Conference Poster

Scaling Analysis of Solving Algorithms for Canonical Problem of Dispatching in the Context of Dynamic Programming

Fedosenko, Y.S., Reznikov, M.B., Plekhov, A.S., Chakirov, R. and Houlden, N.

This is a paper presented at the 7th IEEE Int. Conference on Internet Technologies and Applications ITA-17, Wrexham, UK, 12-15 September 2017.

Scaling Analysis of Solving Algorithms for Canonical Problem of Dispatching in the Context of Dynamic Programming

Yuriy S. Fedosenko, Mikhail B. Reznikov
Volga State University of Water Transportation
5 Nesterov Street, Nizhny Novgorod, 603005, Russia

Aleksandr S. Plekhov
Alekseev Nizhny Novgorod State Technical University
24 Minin Street, Nizhny Novgorod, 603155, Russia

Roustiam Chakirov
Bonn-Rhein-Sieg University of Applied Sciences
20 Grantham-Allee, Sankt Augustin, D-53757, Germany

Nigel Houlden
Glyndwr University
Plas Coch, Mold Road, Wrexham, LL11 2AW, UK

***Abstract*—The paper analyses computational model based on dynamic programming for platforms with multicore processors and heterogeneous architectures with FPGA. The models are applied for solving a canonical problem of dispatching where the computation time significantly depends on the problem scale factor. The parallel algorithms of NP-hard problem of dispatching are complicate and require intensive RAM data exchange. In order to reduce the computation time, it is suggested to use FPGA as a coprocessor providing massively parallel computation and increase the operational performance of the system in one order.**

Keywords—massively parallel calculations, dynamic programming, dispatching problem, calculations modeling, discrete optimisation

I. INTRODUCTION

One of the most common method for solving of the discrete optimisation problems is dynamic programming [1]. However, the optimisation problems related to analysis of transportation systems are characterised by *NP*-hardness [2] leading to an exponential increase in computation time regarding problem scale factor. The increase in computational time significantly affects on the operational efficiency of management systems involved in real-time transportation control and planning. This paper analyses computational procedures for solving a canonical problem of dispatching [3] aimed to reduce the time required for solution finding.

II. MATHEMATICAL MODEL

The analysed model comprises of n-elements of determined flow Z of independent objects $z_1, z_2, \ldots, z_n$. Each object z_i (where $i = \overline{1,n}$) is to be a subject of single-stage servicing stationary processor P and characterised by the following integer parameters:

t_i – the moment of arrival (readiness for servicing),
t_i – duration of servicing,
a_i – penalty value for each time unit when the object is not being serviced.

It is assumed that the inequalities $0 \le t_1 \le \ldots t_i \ldots \le t_n$ are fulfilled without loss of generality. In the first moment of time $t = 0$ processor P is free and ready for servicing the objects of the flow Z. The servicing object z_i can be started by free processor P in any moment of time t ($t \ge t_i$) and is executed without interruptions, $i = \overline{1,n}$. The object can not leave the flow being unserved. The simultaneous servicing by processor P of two and more objects is prohibited as well as idle time spending. The flow Z is accepted as serviced in the only one case if all its objects become serviced.

The schedule of servicing ρ of flow Z matches permutation $p = (p(1), p(2), \ldots, p(k), \ldots, p(n))$ of the set of indexes of objects and must be compact, i.e. moment t'_k of servicing start for each object $z_{p(k)}$, $k = \overline{1,n}$ is defined by following equation $t'_1 = t_{p(1)}$, $t'_k = \max\{t'_{k-1} + t_{p(k-1)}, t_{p(k)}\}$. The moment t'_n of flow Z servicing finish is defined as the moment of the last object $z_{p(n)}$ is serviced.

The canonical problem of dispatching consists of searching for schedule p^* which provides minimisation of overall penalty over all objects of flow Z:

$$W(p) = \sum_{k=1}^{n} a_{p(k)} \left(t'_i - t_{p(k)} \right) \to \min \qquad (1)$$

This task can be solved using methods of Dynamic Programming (DP).

