Parallel Simulations of Optical Communication Systems

Nikolay Karelin, *Member, IEEE*, Gena Shkred, Alexander Simonov, Sergei Mingaleev, Igor Koltchanov*, and André Richter*, *Senior Member, IEEE*

VPI Development Center, Chapaeva str. 5-111, 220034 Minsk, Belarus

** VPIphotonics GmbH, Carnotstr. 6, 10587 Berlin, Germany*

Tel: +49 30 398 058-0, Fax: +49 30 398 058-58, e-mail: andre.richter@vpiphotonics.com

ABSTRACT

Sophisticated numerical simulations represent an indispensable tool for developing new optical communication systems and solutions. With wider availability of parallel computing hardware (most notably, multi-core CPUs and GPUs) it becomes increasingly important to effectively parallelize simulation algorithms.

In this paper, we review the main principles and common pitfalls of parallel simulations for optical systems and components. The addressed topics include signal propagation in fibers, fiber gratings, active and passive semiconductor components, parallel simulations of transmission systems and photonic integrated circuits represented by networks of modules, as well as general-purpose optimization tasks.

Keywords: simulation, parallel, multicore, GPU, nonlinear fiber model, split-step algorithm.

1. INTRODUCTION

Today, development of new photonic communication systems is almost impossible without preliminary numerical simulation and optimization of various designs. With increasing complexity of the components and systems, the simulation performance becomes a limiting factor to develop an optimized solution in reasonable time. On the other hand, the 'raw' performance of modern computers, as characterized by CPU clock rates, is not increasing during the last ten years. The reason for that is that with increasing density of components, and resulting rise of current densities, the heat generation inside a CPU becomes prohibitively large. This is commonly referred to as 'power wall', as the amount of power the CPU can safely dissipate limits the clock rate [1].

The growing complexity of CPUs and the above mentioned 'power wall' problem have resulted in almost ubiquitous use of multicore architectures where CPUs contain several independent processing units with associated high-level cache memory. However, increasing the number of cores does not necessarily lead to an equivalent improvement of the computation performance. In this paper, we try to briefly review the most important ways to speed-up simulations of optical communication systems and components. In addition to multi-core CPUs, graphical processing units (GPUs) can be used for simulation speed-up. Performance factors resulting from the massively-parallel GPU architecture are discussed in this paper as well.

A performance increase due to parallel computation is usually described by the Amdahl law [2]. On a parallel computer with N CPUs (or an N-core CPU), the simulation speed-up is given by the following formula [3]:

$$S_N \equiv \frac{T_1}{T_N} = \frac{1}{1 - a + a/N + c_p + N \cdot c_s}, \tag{1}$$

where T_1 and T_N are the simulation times with a single CPU and multiple CPUs, respectively, a is the fraction of parallelizable operations in the algorithm, c_p is the relative extra time spent by each processor in parallel (such as setup or idle time when waiting for resources or data to be available), and c_s is the relative additional time spent by each processor for sequential operations (such as synchronization). The overall impact of the sequential overhead can depend on the number of processors in various ways. Here we follow [3] assuming a simple proportionality relation. The Amdahl law supposes that the performance of each CPU in the parallel- and sequential processing is the same. It is necessary to note that despite its simplicity, the Amdahl law is still subject of active investigation and discussion in parallel computing community, see for example a recent paper [4]. According to the expression above, the following two statements should be fulfilled for achieving the best performance:

- A computation task should have a significant parallelizable part.
- The overhead associated with the usage of parallel computation should be minimized.

While rather evident at first glance, these conditions can be quite difficult to meet in practice. In the following, we present and discuss several examples of effective parallelization of photonics simulations.

The rest of the paper is organized as follows: in the next section, several approaches for effective utilization of multicore CPU are presented. In Section 3, some aspects of effective utilization of massively-parallel GPUs for nonlinear fiber simulation are discussed. Section 4 briefly discusses the use of multi-core CPUs or computing clusters for optimization of communication systems.

2. SIMULATIONS ON MULTICORE CPU-UNITS

The approaches described in this section are based on employment of multiple computation threads [5], or in other words, different portions of a single program working within the same memory addressing space. Multithreading enables different CPUs (cores) to access the same data concurrently and eliminates the need to communicate data between processes. In some cases, this greatly simplifies the algorithm and may improve the performance.

The price to pay is the necessity to ensure that different threads are not writing data to the same variables simultaneously. This effect ('race conditions') requires special synchronization and is a frequent source of both, degraded performance and more complex code.

2.1 Linear Devices

An important case, where multithreading provides significant benefits, comprises simulations of linear optical devices, such as passive photonic circuits, fiber gratings, and other optical filters. Calculations of the device transfer functions at different frequencies are independent from each other and can be performed on different processor cores. This is an example for 'embarrassingly parallel' computation problems [6] – problems requiring computations with no or very little communication or data synchronization between different processing units.

2.2 Spatial Parallelization

Another case allowing efficient parallelization is when a simulated system or device can be divided into multiple spatially (or logically) separated components. For example, when applying the 'transmission line' model [7] to semiconductor active devices, the whole device is split into a number of small subsections (Fig. 1). Each subsection is driven by electrical and optical input signals produced by neighboring subsections, and is governed by a complex set of multi-physics equations. Simulations inside subsections are time-consuming in comparison with the signal exchange between neighboring subsections, and thus each subsection can be effectively modeled in a separate thread.

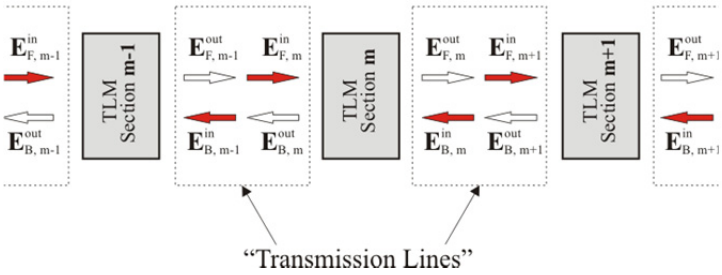

Figure 1. Multi-section model of semiconductor laser (or amplifier).

Another kind of simulation parallelization is possible on topological level, i.e. when several different models (e.g. two fibers with gratings) are independent from each other, as in Fig. 2, and therefore can be simulated in separate threads.

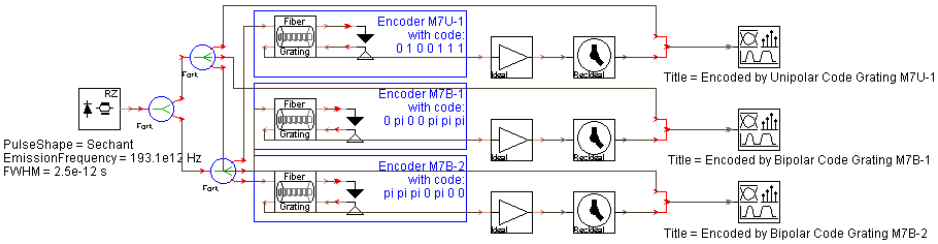

Figure 2. Branched simulation setup.

This approach, that is sometimes referred to as 'parallel scheduling' [8] can be combined with the above mentioned multithreading within specific models, helping to further improve performance because of hidden threads or memory latency.

However, it should also be noted that in many modern multi-core CPU architectures, parallel simulations do not lead to a performance gain. The reason is that CPUs can adjust their clock rates dynamically [9]: when only a single core is used, the clock rate may be higher than in the case when all cores are loaded with computations; otherwise CPU overheating might result.

3. NONLINEAR FIBER SIMULATION

Nonlinearities in optical communication fibers are among the main degradation factors of fiber-based communication systems and are crucial for transmission simulations [10]. Typically, propagation of an optical

signal in nonlinear fiber is efficiently simulated using the split-step Fourier method (SSFM). Often simulating the nonlinear fiber propagation takes most of the simulation time when modeling optical transmission systems. Therefore, an efficient implementation of the SSFM using capabilities of parallel computing is very important. An approach to parallelize the SSFM has been first proposed in [11] and is based on several simple principles:

- Computations at the linear and nonlinear steps of the SSFM are embarrassingly parallel in the frequency and time domains, respectively.
- An efficient parallel implementation of the Fast Fourier Transformation (FFT, as used by the SSFM) is available.

Besides the multi-core CPUs, the SSFM enables an efficient implementation on GPUs [12]. Modern GPUs have massive parallelization features: hundreds of processing units are capable to effectively handle hundreds of thousands of parallel threads.

On the other hand, GPU processing sometimes reveals quite surprising features of its functionality. According to our estimations, the FFT takes 60-70% of the total simulation time on CPU. In contrast thereto, most of the time is spent for the adaptive step size control for small and moderate signal sizes when using a GPU (see Fig. 3). This rather simple operation involves evaluation of either the peak signal intensity (in the 'Maximum Phase Change' method [10]), or of the root-mean-square of a difference between two approximate solutions (in the 'Local Error Method' [13]).

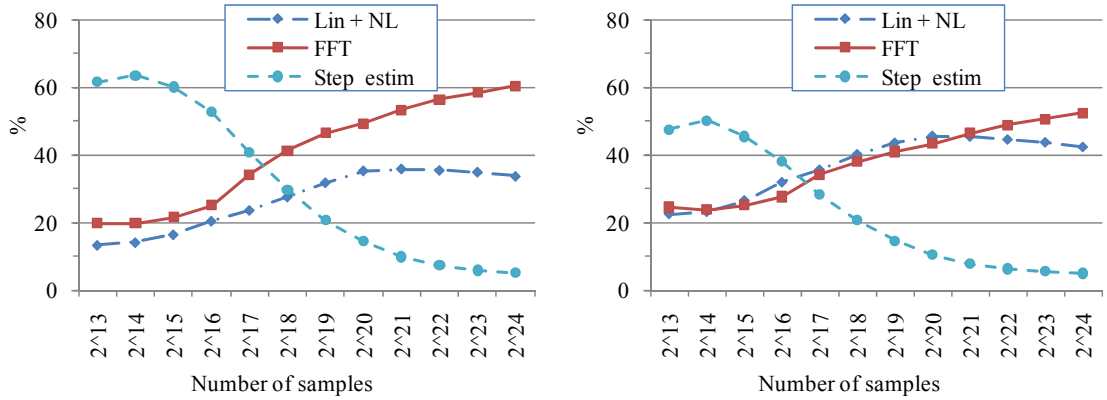

Figure 3. Contribution of different operations to SSFM performance on GPU (left – nonlinear phase change algorithm, right – local error method algorithm).

Both operations are examples of the so-called 'reduction' algorithms, when an array of values is 'reduced' to a single value. In the parallel implementation [14], the reduction is usually performed at several consecutive stages, with the values being processed pair-wise at each stage, as depicted in Fig. 4.

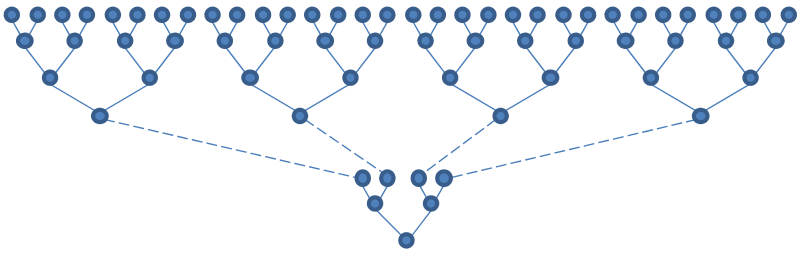

Figure 4. Principle of the parallel reduction.

As can be seen, on the last stages of the reduction, only a small fraction of the initial array is left. As a result, the remaining operations cannot be parallelized efficiently. Asymptotically, the reduction is scaled with $O(S/N + \log S)$, where S is the number of samples per signal and N is the number of processors [14].

It is worth noting that adaptive step-size control is an important factor affecting the overall SSFM performance: usually, the step size can be increased along the fiber length, for example, due to signal attenuation and reducing nonlinear effects. As a result, SSFM with step-size control performs faster than the fixed-step algorithm for the same accuracy, on both CPU and GPU.

In order to improve the computation performance of the reduction algorithm, its alternative implementation based on FFT has been investigated. As a result, the Local Error Method for nonlinear fiber simulation could be further improved. We found that for GPUs and moderate array sizes (up to 2^{16} elements) SSFM performs 1.5-2× faster with the FFT-based reduction implementation compared to using the implementation from [14].

Finally, it is worth noting that data processing on GPUs is still very young area and in many cases, optimal algorithms are not yet found for common problems. Also, some utility operations, such as memory management, require much more attention and optimization than for CPU-only simulations.

4. OPTIMIZATION AND RELATED ALGORITHMS

To complete the discussion on parallel simulation techniques, it is worth mentioning the cases where different tasks are only loosely related and can be implemented by using simulation processes running on different (even distant) computers. Foremost, these are multidimensional optimization tasks [12], but the same is related to parameter sweeps or Monte-Carlo simulations.

The point beyond effective parallel optimization here is that such simulations require running the same task multiple times with different parameter settings. In this case, the extra overheads in formula (1) are small, even if the individual simulations are performed on different computers with slow connections between them. Moreover, some of the modern optimization methods, such as the particle swarm optimization (PSO) or genetic algorithms (GA) are inherently parallel in their design.

The main benefit of the multiprocessing approach from the code development perspective is the good isolation of data, even in a multi-core CPU. It is necessary, though, to implement certain communication procedures to synchronize different sub-tasks.

Finally, when the available hardware resources permit, the multiprocessing approach can be combined with methods described in the previous sections, providing better overall performance for several cases of simulations.

5. CONCLUSIONS

The main conclusion of this paper is that for effective parallelization of simulation algorithms it is necessary to understand which parts of the simulation are better to parallelize with the available hardware. Fortunately, vast amount of data in models of modern communication systems provide enough room to split the simulation onto multiple processors. It is quite reasonable to expect that as parallel hardware architectures and appropriate software libraries mature, parallel simulations will become more straightforward. Presently, good performance on multi-core (or cluster) architectures requires a deep understanding of the algorithms, data and hardware.

REFERENCES

[1] D. Patterson: The trouble with multicore, *IEEE Spectrum*, Jun. 2010, http://spectrum.ieee.org/computing/software/the-trouble-with-multicore

[2] G.M. Amdahl: Validity of the single-processor approach to achieving large-scale computing capabilities, in *Proc. Am. Federation Inform. Process. Societies Conf.*, AFIPS Press, 1967, pp. 483-485.

[3] R.G. Brown: Maximizing Beowulf performance, in *Proc. 4th Ann. Linux Showcase and Conference*, Atlanta, October, 2000, pp. 329–340.

[4] L. Yavits, A. Morad, R. Ginosar: The effect of communication and synchronization on Amdahl's law in multicore systems, *Parallel Computing*, vol. 40, pp. 1-16, Jan. 2014.

[5] S. Akter, J. Roberts: *Multi-Core Programming. Increasing Performance through Software Multi-threading*, Intel Press, 2006.

[6] I. Foster: *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Reading, Mass: Addison-Wesley, 1995

[7] A.J. Lowery: Active photonic integrated circuits, in *Optoelectronic Devices*, J. Joachim, Ed., New York: Springer, 2005, pp. 427-448.

[8] Ch.-J. Hsu, J.L. Pino, Sh.S. Bhattacharyya: Multithreaded simulation for synchronous dataflow graphs, in *Proc. ACM/IEEE 45th Ann. Design Autom. Conf.*, Anaheim, CA, Jun. 2008, paper 19.4.

[9] Intel Corp.: Intel® Turbo Boost Technology 2.0: Quick overview, http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html

[10] G. Agrawal: *Nonlinear Fiber Optics*, 5-th ed, Oxford: Academic Press/Elsevier, 2013.

[11] S. Zoldi, *et al.*: Parallel implementation of the split-step Fourier method for solving nonlinear schrödinger systems," *SIAM News*, vol. 32, No. 1, pp. 8-12, 1999.

[12] S. Pachnicke: Efficient design of fiber optical transmission systems, in *Fiber-Optic Transmission Networks*, Springer-Verlag, 2012.

[13] O. Sinkin *et al.*: Optimization of the split-step Fourier method in modeling optical-fiber communications systems, *J. Lightwave Technol.*, vol. 21, p. 61-68, Jan. 2003.

[14] M. Harris: Optimizing Parallel Reduction in CUDA, https://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf