Parallel Execution of the NS-2 Sequential Simulator on a GPU

Tina Godjoska, Sonja V. Filiposka, *Member*, *IEEE*, and Dimitar R. Trajanov, *Member*, *IEEE*

Abstract—**A wide assortment of network simulators is available and most of them share the same issues when it comes to computation time and scalability. In this paper we propose an idea to alleviate these problems without the need of any special additional hardware resources or complex programming skills. Our proposal is an upgrade of the standard NS-2 simulator, based on parallel execution on the Graphic Processing Unit using OpenCL.**

Keywords — **GPGPU, Network simulator, NS-2, OpenCL, parallel programming.**

I. Introduction

In the modern society, computer networks have become an inevitable part of our life. As the use of computer networks drastically grows, the need for enhancing the existing network protocols and enforcing communication security increases. Tools like network simulators are used by researchers in order to test new scenarios and protocols in a controlled and reproducible environment. They allow the user to represent various topologies, simulate network traffic using different protocols, visualize the network and measure the performances.

Although they are very useful, most of the widely used network simulators do not scale [1]. Trying to simulate medium to large networks will result in a long simulation time unsuitable for investigating protocols. Some of the current network simulators implement various methods for accelerating the simulations by means of using parallel computation or changing the data representation of the nodes. However, most of the optimization techniques require using additional hardware resources as a computational grid or cluster and deep knowledge of the structure of the simulator.

The purpose of this paper is to propose a different concept for addressing the issue of poor simulator performances. Our idea is to gain speedup using only 'common' hardware resources and without having any particular additional knowledge of the simulator architecture and structure or parallel programming. For this purpose, we propose that the protocol extension of the simulator is done using OpenCL in a manner that allows the simulator as a whole to be executed on both the CPU and the Graphical Processing Unit thus inducing parallelism in the simulation system. In this way we address the problem of investigating the performances of new or modified protocols that almost always inherently slows down the network simulation many times over. At the same time, the economical and programming costs of this parallel simulation execution are incomparably small.

The remainder of the paper is organized, as follows: section II gives a brief overview of some of the existing parallel network simulators. Next, the General Purpose Graphic Processing Unit (GPGPU) computation model is described in section III, while section IV describes the OpenCL architecture. Section V elaborates our proposal for ns-2 and OpenCL interoperability followed by the conclusion and future work given in section VI.

II. Parallel network simulators

Discrete event simulation of telecommunications systems is generally a computation intensive task. A single run of a network model with thousands of mobile nodes may easily take several days and even weeks to obtain statistically trustworthy results and many simulation studies require several simulation runs.

Several parallel discrete event simulation systems have been developed to improve the scalability of network simulations. The traditional approach to realizing such a system is to create the parallel simulator "from scratch," where all the simulation software is custom designed for a particular parallel simulation engine. Examples of parallel network simulators using this approach include GloMoSim [2] and SSFNet [3][4] among others. Because new models must be developed and validated, this approach requires a significant amount of time and effort to create a useable system.

Another approach to parallel/distributed simulation involves interconnecting existing simulators. These federated simulations may include multiple copies of the same simulator (modeling different portions of the network), or entirely different simulators. The individual simulators that are to be linked may be sequential or parallel. Examples include the parallel simulators: PDNS [5], GTNetS [6], and OMNET++ [7][8].

In this section we will take a brief look at each of these example simulators.

GloMoSim (Global Mobile Information System Simulator) is a scalable simulation library designed at UCLA Computing Laboratory to support studies of large-scale

Tina Godjoska is with the Faculty of Electrical Engineering and Information Technologies, University of Ss. Cyril and Methodius, ul. Rugjer Boshkovic b.b., 1000 Skopje, R. Macedonia (phone: 389-70-743159; e-mail: tina.godjoska@gmail.com).

Corresponding Sonja V. Filiposka is with the Faculty of Electrical Engineering and Information Technologies, University of Ss. Cyril and Methodius, ul. Rugjer Boshkovic b.b., 1000 Skopje, R. Macedonia (phone: 389-2-3099153; e-mail: filipos@feit.ukim.edu.mk).

Dimitar R. Trajanov is with the Faculty of Electrical Engineering and Information Technologies, University of Ss. Cyril and Methodius, ul. Rugjer Boshkovic b.b., 1000 Skopje, R. Macedonia (phone: 389-2-3099153; e-mail: mite@feit.ukim.edu.mk).

network models, up to millions of nodes, using parallel and/or distributed execution on a diverse set of parallel computers. GloMoSim beside sequential adopts parallel simulation model using libraries and layered API. The libraries are developed using PARSEC [9]; a parallel C based programming language which uses message based approach. Currently GloMoSim supports protocols only for wireless networks, but the wired and hybrid network support is in progress.

SSFNet (Scalable Simulation Framework) claims to be a standard for parallel discrete event network simulation. SSFNet's commercial Java implementation (Renesys Raceway) is becoming popular in the research community, but SSFNet for C++ (DaSSF) does not seem to receive nearly as much attention, probably due to the lack of network protocol models. It is a high performance network simulator designed to transparently utilize parallel processor resources, and therefore scales to a very large collection of simulated entities and problem sizes.

NS-2 [10] is widely used in the networking research community and has found large acceptance as a tool to experiment new ideas, protocols and distributed algorithms. It is a discrete event driven sequential network simulator, developed at UC Berkeley by numbers of different researchers and institutions. NS-2 is suitable for simulating and analyzing either wired or wireless networks and is used mostly for small scale simulations. NS-2 is written in C++ and OTcl. The users define the network topology structure, the nodes, protocols and transmitting times in an OTcl script. The open source model of NS-2 encourages many researchers from institutions and universities to participate and contribute to improve and extend the project. NS-2 plays an important role especially in the research community of mobile ad hoc networks, being a sort of reference simulator [11]. Adding new network objects, protocols and agents requires creation of new classes in C++ and then linking them with the corresponding OTcl objects.

A parallel simulation extension for the traditionally widely used NS-2 simulator has been created at the Georgia Institute of Technology (PADS Research Group), but it is not in wide use. The *PDNS* (Parallel/Distributed NS) was designed to solve the NS-2 problems with large scale networks by running the simulator on a network of workstations connected either via a Myrinet network, or a standard Ethernet network using the TCP/IP protocol stack. In that way the overall execution time of the simulation should be at least as fast at the original single-workstation simulation, allowing simulating large scale networks.

GTNetS (Georgia Tech Network Simulator) is a network simulation environment which uses C++ as a programming language. GTNetS is designed for studying the behavior of moderate to large scale networks. The simulation environment is structured as an actual network with distinct separation of protocol stack layers.

OMNeT++ is a network simulation library and framework, primary used for simulation of communication networks, but because of its flexible architecture can be used to simulate complex IT systems too. OMNeT++ offers an Eclipse based IDE and the programming language used is C++.

Despite about 30 years on research on parallel discrete event simulation, today it is still more of a promise than part of everyday practice. As previously discussed, several tools have been developed targeting the network simulation community. However, they require learning new skills instead of using the more traditional sequential simulators the networking community is familiar with. The GloMoSim/Parsec system requires the user simulation modeler to learn new languages or language extensions to describe the network models. Similarly, when configuring a simulation in PDNS, one creates a TCL script for each federate that instantiates the portion of the network mapped to the federate, and instantiates rlinks to represent the "edge-connections" the span federates. Recasting legacy code in the new tools is also a big burden. Thus, these tools have not seen widespread use in the networking community in spite of their potential for speedup.