Let $W_k^{min}(t, S)$ be the minimal value of overall penalty after servicing of set S by processor P which becomes free in the moment t after object $p(k)$ servicing ($S \in Z$), when k is the serial number of serviced object. In this terms the equations of dynamic programming have the following form:

$$W_k^{\min}(t,S) = \min_{\substack{i=\overline{1..n} \\ z_i \notin S}} \left(W_{k+1}^{\min}(t' + \tau_i, S \cup z_i) + a_i (t' - t_i) \right), \ \ldots,$$
$$W_n^{\min}(t,S) = 0 \qquad (2)$$

978-1-5090-4815-1/17/$31.00 ©2017 IEEE

where the solution of the problem (1) will be taken at the stage $W^{min} = W_0^{min}(0, \varnothing)$.

The expression (t, S) can be considered as servicing system state, since according to Bellman's principle of optimality it matches the minimal possible penalty in case of realisation of any partial schedule p. Same values $W_k^{min}(t, S)$ can be calculated once even if present in states hierarchy multiple times. This treat is used for realization of dynamic programming method either by classical scheme or special scheme with preliminary markup.

The problem (1) belongs to a class of strongly *NP*-hard [2, 3] tasks where the calculation complexity is 2^n. As result, with common single thread realisation of algorithm, the duration θ of optimal schedule p^* solution is exponentially grows up to ten-fifteen minutes even for values $n > 26$.

For many logistics tasks, for example, planning and control in transportation systems, the problem size n can be significantly higher. However, due to technical circumstances a regular time-limit θ for synthesis of the schedule p^* is defined as a value of same order n [4]. Therefore, the effective solution methods and special computing architectures are required to provide fast computation of dispatching tasks under time-limit constrain.

The state of art in solving hard computing problems is parallelisation of works over the array of calculation resources [5, 6]. The effectiveness of such approach is usually estimated with ratio of performance grown in relation to increase of computing resources involved in calculation process.

III. Estimation of Parallelisation Effectiveness of DP Algorithm

Fig. 1a shows a DP algorithm to be estimated in terms of calculation complexity by the number of possible values of the states (t, S) of the system. After limiting integer value t by some maximum moment of time T (discretisation could be selected for desired accuracy) the overall number of states is estimated as $O(T^{2n})$. The calculation of penalty function for each state requires no more than n iterations. The memory required for the calculation is found using similar way. The algorithm flowchart (Fig. 1a) shows that up to n operations of access to memory by random address (RAM) are required for calculation of penalty function in each state (t, S). Step by step from n to 0, the process of calculation of the penalty function for k provides accessibility of all values $W_{k+1}^{min}(t, S)$ on the step k.

In order to provide a parallel algorithm of DP, the computational process is realised in several independent calculation threads. The algorithm is synchronised for independent and sequent calculation of penalty function values for each k step. The graph "operations-operands" of the algorithm is similar to graph of states shown in Fig. 1b. It demonstrates impossibility of algorithm representation as several independent graphs of calculations (threads), i.e. algorithm requires to share memory or intensive data interchange between the threads. Therefore, the performance of parallel DP algorithm will be limited by the memory volume [7], because the increase of number of threads brings the linear increase in the number of memory operations.

For quantitative estimation of effectiveness of the parallel algorithm execution, it was realised using GPU Nvidia GeForce 680 1 GHz with the number of cores 1600 and 4 Gb RAM DDR5 and CUDA [8] with the CPU Intel i5 2 GHz as host. For the parallel resource usage, each core process calculations of W_k^{min} for part of all set of possible states. The values for duration of schedule synthesis are shown in Fig. 2. The synthesis provides calculation of all states by m cores for $T = 32$ и $n = 22$.

