$\mathbf{4.48 \pm 0.01}$ ***s***	$>$ **2** ***days***	1.02 ± 0.05 s
DCell 7 2	GTX 1080	0.36 ± 0.02 ms	0.64 ± 0.03 ms	131.12 ± 2.66 ms	2.84 ± 0.04 ms	2.44 ± 0.24 s	136.95 ± 4.63 ms
	Titan XP	0.34 ± 0.01 ms	0.63 ± 0.01 ms	$\mathbf{79.81 \pm 1.03}$ ***ms***	2.61 ± 0.04 ms	$\mathbf{1.69 \pm 0.26}$ ***s***	134.84 ± 1.47 ms
	Intel i7	0.44 ± 0.03 ms	0.29 ± 0.02 ms	$\mathbf{2.02 \pm 0.01}$ ***m***	27.71 ± 0.36 ms	$\mathbf{2.85 \pm 0.03}$ ***h***	120.47 ± 0.45 ms
DCell 11 2	GTX 1080	0.39 ± 0.00 ms	0.65 ± 0.01 ms	22.57 ± 0.05 s	38.64 ± 0.05 ms	3.24 ± 0.00 m	162.45 ± 1.37 ms
	Titan XP	0.38 ± 0.01 ms	0.65 ± 0.02 ms	$\mathbf{12.52 \pm 0.05}$ ***s***	$\mathbf{26.71 \pm 0.11}$ ***ms***	$\mathbf{1.84 \pm 0.01}$ ***m***	164.77 ± 1.06 ms
	Intel I7	2.68 ± 0.03 ms	1.46 ± 0.02 ms	$\mathbf{5.33 \pm 0.02}$ ***h***	$\mathbf{801.52 \pm 1.34}$ ***ms***	$>$ **2** ***days***	715.18 ± 61.45 ms

4. Speedup Analysis Using Microbenchmarks

We evaluate each algorithm individually to compare their speedup against their CPU counterparts. Two GPUs are used for performance analysis, namely NVIDIA GeForce GTX 1080 (8 GB) and NVIDIA Titan XP (12 GB), on a machine with an Intel i7 2600K and 32 GB of RAM. The machine runs Ubuntu 17.04 Server with CUDA 9.0.176, NVIDIA driver 384.81, and GCC 5. Each experiment is repeated 10 times and the reported results are mean values with standard deviation. To compare the CPU and GPU implementations fairly, the runtime comprises only the execution of the main function, disregarding the initial memory allocation and transfer, and program initialization.

Table 1 summarizes the average runtime results for the microbenchmarks. We omit results on LRC algorithm as it does not pose enough processing load to express any sufficient time measure. Also, as WF and BF present similar behavior we report the results on graph allocation considering WF for vertex and edge comparison and Dijkstra for shortest path (an execution for each virtual link). For the PR algorithm, we use the node capacity metric. We highlight the $9\times$ speedup under the Fat-Tree 48 scenario. Regarding the Dijkstra algorithm, node 0 is the initial node in all scenarios. As the dataset size increases, the GPU outperforms the CPU implementation. On the largest topology, Fat-Tree 48, the GPU speedup is $3.5\times$ compared to the CPU counterpart.

The R-Kleene is executed using the number of edge hops as the metric. For the intermediate-size topologies, the GPU achieved a speedup of $5073\times$ on BCube $8,3$. On the largest topology, the GPU speedup is $1823\times$. For the execution of the K-Means, we arbitrarily select 6 groups. For all scenarios, the GPU algorithms are faster. The speedup is $27\times$ when using the Fat-Tree 48 and $30\times$ on DCell $11,2$. The execution of the MCL used the parameters $p = 2$, $r = 1.2$ and $\epsilon = 0.0$ (empirically defined for the Fat-Tree topology). Moreover, the $\epsilon = 0.0$ expresses the worst-case scenario regarding runtime. We highlight the $6234\times$ speedup achieved on the Fat-Tree 24. Also, the Fat-Tree 48 and DCell $11,2$ CPU executions took more than 2 days and were hence aborted. Last, for the allocation algorithm, we used a 400-nodes request for all topologies. We highlight the speedup of $7\times$ achieved on the BCube $7,4$.

The microbenchmarks highlight that GPU speed up most algorithms for VI allocation. Multiple executions of an algorithm, or combinations of algorithms, may be required to find a suitable allocation.

5. Conclusion

We created a GPU accelerated framework and GPU versions of classical graph algorithms for allocating resources to VIs. Using microbenchmarks, all algorithms were evaluated and their speedup compared against their CPU-based counterparts. R-Kleene and MCL achieved speedups of $5073\times$ and $6234\times$ respectively. In addition to achieving good speedup, the framework allows for considering problem sizes much larger than those found in the literature.

References

[1] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.

[2] C. Fuerst *et al.*, "How hard can it be?: Understanding the complexity of replica aware virtual cluster embeddings," in *23rd Int. Conf. on Network Protocols (ICNP)*. CA, USA: IEEE, Nov 2015, pp. 11–21.

[3] N. K. Govindaraju *et al.*, "A memory model for scientific algorithms on graphics processors," in *Proc. of the ACM/IEEE Conference on Supercomputing*. NY, USA: ACM, 2006.

[4] L. Page *et al.*, "The pagerank citation ranking: Bringing order to the web." *Stanford InfoLab, Technical Report*, no. 1999-66, Nov. 1999.

[5] M. Rost *et al.*, "Beyond the stars: Revisiting virtual cluster embeddings," *SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 12–18, Jul. 2015.

[6] N. M. M. K. Chowdhury *et al.*, "Virtual network embedding with coordinated node and link mapping," in *IEEE INFOCOM 2009*. Rio de Janeiro, RJ, Brasil: IEEE, April 2009, pp. 783–791.

[7] L. R. Bays *et al.*, "A toolset for efficient privacy-oriented virtual network embedding and its instantiation on SDN/OpenFlow-based substrates," *Computer Communications*, vol. 82, pp. 13 – 27, 2016.

[8] R. de Oliveira and G. P. Koslovski, "A tree-based algorithm for virtual infrastructure allocation with joint virtual machine and network requirements," *International Journal of Network Management*, vol. 27, no. 1, pp. e1958–n/a, 2017, e1958 nem.1958.