Experimental Assessment of a High Performance Back-end PCE for Flexgrid Optical Network Re-optimization

Lluís Gifre1,2*, Luis Velasco1, Nacho Navarro2,3, Gabriel Junyent1
1Optical Communications Group (GCO), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
2Barcelona Supercomputing Center (BSC), Barcelona, Spain
3High Performance Computing Group, Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
*e-mail: lgifre@ac.upc.edu

Abstract: A specialized high performance Graphics Processing Unit (GPU)-based back-end Path Computation Element (PCE) to compute re-optimization in Flexgrid networks is presented. Experimental results show 6x speedups compared to single centralized PCE.

© 2014 Optical Society of America
OCIS codes: (060.4251) Networks, assignment and routing algorithms; (060.4256) Networks, network optimization

1. Introduction

The flexgrid optical technology [1] is a promising candidate to be used on future optical networks. This technology can assign a variable amount of contiguous slices to each optical connection depending on the requested bandwidth and used modulation format, resulting in an efficient usage of the optical spectrum.

A control plane, responsible for handling both network resources and optical connections, is commonly used in dynamic scenarios. This control plane can include either a centralized Path Computation Element (PCE) [2] or a Software Defined Network (SDN) controller. In addition to computing the physical route and the spectrum allocation (RSA) for new connection requests, some other tasks are performed, such as computing alternative RSA for a set of connections for restoration [3] or for re-optimization (e.g. defragmentation) [4] purposes. These tasks require intensive computations to be performed in short time.

Graphic Processing Units (GPUs) are lately used in intensive computational tasks. State-of-the-art GPUs can perform computations in the range of Tera-FLOPS (FLoating-point Operations Per Second) consuming a fraction of energy and physical space compared to CPU-based computers.

In this paper, we propose extend the centralized PCE architecture with a specialized back-end PCE to perform intensive computation. The resulting architecture consists then in a front-end PCE computing simple RSA to serve connection requests and in a back-end PCE focused on computationally intensive tasks (e.g. recovery or re-optimization). Note that although we focus on PCE-base control plane, this architecture is also applicable to SDN. We experimentally compare the performance of the proposed high performance (HP) back-end PCE on two real backbone networks.

2. PCE Architecture

In the centralized architecture (Fig. 1a), the PCE executes all tasks to serve connection requests using its internal CPU. Although parallelism is available in current CPU architectures (e.g. multi-core or even multi-processor), the amount of threads that can be executed in parallel is relatively low, e.g. 4/8 cores or processors. Therefore, in case of an intensive computation needs to be done the PCE can either dedicate all the available resources to that computation thus introducing additional delay to any other request arriving during such computation, or dedicate only part of the resources thus enlarging the computation time of that intensive computation.

A solution to the above problem is to split the centralized PCE into two parts (Fig. 1b): a front-end PCE and a back-end PCE. While the front-end PCE is responsible for simple computations (e.g. single path computation), intensive computations are delegated to the back-end PCE. An additional advantage of this architecture is that back-end PCEs can be based on specific hardware devices and/or software applications. In that regard, we used GPU devices [5] able to execute thousands of threads in parallel attaining peaks of performance in the range of Tera-FLOPS. GPUs are organized as blocks of threads: threads within a block collaborate among them, and different blocks are executed independently in parallel.

When an intensive task need be computed, the front-end PCE forwards the set of path computation requests to the back-end PCE to solve the task. To this end, the PCE Protocol (PCEP) [6] can be used; a single PCReq message can contain the whole set of path computations and a PCEP Synchronization VEctor (SVEC) object [7] is included to group together several path requests so as to the computation is performed for all those requests (bulk), attaining thus a global optimum.

It is clear that separating the PCE into two parts introduces some additional delay to intensive path computation. Therefore, an estimation of the minimum size to decide whether a computation is better performed...
3. The Bulk Path Computation Problem

To compare the above PCE architectures, let us solve a simple single layer optical network bulk path computation problem. Note that this problem can be easily applied to many scenarios, including failure recovery and re-optimization. The bulk path computation problem consists on:

**Given:**
- a network topology represented by a graph $G(N, E)$ where $N$ is a set of nodes and $E$ is a set of fiber links each one connecting two nodes;
- the spectrum characteristics of each link including the set $S$ of slices and the spectrum and the slice widths;
- a set $D$ of demands (bulk) to be routed and allocated on the network; each demand $d$ is identified by the tuple $(s_d, l_d, n_d)$, where $s_d$ and $l_d$ are the source and destination node, respectively, and $n_d$ is the amount of slices required for the bitrate requested.

**Output:** the route and spectrum allocation for each demand $d$ in the bulk.

**Objective:** Minimize the amount of rejected slices and the spectrum resources (i.e., each slice in each link) used.

The problem need to be solved in stringent times (e.g. ms); as a consequence, some heuristic algorithm need to be devised. Although several meta-heuristics can be used (see e.g. the algorithms in [3], [4]), for the sake of clarity, we have developed a simple algorithm that performs a number of iterations, randomly sorting the demands in the bulk at each iteration. Following the resulting ordering, the RSA problem is solved for each demand in order. This approach is very simple and, therefore, can be easily implemented. The heuristic algorithm has been implemented in C++ for CPUs and GPUs and placed in the front-end and back-end PCE modules, respectively, in our iONE test-bed. Several bulk path computation requests were created by the front-end PCE. When the front-end PCE is in charge of performing the bulk computation, one single CPU thread is used; therefore, iterations of the above heuristic algorithm are performed sequentially in the front-end PCE. In contrast, when the front-end PCE delegates the computation to the back-end PCE, the former opens a new PCEP session, creates a PCReq message containing every path request and a SVEC object, and forwards the message to the back-end PCE. Upon receiving a PCReq message, the HP back-end PCE uses the GPU device to accelerate the computation; each iteration is executed into an independent thread block in the GPU, and all the threads inside a block collaborate in solving the RSA problem in parallel for each demand in the bulk following the defined order. The back-end PCE sends a PCRep message containing the route and spectrum allocation for each demand or the object NO_PATH when no feasible solution was found for a demand (Fig. 3). Due to GPU’s architecture, huge amount of blocks must be computed to achieve the peak performance, so we expect better results when the number of iterations is high.
We assume that the time available for bulk computation is 100ms. Therefore, that is the maximum time in the case the computation is performed in the front-end PCE’s CPU. Nonetheless, since PCEP time, including PCEP session opening and PCReq and PCRep transmission, is in the order of 2ms on average, the available computation time in the case the back-end PCE performs the computation is limited to 98ms.

A large number of bulk path computations were performed by both the front-end and the back-end for each of the considered networks. Several bulk sizes and number of iterations were also considered. In addition, each bulk was solved 10 times and so, average figures are provided.

Fig. 4a plots the solving times for bulks of 50 demands on the BT network computed on the front-end PCE’s CPU (dotted line) and on the back-end PCE’s CPU+GPU (solid line). As observed, 120 heuristic iterations can be done within 100ms when the computation is performed in the front-end PCE, whereas 990 iterations are done in the back-end PCE within 98ms (i.e. 8.4x speedup). Similarly, 6.7x speedups are shown for the TEL network topology (Fig. 4b). The jerk-like behavior of GPU plots are as a consequence of the amount of blocks that need to be allocated in the GPU.

It is clear that the larger the number of iterations done, the better the quality of the solution. Aiming at studying that quality, we have evaluated the probability of obtaining an optimal solution as a function of the number of iterations computed (Fig. 5). In the case of the BT network, the probability of finding the optimal solution with the front-end PCE is relatively high with only 120 iterations (78.5%), although the back-end PCE reaches 91.5% of probability as a result of computing 990 iterations. For the TEL network, however, the obtained probabilities are lower as a result of its larger size. Notwithstanding that, the back-end PCE reaches 85.9% of probability in contrast to the poor 55.8% obtained when the front-end PCE was used.

Table 1 details the number of iterations and solving times to obtain 90% of probability of finding an optimal solution for three different bulk sizes. As observed, when both the size of the bulk and the size of the network are low, the front end PCE is able to compute the amount of iterations needed to reach 90% of probability of optimality. However, as soon as either the size of the bulk or the topology increases, the front-end PCE cannot compute the number of iterations required within 100ms. Note that bulk sizes in the order of 40-50 demands are common in restoration and re-optimization.

5. Conclusions
A specialized HP GPU-based back-end PCE has been presented to compute re-optimization in Flexgrid networks. Experimental results performed in the iONE test-bed showed 6x speedups compared to front-end CPU running times, resulting in an increased probability of finding optimal solutions.

References