The results of experiment have showed that:

- the effect of involved core number grows is present if it is less or equal to 352 which is about quarter of available calculation resources;
- overall benefit of GPU usage in comparison to CPU is less than 2 times while GPU has magnitude of order more cores.

The estimation of algorithm complexity shows that the calculation of W_k^{min} for 268`435`456 states requires n times

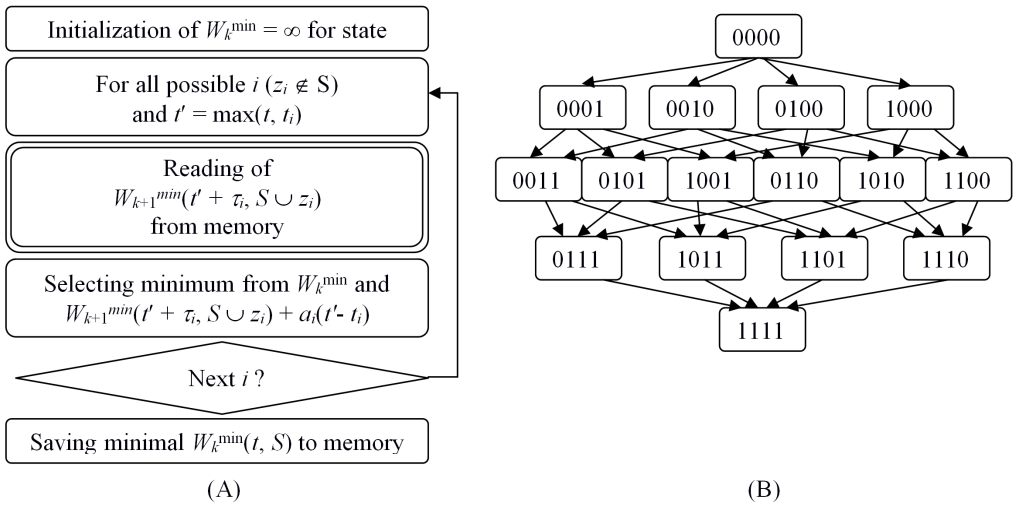

Fig. 1. a) Algorithm of $W_k^{min}(t, S)$ calculation, b) Example of states (t, S) graph for $n = 4$.

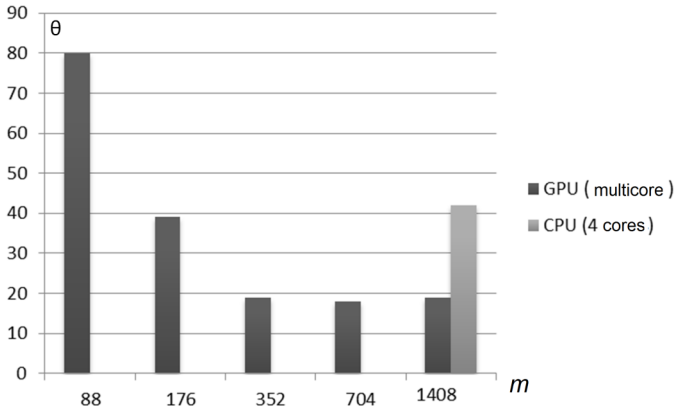

Fig. 2. Duration (sec.) of schedule synthesis by DP algorithm depending on *m*.

more commands to RAM. For the RAM frequency equal to 2 GHz the best estimation of processing time is about 5 seconds. Taking in to account that the standard technologies rely on atomic reading of 64/128 bytes for CPU (256 bytes for GPU) while needs only about 4 bytes, the overhead increases the time of calculations in ten times [9]. Thus, the presence of scaling limit for CPU and GPU technology confirms the assumption about inefficiency of using super multicore processor for realization of DP algorithm.

IV. REALIZATION OF DP ALGORITHM USING FPGA

Let's consider special computational system as alternative to multicore processor. The main requirements for this system are:

- no waiting time for reply on memory commands while calculating minimal value of penalty function over all objects available for servicing;
- ability to work with memory using 8 byte (64 bit) words.

Such calculation platform could be developed using FPGA with connected RAM DIMM module. Calculation process, like in common algorithm, consists of sequence of n stages, each is providing calculation of all correspondent values $W_k^{min}(t, S)$ while k changes from n to 0. The value W_0^{min} of the final stage is minimal possible overall penalty after servicing of all objects of flow Z. For preparing the process, the n arrays formed in RAM (for number of stages), each consists of all possible states (t, S) and uninitialised value of minimal possible penalty W_k^{min} for following calculation. For transportation systems the following dimensions will be enough: t – 8 bit, S – 32 bits, W_k^{min} – 16 bits (overall element size is 8 bytes).

At each calculation stage k all elements $\{(t, S), W_k^{min}\}$ of array sequentially arrives to FPGA. In the process of calculations the new array $\{(t, S), W_{k-1}^{min}\}$ is formed in FPGA

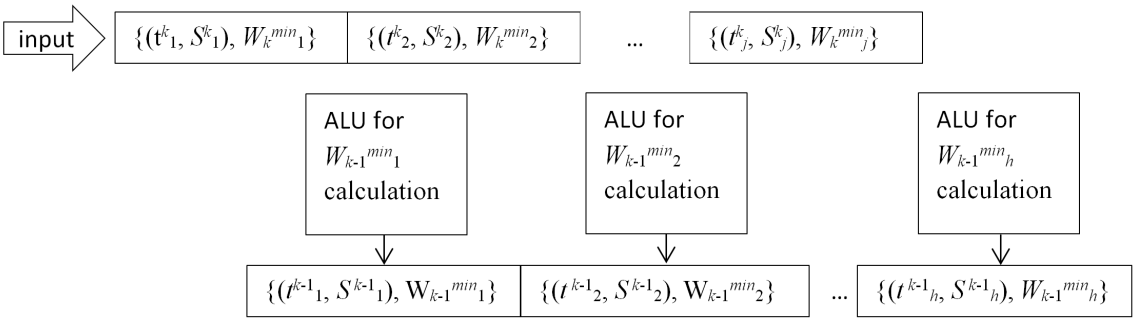

Fig. 3. The principle scheme of realization of *DP* algorithm for FPGA.

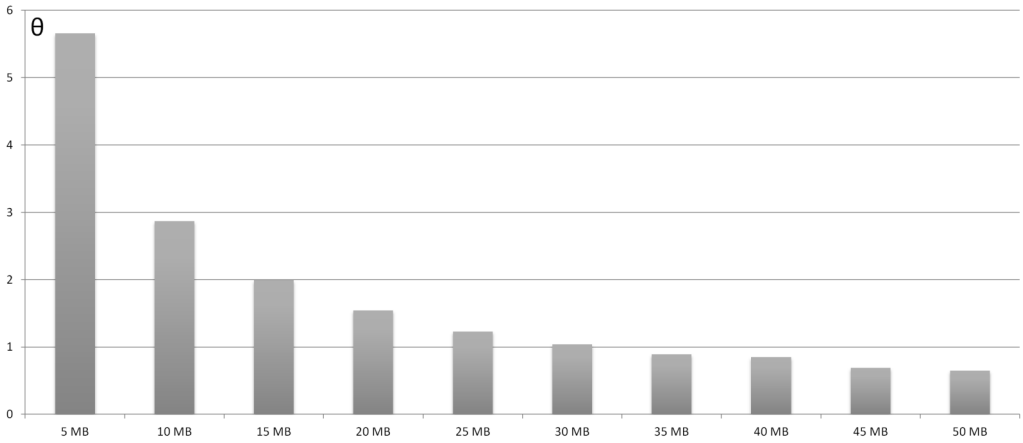

Fig. 4. The relation of synthesis time to the FPGA internal memory size.

and later is uploaded to RAM will be the source for next stage. Thus, as result of iterative process on the final stage in RAM we have last array of one element $\{(0, \varnothing), W_0^{min}\}$ which is required as task solution.

