Fundamenta Informaticae 133 (2014) 1–19
DOI 10.3233/FI-2012-998
IOS Press

A Bi-objective Optimization Framework for Heterogeneous CPU/GPU Query Plans

Piotr Przymus[*][†]
Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Chopina 12/18, Torun, Poland
eror@mat.umk.pl

Krzysztof Kaczmarski[†]
Faculty of Mathematics and Information Science
Warsaw University of Technology
Koszykowa 75, 00-662 Warszawa
k.kaczmarski@mini.pw.edu.pl

Krzysztof Stencel[‡]
Institute of Informatics
Warsaw University
Banacha 2, 02-097 Warszawa, Poland
stencel@mimuw.edu.pl

Abstract. Graphics Processing Units (GPU) have significantly more applications than just rendering images. They are also used in general-purpose computing to solve problems that can benefit from massive parallel processing. However, there are tasks that either hardly suit GPU or fit GPU only partially. The latter class is the focus of this paper. We elaborate on hybrid CPU/GPU computation and build optimization methods that seek the equilibrium between these two computation platforms. The method is based on heuristic search for bi-objective Pareto optimal execution plans in presence of multiple concurrent queries. The underlying model mimics the commodity market where devices are producers and queries are consumers. The value of resources of computing devices is controlled by supply-and-demand laws. Our model of the optimization criteria allows finding solutions of problems not yet addressed in heterogeneous query processing. Furthermore, it also offers lower time complexity and higher accuracy than other methods.

Keywords: heterogeneous environment, GPU, CUDA, query processing

[*]The project was partially funded by Marshall of Kuyavian-Pomeranian Voivodeship in Poland with the funds from European Social Fund (EFS) in the form of a PhD scholarships. “Krok w przyszłość – stypendia dla doktorantów V edycja” (Step in the future – PhD scholarships V edition)
[†]The project was partially funded by National Science Centre, decision DEC-2012/07/D/ST6/02483
[‡]Address for correspondence: Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland

1. Introduction

General-Purpose computing on Graphics Processing Units (GPGPU) involves utilization of graphics processing units (GPU) in tasks traditionally handled by Central Processing Units (CPU). In numerous applications, including databases, GPU processor became a full flagged parallel coprocessor, offering massive number of threads with advanced synchronization methods, two level caching, automatic parallel thread scaling and much higher memory bandwidth.

In case of database systems current research focuses on using GPU to support general tasks in database engines [7, 14, 15, 5, 4], like relational query processing, query optimization, database compression and other. Also specialized systems, for example time series databases, may benefit from parallel execution of numerical computations like interpolation, approximation, pattern matching or compression [7, 1, 12, 5, 13, 14].

A utilization of GPU requires transferring data from the CPU memory to the graphical device's memory. This operation in case of big data volumes is time-consuming. It may diminish the gain of the acceleration credited to GPU. In our previous works we showed that this data transfer cost can be reduced using lightweight compression methods [12, 13, 14]. However, this does not solve all the problems since not all tasks may be effectively faster on GPU. In particular, a graphical device is optimized for parallel numerical computations and certain memory access patterns. Another problem are small data sets for which data transfer time may dominate processing and destroy the performance gain. It is clear that only selected operations may benefit when executed on a GPU. Therefore, joint processing capabilities of both CPU and GPU are worth considering. Furthermore, as it is common to have more than one CPU and one GPU in a single machine a potential use of various processing devices should be analysed. Each database query is processed using a query plan created by a dedicated database modules and optimizers. A *heterogeneous query plan* includes processing on heterogeneous (CPU, GPU and possibly other) devices.

The previous research efforts focused on the creation of query plans based on a cost model. This approach finds plans with the best throughput. However, it does not allow modelling all phenomena that can occur in heterogeneous systems. Performing a query as soon as possible is not always cost effective [8]. For this reason, we propose a query processing model based on concepts of markets that are known to be suitable for describing the interactions in a heterogeneous world. They have already gained a considerable interest in the context of task processing in heterogeneous systems [6]. In market models, manufacturers (processing devices) compete with each other for customers (query plans). Similar competition occurs among customers.

In this paper, we propose a query optimization model dedicated to heterogeneous processing based on the commodity market. A query plan is bi-objectively optimized to minimize both processing time and cost (a value of consumed resources). A cost may be, e.g. the responsiveness of the system, power consumption, heat production, etc. For other purposes, one may also consider expressing it in financial terms.

This work, an extended version of [15], is organised follows. Section 2 rolls out the problem space. Section 3 presents a bi-objective optimization framework for heterogeneous CPU/GPU query plans. Section 4 discusses experimental evaluation of the proposed framework. Section 5 addresses the related work. Section 6 concludes.

2. Preliminaries

2.1. GPU And Heterogeneous Query Processing

From the point of view of parallel processing, CPU accompanied by a GPU coprocessor is a *shared nothing architecture*. A GPU card has its own memory or a separate area in the CPU memory space. The data has to be explicitly transferred between the CPU and GPU main memory. Similarly, the results produced by GPU have to be transferred back to the CPU main memory. This data transfer often introduces significant overhead and may require some tasks to be run on a CPU. Therefore these costs, as components of the execution time, must be included in the prediction of the total time of an operation.

Contemporary computer systems often include more than one GPU. It is possible to combine multiple computational units in a single query plan. Each device may have a different communication cost (e.g. PCIe or shared memory) with the CPU main memory. Furthermore, devices can often communicate directly between each other. Thus, the main problem of heterogeneous query processing is the construction of such a query plan that uses only computational units from which query performance will benefit most and yet will minimize the resources used.

Bress et. al. [5] identified problems of hybrid (CPU/GPU) query processing that also concern the heterogeneous query processing:

Problem 1 Execution Time Prediction - as multiple database operations may be executed concurrently it is hard to predict influence of concurrent tasks on execution times.

Problem 2 Critical Query - since the GPU memory, the concurrent GPU kernels execution and the PCIe bus bandwidth are all limited, only the *critical queries* should be selected to use GPU (i.e., queries that benefit from GPU usage and are *important* from global perspective).

Problem 3 Optimization Impact - as concurrent heterogeneous queries will influence each other, it is important to consider this aspect in the planning process.

Problem 4 Copy Serialization Challenge – GPU does not allow concurrent copy operations in the same direction. Hence, such copy operations are serialized. As a result the utilization of the PCIe bus may be significantly reduced. A solution to this consists in combining multiple data streams into one copy operation and then reorganizing the transferred data in the GPU memory [3]. Another solution involves usage of lightweight data compression methods where copy operations may be concurrently mixed with decompression operations [13, 14].

2.2. Commodity Market Approach In Query Processing Context

In this paper we address the problems enumerated in the previous section by showing that they may be solved by applying a supply-and-demand pricing model taken from commodity markets. Economic models are known to be appropriate for describing the heterogeneous environments and already gained a considerable interest in the context of heterogeneous grid computing systems [6, 16], where they were used for scheduling and resource management. This concept has also reached the field of distributed database systems, and resulted in several interesting prototypes [17, 11, 6].

