Optimisation of Water Management Systems Using a GPU-Accelerated Differential Evolution

Jiri Jaros
Department of Computer Systems
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno
Czech Republic
Email: jarosjir@fit.vutbr.cz

Jan Marek
Department of Computer Systems
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno
Czech Republic
Email: mrjmarek@gmail.com

Pavel Mensik
Institute of Landscape Water Management
Faculty of Civil Engineering
Brno University of Technology
Veveri 331/95, 662 00 Brno
Czech Republic
Email: mensik.p@fce.vutbr.cz

***Abstract*—The aim of this paper is to present a tool that could optimise operation of multi-reservoir systems to provide the inhabitants and industry with enough water during drought periods. Since the multi-reservoir systems may be quite complicated and the water outflow from reservoirs have to be precise, we developed a GPU-accelerated algorithm based on the differential evolution to solve this task. The experimental results show that a twelve optimal average monthly water outflows from reservoir can be obtained within a minute, almost 18 times faster than on a hex-core CPU.**

I. Introduction

The global warming and subsequent climate change has a high impact on the hydrologic cycle. Changes in the hydrologic cycle in different parts of the world can cause new water management problems that we did not have to solve before. In some areas of the world such changes may lead to the more frequent occurrence of extreme floods, while other parts of the world may be afflicted by the increased incidence of periods of drought. One way of preventing or completely avoiding the appearance of these problems is the re-evaluation of the size of the storage and protective capacity of existing reservoirs [1].

The task, we want to solve, can be summarised as a task of optimal control. We are looking for an optimum water flow via an existing system with a defined structure composed of a river system and a handful of water reservoirs such as dams, ponds, lakes, etc [1]. The goal is to keep the water level in the rivers and reservoirs at a convenient level during the whole period (year). It means we use the strategic control of the multi-reservoir system [2]. We can use the strategy control in the future period if we have predictions of water inflow into the system. The criteria used reflect the fact that in terms of water reservoir operation it is much more convenient to introduce a shallow and long failure in advance than to have even a single but very deep failure (critical failure) when the reservoir is not able to improve the outflow from the reservoir [3].

There are many ways this optimisation problem can be addressed. Basic methods in the field of the control of water management systems are summarised in [4]. The presented methods cover mainly mathematical programming and simulation modelling. The algorithm of seasonal water management reservoir operation by means of the deterministic optimization was derived by the authors of [5]. The algorithm of optimal time water distribution depending on the priorities of water users during the filling of reservoirs in the dispatcher graph of reservoir system was solved by the authors [6]. The general global model for development of the water resource used with indirect optimization was deduced by the authors [7].

The gradual development of computer technology has allowed creation of large and complex water reservoirs system models. For solving these complex systems, it was required to use appropriate modern software products. Unfortunately, none of the existing programmes are designed specifically to solve multi-reservoir system storage capacity. Current programmes use the overcomplicated river network model, which needs much more inputs than it is required by our own simplified solution of multi-reservoir system storage capacity.

In [8] we introduced a novel algorithm that can be used in practice to optimise the water flow in rivers and associated water reservoirs. The algorithm is based on the differential evolution and thus uses a bio-inspired stochastic optimisation method to produce better results than both deterministic and random algorithms. Although the algorithm has reached high quality results, the main bottleneck remains its computational requirements. To accelerate the evolution process, we decided to use modern graphics processing units (GPUs) that offer up to 20 times higher performance compared to classic servers while maintaining low running cost. The description and evaluation of this algorithm is the contribution of this paper.

II. Problem Definition

The goal of this work is the optimisation of the strategic control of multi-reservoir system storage capacity. We will solve systems of real reservoirs whose parameters have to meet several restrictions such as (1) maximum and minimum volume (water level) in the reservoir, (2) maximum and minimum outflow and (3) maximum and minimum water supply taken from the reservoir. The reservoirs have time varying inflows. As an abstraction, let's divide the whole year into a given number of periods (months, weeks or days) and consider the level of inflow/outflow/supply to be constant in these periods. The goal is then to find such outflow and supply levels that do not violate the restrictions in any time period. Let first define a system with a single reservoir, and consequently a system with multiple reservoirs, rivers and river confluences.

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.266

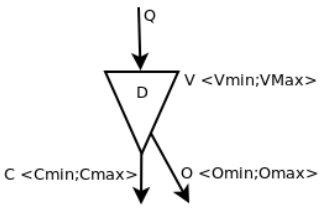

Fig. 1. A schema of a single water reservoir system with a single inflow, outflow and water supply.

A. Water Reservoir Definition

A water reservoir (dam) D can be defined as an n-tuple:

$$D = \{V_0, \tau, Q, Q_\beta, V, V_{min}, V_{max}, \\ O, O_{min}, O_{max}, C, C_{min}, C_{max}\} \quad (1)$$

where

- V_0 is an initial water volume in the reservoir.
- τ is the number of time periods we optimise for.
- $Q = \{Q_0, .., Q_{\tau-1}\}$ is a set of water inflows (from rivers) in given time periods.
- $Q_\beta = \{Q_{\beta_0}, .., Q_{\beta_{\tau-1}}\}$ is a set of water inflows from neighbour water sources (other reservoirs) in given time periods.
- $V = \{V_0, .., V_{\tau-1}\}$ is a set of water volumes in the reservoir in given time periods.
- $V_{min} = \{V_{min_0}, .., V_{min_{\tau-1}}\}$ is a set of min allowed water volumes in given time periods.
- $V_{max} = \{V_{max_0}, .., V_{max_{\tau-1}}\}$ is a set of max allowed water volumes in given time periods.
- $O = \{O_0, .., O_{\tau-1}\}$ is a set of water supplies taken in given time periods.
- $O_{min} = \{O_{min_0}, .., O_{min_{\tau-1}}\}$ is a set of min allowed water supplies in given time periods.
- $O_{max} = \{O_{max_0}, .., O_{max_{\tau-1}}\}$ is a set of max allowed water supplies in given time periods.
- $C = \{C_0, .., C_{\tau-1}\}$ a set of outflows in given time periods.
- $C_{min} = \{C_{min_0}, .., C_{min_{\tau-1}}\}$ a set of min allowed outflows in given time periods.
- $C_{max} = \{C_{max_0}, .., C_{max_{\tau-1}}\}$ a set of max allowed outflows in given time periods.

A scheme of such a single water reservoir problem is depicted in Fig 1. The goal is to find the best values for the pair of parameters $O_t \in O$ (actual water supply) and $C_t \in C$ (actual water outflow). The other parameters are the restrictions for the system. The parameter Q_β does not have any use in this simple situation, however, it will be necessary for systems with multiple reservoirs.

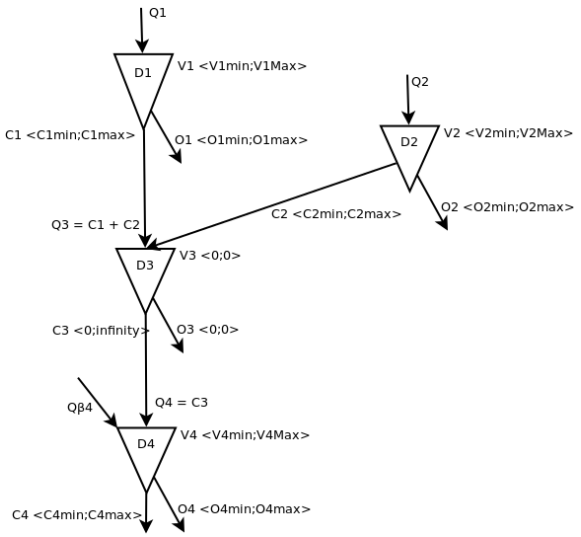

Fig. 2. A schema of a system of four water reservoirs with their inflows, outflows and water supplies.

B. System of Water Reservoirs Definition

We will model a system of water dams using an oriented graph $G = \{V, E\}$ where vertices represent reservoirs and river confluences in the systems while edges do the connections between these reservoirs. For each reservoir

$$\forall v = \{V_0^v, \tau^v, Q^v, Q_\beta^v, V^v, V_{min}^v, V_{max}^v, \\ O^v, O_{min}^v, O_{max}^v, C^v, C_{min}^v, C_{max}^v\} \in V, \quad (2)$$

the following rules hold:

- $\tau^v = c$ where c is an integer constant.
- If $\{\forall x, y | (x, y) \in E\}$ holds that $v \neq y$ then $\{\forall q_b \in Q_\beta^v | q_b = 0\}$ and $\{\forall q \in Q^v | q$ is one of input parameters of the system$\}$.
- Otherwise $\{\forall q_b \in Q_\beta^v | q_b$ is one of the output parameters of the system$\}$ and $\{\forall q_i \in Q^v | q_i$ is a sum of all outflows of reservoirs x such that $\exists (x, y) \in E\}$.

Now, we can model a cascade of water reservoirs. However, in order to be able to describe even more complicated systems with realistic river tries, we have to model river confluences. A river confluence can be modelled as a special reservoir with a zero volume (capacity)

$$N = \{V_0, \tau, Q, Q_\beta, V, V_{min}, V_{max}, \\ O, O_{min}, O_{max}, C, C_{min}, C_{max}\} \quad (3)$$

where $V_0 = 0$ and also V_{min}, V_{max}, O_{min}, O_{max}, C_{min} hold zero in all time periods while C_{max} holds ∞ in all time periods.

Fig. 2 shows a realistic schema of a water reservoirs system. Reservoirs $D1$ and $D2$ represent ordinary water dams while $D3$ represents a river confluence. The reservoir $D4$ is also a water dam, however with two inflows (one from the confluence $D3$ and another side inflow $Q_{\beta 4}$.

III. Optimization Algorithm

The optimisation algorithm is based on the Differential Evolution (DE) [9] and [10]. DE is a stochastic bio-inspired method that optimizes a problem by iteratively trying to improve a candidate solution, referred to as vector, with regard to a given measure of quality. DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require for the optimization problem to be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods.

At each iteration, also called a generation, new vectors are generated by the combination of vectors randomly chosen from the current population (selection). The outcoming vectors are then mixed with a predetermined target vector. This operation is called recombination and produces a trial vector. Finally, the trial vector is accepted for the next generation if and only if it yields a reduction in the value of the objective function. This last operator is referred to as replacement.

In order to be able to employ DE in the optimisation process, a suitable solution encoding and fitness function have to be developed.

A. Design of the Optimisation Algorithm

A candidate solution for a single reservoir problem is represented by a vector of structures, each of which holding the state of the water system in a given time period (outflow, water supply and volume). Although the volume can be calculated for each time period from the previous values of inflow, outflow and water supply, for the sake of performance, it is suitable to store it along with the inflow and outflow. The values of inflow are the simulation parameters that are not stored in the candidate solution.

The algorithm starts with a random set of candidate solutions (initial population). The values of outflow and water supply are generated using a uniform random generator from intervals $\langle C_{min}, C_{max}\rangle$ and $\langle O_{min}, O_{max}\rangle$, respectively. The current water volume is calculated based on the inflow, outflow and supply, however the values are not checked against the limit here - it's up to the evolution process to eliminate solutions that violate any restriction.

The evolution starts by generating trial vectors. With a predefined probability CF, a trial vector for an individual is created based on two parents and itself, and their contents are recombined using the formulae

$$v_t = x_{r_1} + F * (x_{r_2} - x_{r_3}) \tag{4}$$

where x_{r_1} is the actual individual whereas x_{r_2} and x_{r_3} are two distinct parents. F is a parameter allowing for the variance control. Every trial vector is evaluated by the fitness function. The better trial vectors finally replace the original individuals.

The critical part of the algorithm is the fitness function. The fitness function, based on the work by Mensik [11], is divided into two parts. The first part is the penalisation Y which is applied on individuals violating the prescribed restrictions. The penalisation can be tuned for every lower and upper limit of all three restrictions (outflow, supply, volume) by choosing a different penalisation factor PF. The penalisation follows formulas

$$Y_x = \begin{cases} (x - max) * PF & \text{if } x > max \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$Y_x = \begin{cases} (min - x) * PF & \text{if } x < min \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$Y = \sum_{t=0}^{\tau-1} \sum_{a \in \{V,O,C\}} Y_{a,t} \tag{7}$$

The overall value of the penalisation is the sum over all restrictions and time periods, see eq. (7).

The second step of the fitness function is the quality assessment of X. Here, we only work with the outflow and water supply measuring how close we are to the desired values of Z_C and Z_O in all time periods. The precise formulae is shown in eq. (8).

$$X = \sum_{i=0}^{\tau-1} ((C_i - Z_c)^2 + (O_i - Z_o)^2) \tag{8}$$

The final fitness function value is the sum of the quality X and penalisation Y.

The simulation is guided by a list of control parameters:

- Zc is a desired value of outflow.
- Zo is a desired value of water supply.
- $PF_{V_{min}}$ is the low volume penalisation factor.
- $PF_{V_{max}}$ is the high volume penalisation factor.
- $PF_{O_{min}}$ is the low supply penalisation factor.
- $PF_{O_{max}}$ is the high supply penalisation factor.
- $PF_{C_{min}}$ is the low outflow penalisation factor.
- $PF_{C_{max}}$ is the high outflow penalisation factor.

The evolutionary process is controlled by several other parameters:

- NP is population size.
- CF is recombination probability.
- F is variance coefficient.
- i is number of generations.

When solving a system of water reservoirs, we have to extend the list of parameters. First of all, we have to define the number of water reservoirs, second we specify penalisation factors for lower and upper restrictions on all reservoirs. The penalisation function Y and quality function X then read

$$Y = \sum_{v=0}^{D-1} \sum_{t=0}^{\tau^v-1} \sum_{a \in \{V,O,C\}} Y_{a,t}^v \tag{9}$$

$$X = \sum_{v=0}^{D-1} \sum_{i=0}^{\tau^v-1} ((C_i^v - Z_c^v)^2 + (O_i^v - Z_o^v)^2) \tag{10}$$

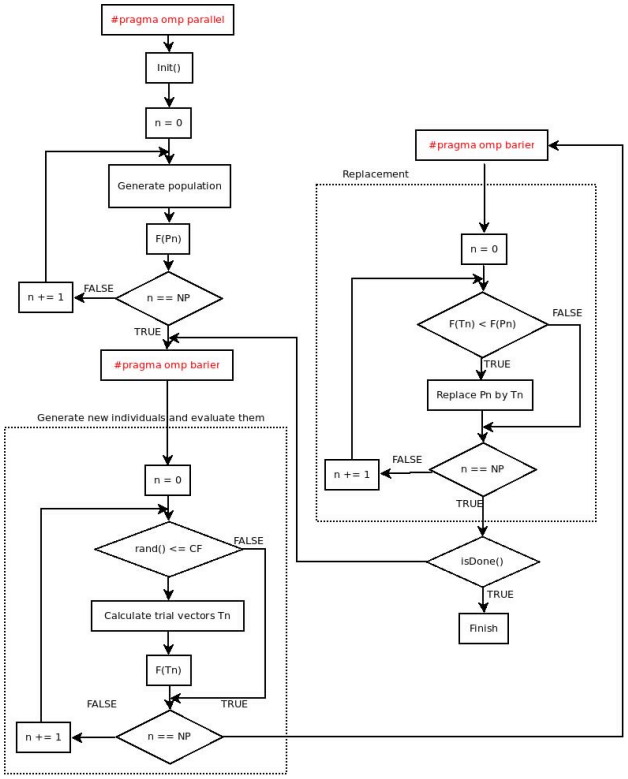

Fig. 3. Implementation of the CPU multi-threaded DE.

B. CPU Implementation of the DE Algorithm

In the paper, we present two different implementations of differential evolution. First, we created a multi-threaded C++ code using OpenMP pragmas[1], and then, we ported the code in the CUDA language[2] to employ current GPUs.

The structure of the CPU code is shown in Fig 3. After launch, a parallel OpenMP region with a predefined number of threads is created. These threads then work almost independently, synchronising only twice per generation at barrier synchronisation points, which yields high parallel efficacy.

The CPU algorithm first initialises the population by random individuals and then waits at a barrier. As soon as all threads are done with the initialisation, the trial vector generation process starts. Each thread receives a portion of individuals to perform genetic manipulation on and evaluate the fitness. As the genetic manipulation only takes place with a given probability and the fitness function evaluation may take a variable time, we used dynamic workload balancing to improve the efficacy. After a barrier synchronisation, the replacement process starts. This process brings better individuals into current population. After the replacement has finished, all threads wait at a barrier and a new generation starts. This process repeats until a given number of generations is evaluated.

Finally, C++ STL vectors maintain populations and the standard C++ random engine generates random numbers.

[1] http://openmp.org/wp/

[2] https://developer.nvidia.com/cuda-zone

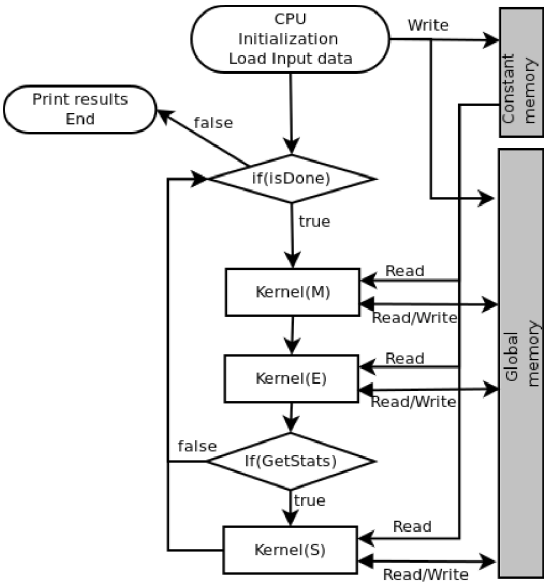

Fig. 4. Implementation of the GPU-accelerated DE in the CUDA language.

C. GPU Implementation of the DE Algorithm

The GPU implementation was realized in the CUDA language and can thus be used with Nvidia GPU cards. The design of the GPU accelerated DE was inspired by work of Qin [12], however, not all principles were applicable in our case. From the CPU flat profile (see sec IV-B1) we had known that the fitness evaluation run time is comparable with the rest of the DE algorithm. From Fig 3, we had also learnt that only two synchronization points are necessary for the whole algorithm. This led to the design of a DE algorithm that entirely runs on the GPU and is structured into only two main kernels. Kernel (M) performs genetic manipulation while kernel (E) carries out evaluation and replacement. In order to collect statistical information about the evolution process, Kernel (S) is occasionally executed. The CPU only helps with reading input files, preprocessing data and controlling the progress of the evolution, see details in Fig 4.

At the beginning, the CPU populates the GPU's constant memory with simulation parameters, restrictions and penalisation factors for a given water system complemented by the parameters necessary for the run of DE. Consequently, several data structures are allocated in the GPU global memory to maintain the population of candidate solutions, the population of trial vectors and some scratch place.

1) Kernel configurations: In CUDA, all kernels (GPU routines) are executed by a bunch of lightweight threads grouped into thread blocks and organised into a grid, which is called kernel configuration. In order to fully exploit GPU potential, tens of thousands of threads must run concurrently. The threads within a single thread block can communicate and collaborate via shared on-chip memory while the communication and synchronisation of thread blocks is not possible (only by stopping the kernel and invoke it again).

Our kernel configuration tries to maximize the amount of work that could be done in parallel by maximizing the number

of threads. The first source of parallelism comes from the individuals which can be processed independently. Moreover, a lot of computation within an individual can be done in parallel for particular water reservoirs (graph vertices) and a subset of operations is also time independent (multiple time periods can be processed concurrently). This altogether allows a creation of up to thousands of CUDA threads even for small systems.

However, both the GPU architecture and the CUDA language introduce a few restrictions on how the threads may be packed and what number of threads can be associated with a thread block. Thus, a simple heuristics to create an appropriate kernel configuration was developed taking into account the current water system being solved. The rules are as follows:

- The number of individuals per block is at least 1.
- An individual cannot be processed by multiple thread blocks due to the use of shared memory.
- The smallest number of threads per block is 32.
- The number of threads cannot exceed the maximum number given by the GPU architecture, usually between 1024 and 2048.

2) Kernel (M) - generation of new individual: This kernel caries out the generation of new trial vectors. The kernel first generates random numbers using the well known Random123 CUDA generator [13] and then applies genetic manipulation according to eq. (4). The random seed is a combination of processor time and the individual index in the population. The random numbers are shared among CUDA threads via shared memory. The advantage is that the kernel can employ all CUDA threads during the generation of new individuals. The performance limitation of this kernel is uncoalesced access to the global memory due to memory accesses controlled by random numbers.

3) Kernel (E) - evaluation and replacement: The kernel starts with the evaluation of current individuals. In order to save memory bandwidth, we do not store the actual water volume in the individual, but instead, calculate them on the fly. Thus, the kernel first calculates the actual water volumes for all water reservoirs at all time periods. Although a significant portion of this calculation (summing up all inflows, outflows and supplies) could be done in parallel, there are data dependencies between subsequent time periods. This means that it is not possible to start the calculation for the time period $t+1$ until the calculation for t has finished. Thus, the level of parallelism per individual is limited by the number of vertices in the graph (number of water dams). In practice, only a subset of CUDA threads is employed while the rest remains idle.

The second part of the kernel is the evaluation of individuals. Here, the level of parallelism is not limited and the number of concurrent operations (exploitable CUDA threads) can scale up to the product of the water dams and the time periods. This means that particular threads calculate penalisation for a single graph vertex at one time period. The partial results are stored in on-chip shared memory and summed up using a custom parallel reduction.

Finally, the replacement is performed. If a generated trial vector is better than the original individual, it replaces the original one.

IV. Experimental Results

This section investigates the quality and performance of the proposed CPU and GPU DE algorithms applied on the multi-reservoir system storage capacity problem. First, we describe the benchmark, the investigated hardware and the parameters of DE. Then, we show the performance compassion of both CPU and GPU implementation on a few different processors and graphics cards. Finally, we show the quality of evolved solutions and compare the implementations against each other.

A. Experimental Setup

1) Benchmark water systems: In order to investigate the the quality of the proposed algorithm, we created two benchmark water systems. This benchmarks were inspired by real water systems located in the south-east part of the Czech Republic.

The water system I, shown on the left hand side in Fig. 5, represents a simplified model with two water reservoirs called Vir (D1) and Brno (D2), and a confluence of two rivers (D3). The water reservoir Vir has also a water supply function while the water reservoir Brno only serves as a protective and recreation water dam. The systems also contains three rivers (Q1, Q2 and Q3).

The water system II, shown on the right hand side in Fig. 5, represents a much complicated water system with four reservoirs Vir (D1), Brno (D2), Letovice (D3) and Boskovice (D4), four river confluences (D4, D6, D7, D8) and several rivers. We can also see that only the water reservoir D1 has the water supply function while the others have the protective and regulatory function only.

2) Parameters of differential evolution: In order to set the suitable parameters of the differential evolution, we performed a deep investigation of the evolution convergence and statistically evaluated multiple parameter configurations over 50 independent runs. However, for the sake of brevity, we are only going to summarize the best parameter setup.

The DE population size NP was investigated on the interval $\langle 100, 50000 \rangle$. The convergence tests showed that a reasonable population size still producing high quality results

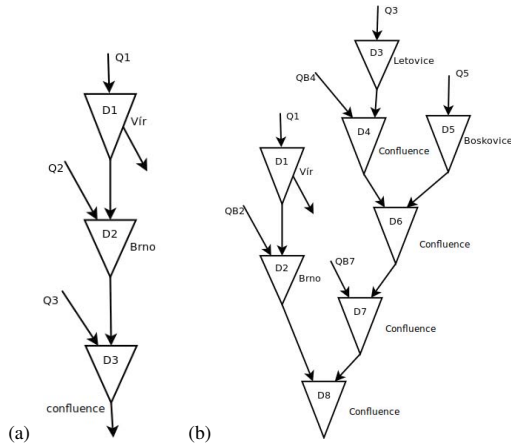

Fig. 5. Optimised water systems (a) Water system I with two reservoirs and a confluence and (b) Water system II with four reservoirs and four confluences.

TABLE I. INVESTIGATED PROCESSORS AND THEIR PARAMETERS.

Name	edesign1	edesign2	edesign3
Model	X5650	X5355	E5-2630
Architecture	Westmere	Clovertown	Sandy Bridge
Cores/Threads	6/12	8/8	6/12
Frequency [GHz]	2.66	2.66	2.3
Performance [GFLOPS]	64	85.2	110
Memory BW [GB/s]	32	44	42.6

TABLE II. INVESTIGATED GRAPHICS CARDS AND THEIR PARAMETERS.

Name	GTX660M	GTX580	Tesla C2070	Tesla K20s
Architecture	Kepler	Fermi	Fermi	Kepler
CUDA Cores	384	512	448	2496
Freqequency [MHz]	835	772	575	706
Performance [GFLOPS]	641	1581	1030	3524
Memory BW [GB/s]	64	192	144	208

while saving computation resources lies close to 1000 and 30,000 individuals for water system I and II, respectively. Higher population sizes do not bring significant improvement, while smaller populations tend to premature convergence.

The probability a trial vector is generated for a given individual CF was investigated on the interval $\langle 0.2, 1.0 \rangle$ with a step size of 0.2. The convergence tests found the most suitable value close to 0.6. Higher values do not bring significant improvements but linearly increase the number of evaluations and the execution time.

The variance coefficient F was investigated on the interval $\langle 0.2, 2.0 \rangle$. The convergence tests showed that for values higher than 1 the quality of individuals rapidly decreases. The quality of the individuals reaches its maximum for F approaching 0.5.

Finally, the necessary number of iterations/generations to reach a suitable solution was investigated. For both water systems, 10,000 iterations proved to be enough time for the algorithm to converge.

3) Hardware definition: Table I and II show the CPU and GPUs hardware investigated in this papers. We used three different servers based on different CPU architectures featuring 4 and 6 cores. Edesign1 and edesign3 represent a single socket hex-core machines while edesign2 is a dual-socket quad-core machine. We can see that the performance in terms of GFLOPS (Billion floating point operations per seconds) varies between 64 and 110 GFLOPS. The memory bandwidth ranges between 32 and 44 GB/s. However, in the case of edesign2, this is the sum of two processors connected by a slow front side bus.

The list of GPU cards investigated in this paper includes a laptop GTX660M, a desktop GTX580 and two server cards Tesla C2070 and Tesla K20s. The performance varies between 641 GLFOPS and 3524 GFLOPS. The memory bandwidth reaches its peak between 64 and 208 GB/s.

Comparing the theoretical parameters of the CPUs and GPUs, we can anticipate the speed-up provided by particular GPUs. Compared to the fastest CPU server (edesign3), the speed-up provided by the GPU version might sit between 6 and 32 int the case the evolution was compute bound, or between 1.5 and 4.9 in the case the memory bandwidth was the bottleneck. The actual results can be seen in sect IV-B3.

B. Performance evaluation

Two water systems presented in Fig. 5 were used for performance comparisons of the CPU and GPU implementations. While developing both codes we profiled on edesing3 (Table III) and GPU GTX580 (Table IV) to find out where the most time is spent. These routines were subsequently tuned and the final comparisons is shown in sections IV-B3 and IV-B2.

1) CPU profile: Table III lists the most significant routines of the proposed multi-threaded differential evolution obtained by the Linux tool `gprof`. From the table, we can see that the genetic manipulation and fitness function evaluation are comparable, 55% vs 42%. This observation suggests that both part of DE must be implemented and optimised with same care. Looking more closely at the fitness function evaluation, we can conclude that the penalisation routine is more time consuming. This is caused by checking the list of restrictions that has to be satisfied. This leads to a lot of branches which are virtually unpredictable. The CPU thus loses significant performance on the branch miss-prediction penalty. On the contrary, the evaluation of how close we are to the desired solution (the level of outflow and supply) is purely mathematical (sum of error squares). This kind of calculation can be performed with high efficacy using the processor vector units (SSE or AVX).

The genetic manipulation process comprises of the trial vector generation (routine Manipulate) and the calculation of the total inflow. Here, the genetic manipulation routine is the dominant part due to random number generation and random memory accesses causing high percentage of cache misses.

Finally, we can conclude that there is not a significant difference between water system I and system II.

2) GPU profile: The GPU profile was taken by the Nvidia visual profiler. Since there are only two CUDA kernels implementing whole DE, we also present detail information on the execution in Table IV. First, we should mention that the kernel (M) took about 41% of the execution time while the kernel (E) took about 59%. Second, let us examine the global memory efficiency that tells how much is the memory bandwidth used. For kernel (M), we reached 81% and 84% efficiency for water system I and II, respectively. These values represent relatively good results, slightly deteriorated by the need of uncoalesced memory accesses during the parent selection. In the case of kernel (E), we can see that the efficiency is even higher than 100% (actually 126% and 131%). How is this possible? It is caused by the presence of L1 and L2 cache that are very helpful when temporal and/or spatial data locality is preserved. This is confirmed by the third row, where the cache hit is almost 88%. On the other hand, kernel (M) has quite a low cache hit, especially for the bigger water system.

TABLE III. THE FLAT PROFILE OF THE CPU VERSION ON EDESIGN3.

	wall clock time[%]	
Routine	System I	System II
Penalisation	38.21	36.66
Evaluation	16.84	15.24
Manipulate	30.9	32.65
TotalInflow	12.03	12.96
Fitness function	55.05	53.22
Genetic manipulation	42.93	45.61
Grand total	97.98	98.83

TABLE IV. THE PERFORMANCE METRICS FOR THE GPU IMPLEMENTATION RUNNING ON NVIDIA GTX580 GPU.

	kernel(M)		kernel(E)	
	System I	System II	System I	System II
Global mem load efficiency	81.60%	83.70%	126.00%	131.00%
Global mem store efficiency	78.60%	86.70%	78.00%	79.20%
L2 hit rare (L1 reads)	56.70%	11.21%	86.80%	88.70%
Achieved occupancy	86.00%	48.80%	93.00%	89.60%
MP activity	96.50%	99.80%	96.80%	99.90%
Branch efficiency	96.80%	90.30%	99.80%	100.00%

The achieved occupancy is a measure of how much parallelism can be exposited by the code while the MP activity shows the how much the CUDA cores are being utilised. In our case, the occupancy is very high. The only exception is the kernel (M) creating individuals for the water system II. Here, a lot of hardware resources are necessary (registers and on-chip shared memory). Looking, at the MP activity we reached over 96% utilisation of the CUDA cores. That signals that the GPU is being kept busy not idling when waiting for memory.

The last very important performance measure is the branch efficiency. Since GPU architectures are based on very long vector units, they are very sensitive to branch divergence (at least two threads taking a different branch of the if statement). Here, we can see that the code achieves a very high branch efficiency above 90 % for the kernel (M), although there are quite a lot decisions depending on random numbers here. The branch efficiency for the kernel (E) is excellent.

3) Execution time: This subsection evaluates the execution time needed to find an acceptable solution and compares the speed-up reached by the GPUs to multi-core CPUs. Fig. 6 shows the execution time for the water system I. First, we can see that the edesign2 is significantly slower than other CPU machines, although the paper values promise a good performance. Of course, we don't have to forget that it is a dual socket system with quite outdated CPUs interconnected with slow front side bus. The newer CPUs based on Westmere and Sandy Bridge architectures outperform the Clavertown by a factor of 2.1.

The investigated GPUs perform significantly better. The fastest is the GTX 580. This GPU was able to produce an acceptable solution 12 times faster than edesing2 and almost 5 times faster than other hex-core servers. The C2070 is a bit slower than GTX580, but the performance drop is proportions to the reduced performance. Interestingly, the laptop GTX660M achieves a very good performance considering its performance, reaching up to 3 times execution time reduction. What was on the other hand a disappointment, was the performance of K20s GPU. It seems that the GPU was not fully utilised by the small benchmark, or tuning the code for Fermi architecture (GTX580 and C2070) had caused ill-optimised code for the more recent Kepler architecture. Although, the Kepler architecture almost triples the number of CUDA cores, the chip clock was slightly decreased and the hot clock (CUDA cores and on chip memory running at double frequency) was removed.

Figure 7 shows the same performance comparison for the water system II. Since the optimisation problem is significantly harder, it offers more opportunities for the parallel calculation. And really, the fastest GPU GTX580 outperforms the slowest CPU by almost a factor of 42 and the fastest CPU by a factor

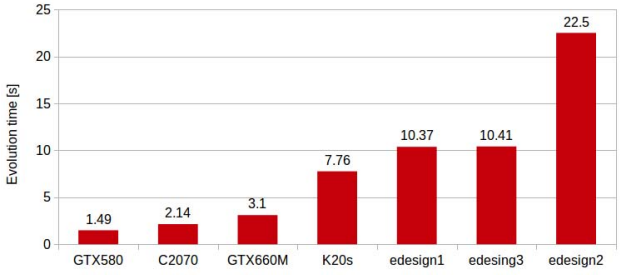

Fig. 6. Execution time of DE for water system I on different architectures.

of 18. This is believed to be an excellent results taking into account that the theoretical maximum acceleration factor of the GTX580 compared to edesing1 is 24. The code thus tends to be compute bound. Interestingly, edesing3 is on the par with the edesing1, although paper parameters would suggest that a newer Sandy Bridge offers almost twice as high performance. The reason is that this performance is delivered by the AVX instruction and our code could not exploit all opportunities to use these vector units.

When we compare the other GPUs we can see that their speed is proportional to the theoretical performance, but the Kepler K20s. Here, we can see a significant improvement in the evolution speed compared to the previous benchmark. It is believed that running even more complicated benchmark would show the true potential of this GPU. For now, we can conclude that the K20 is on par with GTX580.

C. Quality of the evolution process

Finally, we compared the quality of the evolutionary process in the search for an acceptable solution, see Figures 8 and 9. The x-axis represents the iteration and the y-axis represents the fitness value. The plots show the development of the best and average individuals over 10,000 iterations collected and aggregated over 20 independent runs.

Fig. 8 shows that the CPU and GPU implementations behave similarly. Both algorithms converge after about 2000 generations after an optimal solution has been found.

From Fig. 9, we can see two phenomenons. First, both implementations have problems in finding an optimal solution. Although none of the restrictions was violated, the evolved

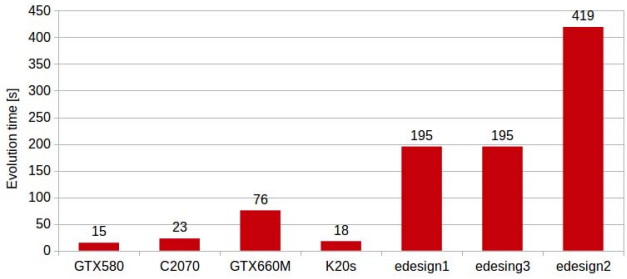

Fig. 7. Execution time of DE for water system II on different architectures.

values of outflows and supplies seem to be far from the desired levels. We have to admit that these values were set to be very restrictive and made the evolution run for a very long time. The second phenomenon deserving an explanation is the different convergence of both algorithm. This can be attributed to different random number generators for CPU and GPU and small deviations in the code for different architectures. The GPU implementation produces significantly better solutions, which is great success considering the evolution time reduction.

V. Conclusions

The paper has presented an innovative algorithm for optimisation of multi-reservoir system storage capacity. Such systems are getting significant importance with the growing population density and climate changes. The proposed algorithm allows us careful strategic control of the multi-reservoir system storage capacity for population and industry supplies during twelve months. In practise, we can use the proposed algorithm in case that we have predictions of monthly inflows into the system. Our algorithm was tested on realistic water systems built on rivers in the Czech republic and shown the ability to plan the outflows and supplies in monthly time steps.

The optimisation algorithm was carefully tuned for current processors and graphics cards. The deployment of GPUs allows us to optimise moderate systems within a minute, almost 18 times faster than a hex-core processor. Moreover, the GPU implementation produces a significantly better solutions (outflow plans).

In the future we would like to test the optimisation algorithm on large water systems and try to strategy control multi-reservoir systems with predictive model of mean monthly inflows into the system. We also could modify the algorithm to the adaptive control for testing of strategic control of multi-reservoir system storage capacity and hydropower in the long time period.

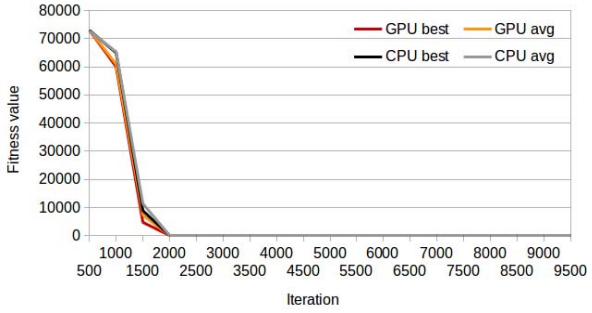

Fig. 8. Solution quality development for the water system I.

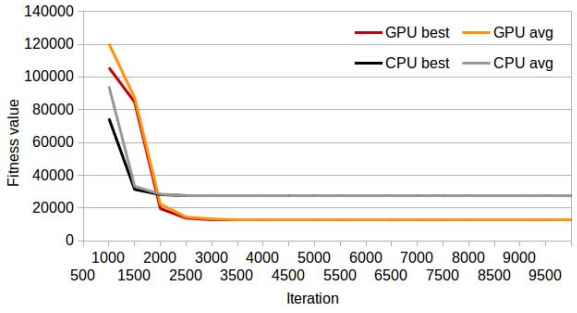

Fig. 9. Solution quality development for the water system II.

Acknowledgment

This work was supported by the research project "Architecture of parallel and embedded computer systems", Brno University of Technology, FIT-S-14-2297, 2014-2016, and the research project "Operation control of storage function of large open reservoirs system by using optimization model", Brno University of Technology, FAST-S-14-2454, 2014-2015.

This work also was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

References

[1] M. Stary, *Nadrze a vodohospodarske soustavy*. Brno University of Tehchnology, Brno Czech republic, 1986.

[2] D. Marton, P. Mensik and S. Stary, Using Predictive Model for Strategic Control of Multi-reservoir System Storage Capacity, *Procedia Engineering*, 2015, vol. 2015, no. 119, pp. 994–1002, ISSN: 1877-7058.

[3] P. Mensik, M. Stary and D. Marton, Using Predictive Models of Mean Monthly Flows for Operative Outflows Control from Large Open Reservoirs, *Proceedings ITISE 2014, International work- conference on Time Series*, Spain, Granada, 2014, ISBN 978-84-15814-97-9, pp. 382–395.

[4] W.W.G. Yeh, *Reservoir management and operations models: a state-of-the-art review*, Water Resour. Res.,1985, vol. 21, no. 12, pp. 17971818.

[5] M. Karamouz and M.H. Houckm, Annual and monthly reservoir operating rules generated by deterministic optimization, *Water Resour. Res.*, 1982, vol. 18, no. 5, pp. 1337–1344.

[6] A.L. Velikanov and J.C. Oziranskij, Compiling perspective water management balances considering water resources management, *Vodn. Resur.*, 1985, no. 2, p. 514.

[7] M.A. Kornjuchin, K.K. Mosevi and A.I. Safonov, Water balance model of the operation regime of combined water management systems, *Vodn. Resur.*, 1985, no. 2, pp. 15–19.

[8] P. Mensik, M. Star and D. Marton, Water Management Software for Controlling the Water Supply Function of Many Reservoirs in a Watershed. *Water Resources*, 2015, vol. 42, no. 1, p. 133–145. ISSN: 0097-8078.

[9] V. Feoktistov, *Differential Evolution: In Search of Solutions*. Springer. 2006, ISBN 978-0-387-36895-5.

[10] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces.*Journal of Global Optimization* 11: 341–359. 1997. doi:10.1023/A:1008202821328.

[11] P. Mensik, *Automatizace reseni zsobn funkce vodohospodarske soustavy*. PhD Thesis, Brno University of technology, Brno, Czech Republic, 2012.

[12] A. K. Qin, F. Raimondo and F. Forbes, An improved CUDA-based implementation of differential evolution on GPU. *2012 GECCO Genetic and evolutionary Compuation Conference*, editace T. Soule; J. H. Moore, ACM, 2012, ISBN 978-1-4503-1177-9, pp. 991–998.

[13] J.K. Salmon, M.A. Moraes and R.O. Dror and D.E. Shaw, Parallel Random Numbers: As Easy as 1, 2, 3, *Proceedings of 2011 International Conference for High Performance Computing*, Networking, Storage and Analysis on - SC11, 2011, pp. 16:1-16:12.