In contrast to these efforts, our work attempts to parallelize a very widely used network simulator transparently to the simulation modeler. NS-2 is considered to be a de facto standard simulator for internetworking protocol research. A large number of legacy simulation models exist using this simulator. The networking community is not inclined to rewrite them in another platform. Thus, a transparent parallel execution of NS-2 could fill an important gap in simulation research. Our parallel version of NS-2 is likely to see wide use even if the speedup is less than optimal as such speedup will be obtained without any programming effort on the part of the simulation modeler.

The fundaments of our idea lay on using parallel computation similar to the federated attempts, but instead of using a network of computers (which is not always available), we attempt to use one desktop computer wherein, additionally to the CPU, we will make use of the parallel capabilities of the GPU. The next section describes the promising, yet not very novel, idea of General Purpose computation on Graphic processing units (GPGPU).

III. GPGPU

Graphics Processing Units (GPUs) are high-performance many-core processors that can be used to accelerate a wide range of applications. Today, the GPU evolved into a programmable device which uses effects similar to the ones supported in the CPU. The highly parallel structure, the flexibility of floating point computation, the increased bandwidth and improving performance over the current CPU led to the interest of using the GPU not only for graphic purposes, but for general programming.

GPGPU [12] is a powerful technique that can take advantage of the programmability of the GPU and accelerate algorithms and complex computations gaining better performances from the same hardware resources. Data parallelism can be naturally implemented allowing simultaneous calculations using the same operation on different data sets. Furthermore, one can benefit from using all hardware resources available in the system. We can make an empirical use of these technologies combining the massive parallel computation capabilities with improved floating point computation on the GPU and task parallel computation on the CPU.

The complicated programmability of the GPU was the reason for emerging of a new high-level approach to abstract the GPU and treat it like a compute device. The GPU operates like a coprocessor of the CPU, and the calculations could be split into parts that can run separately on the devices. The sequential code can run on the CPU while the most time-consuming routines that run many times on different independent data sets can be isolated and computed on the GPU as many threads [13]. Using the GPU for programming is available for public use, since the programmer can use C as a programming language with few additional extensions, types and functions.

To take advantage of such powerful hardware resources there is a need of suitable software tools. As a high-level programming environment for GPGPU we chose to use OpenCL described in details in the next section.

IV. OpenCL

The Khronos Group consortium in collaboration with Apple, NVIDIA, AMD and Intel, developed the standard for Open Computing Language (OpenCL) [14]. The first public release of the royalty-free, cross-platform standard for OpenCL was on December 2008.

A. Platform model

The OpenCL platform model consists of a set of devices and a host on which the controlling application is running [15]. The host is usually the CPU but can also be an OpenCL device. The devices are divided into compute units, which are further divided into processing elements responsible for the computations. The main application runs from the host and submits commands to the processing element of the device, where the commands are executed as Single Instruction, Multiple Data or Single Process, Multiple Data.

B. Programming model

The programming model of OpenCL can be data parallel, task parallel or a hybrid of the two [16][17]:
Data parallel is used when the same calculation defined in the kernel is performed to multiple elements of the memory object. Each element is associated to one *work-item* with one-to-one mapping. The data parallel programming model can be hierarchical.
Task parallel is performed by enqueueing multiple kernels and running them in parallel using different available processors.

C. OpenCL Framework

The framework allows creation of a single parallel heterogeneous compute system, with one host and multiple devices [16][17].
OpenCL compiler – the language used for the kernels is an extension of ISO C99 standard with support for parallelism. Due to the fact that OpenCL is hardware independent, each manufacturer uses different compilers.

OpenCL runtime- ones the contexts have been created the runtime allows the host to manipulate them, manage scheduling, compute memory resources. The runtime transfers data between main memory and the dedicated VRAM of the GPU, directs the execution of the kernel. The runtime is also responsible for utilizing the GPU processing elements in the best possible way and managing the kernel dependences.

OpenCL API- OpenCL has a very rich API to abstract the hardware level over diverse computational resources. The API includes functions to query the runtime and platform in order to get the available devices in the system with a set of information that could be useful for further optimizations. Functions exists for creation and manipulation of memory objects (buffer, image and sampler objects), and devices. Also functions for building the program object, creating the kernel, setting the arguments and retrieving the results are included.

The basic structure of an OpenCL application includes:
Platform layer- On the host: Query the platform, compute devices and create the contexts.
Runtime- On the host: Create memory objects associate to the context, compile and create the command objects, issue commands to command queues, synchronize the commands and clean up the OpenCL resources.
Language- Consist of the kernel code written in C99 with some restrictions and OpenCL extensions.

To use OpenCL one only needs the OpenCL library and drivers, and does not require any special tools. At this time, debugging and profiling tools are not available, but having in mind that it is a still young technique this problem will be solved in short time.

V. Proof of concept: Interoperability between NS-2 and OpenCL

As previously mentioned, both NS-2 and OpenCL are written in C++ which makes them easier to integrate into one program. Several modifications should be done in the NS-2 files to allow the compiler to prepare and use the multiple processors installed on the machine (CPU, GPU and other processors which support OpenCL if present) for computation. Because NS-2 is mainly designed to work on a Linux machine there are only a few steps to prepare the system and exploit the NS-2 and OpenCL interoperability:

1. *Installing OpenCL drivers* – keep in mind that this is a hardware dependent task.
2. *Installing NS-2* - standard installation of the NS-2 package.
3. *Adding the OpenCL headers in ns-2 and making changes in the Makefile*: The necessary headers are available at the official Khronos web site [17]. (Specifically, the files *cl.h, cl_gl.h, cl_platform.h* and *clext.h* should be saved in the ns-2.34/apps folder). In order to instruct the compiler to use them, several additional changes should be done in the NS-2 Makefile (the line *LDFLAGS = -L$(LIBDIR) –lOpenCL* should be added along with the folder that contains all of the necessary OpenCL files).
4. *Creating a new protocol in ns-2*: the main idea of our work is not to change the existing NS-2 code, but add the parallelism to the new protocols we want to analyse using the simulator.

However this is not a strict restriction in which case this step should be skipped. Please keep in mind that this is the part that is going to be executed in parallel on the GPU and, thus, the written code should be such that exploits this advantage.

5. *Adding OpenCL functions in the protocol classes*: In order to make the computations run on GPU and other devices available, the class of the new protocol should include the *cl.h* header file. When the OpenCL environment is correctly set up, the specific OpenCL functions for setting and preparing the devices can be used. For getting the maximum of OpenCL, it is desirable to gather all necessary information about the involved network nodes in the current simulation in one or more arrays (one, two or three dimensional) and let the GPU do parallel computations.
6. *Adding the kernel code*: The code which is run in parallel against the nodes should be written in a separate file with *.cl* extension and saved in the ns-2.31 folder. The well structured kernel code and the memory organization are very important for gaining speed up. Attention should be given to the restrictions implied by OpenCL and the GPU limitations

For the purposes of testing the interoperability, we created a new radio propagation model in NS-2 intending to do the signal strength calculation at the receivers on the GPU. The tested radio model was fairly simple and served as a proof of concept only. The test of this interoperability was validated in the way that we compared the output result for a standard test wireless network with random mobility pattern and random cbr traffic using the NS-2 mobility and traffic generators. When compared to the output result for the standard sequential NS-2, the output result of our parallel version for the exact test scenarios was a perfect match.

Following the given steps, we found that once setup and running, the NS-2 user can simply forget about the OpenCL extension. The format of the TCL scripts needed to setup the simulation scenarios needs not to be changed in any way.