Supply and demand pricing model is an economic model of price determination in competitive markets. In such a market, resource owners (processing devices) price their assets and charge their customers (queries) for consumed resources (see [6] for other pricing models that may be used). In this model when supply (available resources) or demand (needed resources) change, prices will be changed until an equilibrium between supply and demand is reached. Typically the value of a resource is influenced by: its strength, physical cost, service overhead, demand and preferences [6]. A consumer may be charged for various resources like CPU cycles, memory used, the bus or network usage. Usually, a broker mediates between the resource owners and the consumer. The resource owners announce their valuation and the resource quality information (e.g. estimated time) in response to the broker's enquiry. Then, the broker selects resources that meet the consumer utility function and objectives, like cost and estimated time constraints or minimization of one of the objectives.

2.3. Bi-objective Optimization

Bi-objective optimization is a special case of multiple criteria decision making. The simplified version of general multi-objective optimization problem, may be defined as[1]:

$$\text{optimize } F(x) = \big(F_1(x), F_2(x), \dots, F_k(x)\big),$$

where $k \geq 1$ is the number of objective functions, $x \in \mathbb{R}^n$ is a vector of design variables, n is the number of independent variables x_i, and $F(x) \in \mathbb{R}^k$ is a vector given by the values of objective functions $F_i \colon \mathbb{R}^n \to \mathbb{R}$.

Usually there are no solutions that meets all objectives. Thus, a definition of an optimum solution set should be established. In this paper, we use the predominant Pareto optimality [9]. A point $x^* \in R^n$ is called *Pareto optimal* if there does not exist another point $x \in R^n$ such that $F(x) \leq F(x^*)$ and $F_i(x) < F_i(x^*)$ for at least one function. Note that, for some problems, there may be an infinite number of Pareto optimal points. Given a set of choices and a way of valuing them, the *Pareto set* consists of choices that are Pareto optimal.

In numerous computational problems it is sufficient to find the most preferred Pareto optimal solution according to subjective preferences. These preferences are usually given by a decision maker. This problem may be solved using a number of different approaches. Two of them are worth mentioning. They are *a priori* methods and methods with no articulation of preferences (examples of such methods may be found in [9]). In *a priori* methods the preferences are specified at the beginning by the decision maker and are used to find the best solution, while in *methods with no articulation of preferences* a neutral compromise solution is identified without preference information.

A special case of multi-objective decision making is a *bi-objective optimization*, where optimal decisions need to be taken in the presence of trade-offs between two conflicting objectives.

The concept of multi-objective optimization has applications in disparate engineering and computational problems [9]. One of the them is the multi-objective query plan optimization [17, 11]. As this problem is NP-hard (it reduces to the knapsack problem), it is often necessary to propose an approximate solution [17, 11].

[1]A more general and complete definition may be found in [9]

3. A Bi-objective Heterogeneous Query Planer Framework

The main aim of the new query planner is to propose a solution to the problems listed in Section 2.1, i.e., Execution Time Prediction, Critical Query and Optimization Impact. Furthermore, this planner also addresses heterogeneous GPU cards and distributed processing. In this paper we propose a method to build heterogeneous query plans based on the economics of commodity markets. It is characterized by the fact that the resource producers determine the cost of their resources and resource consumers jostle for resources. Furthermore, the resources owners provide information on the quality of their resources, i.e., the estimated processing time.

3.1. Notation

Table 1 contains a summary of the notation used in this paper. Assume a set of units U, a logical query sequence QS_{log} and a dataset D. The goal is to build a heterogeneous query sequence. Let QS_{het} be a heterogeneous query sequence defined in Equation (1). Let D_{k+1} be the data returned by an operation $A^{o_k}_{u_{i_k}}(D_k)$. The first row of QS_{het} is created by replacing each operation $o_i \in QS_{log}$ with an algorithm $A^{o_i}_{u_j} \in AP_{o_i}$. The second row is created by inserting an operation $M_{u_k,u_{k'}}(D)$ that copies the output of an algorithm on the unit u to the input of the current algorithm on the unit u'.

Table 1: Symbols used in the definition of our optimization model

Symbol	**Description**
$U = \{u_1, u_2, \dots u_n\}$	set of computational units available to process data
D	dataset
$M_{u_k,u_{k'}}(D)$	**if** $u_k \neq u_{k'}$ move D from u_k to $u_{k'}$ **else** pass
$o_i \in O$	database operation o_i from set of operations O
$A^{o_i}_{u_k}$	algorithm that computes the operation o_i on u_k
$AP_{o_i} = \{A^{o_i}_{u_1}, A^{o_i}_{u_2}, \dots, A^{o_i}_{u_n}\}$	algorithm pool for the operation o_i
$t_{run}(A^{o_i}_{u_j}, D)$	estimated run time of the algorithm $A^{o_i}_{u_j}$ on the data D
$t_{copy}(M_{u_i,u_j}, D))$	estimated copy time of the data D from u_i to u_j
$c_{run}(A^{o_i}_{u_j}, D)$	estimated run cost of the algorithm $A^{o_i}_{u_j}$ on the data D
$c_{copy}(M_{u_i,u_j}, D)$	estimated copy cost of the data D from u_i to u_j
$QS_{log} = o_1 o_2 \dots o_n$	logical query sequence
QS_{het}	heterogeneous query sequence see Equation 1
f_t, g_t	estimated algorithm run time and copy time see Section 3.3
f_c, g_c	estimated algorithm run cost and copy cost see Section 3.4
f_b, g_b	estimated algorithm run and copy bi-objective scalarization see Section 3.5
$F_x(QS_{het})$	sum of f_x and g_x over columns of QS_{het} where $x \in \{t, c, b\}$ see Equation 2

$$QS_{het} = \begin{pmatrix} A^{o_1}_{u_{i_1}}(D_1), & A^{o_2}_{u_{i_2}}(D_2), & A^{o_3}_{u_{i_3}}(D_3), & \dots, & A^{o_n}_{u_{in}}(D_n) \\ M_{u^*,u_{i_1}}(D_1), & M_{u_{i_1},u_{i_2}}(D_2), & M_{u_{i_2},u_{i_3}}(D_3), & \dots, & M_{u_{in},u^*}(D_n) \end{pmatrix} \tag{1}$$

$$F_x(QS_{het}) = \sum_{A^o_u(D)} f_x(A^o_u, D) + \sum_{M_{u',u''}(D)} g_x(M_{u',u''}, D) \tag{2}$$

3.2. Single Objective Heterogeneous Query Planer

```
Procedure OptimalSeq($QS_{log}, u^*, x$)
Input: $QS_{log} = o_1 o_2 o_3 \dots o_n$ - logical query sequence, $u^*$ - base unit,
$x \in \{$ t - time, c - cost, b - bi-optimization $\}$ - optimization type
Result: $QS_{hybrid}$
1  seq_list = [];
2  for u in U do
3      $Q_u = S_u(QS_{log}, u^*)$;
4      $QF_u = F_x(Q_u)$ ;                                          /* e.g.  $F_t(Q_u)$ */
5      append $(u, Q_u, QF_u)$ to seq_list;
6  end
7  $QS_{hybrid}$ = pop minimum $Q_u$ (by $QF_u$) sequence from seq_list;
8  for ($u$, $Q_u$, $QF_u$) in seq_list do
9      E, F, G = DiffSeq($QS_{hybrid}, Q_u, u, x$);
10     val, start, end = MaxSubseq(E, F, G);
11     if val > 0 then
12         $Q_{hybrid}(start : end) = Q_u(start : end)$ ;         /* subarray subsitution */
13     end
14 end
15 return $Q_{base}$
```

In this section, we introduce the algorithm that searches for a heterogeneous query plan, i.e., a plan that operates on more than two devices.

Let $S_u(QS_{log}, u^*)$ return such QS_{het} that each operation $o_i \in QS_{log}$ is replaced with an algorithm from unit u algorithm pool, i.e., $A^{o_i}_u \in AP_{o_i}$ and the base device is set to u^*. Note that there is a specially designated computing unit u^* from which the processing starts. It also collects the data in the end of processing, since the GPU computing is controlled by a CPU side program.

The algorithm OptimalSeq starts by creating a query sequence for each computing unit and estimating the processing cost for each item of this sequences (lines 2-6). Next, one sequence (which minimizes $F_x(Q_u)$) is selected as the base sequence. It will be improved in later steps (line 7). Then, the algorithm iterates over remaining query sequences in QS_{hybrid} in order to find such segments in the remaining query sequences which improve original sequence (by replacing corresponding segment of the original sequence). This is done by calculating the improvement and copy cost arrays in DiffSeq and finding

```
Procedure DiffSeq($seq_{base}, seq_u, u, x$)
  Input: $seq_{base}$ - base sequence, $seq_u$ - unit query sequence, $u$ - $seq_u$ unit,
  $x \in \{$ t - time, c - cost, b - bi-optimization $\}$ - optimization type
  Result: E - operations improvement array; F, G - copy to/from unit arrays
1 E, F, G = [], [], [];
2 for i in enumerate columns $seq_{base}$ do
3     $A^o_{u_b}(D_i), M_{u_f,u_t}(D_i) = seq_{base}[i]$;
4     $A^o_u(D_i), M_{u,u}(D_i) = seq_u[i]$;
5     append $f_x(A^o_{u_b}, D_i) - f_x(A^o_u, D_i)$ to E ;        /* e.g. $f_t(A^o_u, D)$ */
6     append $g_x(M_{u_f,u}, D_i)$ to F ;                        /* e.g. $g_t(M_{u,u'}, D)$ */
7     append $g_x(M_{u,u_t}, D_i)$ to G;
8 end
9 return E,F,G
```

maximal sequence segment in MaxSubseq. A following variant of the proposed algorithm should also be considered. Suppose that only one query sequence segment may be inserted (i.e., choose one sequence segment from remaining $k-1$ sequences with the biggest), this minimizes number of involved computational units and reduces overall communication costs.

The procedure DiffSeq simply calculates element wise difference between two query sequences and copy costs from/to unit, that is:

$$f_x(A^o_{u_b}, D_i) - f_x(A^o_u, D_i) \tag{3}$$

$$g_x(M_{u,u'}, D). \tag{4}$$

The procedure MaxSubseq is based on Kadane's algorithm for maximum subarray problem [2]. It scans through the improvement array, computing at each position the maximum subsequence ending at this position. This subsequence is either empty or consists of one more element than the maximum subsequence ending at the previous position. Additionally, the *copy to* and *copy from* costs are included in the calculation of the maximum subsequence (F and G arrays). The algorithm returns the maximum improvement for a subsequence (which may be zero if the subsequence does not improve the original query), the start and end items of subsequence.

The complexity of OptimalSeq is $O(k*n)$ where k is the number of devices (usually small) and n is the number of operations of the sequence. S_u,F_x, DiffSeq and MaxSubseq have the complexity $O(n)$.

3.3. Time Objectives

Let f_t and g_t be the estimated running time and copying time of an algorithm defined as

$$f_t(A^o_u, D_i) := t_{run}(A^o_u, D_i), \tag{5}$$

$$g_t(M_{u,u'}, D) := t_{copy}(M_{u,u'}, D) \tag{6}$$

where $t_{run}(A^{o_i}_{u_j}, D)$ is the estimated running time of the algorithm $A^{o_i}_{u_j}$ on the data D and $t_{copy}(M_{u_i,u_j}, D)$ is the estimated copying time of the data D from u_i to u_j. Both t_{run} and t_{copy} are interpolated from sample measurements taken in perfect conditions (i.e. with the exclusive access to devices). Later in this

```
Procedure MaxSubseq(E, F, G)
Input: E - operations improvement array; F, G - copy to/from unit arrays
Result: maximum_improvment, start, end
max_ending_here = max_so_far = 0 ;
begin = tbegin = end = 0 ;
for i, x in enumerate E do
    max_ending_here = max(0, max_ending_here + x) ;
    if max_ending_here = 0 then
        tbegin = i ;
        max_ending_here -= F[i];
    end
    if max_ending_here - G[i] >= max_so_far then
        begin = tbegin ;
        end = i ;
        max_so_far = max_ending_here ;
    end
end
return max_so_far - G[end], begin, end
```

article we will define functions f_c, g_c and f_b, g_b to fit the model of the commodity market and the bi-objective optimization.

3.4. Economics In Heterogeneous Environment And Construction Of Cost Objective

To cope with the problems mentioned in Section 2.1, additional criteria are necessary – in this work an approach based on a simple economic model is proposed. Each consumer (client) has a query budget that can be used to pay for the resources used to process queries. Each computational unit is a service provider (producer) of services available in units algorithm pool AP_{u_i}. Each service provider establishes its own pricing for execution of any service from AP_{u_i}. Pricing of the service depends on:

- the estimation of needed resources (the size of the data D, the performance of the task $A_{u_i}^{o_i}$),
- the pricing of needed resources (the load of device u_i – the greater the load on the device, the higher cost of using the device),
- the preference of the device (e.g. device may prefer larger jobs and/or tasks that give a greater acceleration on the GPU).

