Optimizing Maximum Shared Risk Link Group Disjoint Path Algorithm using NVIDIA CUDA Heterogeneous Parallel Programming Platform

Vedran Miletić*, Tomislav Šubić* and Branko Mikac†
*University of Rijeka Department of Informatics
Radmile Matejčić 2, 51000 Rijeka, Croatia
vmiletic@inf.uniri.hr, tsubic@student.uniri.hr
†University of Zagreb Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
branko.mikac@fer.hr

***Abstract*—Network availability is an essential feature of an optical telecommunication network. Should a failure of a network component occur, be it a link or a component inside a node, network control plane must be able to detect the failure and reroute the traffic using spare components until a repair is done. Shared risk link groups (SRLGs) are used to describe a situation where seemingly unrelated logical failures happen due to a single physical failure. For example, two or more links might share a bridge crossing; should a failure happen, all of them will be damaged. Routing algorithms were proposed to ensure working and spare paths of a connection in a network are SRLG-disjoint to avoid such common cause failures. However, complete SRLG-disjointness of working and spare path is not always possible due to limited number of links or limited capacity available in the network, so maximum SRLG-disjoint paths algorithm is taken instead. Maximum SRLG-disjoint path problem is in general NP-hard. In terms of solution quality greedy algorithms for maximum SRLG-disjoint path problem are as good as more complicated heuristics. To improve the performance of maximum SRLG-disjoint path greedy algorithm, it was implemented using NVIDIA CUDA heterogeneous parallel programming platform and executed on graphics processing unit. The implementation of maximum SRLG-disjoint path algorithm on GPU increases performance significantly compared to implementation utilizing only CPU, especially in simulations of large networks.**

***Index Terms*—optical networks, reliability, shared risk link group, modelling, simulation, ns-3, maximum SRLG-disjoint path algorithm, algorithm optimization, heterogeneous paralell programming, NVIDIA CUDA**

I. Introduction

With continuous increase in Internet traffic, enabled by capacity growth of optical transport networks, network resilience becomes an important consideration. A failure of any network component (e.g. a fiber or a switching element in network node) can cause outage for many lightpaths, and lead to user dissatisfaction and effectually decreased operator revenues.

In case a working lightpath goes down due to a component failure, a spare lightpath is used until the working is repaired. Routing of working and spare lightpaths is a non-trivial problem and, combined with wavelength assignment, it can be shown to be NP-complete [1]. Many heuristics for solving this problem have been developed over the years, and many special case optimizations have been made.

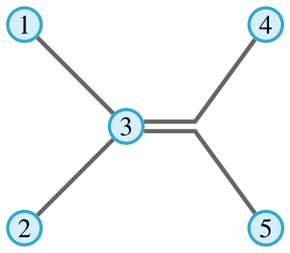

Figure 1. Example SRLG containing two cables (3–4 and 3–5) that share an exit at a particular network node. For comparison, there is no SRLG containing cables 1–3 and 2–3.

A particularly interesting special case optimization is routing and wavelength assignment (RWA) for working and spare lightpath in presence of shared risk link groups (SRLGs), groups of links that have a common physical location [2]. Example of a SRLG where two cables share a common exit at a node can be seen in Figure 1. Due to shared location, be it a cable, duct or bridge crossing, SLRGs are prone to failing at the same time due to a single physical damage. In effect, multiple seemingly unrelated logical failures can occur, for example two link-disjoint (but not SRLG-disjoint) lightpaths can fail at the same time. Therefore, an algorithm for RWA should be designed to avoid common shared risk link groups in working and spare lightpaths, to prevent them from failing at the same time due to a common physical force. Since link- and SRLG-disjoint paths might not exist or be possible to set up in a network, maximum disjoint paths are usually are reasonable substitute. However, finding maximum link disjoint paths is NP-hard problem [1]. Furthermore, it is known that greedy algorithms for it are performing as well as (much more complex) heuristics.

Simulation methods are often employed for studies of network resilience [3]–[5]. In particular, Monte Carlo simulation can be used to give an estimate of network availability and comparison of different network scenarios using different RWA strategies. Monte Carlo simulation require many runs of the same scenario to give a good estimate, so reduction of simulation execution time becomes crucial. One approach is parallelization of suitable parts of simulation, utilizing multi-core central processing units (CPUs) and one or more graphics processing units (GPUs) on one or more compute nodes.

Horizontal scaling means adding more compute nodes to a computer cluster used for running simulations. Vertical scaling, on the other hand, implies adding resources to a single compute node in the cluster, meaning additional CPUs, GPUs, memory etc. When scaling is required to satisfy computation demands, one can utilize horizontal or vertical scaling, or combine both.

This paper presents our approach to performance optimization of best-effort RWA algorithm using CUDA heterogeneous parallel programming platform enabling code to run on both GPU(s) and CPU(s). Part of the algorithm is moved to GPU for computation to reduce overall execution time. Meanwhile, CPU handles computations not suitable for the GPU. Our approach is based on extending models implemented by ns-3 network simulator [6] with GPU-enabled code, utilizing NVIDIA CUDA programming platform [7]. Compute clusters are becoming increasingly heterogeneous over time, with computation power divided over a number of different processors of vastly disparate computational features [8].

The paper is organized as follows: first we overview related work in Section II, then we describe the application of maximum disjoint path algorithms in RWA in Section III. We follow up with description of our approach to algorithm parallelization in Section IV. We do performance benchmarks in Section V, and finally conclude along with possible directions for future work.

II. Related Work

The usage of GPUs for general purpose computing has been on the rise in recent years [9]. Many application domains, especially computational science and engineering, have benefited greatly and expanded their scope significantly from computational performance increase due to GPUs.

In domain of computer networks, usage of GPUs for IP routing has been studied by Han et al. [10] using custom PacketShader software. Benchmarks have shown that peak performance of NVIDIA GeForce GTX 480 consumer-grade GPU is roughly comparable to ten Intel Xeon X5500 processors. In effect, this result enables a well-designed PC-based router to forward IP packets at 40 Gbps.

A. Parallelization of Graph Search

Swenson and Riley provided an implementation of CUDA-enabled computation of Floyd-Warshall algorithm used for solving all pairs shortest path problems [11]. The goal was performance improvement of IP routing in ns-3 network simulator and therefore decrease of simulation runtime. It was shown that CUDA-enabled routing reduced simulation runtime compared to CPU-only Nix-vector routing [12] consistently by a factor over three.

Harish and Narayanan described the approach to parallelization of breadth-first search, single source shortest path and all pairs shortest path using CUDA [13]. They parallelized Dijkstra algorithm using two kernels and found a two orders of magnitude speedup in GPU-enabled code over the code that utilizes only the CPU.

B. Algorithms for Maximum Link and Shared Risk Link Group Disjoint Paths

Maximum edge (link) disjoint path problem is a variant of k-shortest path problem (in most applications $k = 2$). Say two (disjoint) shortest paths can not be found in a given network; one can ask for maximum link or shared risk link group disjoint paths instead. algorithms have been studied for RWA in optical networks for many years [1], [14]–[16]. In particular, Oki et al. study RWA in presence of SRLGs, introducing the concept of weighted SRLGs. Two paths sharing many SRLGs have low probability to be selected as working and spare path pair, since weight of SRLGs contained on links is added to link cost. Shao et al. [16] present a custom maximum link disjoint path algorithm to RWA problem in optical telecommunication network in presence of SRLGs, taking a different approach than Oki et al. and using number of SRLGs as a metric independently of path length.

III. Routing and Wavelength Assignment in Presence of Shared Risk Link Groups

We now turn our attention to RWA of logical connections in the network, namely RWA of working and spare paths required for establishment of each connection. Routing requires link- and SRLG-disjointness of working and spare paths. On the other hand, wavelength assignment requires a common unused wavelength on each of the links the path traverses for both paths. The first requirement can be relaxed to maximum disjointness, if completely disjoint paths do not exist. The second requirement can be relaxed if optical network contains support for wavelength conversion, so only an unused wavelength on each link is required, i.e. the wavelength does not need to be the same one in all the links used by the path.

A. Shared Risk Link Group Disjoint Paths

Despite the fact a network might offer many options for routing working and spare paths for of a particular connection, generally not all of them need to be link- and SRLG-disjoint or satisfy certain limit of path length.

To illustrate link- and SRLG-disjointness, we turn our attention to network shown in Figure 2 which offers four possible paths between nodes 1 and 8. Out of those four we need to pick one for working path and one for spare path. One option would be to route two paths as $1-2-3-8$ and $1-4-5-8$. We can see that links $1-2$ and $1-4$, and also $3-8$ and $5-8$ each have a common SRLG, see these two paths despite being link-disjoint share two SRLGs. If instead of $1-2-3-8$ one picks $1-6-2-3-8$ as

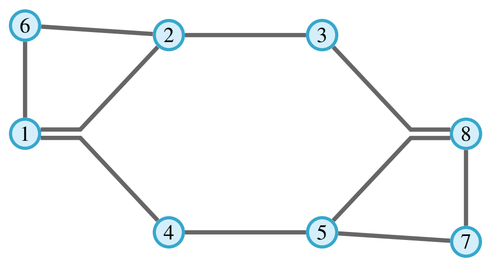

Figure 2. Example network used to illustrate the concepts of link- and SRLG-disjointness.

working path and spare path remains unchanged, only one common SRLG remains between two paths. Finally, paths $1-6-2-3-8$ and $1-4-5-7-8$ are both link- and SRLG-disjoint. Despite the fact that they are longer, the requirement to avoid simultaneous failure of working and spare path is quite often more significant than increase in path length.

RWA problem in presence of shared risk link groups can generally be written as an integer linear program, and software solvers can be applied [17]. However, due to NP-completeness of RWA problem, relaxation techniques and heuristics are commonly used. The maximum SRLG-disjoint path algorithm we utilize and optimize in this work is similar to algorithm proposed by Shao et. al. [16]. The algorithm is as follows:

1) Route the working path using Dijkstra shortest path algorithm. Let the set of links used by working path be L_w, and let $S(L_w)$ be a set of all links that contain at least one common SRLG with a link in L_w.
2) To route the spare path, remove from graph links in L_w (links used by working path) and also remove links in $S(L_w)$ (links with commons SRLG with working path). If possible route the spare path using Dijkstra shortest path algorithm and exit with success.
3) Let $s = |S(L_w)|$. Then there are $\binom{s}{r}$ r-subsets of links in $S(L_w)$, and 2^s subsets total. Let $i = 1, 2, \ldots, s$. In $i-th$ step do the following:
    a) Select next $(s-i)$-subset of links in $S(L_w)$. If all $(s-i)$-subsets have been tried, increment i and continue.
    b) Remove from graph links in L_w and $s-i$ links selected in subset. If possible route the spare path using Dijkstra shortest path algorithm and exit with success.

It is reasonable to route the working path as shortest path since it is used most of the time. It is easy to see that algorithm ends either upon finding a maximum SRLG-disjoint and completely link-disjoint spare path or concluding no link-disjoint path exists.

Note also that it would be trivial to extend the algorithm to find maximum link-disjoint path in addition to maximum SRLG-disjoint path if there is such requirement. Combining link- and SRLG-disjointness with particular weight or coefficient assigned to each is also a possibility.

B. *Simulation Models*

We used PWNS, an extension of ns-3 network simulator intended for availability study of optical telecommunication network. Models for optical network components such as demultiplexers and multiplexers, fiber, edge devices, optical cross connects, path computation element and control plane are included [18], as well as models for network cables and SRLGs [5]. Component failure and repair events can be simulated; component uptime and downtime are tracked and used for availability estimation.

We extended path computation element available in PWNS with support for CUDA-based Dijkstra shortest path finding, which is used in maximum disjoint path algorithm.

IV. Maximum Disjoint Path Algorithm Parallelization Approach

Graphics processors began as general-purpose computing processors with programmable shaders on NVIDIA GeForce FX and AMD Radeon series of graphics cards in 2003 [19]. Three programming languages were used: NVIDA Cg, OpenGL Shading Language (GLSL), and High-level shading language (HLSL), part of Microsoft DirectX suite. Regardless of the requirement to significantly alter algorithms to fit them for the GPU, usage of GPUs for non-graphics computations started to grow and NVIDIA saw the potential in it. GeForce 8 series introduced an application programming interface (API) called Compute Unified Device Architecture (CUDA) intened for general purpose computing on the GPU [7].

A. *Compute Unified Device Architecture*

GPUs are very different from commonly used CPUs. GPUs are essentially single instruction, multiple data (SIMD) parallel processors, meaning they have many processing elements able to do the same operation on multiple data elements simultaneously. As we have seen in examples mentioned in SectionII, gains from using GPUs vary a lot depending on amount of data-level parallelism present in algorithm one is aiming to accelerate.

Roughly a year after the introduction of CUDA an open standard called OpenCL (short for Open Compute Language) appeared. OpenCL is very similar to CUDA both in application domain and syntax, but has not so far gained market share comparable to CUDA. In addition to the fact that CUDA appeared first, relative unpopularity of OpenCL is also due to lesser amount of literature and advanced programming tools compared to CUDA. While both standards are very similar, they are not compatible [20].

We picked CUDA for this work, and from now on we focus solely on it. CUDA is as an extension of programming languages C/C++ and Fortran. CUDA Application Programing Interface (API) enables programmer to use threads, grouped in blocks of threads. Threads can share memory if required, and thread synchronization mechanism is provided. On the other hand, blocks do not have these features, and execute independently of each other. CUDA programming model is particularly suited for multidimensional arrays. Functions written in CUDA intended for GPU execution are called kernels. When a kernel is called from the code, the number of blocks and threads used for execution is specified. This allows writing kernels once for data arrays of different shapes and sizes.

B. Algorithm Parallelization Approach

Due to many academic and open source efforts utilizing CUDA, a number of libraries with highly optimized versions of commonly used algorithms (such as reduction, transformation, and sorting) have appeared. However, due to our particular needs we describe below, we implemented our work in pure CUDA C/C++ without using any additional libraries.

To fit our problem into data-parallel framework, we opted for parallelizing the Dijkstra algorithm in maximum disjoint spare path routing stage. Algorithm described in Section III remains unchanged in stages 1 and 2. Stage 3 is done on the GPU in way that:

- CPU generates 2^s subset of links and stores them in an array, which is copied to GPU.
- GPU kernel is called in $\frac{2^s}{512}$ blocks with 512 threads in each block.[1]
  - Each thread takes its subset from the array of subsets stored in GPU memory, and stores a copy of the graph in statically allocated array contained in per-thread local memory.
  - Each thread does Dijkstra shortest path algorithm on graph stored in per-thread local memory. If the shortest path is found, it is stored in global memory.
- Array of paths that were found is copied back from GPU.

To simplify the implementation, we also convert link weights to integer. To contain decimals, prior to conversion we multiply the weights by 1000. In our test networks links weights (lengths) are in order of magnitude of 100 (i.e. kilometers), and sum of lengths of all links is in order of magnitude of 1000. Multiplied by 1000, this gives numbers in order of magnitude 10^6 which is way below 10^9 order of magnitude of 32-bit integer maximum value.

One might possibly be concerned here by the amount of memory used for graph copies. However, used memory consistently remained under 1 GB for all scenarios we tested. Since modern entry level domain GeForce GPUs come with over 1 GB of video memory, we did not consider this a big issue. However, GPU memory usage can be reduced further by utilizing dynamic instead of static memory allocation for storing per-thread arrays representing graphs.

V. Performance Measurements

Our testing and benchmarking system consists of AMD FX-6100 6-core CPU and NVIDIA GeForce GTX 480 GPU. Since we work only with integers, neither 64-bit floating point precision nor extremely large amounts of GPU memory are required for our implementation. Therefore, consumer grade GeForce GPUs work just as well as more expensive professional grade Teslas and Quadros.

[1]Early GeForce and Tesla cards support a maximum of 512 threads per block. Later cards allow 1024 threads per block; regardless, we opted for 512 threads per block to gain wider compatibility, since we had no particular requirement to increase number of threads per block.

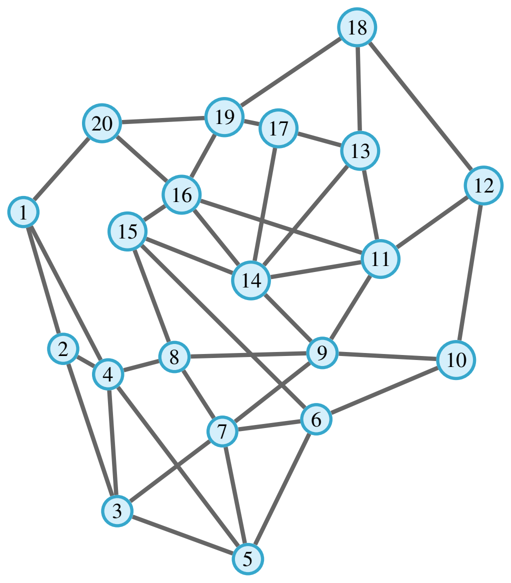

Figure 3. Test network topology containing 20 nodes and 40 links.

For the performance benchmarking we use three networks: 20 nodes and 40 links (Figure 3), 25 nodes and 50 links (Figure 4), and 30 nodes and 60 links (Figure 5). All three networks were first used by Grover et al. [21], [22].

We evaluate performance using the scenario where bidirectional connections are established between all pairs of nodes. For the the test networks, 20, 25 and 30 nodes implies 190, 300 and 435 bidirectional connections established. We benchmark using scenarios with 20, 40 and 80 SRLGs existing in the network. We assume each SRLG contains two cables.

Program execution time of CPU and GPU versions of the algorithm for 20 node 40 link network is shown in Figure 6. We can see that even GPU performance is consistently better, despite large variance in magnitude of difference. If we compare 80 SRLG scenario, GPU computation time is only 3 seconds, which is 7 times better than CPU computation time of 21 seconds. However, in case of 100 SRLGs, GPU takes 39 seconds and CPU takes 72 seconds, so the difference isn't nowhere as large.

Program execution time of the algorithm for 25 node 50 link network is shown in Figure 7. Here we can see that speed is much more consistent, and ends up at nearly 10 times in 100 SRLG case with CPU execution taking 107 and GPU taking only 11 seconds.

Finally, we take a look at 30 node 60 link network results in Figure 8. Up to 80 SRLGs GPU is consistently faster, coming again up to 10 times in scenarios 60 and 80 SRLGs. However, for 100 SRLGs scenario this is not the case, and speedup is around 1.5 times.

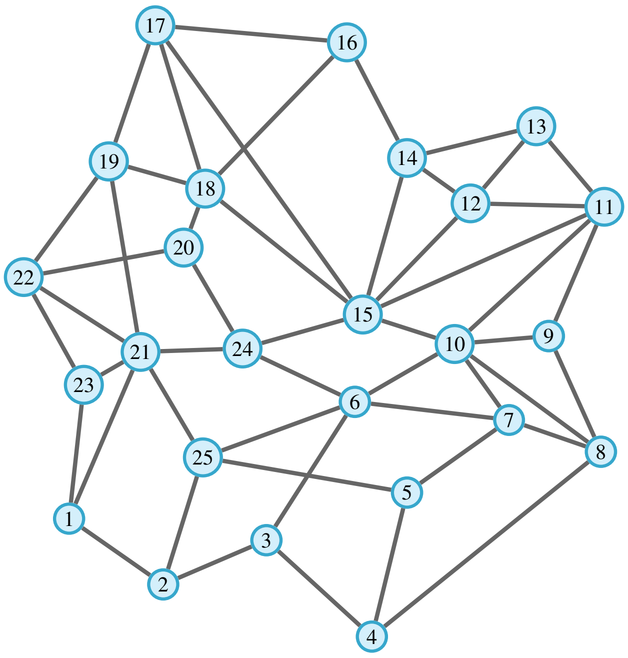

Figure 4. Test network topology containing 25 nodes and 50 spans.

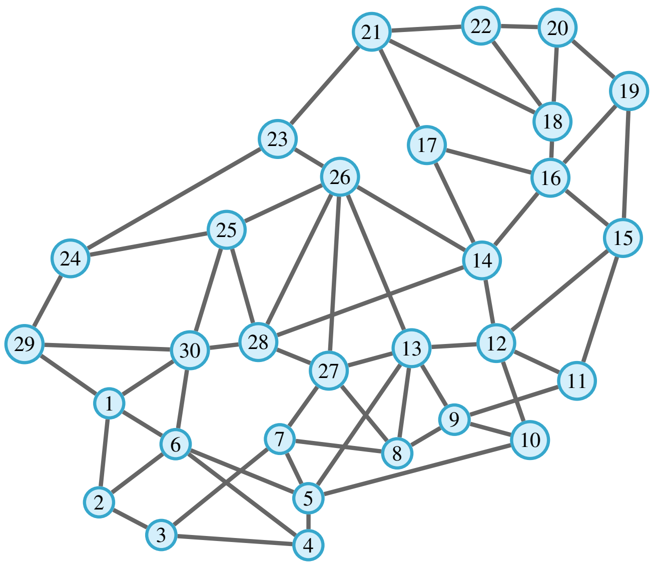

Figure 5. Test network topology containing 30 nodes and 60 links.

VI. Conclusion, Discussion and Future Work

We presented an approach to optimization of maximum shared risk link group-disjoint path algorithm by offloading a part of algorithm to GPU for execution. We believe this approach to be future-proof, considering the increasing heterogeneity of compute components inside computer systems over time, each chip suited for different kind of work. We found the optimization approach we took improving performance very significantly, and decreasing simulation runtime. Increasing number of SRLGs has shown an expected impact on performance; on average, more SRLGs increases the number

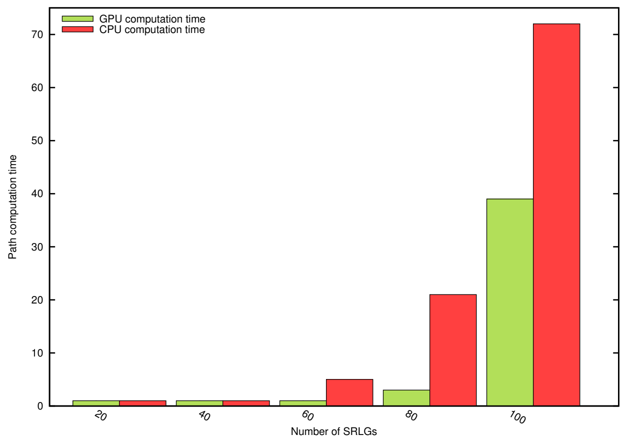

Figure 6. Performance measurements for 20 node 40 link topology.

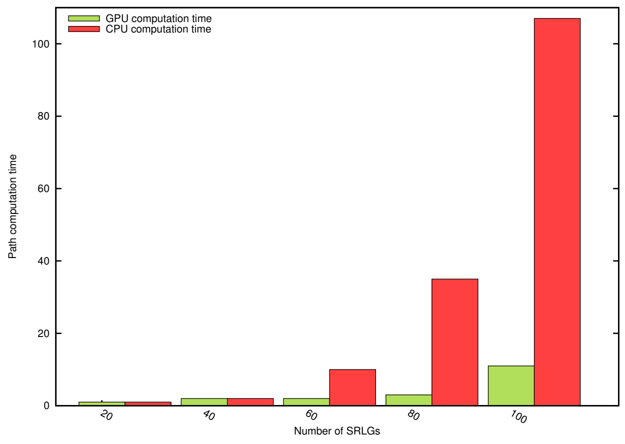

Figure 7. Performance measurements for 25 node 50 link topology.

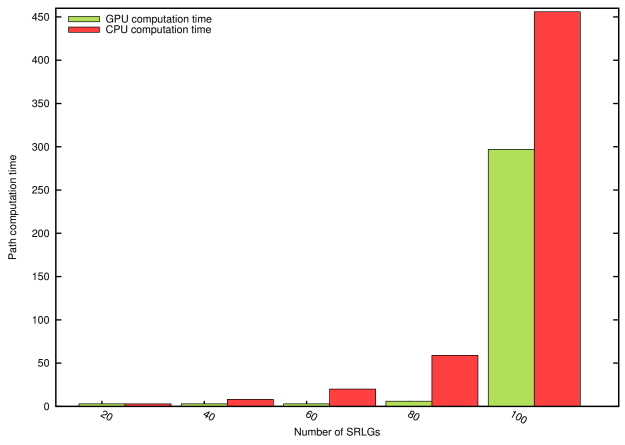

Figure 8. Performance measurements for 30 node 60 link topology.

of subsetss the algorithm has to process. However, to give definitive performance assessment and select code "hotspots" for optimization further study will be required. Specifically, we are interested in evaluating larger scenarios with more nodes and links in the network.

Our future work will be focused on further optimizing the implementation by increasing amount of parallelism and decreasing memory usage. Dynamic parallelism available on NVIDIA Kepler and subsequent chips, which we have not utilized so far to ensure broader compatibility, is potentially useful for increasing amount of parallelism.

With GPUs making their way into embedded hardware such as NVIDIA Tegra and Adapteva Parallella, it could be possible to use GPUs also for routing in control plane of the optical telecommunication network. Considering the performance and energy efficiency of the GPUs, usage of them in real world optical network control plane is an interesting research direction for the future.

Acknowledgment

Performance benchmarks were done at University of Rijeka CUDA Teaching Center provided in large part by NVIDIA. The authors would like to thank NVIDIA for providing hardware and support that made this research possible.

References

[1] P. Manohar, D. Manjunath, and R. Shevgaonkar, “Routing and wavelength assignment in optical networks from edge disjoint path algorithms,” *Communications Letters, IEEE*, vol. 6, no. 5, pp. 211–213, 2002.

[2] P. Sebos, J. Yates, A. Greenberg, and D. Rubeinstein, “Effectiveness of shared risk link group auto-discovery in optical networks,” in *Optical Fiber Communication Conference and Exhibit, 2002. OFC 2002.* IEEE, 2002, pp. 493–495.

[3] M. Lackovic and B. Mikac, “Analytical vs. simulation approach to availability calculation of circuit switched optical transmission network,” in *Telecommunications, 2003. ConTEL 2003. Proceedings of the 7th International Conference on*, vol. 2. IEEE, 2003, pp. 743–750.

[4] E. K. Çetinkaya, D. Broyles, A. Dandekar, S. Srinivasan, and J. P. Sterbenz, “Modelling communication network challenges for future internet resilience, survivability, and disruption tolerance: A simulation-based approach,” *Telecommunication Systems*, pp. 1–16, 2011.

[5] V. Miletic, B. Mikac, and M. Dzanko, “Impact evaluation of physical length of shared risk link groups on optical network availability using monte carlo simulation,” in *Network and Optical Communications (NOC), 2013 18th European Conference on and Optical Cabling and Infrastructure (OC&I), 2013 8th Conference on*. IEEE, 2013, pp. 249–255.

[6] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, “Network simulations with the ns-3 simulator,” *SIGCOMM demonstration*, 2008.

[7] D. Kirk, “Nvidia cuda software and gpu parallel computing architecture,” in *ISMM*, vol. 7, 2007, pp. 103–104.

[8] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, “Gpus and the future of parallel computing,” *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[9] W. J. Dally, “The end of denial architecture and the rise of throughput computing,” in *Keynote speech at Desgin Automation Conference*, 2010.

[10] S. Han, K. Jang, K. Park, and S. Moon, “Packetshader: a gpu-accelerated software router,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 195–206, 2011.

[11] B. P. Swenson and G. F. Riley, “Simulating large topologies in ns-3 using brite and cuda driven global routing,” in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 159–166.

[12] G. F. Riley, M. H. Ammar, and E. W. Zegura, “Efficient routing using nix-vectors,” in *High Performance Switching and Routing, 2001 IEEE Workshop on*. IEEE, 2001, pp. 390–395.

[13] P. Harish and P. Narayanan, “Accelerating large graph algorithms on the gpu using cuda,” in *High performance computing–HiPC 2007*. Springer, 2007, pp. 197–208.

[14] E. Oki, N. Matsuura, K. Shiomoto, and N. Yamanaka, “A disjoint path selection scheme with shared risk link groups in gmpls networks,” *Communications Letters, IEEE*, vol. 6, no. 9, pp. 406–408, 2002.

[15] G. Xiao and X. Pan, “Heuristic for the maximum disjoint paths problem in wavelength-routed networks with shared-risk link groups [invited],” *Journal of Optical Networking*, vol. 3, no. 1, pp. 38–49, 2004.

[16] X. Shao, Y. Bai, X. Cheng, Y.-K. Yeo, L. Zhou, and L. H. Ngoh, “Best effort srlg failure protection for optical wdm networks,” *Optical Communications and Networking, IEEE/OSA Journal of*, vol. 3, no. 9, pp. 739–749, 2011.

[17] H.-W. Lee, E. Modiano, and K. Lee, “Diverse routing in networks with probabilistic failures,” *Networking, IEEE/ACM Transactions on*, vol. 18, no. 6, pp. 1895–1907, 2010.

[18] V. Miletic, B. Mikac, and M. Dzanko, “Modelling optical network components: A network simulator-based approach,” in *Telecommunications (BIHTEL), 2012 IX International Symposium on*. IEEE, 2012, pp. 1–6.

[19] E. Wu and Y. Liu, “Emerging technology about gpgpu,” in *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*. IEEE, 2008, pp. 618–622.

[20] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, “From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming,” *Parallel Computing*, vol. 38, no. 8, pp. 391–407, 2012.

[21] W. D. Grover, *Mesh-based survivable networks: options and strategies for optical, MPLS, SONET, and ATM Networking*. Prentice Hall, 2004.

[22] J. Doucette and W. D. Grover, “Shared-risk logical span groups in span-restorable optical networks: Analysis and capacity planning model,” *Photonic Network Communications*, vol. 9, no. 1, pp. 35–53, 2005.