Using this test trial as an example, we would like to stress the possibilities for a great speedup using the GPU. Namely, in the radio propagation model example when a node is sending a packet over the air, the NS-2 simulator needs to calculate the received strength at every other node in the network in order to establish whether there is interference or the packet can be received correctly. Using the new radio model and the GPU, we are able to perform all of the separate calculations (sender to each node in the network) in parallel and thus create a result cache which is especially useful when simulating static networks. The gained speedup depends of the nature of the problem, the implementation of the algorithm, the portion of the code that can be parallelized, the size of the data and the technique used to avoid the memory bandwidth limitations. At this time is early to give numbers, but the growing interest for OpenCL applications shows that promising speedup is to be expected.

VI. Conclusion and Future Work

In this paper, a proposal of a parallel extension of NS-2 is given. Our idea differs from the existing parallel solutions in several points: it needs only modest hardware to run on (an OpenCL compliant GPU) and it is completely transparent to the end user. We believe that this approach can gain a lot of attention from the community exploiting the attention given to the general purpose computing.

Having our proof of concept, our future work is to implement protocols for NS-2 that will exploit the maximum of the GPU's parallel capabilities. We intend to use OpenCL for an existing implementation of the Durkin's terrain aware radio propagation model that has proven to be very time consuming in its traditional implementation. If we can accelerate the propagation model and simultaneously apply some changes to optimize the simulator itself, the gained speedup could have a significant value, allowing us to investigate large scale networks while using 'common' hardware resources.

VII. References

[1] Weingartner, E., vom Lehn, H., Wehrle, K., "A Performance Comparison of Recent Network Simulators," in *Conf. Rec. 2009. ICC* '09. *IEEE Int. Conf. Communications,* pp. 1–5.

[2] Zeng, X., Bagrodia R., and Gerla M., "GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks", in *Proc. 12th Workshop on Parallel and Distributed Simulation*, Banff, Alta. Canada, 1998, p. 154-161.

[3] Cowie, J.H., Nicol D.M., and Ogielski A.T., "Modeling the Global Internet", *Computing in Science and Engineering*, 1999

[4] Sunghyun Y., Young B. K., "A Design of Network Simulation Environment Using SSFNet", *First Int. Conf. on Advances in System Simulation*, SIMUL '09, Porto, 2009, pp. 73-78.

[5] Riley, G., R.M. Fujimoto, and M. Ammar, "A Generic Framework for Parallelization of Network Simulations", in *Proc. 7th Int.Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 1999, p. 128-135.

[6] Riley, G.F., "The Georgia Tech Network Simulator", in *Proc. of the Workshop on Models, Methods, and Tools for Reproducible Network Research (MoMe Tools)*, 2003.

[7] Varga A., "The OMNeT++ discrete event simulation system", *Proc. of the European Simulation Multiconference* (ESM'2001), Prague, Czech Republic, 2001.

[8] Sekercioglu Y. A., Varga A., Egan G. K., "Parallel Simulation Made Easy With Omnet++", in *Proc. of the European Simulation Symposium (ESS2003)*, Oct. 2003, Delft, The Netherlands.

[9] Parallel Simulation Environment for Complex Systems (PARSEC). http://pcl.cs.ucla.edu/projects/parsec/.

[10] NS-2 Newtork simulator. http://www.isi.edu/nsnam/ns/.

[11] Di Caro G. A., "Analysis of simulation environments for mobile ad hoc networks", Technical Report No. IDSIA-24-03, IDSIA / USI-SUPSI, BISON project, Switzerland, 2003.

[12] General-Purpose Computation on Graphics Hardware, http://gpgpu.org/

[13] OpenCL - The open standard for parallel programming of heterogeneous systems, http://www.khronos.org/opencl/

[14] Apple Inc., "OpenCL-Taking the graphics processor beyond graphics." ,August 2009

[15] Khronos OpenCL Working Group," The OpenCL Specification Version: 1.0", 16.5.2009

[16] NVIDIA Corporation, "NVIDIA OpenCL Best Practices Guide Version 2.3 ", 2009

[17] Khronos OpenCL API Registry, http://www.khronos.org/registry/cl/