First, the pricing for using the resources of a computational unit is established. This depends on the previous load of the device: the higher demand for a computational unit, the higher price for using it. This is a periodic process which calculates prices every Δt_{up} seconds by calculating the price P_u of a unit price u. Let $0 < L_{curr} < 1$, $0 < L_{prev} < 1$ be the current and previous load factors of a computational unit. Additionally, let L_{th} be a threshold below which prices should decrease, and P_{min}

be the minimal price. Then, the new price P'_u is calculated using the following formula:

$$P'_u := \begin{cases} max(P_{min}, P_u \cdot (1 + \frac{\Delta P_u}{(1-\Delta U)}) & \text{if } (\Delta P > 0 \wedge \Delta U > 0) \vee (\Delta P < 0), \\ P_u & \text{otherwise,} \end{cases} \tag{7}$$

where $\Delta P_u = L_{current} - L_{threshold}$ and $\Delta U_u = L_{current} - L_{Previous}$. This is similar to the dynamic pricing model proposed in [16] with an exception to the pricing formula. We use $max(P_{min}, P_u \cdot (1 + \frac{\Delta P_u}{(1-\Delta U)})$ instead of $max(P_{min}, P_u \cdot (1 + \Delta P_u))$ This modification reduces the excessive growth of prices.

To reflect the preference of the device in the price we need to define a function returning the speedup factor between the base device u^* (defined in the previous section) and the current device:

$$speedup(A^o_u, D_i) := \frac{t_{run}(A^o_{u^*}, D_i)}{t_{run}(A^o_u, D_i)}. \tag{8}$$

Then we define the cost function as

$$c_{run}(A^o_u, D_i) := \frac{\#D_i}{speedup(A^o_u, D_i)} \cdot P_u, \tag{9}$$

where the part $\frac{\#D_i}{speedup(A^o_u, D_i)}$ combines the estimation of needed resources and the preference of the device. A computational unit with a high speedup on a given operation will get a discount per data size when pricing this operation. Similarly, operations with a lower speedup factor will be charged more per quantity. Additionally it is observed [13] that the speedup often depends on the size of the processed data (usually a low speed-up on small datasets) so the discount depends on the data size.

It is also important to include the cost of the data transfer, let us define it as

$$c_{copy}(M_{u,u'}, D) := \begin{cases} 0 & \text{if u,u' share memory} \\ \frac{\#D_i}{bandwidth(\#D_i, u, u')} \cdot (P_u + P_{u'})/2 & \text{otherwise} \end{cases} \tag{10}$$

where $bandwidth$ returns the estimated byte count per second between the computational units u and u'. If the direct data transfer is not available between the devices u and u', then a transit device will be used (e.q. two GPU cards without direct memory access will communicate using CPU RAM).

Now let us define the estimated run cost and copy cost functions of an algorithm,

$$f_c(A^o_u, D_i) := c_{run}(A^o_u, D_i), \tag{11}$$

$$g_c(M_{u,u'}, D) := c_{copy}(M_{u,u'}, D). \tag{12}$$

A solution minimizing the cost of the query plan may be found under the previous assumptions and using the procedure OptimalSeq with the optimization type set to cost (i.e. $x = c$).

3.5. Construction Of Bi-objective Optimization

As finding Pareto optimal bi-objective query plan is NP-hard (it is a bi-objective shortest path problem) [11], we will use the previously described OptimalSeq single objective approximation algorithm and extend it to the bi-objective case.

We will use *a priori* articulation of the preference approach which is often applied to multi-objective optimization problems. It may be realized as a scalarization of objectives, i.e., all objective functions are combined to form a single function. In this work we will use the *weighted product* method, where weights express the user preference [9]. Let us define:

$$f_b(A_u^o, D) := c_{run}(A_u^o, D)^{w_c} \cdot t_{run}(A_u^o, D)^{w_t}, \tag{13}$$

$$g_b(M_{u,u'}, D) := c_{copy}(M_{u,u'}, D)^{w_c} \cdot t_{copy}(M_{u,u'}, D)^{w_t}. \tag{14}$$

where w_t and w_c are weights which reflect the relative importance of the cost and the time. The bigger the weight is, the more important is the feature. Values of f_b, g_b are higher than 1. It is worth mentioning that a special case with $w_t = w_c = 1$ (i.e., without any preferences) is equivalent to the Nash arbitration method (or the objective product method) [9].

3.6. Framework Summary

The notable flexibility of the presented framework allows for its utilization in different scenarios. In cases, when all queries are equally important the *heterogeneous query planner with bi-objective optimization* should be used. If there is a diversity of expectations for the query optimization (e.g. there is some prioritization of queries) an approach of cooperative usage of all three optimization objectives (Time, Cost and Bi-objective) should be applied. High priority queries should be optimized using the *Time objective*. Low priority queries should be optimized using the *Cost objective*. All other queries should be optimized using the *double (bi-) objective*. Such approach will be called *heterogeneous query planner with mixed objective optimization*. It is also possible to create a tunable query prioritization model based on weights in the bi-objective function. It is planned as our future work.

4. Experimental Results

4.1. The Setting

In order to evaluate the proposed framework we prepared a *proof-of-concept* implementation and evaluated it using a custom developed simulation environment. The simulation environment was developed using Python and the SimPy framework [10]. We defined four devices in the experimental environment: CPU_1, CPU_2, GPU_1 and GPU_2. The bandwidth of the data transfer between $CPU_i \leftrightarrow GPU_k$ was measured on a real system. The bandwidth of $GPU_1 \leftrightarrow GPU_2$ was calculated using CPU_1 as the transit device (i.e. $GPU_1 \leftrightarrow CPU_1 \leftrightarrow GPU_2$). We used the following weights of the bi-objective scalarization: $w_t = w_c = 1$ (i.e., without any preferences setting). Other settings of the simulation environment are gathered in Tables 2a and 2b. The simulation environment generates new query sequences, when a spawn event occurs. A spawn event is generated randomly in a fixed interval. It produces a random set of query sequences (with a fixed maximum). Every query sequence consists of up to six operations and operates on a random data volume. In the simulation the processed data size has a direct (linear) influence on the speed of processing. Each device has a limited number of resources. A database operation can be performed only if needed resources are available. In other cases the operation waits. After generating the desired number of query sequences the simulation stops spawning of new tasks and waits until all generated query sequences are processed.

Table 2: The configuration of the simulation environment

Device	o_1	o_2	o_3	o_4	o_5	o_6
GPU_1	20	11	6	0.38	14	15
GPU_2	5	11	6	0.33	4.66	5
CPU_2	1	1	1.09	1	1.27	1.36
CPU_1	1	1	1	1	1	1

(a) The average speedup of the operation o_i on a given device compared to CPU_1

Option	CPU_1	CPU_2	GPU_1	GPU_2
Unit threshold	0.75	0.75	0.4	0.4
Unit minimal price	5	5	70	70

(b) The configuration of the pricing model

4.2. The Results

4.2.1. A Comparative Evaluation Of Query Planners

Figure 1 presents simulated execution times of five scheduling frameworks processing a pool of generated query sequences. Every scheduling framework processes exactly the same pool. After every 5000 ticks we count all processed tasks in each framework. This experiment was repeated 100 times. The presented plot is an average of all runs.

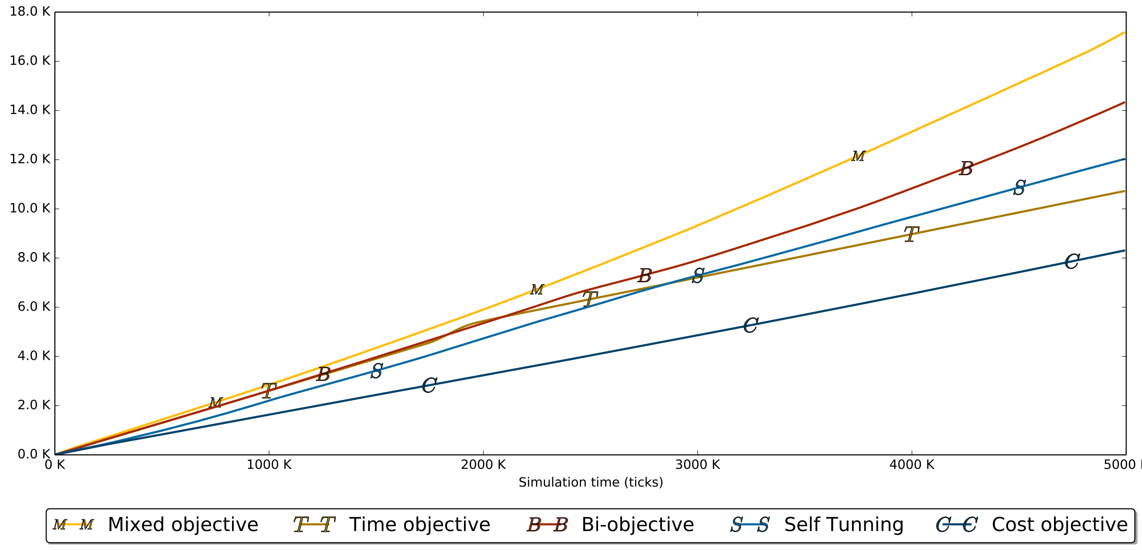


Figure 1: The simulated efficiency of the heterogeneous query planer with various optimization tasks (time, cost, bi-objective and mixed) and the self-tuning query planner

Compared frameworks are based on the algorithm OptimalSeq but use different objective function:

- *Mixed objective* – The heterogeneous query planer with the mixed objective optimization (see Section 3.6). To each generated query sequence an optimization criterion (time, cost or bi-objective) was assigned with equal probability $1/3$.

- *Time objective* – The heterogeneous query planer with the time objective optimization (see Section 3.3). It uses only f_t and g_t functions as optimization criteria; this means that it has no idea on the load of each of devices.

- *Cost objective* – The heterogeneous query planer with cost objective optimization (see Section 3.4). It uses only f_c and g_c functions as optimization criteria; this means that it deliberately chooses the cheapest devices to run the query.

- *Bi-objective* – The heterogeneous query planer with bi-objective optimization (see Section 3.5). It uses f_b and g_b functions as optimization criteria; this means that it always optimizes the trade-off between the cost and time execution.

- *Self Tuning* – is based on the idea presented in Breß et. al. [3]. It maintains a list of observed execution times on the data D for each algorithm $A^{o_i}_{u_j}$. Observations are interpolated (using e.g. cubic splines) to form a new function of the estimated execution time.

As expected the heterogeneous query planer with the *cost objective* optimization is the slowest one. It is designed only to minimize the execution cost of a query and the execution time is not under consideration. This leads to under-utilization of capabilities of the devices in the environment.

The second slowest is the *time objective* optimization. It uses only the theoretical throughput of operations as a criteria. Since these criteria do not change over time or load on the environment, there is no load balancing between devices. This leads to the poor utilization of capabilities of less powerful devices in the environment.

The *Self Tuning Planner* suggested in [3] performs better. However, there are two problems with this approach. Firstly, it adds an additional overhead due to the interpolation of the observed execution times [3]. Secondly, as Figure 1 shows, it takes some time before the planner adapts to a new situation. At an early stage it performs similarly to the *Time Objective Planner*. This is due the fact that it does not immediately react to load changes on the device. Instead, it has to gather enough observations before adapting. This is presented as a reference framework.

The best performance is gained when using the heterogeneous query planner with the *mixed objectives* optimization and the *bi-objective* optimization. When using the *mixed objective* optimization, each query sequence is optimized using one of three supported optimization criteria: time, cost and bi-objective. For the purposes of this simulation, we assumed that for each query sequence an optimization criterion is chosen randomly (with equal probability). This simulates the cases where there exists a priori queries prioritisation (as discussed in Section 3.6). On the contrary, the heterogeneous query planner with the *bi-objective* optimization, optimizes all queries using only the bi-objective criteria. The optimization using both approaches gave a significant improvement compared to a single objective (Cost or Time) criterion and the reference framework.

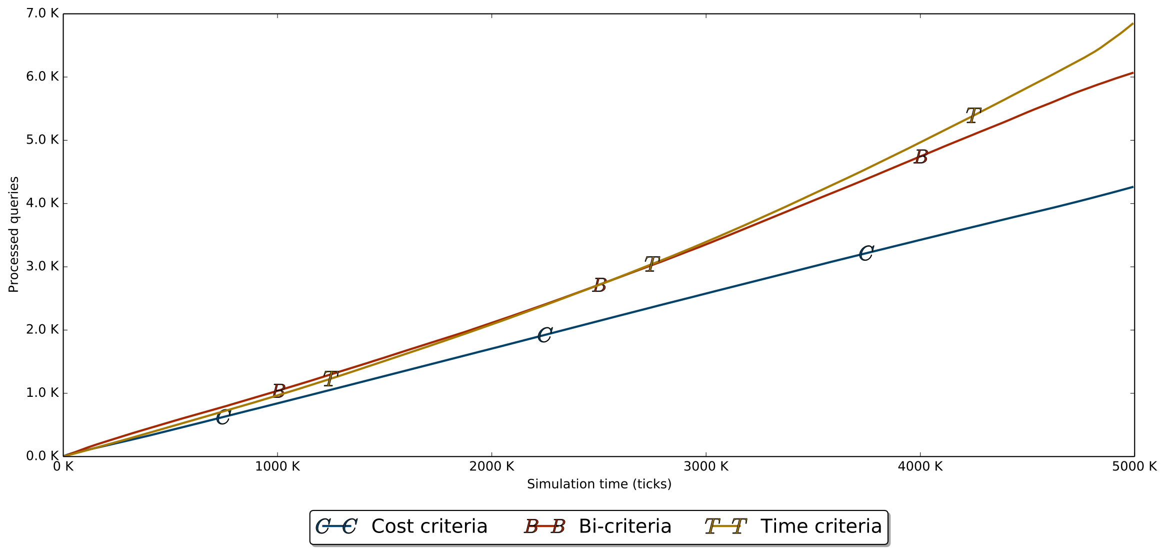


Figure 2: The efficiency of components combined in the heterogeneous query planer with the mixed objectives optimization

4.2.2. Heterogeneous Query Planer: Mixed Objectives Vs Bi-objective Optimization

In Section 3.6 we discussed possible use cases for the presented query planner. One possible use case is for applications with known a priori prioritization of queries (e.g. batch queries with low priority and interactive queries with high priority). This can be done by using the heterogeneous query planner with the *mixed objectives* optimization. Figure 2 presents the isolated efficiencies of various optimization objectives used within the mixed objective optimization. In our simulation of the mixed planner each query was assigned one optimization objective with equal probability $1/3$. As it may be observed in such configuration, the time optimization is the most appropriate for the query processing with high priority or with the execution time constraint. The cost optimization is appropriate for operations with low priority or without a time constraint. The optimization of both cost and time (without preferences) leads to a moderate processing speed and load balancing. The heterogeneous query planner with the *mixed objectives* optimization may naturally adopt to situations when there are different ratios between queries optimized using time, cost or both objectives. However, this can significantly affect the performance of this solution. The domination of one of objectives in the mixed query planer may resemble the performance of a planer which uses only that objective.

In order to explain the similarity of the performance of the mixed objectives heterogeneous query planer and the bi-objective optimization heterogeneous query planner, we prepared another experiment. We repeated experiments from Section 4.2.1. However, this time we summarized the execution times for all optimization objectives in the mixed query planner. Simultaneously we gathered identifiers of the queries from each group. Then we summarized the execution times of queries when using the *bi-objective* query planer. We used identifiers of queries gathered in the mixed planner to create the group summation for the bi-objective planner.

As a result we have found out that the mixed planner better adapts to the potential priority of queries than the bi-objective planner. Assume a query that has been drawn for the cost optimization by the mixed planner. When compared to the bi-objective planner the mixed planner generates the plan that on average is 31% slower. This reasonably reflects the expected handling of batch queries. On the other hand assume a query drawn to be time optimized. When compared to the bi-objective planner the mixed planner generates the plan that on average is 19% faster. This reasonably reflects the expected handling of interactive queries. Furthermore, queries chosen to be bi-objectively optimized are handled 4% faster by the mixed planner than by the bi-objective planner. Therefore, when an application has a closed set of queries divided into priority classes (batch, interactive, etc.), the mixed planner seems to be the best choice. Obviously the optimization objective is not randomly selected in such a setting.

4.2.3. The Evaluation Of The Economics Model

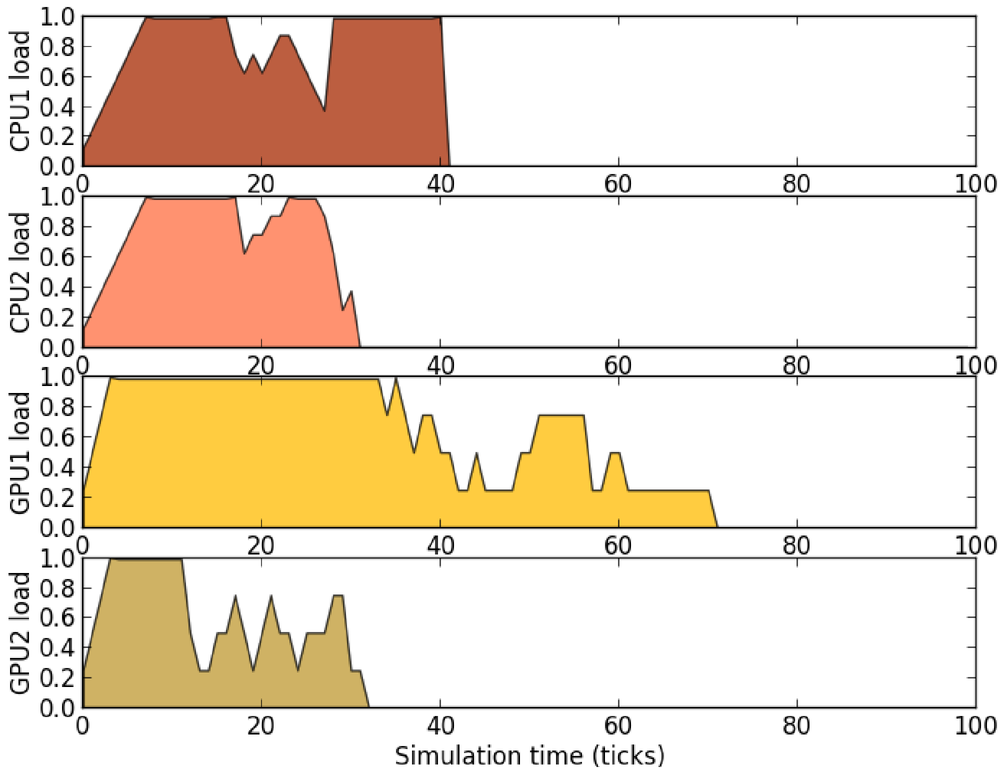


Figure 3: The simulated load of devices ($0 < load < 1$). Note that the time is given in simulation ticks.

Since the proposed economic model is an important part of the presented framework, we present Figures 3 and 4. They show the interaction between device load (Fig. 3) and the established device pricing (Fig. 4). Indeed, the increased load influences the unit pricing according to the formula 7.

It is worth noting that the pricing model may be tuned for specific applications (see Table 2b for this simulation settings) by modifications of minimal unit prices and thresholds that indicate the level of the desired device load.

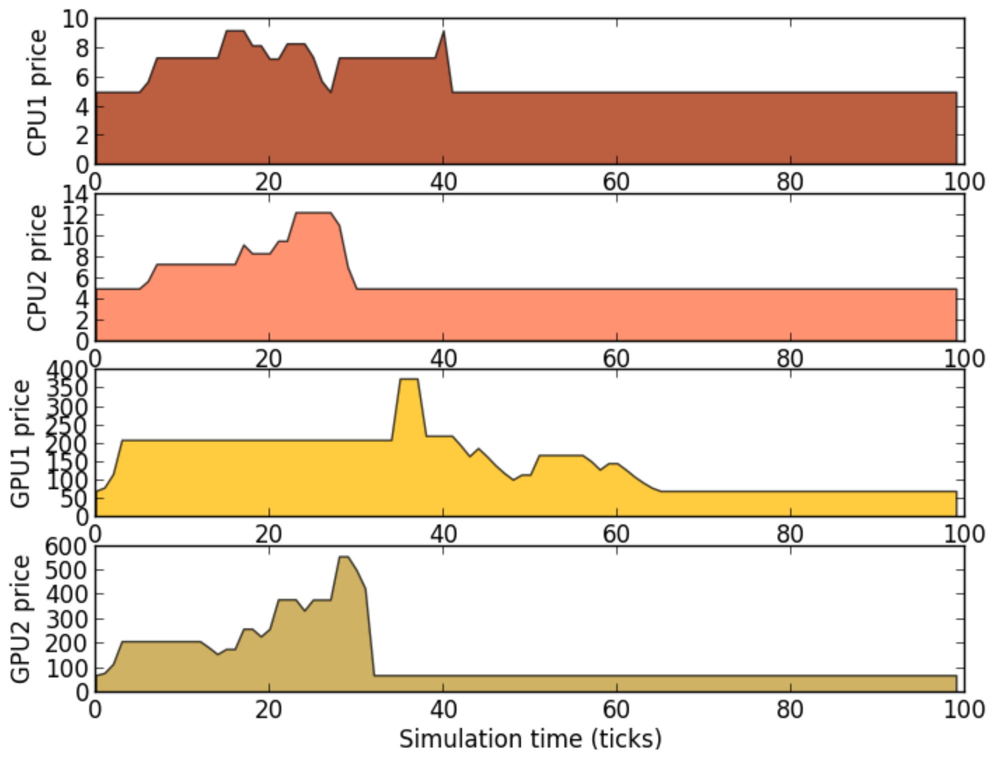


Figure 4: The simulated pricing of devices. Note that the time is given in simulation ticks.

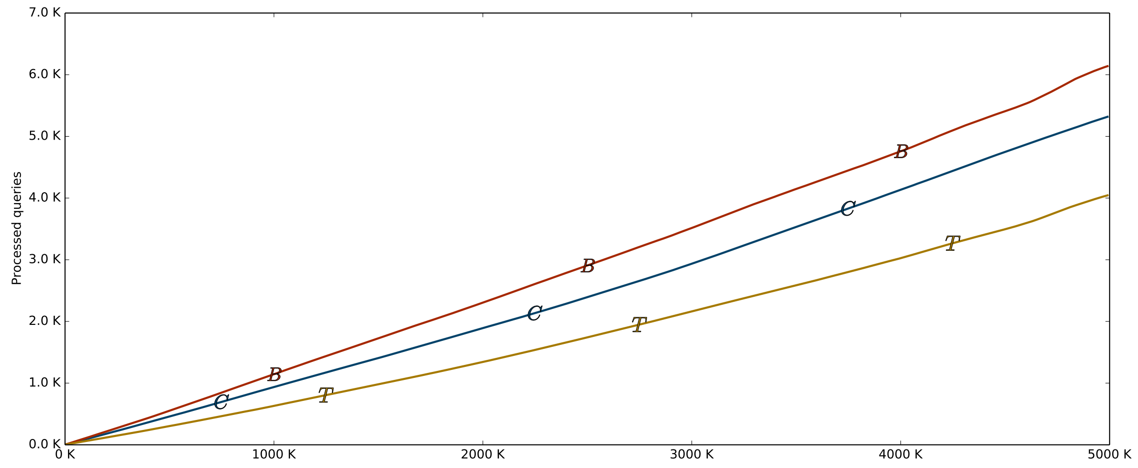
Especially, the behaviour of the mixed model strongly depends on the threshold setting for each device. Figure 5 presents how the model behaves for two different settings of the threshold. In the first case (Fig. 5a) the system is expected to put more load onto GPU and keep CPU less occupied. In the second case (Fig. 5b) GPU load is intended to be low and CPU may be more occupied.

We may observe that in the first case (Fig. 5a) the time objective optimization is not working as intended since other modes of optimization are able to process more queries. This situation is a natural consequence of CPU/GPU threshold levels, as high GPU threshold promotes GPU (by competitive price) and low CPU threshold demotes CPU (i.e. CPU protects resources by rising the prices). As time objective depends on measurements taken in perfect conditions (see Section 3.3), in most cases GPU will be selected as processing device. This observation explains the overload of GPU.

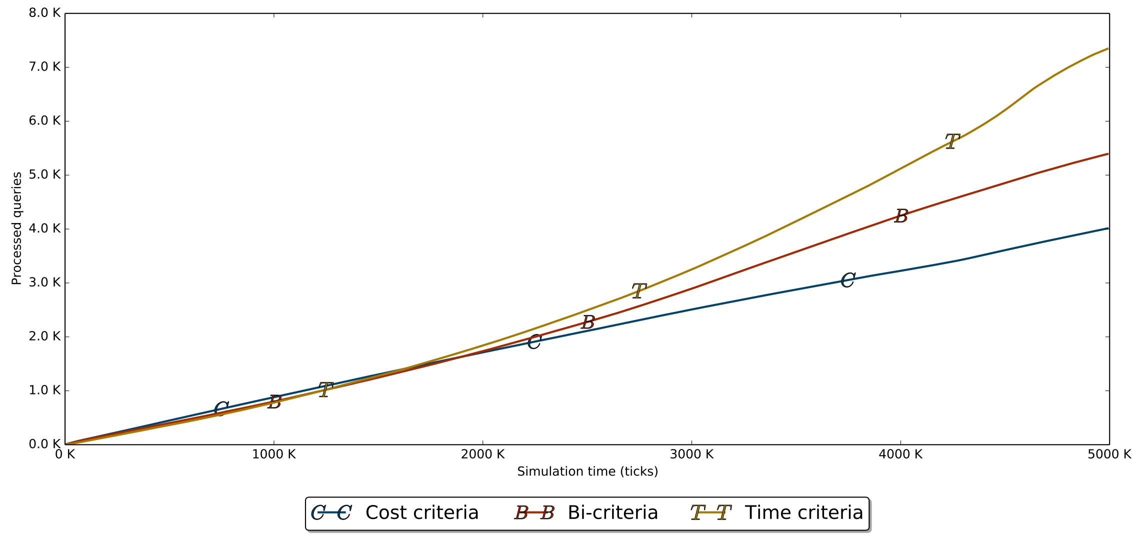
In the second case (Fig. 5b) GPU is able to offer resources almost all the time (protecting resources by raising the price) and queries which require fast processing may benefit from it. We may see that the time objective optimization processed significantly more queries than other optimizations.

4.3. Evaluation Of Challenges Of Heterogeneous (CPU/GPU) Query Processing

In Section 2.1 we have cited challenges of Hybrid Query Processing initially presented in Breß et.al. [3]. We have also noted that these challenges are concern heterogeneous query plans. As our bi-objective optimization framework was designed in order to address this challenges, an evaluation in the context of them is needed.



(a) Thresholds – CPU:0.25, GPU:0.75.



(b) Thresholds – CPU:0.75, GPU:0.25.

Figure 5: The efficiency of various optimization objectives for mixed objectives optimization depending on the threshold.

We address the *Critical Query* problem by allowing different optimization targets for queries (when using the mixed query planner). Choosing the time optimization allows to articulate a priori the importance of a query (see Section 4.2.2). Also, the bi-objective optimization tends to promote queries which may gain more on particular devices (due to the cost-delay trade-off and the fact that the cost objective is designed to promote tasks with greater speed-up – see Section 3.4).

The problem of *Execution Time Prediction* is addressed indirectly with bi-objective optimization. This is because the bi-objective optimization combines the cost objective function, which uses the current device load when pricing a device, with the execution time objective. Therefore, in most cases it is preferred to optimize both cost and time (without preferences towards any) through the time/cost trade-off.

Copy Serialization Challenge is partially addressed in the set of our publications on lightweight compression methods for GPU environment [12, 13, 14]. CUDA programming framework allows, in certain circumstances, to perform the computation in parallel to the data transfer. Thus, it is possible to mix copy operations with decompression operations. This may reduce the negative impact of the copy serialization on the PCIe bus.

Lastly different types of optimization in the mixed query planner apply also to the *Optimization Impact* challenge. The choice of optimization criteria specifies a possible impact on other queries (see the results in Section 4.2.2).

5. Related Work

The multi-objective query optimization was considered e.g. in Stonebraker et.al. [17] where a wide-area distributed database system (called Mariposa) was presented. An economic auction model was used as cost model. To process a query a user supplied a cost-delay trade-off curve. Because defining this kind of input data was problematic, Papadimitriou et.al. [11] proposed a new approach where an algorithm for finding ϵ-Pareto optimal solutions was presented. Then a user manually chooses one of the presented solutions. This work differs both in the optimization method and the economic model involved.

In our framework a user supplies an optimization objective for a query a priori (time, cost or bi-objective). Also as our model addresses the optimization of the co-processing interaction a simpler commodity market model could be used instead of a bidding model.

An extended overview on utilization of a GPU as a coprocessor in database operations may be found in [3]. Breß et. al. [3] proposed a framework for optimization of hybrid CPU/GPU query plans and present two algorithms for constructing hybrid query sequences. The first algorithm selects the fastest algorithm for every element of a query sequence (including the cost of transfer between devices) with the complexity $O(n)$. Unfortunately, this algorithm has two flaws [3]: the constructed plan could generate too frequent data transfers between devices. This may significantly affect the performance of the data processing. It also means that a suboptimal plan is sometimes generated. To overcome those problems they proposed a second algorithm. It searches for a continuous segment of operations on GPU that could improve the base CPU sequence. In order to find an optimal solution this algorithm generates all possible GPU sequences. Its complexity is obviously higher, as it is $O(n^2)$. Our work extends this approach by allowing numerous co-processing devices (the Heterogeneous Query Planer in Section 3.1). Moreover, our work incorporates the commodity market model as well as the bi-objective optimization for better performance overcoming problems mentioned in Section 2.1. Additionally, the algorithm OptimalSeq presented in our work may be used to produce a similar solution as the second algorithm by Breß et. al. [3] but with better complexity (in case of two devices $O(n)$). If $U = \{GPU_1, CPU_1\}$ and $u^* = CPU_1$, then the results returned by the algorithm OptimalSeq are equivalent to the results returned by the second algorithm by Breß et.al.[3].

Two surveys are worth mentioning. The first is an extended overview of economic models in grid computing [6]. The second thoroughly describes methods for the multi-objective optimization [9].

6. Conclusions And Future Work

6.1. Conclusions

In this paper, we proposed a bi-objective optimization framework for heterogeneous query plans. We also presented an algorithm to create query sequences in a heterogeneous environment with a single objective. This algorithm may be used to construct query sequences similar to [3] but with a better complexity. For the purposes of this bi-objective optimization we designed a model including the time and cost objective function. The cost objective function and the pricing model are built on the foundations of the economic model of commodity markets.

The effectiveness of the proposed model has been tested in a simulation that mimics the heterogeneous environment. The results of experiments are satisfactory. We achieved good load balancing of simulated devices combined with better optimization results than previously presented processing models. The obtained results attest that the proposed model can be an interesting and effective option for databases operating in heterogeneous computing environments.

6.2. Future work

In this work, we put the main emphasis onto the database systems using the GPU as a coprocessor. However, the presented model can be used for arbitrary coprocessors (e.g. GPU, CPU, FPGA, Intel Xeon Phi, etc.) and unrestricted numbers and combinations of them. Therefore, we plan to extend the presented model beyond CPU/GPU co-processing.

The presented model has a number of essential parameters. In order to tune them, we plan to perform a detailed analysis of the performance of the model for different factors and settings.

The most important aspect of future work is to test the presented model in a real heterogeneous database environment. Candidate environments are a prototype time-series database [14, 13] and database solution described by Bress et. al. [5].

References

[1] W. Andrzejewski and R. Wrembel. Gpu-wah: applying gpus to compressing bitmap indexes with word aligned hybrid. In *Database and Expert Systems Applications*, pages 315–329. Springer, 2010.

[2] J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, Sept. 1984.

[3] S. Breß, I. Geist, E. Schallehn, M. Mory, and G. Saake. A framework for cost based optimization of hybrid cpu/gpu query plans in database systems. *Control and Cybernetics*, pages 27–35, 2013.

[4] S. Breß and G. Saake. Why it is time for a hype: a hybrid query processing engine for efficient gpu coprocessing in dbms. *Proceedings of the VLDB Endowment*, 6(12):1398–1403, 2013.

[5] S. Breß, E. Schallehn, and I. Geist. Towards optimization of hybrid cpu/gpu query plans in database systems. In *New Trends in Databases and Information Systems*, pages 27–35. Springer, 2013.

[6] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1507–1542, 2002.

[7] W. Fang, B. He, and Q. Luo. Database compression on graphics processors. *Proceedings of the VLDB Endowment*, 3(1-2):670–680, 2010.

[8] D. Florescu and D. Kossmann. Rethinking cost and performance of database systems. *ACM Sigmod Record*, 38(1):43–48, 2009.

[9] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[10] K. Muller. Advanced systems simulation capabilities in simpy. *Europython 2004*, 2004.

[11] C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 52–59. ACM, 2001.

[12] P. Przymus and K. Kaczmarski. Improving efficiency of data intensive applications on gpu using lightweight compression. In *On the Move to Meaningful Internet Systems: OTM 2012 Workshops - Lecture Notes in Computer Science*, volume 7567, pages 3–12. Springer Berlin Heidelberg, 2012.

[13] P. Przymus and K. Kaczmarski. Dynamic compression strategy for time series database using gpu. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1-4, 2013 - Genoa, Italy, 2013.

[14] P. Przymus and K. Kaczmarski. Time series queries processing with gpu support. In *New Trends in Databases and Information Systems*. 17th East-European Conference on Advances in Databases and Information Systems September 1-4, 2013 - Genoa, Italy, 2013.

[15] P. Przymus, K. Kaczmarski, and K. Stencel. A bi-objective optimization framework for heterogeneous cpu/gpu query plans. In *CS&P 2013 Concurrency, Specification and Programming*. Proceedings of the 22nd International Workshop on Concurrency, Specification and Programming, September 25-27, 2013 - Warsaw, Poland.

[16] O. O. Sonmez and A. Gursoy. Comparison of pricing policies for a computational grid market. In *Parallel Processing and Applied Mathematics*, pages 766–773. Springer, 2006.

[17] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):48–63, 1996.