Calculation process shown in Fig. 3 requires usage of internal FPGA memory for saving array of size $T \times C_k^n$ of elements with size 8 bytes.

The work problem considered in this paper has sizes $n = 22$ and $T = 32$. At the stage $k = 11$, FPGA requires internal memory size of 180 Mb. Currently mass production FPGA contains significantly lower amount of internal memory (BRAM). This limitation could be avoided by sequent repeating of one calculation stage for computing different intervals of output array $\{(t, S), W_{k-1}^{min}\}$ where the considered example requires 32 repetitions. FPGA devices of last generation such as Xilinx Ultrascale+ contain significantly more BRAM – from 57 MB to 8 Gb [10].

Fig. 4 shows the results of estimation of synthesis duration for FPGA-based DP algorithm depending on internal memory size. If the system utilises FPGA with the internal memory size equal to 35 Mb it is expected that system performance is increased in one order in comparison to GPU realisation.

V. Conclusion

The realisation of parallel algorithms of computation of *NP*-hard problem of dispatching is extremely complicate due to requirement of intensive RAM data exchange. The increase in processor cores involved in the computational process provides insignificant synthesis time reducing for dynamic programming algorithm. It has been shown that an alternative model based on FPGA as coprocessor to provide massively parallel calculations increase the operational performance of the computational system in one order.

Acknowledgment

This work was financially supported by the Russian Fund for Fundamental Research (Project 15-07-03141) and the Russian Scientific Fund (Project 15-19-10026).

References

[1] S.E. Dreyfus, and R.E. Bellman, *Applied Dynamic Programming*. Princeton, N.J.: Princeton Univ. Press, 1971.

[2] M.R. Garey, and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Co., 2003.

[3] D.I. Kogan, and Yu.S. Fedosenko, "The discretization problem: analysis of computational complexity and polynomially solvable subclasses," *Discrete Mathematics and Applications*, vol. 6, no. 5, pp. 435–447, 1996.

[4] D.I. Kogan, and Yu.S. Fedosenko, "Optimal servicing strategy design problems for stationary objects in a one-dimensional working zone of a processor," *Automation and Remote Control*, vol. 71, no. 10, pp. 2058–2069, Oct. 2010.

[5] S. Lang. (2015). *Parallel Computer Architecture I* [Online]. Available: https://conan2.iwr.uni-heidelberg.de/old-site/teaching/phlr_ws2015/lecture02.pdf.

[6] P.N. Mehra, "Massively parallel processing: Architecture and technologies," in *Handbook of Systems Development*, P.C. Tinnirello, Ed. Boca Raton, FL: Auerbach Publications, 1998, pp. 483–503.

[7] L.M. Silva, and R. Buyya, "Parallel programming models and paradigms," in *High Performance Cluster Computing: Architectures and Systems*, R. Buyya, Ed. Upper Saddle River, NJ: Prentice Hall, 1999, pp. 4–27.

[8] R. Hochberg. (2012). *Dynamic Programming with CUDA. Part I* [Online]. Available: http://www.shodor.org/media/content//petascale/materials/UPModules/dynamicProgrammingPartI/dynProgPt1ModuleDoc.pdf

[9] S. Settle, "High-performance dynamic programming on FPGAs with OpenCL," in *Proc. IEEE 17th Annual Conference on High Performance Extreme Computing*, Waltham, MA, 10-12 Sept. 2013. [Online]. Available: http://ieee-hpec.org/2013/index_htm_files/29-High-performance-Settle-2876089.pdf

[10] Xilinx. (2017). *UltraScale Architecture and Product Data Sheet: Overview*. Preliminary Product Specification [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf