Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 2463–2467

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Modeling and Simulation for Exploring Power/Time Trade-off of Parallel Deep Neural Network Training

Paweł Rościszewski

Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology, Gdańsk, Poland
pawel.rosciszewski@pg.edu.pl

Abstract

In the paper we tackle bi-objective *execution time* and *power consumption* optimization problem concerning execution of parallel applications. We propose using a discrete-event simulation environment for exploring this power/time trade-off in the form of a Pareto front. The solution is verified by a case study based on a real deep neural network training application for *automatic speech recognition*. A simulation lasting over 2 hours on a single CPU accurately predicts real results from executions that take over 335 hours in a cluster with 8 GPUs. The simulations allow also estimating the impact of data package imbalance on the application performance.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: high performance computing, energy efficiency, Pareto optimization, deep neural networks

1 Introduction and Motivations

Bridging the gap between *computational science* and *artificial intelligence* has probably never been more relevant than it is right now. Due to recent advances in *machine learning*, big emphasis in the field of *high performance computing* (HPC) has been put on improving performance of *deep learning*. Being able to efficiently utilize multiple computing devices, *machine learning* engineers face a trade-off between performance and power consumption. They need to decide if in given circumstances the computations should finish as soon as possible or if it is acceptable that they run longer but consume less resources, which could be used more efficiently for other applications or unused due to energy costs or imposed power limits. Energy-awareness has been recently gaining importance in HPC. Hardware manufacturers focus on energy efficiency of the computing devices, measured in performance per watt. Extensive work is held also in the field of software engineering to develop *multi-objective scheduling* and *parameter auto-tuning* strategies for finding optimal task assignments and execution parameters, taking into account not only performance, but also power consumption.

In previous work we introduced MERPSYS, a modeling and discrete-event simulation environment as a proposed method for estimating execution time [2] and power consumption [1] of parallel applications. In this paper we propose using MERPSYS for exploring power/time

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.214

trade-off of parallel applications on the example of training deep neural network for automatic speech recognition. Chosen existing approaches to multi-objective optimization of parallel application execution are discussed in Section 2. The proposed approach is described in Section 3, while in Section 4 we summarize the paper and propose future work directions.

2 Related Work

Multi-objective auto-tuning optimization has been proposed in [4] for parallel codes, where configurations selected by an evolutionary algorithm were iteratively evaluated through their execution on the target system. In case of long lasting applications, such approach might be infeasible, so a method for fast configuration evaluation is needed. Many existing approaches to optimization of parallel application execution are based on an assumption that a deterministic analytical model of the application is available. For example, tackling the problem of analyzing trade-offs between maximizing performance and minimizing energy consumption in a heterogeneous resource allocation problem, authors of [3] assume that an application is a static collection of tasks. The estimated times to compute (ETC) and estimated energy consumed (EEC) are given in the form of matrices defining the values for each task and device type. A similar approach is used in [9], where the energy and makespan trade-offs in heterogeneous computing systems are solved by linear programming techniques. In practice such a precise application model is often not available and instead there is a working implementation of the application as well as results from previous executions with specific configurations on the available hardware. In order to allow engineers to fully explore the resource management possibilities, a method is needed for estimating execution time and power consumption for different, often currently unavailable configurations. Some works assume lack of exact prior knowledge about the application. Task execution times are treated as stochastic variables in [5], where energy-efficient task scheduling for heterogeneous computing systems is proposed. A static resource allocation of bag-of-tasks application presented in [6] is robust towards the energy and makespan, so that instead of hard constraints, probabilities of violating the deadlines are considered.

3 Proposed Solution and Results

In our approach we use the MERPSYS discrete-event simulation environment. In this paper we describe a case study based on a real application and system described in Section 3.1. We describe the model used in our simulations in Section 3.2 and the process of tuning the model in Section 3.3. Finally, in Section 3.4 we show how the proposed *ParetoVisualizer* tool can provide insights to the power/time trade-off of the simulated application.

3.1 Testbed Application and System

The application considered in the experiments is training an acoustic model based on a deep recurrent neural network with 5 layers of 768 LSTM (Long short-term memory) cells each. The training database consists of 4200 hours of transcribed speech recordings in Polish language divided into 1061 archives, each of roughly 150MB size. The used training method is NG-SGD (natural gradient stochastic gradient descent) with parameter averaging [8]. The training is performed using the *nnet3* implementation from the Kaldi [7] speech recognition toolkit with the CTC (connectionist temporal classification) loss function. The number of GPUs used in the computations is gradually increased from a certain initial to a final value.

Real application executions for tuning the simulation model were run on 2 workstations, each equipped with four NVIDIA GeForce GTX TITAN X GPUs with NVIDIA Maxwell architecture and 12 GB of memory. Both workstations had 128 GB of RAM, were connected by Gigabit Ethernet interconnect and running 4.4.0-53 Linux kernel, 361.93.02 NVIDIA driver, CUDA 8.0 and OpenMPI 1.6.5. First workstation was equipped with two Intel Xeon E5-2620 v3 6-core CPUs and the second with two Intel Xeon E5-4650 v2 10-core CPUs.

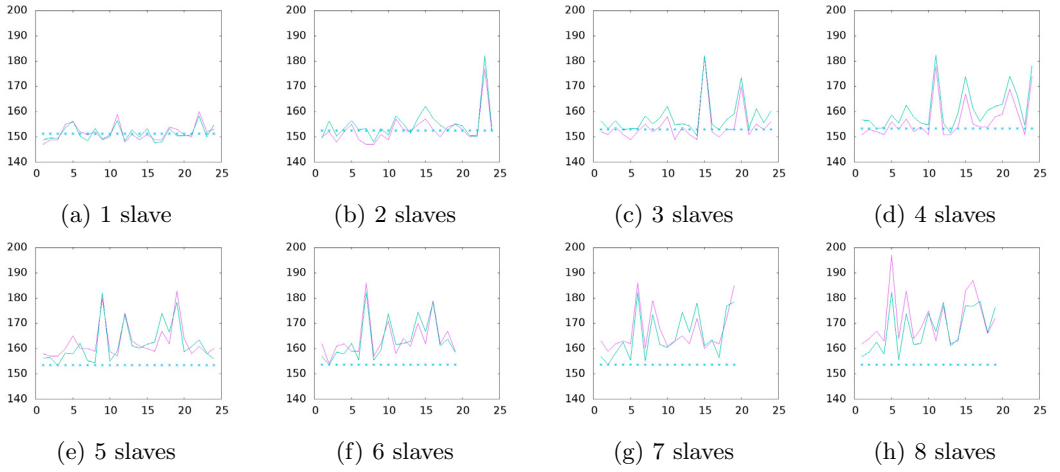
3.2 Simulation Model

Modeling an application using the MERPSYS environment requires writing code in a Java based meta-language reflecting the code of the real application where chosen fragments are replaced by API calls representing *atomic operations*. In our case study the application model consists of two *process implementations*: *master* responsible for orchestrating the training through distributing and averaging the neural network models and ordering data archive numbers, and *slave* responsible for performing the NG-SGD model optimization. The *master* process runs multiple iterations of *p2pCommunicationSend* and *p2pCommunicationReceive atomic operations* representing sending and receiving the model for each instance of the *slave* process. The *master* process controls the number of *slaves* taking part in the computations, gradually increasing it using the same algorithm that the original application. The *slave* process consists of a loop performing *p2pCommunicationReceive* and *p2pCommunicationSend* operations representing receiving and sending the model to the *master* process and a *computation* operation between them. The computational complexity of the *computation* operation is a linear function of the *dataSize atomic operation parameter*, which represents the size in bytes of the processed training data package and depends on the archive number sent by the *master* process. The *space of application parameters* consists of the neural network model size, number of training epochs, frame subsampling factor, number of training data archives and initial and final numbers of slaves. The latter two specify how many *slaves* actually take part in the computations.

Modeling a system in the MERPSYS environment requires building a *hardware graph* using a graphical *Editor* tool and providing functions for *atomic execution time* and *atomic power consumption* of the *atomic operations*. In our case the *hardware graph* consists of 2 workstations connected through a Gigabit Ethernet interconnect, each consisting of 4 GTX TITAN X devices connected with a Xeon CPU through an artificial "CUDA Device To Host" interconnect. The *atomic execution time* is a linear function of the computational complexity divided by performance of the *computing device*. The *atomic power consumption* is the startup time of a *network link* plus a linear function of data size divided by the bandwidth of the *network link*.

3.3 Tuning the Model

Before performing simulations, the MERPSYS environment requires tuning the coefficients used by the *atomic execution time* and *atomic power consumption* functions. In our case study the model has been tuned using results from real executions of the training application on one GPU (Fig. 1a). Execution times of *atomic operations* have been measured and averaged from four runs of 25 consecutive iterations. As shown in Fig. 1 the model tuned on the results from executions on one *slave* is accurate also in case of multiple *slaves* with mean percentage error ranging from 1.5% to 2.7%. The gain from balancing data archives grows with the number of slaves reaching up to 8,25% in case of 8 slaves. The MERPSYS *Simulator* returns also an estimation of total energy consumed by the application execution, as presented in [1]. In this paper we used the same method, but instead of total consumed energy we consider average power consumption calculated as the total consumed energy divided by total execution time.



(a) 1 slave (b) 2 slaves (c) 3 slaves (d) 4 slaves

(e) 5 slaves (f) 6 slaves (g) 7 slaves (h) 8 slaves

Figure 1: Execution times [s] (vertical axis) for a number of consecutive iterations (horizontal axis) depending on the number of *slaves*. Red lines represent the real results, green lines the simulation results and dotted lines represent the simulation results with balanced data archives.

3.4 Visualizing the Trade-off

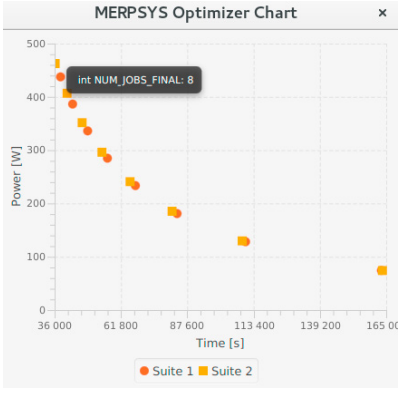


Figure 2: Screenshot from the MERPSYS *ParetoVisualizer*.

In order to visualize the trade-off we first executed a simulation of one training epoch where only one GPU took part in the computations. Then we used the MERPSYS *Web* interface to order an *optimizer suite* based on the aforementioned single *simulation instance*, but we allowed the final number of jobs to vary from 1 to 8 with a step of 1. We started the MERPSYS *Optimizer* component configured to run a *ParetoVisualizer* module. The above steps have been repeated twice: first using real archive sizes and then always using one, average archive size in the application model.

Each *optimizer suite* is passed to the *Optimizer* component, which is equipped with a programming API that allows to iteratively enqueue chosen *simulation instances* from the given range in the *simulation queue* and decide which parameter combinations should be tested next. In our case study simulations in Suite 1 were configured with real archive sizes and Suite 2 with hypothetical ideally balanced archive sizes. The *Optimizer* enqueued all 8 feasible *simulation instances* for each of the suites and passed the results to the *ParetoVisualizer*. A screenshot from the *ParetoVisualizer* is shown in Fig. 2. Hovering with the cursor over a point, the user can see values of those parameters which were varying in the given suite. It should be noted that total time of the simulations run on an Intel Core i7-4712HQ CPU used to draw the chart in Fig. 2 was 2 hours and 10 minutes, while real execution times for the 16 points vary from 10 to 45 hours, giving the total execution time of 335 hours. This shows that the proposed simulation method can predict the execution parameters in significantly shorter time than the proper application execution.

4 Summary and Future Work

The proposed method employing a discrete-event simulation environment allowed to accurately predict *execution time* and *power consumption* of a deep neural network training application and thus, exploring power/time trade-off of its execution. It allowed also estimating how balancing data packages would improve the application performance, giving motivation for further code improvements. Future work includes using the proposed method for other parallel applications and changing parameters. The *Optimizer* component might prove useful also in cases when *exhaustive search* of the parameter space is too computationally intensive. It may be a good environment for developing optimization algorithms for *parameter tuning* and *task mapping*, allowing to iteratively enqueue groups of simulations which can be performed in parallel.

Acknowledgments

The work was partially performed within grant funded by the home faculty of the author based on decision no. MNiD/2016/2017/27. The work has been supported partially by the Polish Ministry of Science and Higher Education. Sincere thanks to the VoiceLab.ai company for providing computing resources and training data for the experiments.

References

[1] Paweł Czarnul, Jarosław Kuchta, Paweł Rościszewski, and Jerzy Proficz. Modeling energy consumption of parallel applications. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *FedCSIS*, pages 855–864, 2016.

[2] Paweł Czarnul, Paweł Rościszewski, Mariusz Matuszek, and Julian Szymanski. Simulation of parallel similarity measure computations for large data sets. In *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)*, pages 472–477, June 2015.

[3] Ryan Friese, Tyler Brinks, Curt Oliver, Howard Jay Siegel, and Anthony A. Maciejewski. Analyzing the trade-offs between minimizing makespan and minimizing energy consumption in a heterogeneous resource allocation problem. In *INFOCOMP, The Second International Conference on Advanced Communications and Computation*, pages 81–89, 2012.

[4] Philipp Gschwandtner, Juan J. Durillo, and Thomas Fahringer. Multi-objective auto-tuning with insieme: Optimization and trade-off analysis for time, energy and resource usage. In *European Conference on Parallel Processing*, pages 87–98. Springer, 2014.

[5] K. Li, X. Tang, and K. Li. Energy-Efficient Stochastic Task Scheduling on Heterogeneous Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 25(11):2867–2876, 2014.

[6] M. A. Oxley, S. Pasricha, A. A. Maciejewski, H. J. Siegel, J. Apodaca, D. Young, L. Briceño, J. Smith, S. Bahirat, B. Khemka, A. Ramirez, and Y. Zou. Makespan and Energy Robust Stochastic Static Resource Allocation of a Bag-of-Tasks to a Heterogeneous Computing System. *IEEE Transactions on Parallel and Distributed Systems*, 26(10):2791–2805, 2015.

[7] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, and others. The Kaldi speech recognition toolkit. In *IEEE 2011 ASRU workshop*. IEEE Signal Processing Society, 2011.

[8] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of deep neural networks with natural gradient and parameter averaging. *CoRR, vol. abs/1410.7455*, 2014.

[9] K. M. Tarplee, R. Friese, A. A. Maciejewski, H. J. Siegel, and E. K. P. Chong. Energy and Makespan Tradeoffs in Heterogeneous Computing Systems using Efficient Linear Programming Techniques. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1633–1646, June 2016.