A Bi-objective Optimization Framework for Query Plans

Piotr Przymus[1], Krzysztof Kaczmarski[2], and Krzysztof Stencel[3]

[1] Nicolaus Copernicus University, Poland, eror@mat.umk.pl
[2] Warsaw University of Technology, Poland, k.kaczmarski@mini.pw.edu.pl
[3] The University of Warsaw, 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 optimisation 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.

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). GPUs offer a notable processing power for streams.

Execution of database queries is an example of a successful application of GPGPU. The current research focuses on using the GPU as a co-processor [5]. GPU as co-processor may accelerate numerous database computations, e.g. relational query processing, query optimization, database compression or supporting time series databases [5,13,14].

An application of GPU requires transferring data from the CPU memory to the graphical device memory. The data transfer is usually time-consuming. It may diminish the gain of the acceleration credited to GPU. This situation can be improved by using lightweight compression methods that can significantly reduce the costs associated with communication [13,14]. However, this does not solve all the problems. In particular, GPU is optimized for numerical computation. Thus, only selected operations will benefit from GPU. Small data sets are another problem. For such sets the data transfer may dominate processing time and

destroy the performance gain. Therefore, joint processing capabilities of both CPU and GPU are worth considering. Furthermore, as it is common to have more than one GPU in a computer, a potential use of various GPU devices should be considered. This type of query plans is called heterogeneous.

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 based on the commodity market. A query plan is bi-objectively optimized to minimize: the processing time and the value of consumed resources. For the user, a small difference in execution time can be negligible. Thus, it is worth optimizing a query, so that the execution time satisfies the user while other costs are minimized. In this case, the cost may be, e.g. the responsiveness of the system, power consumption, heat production, etc. One can also consider expressing the cost in financial terms.

2 Preliminaries

2.1 GPU and Heterogeneous query processing

From the parallel processing's point of view, CPU accompanied by a GPU co-processor is a *shared nothing architecture*. A GPU card has its own memory or a separate area in the CPU main memory. Thus, the data has to be explicitly transferred from the CPU main memory to the 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. Thus, it is important to include the transfer cost in the total execution time of an operation. This cost is also a component of the execution time prediction.

Contemporary computer systems often include more than one GPU. Then, it is possible to combine multiple computational units in a single query plan. Such plans are called *heterogeneous query processing*. 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. Therefore, 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 used resources.

Bress et. al. [5] identified problems of hybrid (CPU/GPU) query processing which are also true in 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.

2.2 Commodity market approach in query processing context

In this paper we address these problems by showing that they may be solved by applying a supply-and-demand pricing model taken from a commodity market. In such a market resource owners (processing devices) price their assets and charge their customers (queries) for consumed resources. Other pricing models may also be used [6].

In the supply-and-demand model when supply (available resources) or demand (needed resources) changes, the prices will be changed until an equilibrium between supply and demand is found. 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 usage or the network usage. Typically, 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 problem where optimal decisions need to be taken in the presence of trade-offs between two conflicting objectives. It is a special case of multiple criteria decision making. Typically 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 [11]. Given a set of choices and a way of valuing them, the Pareto set consists of choices that are Pareto efficient. A set of choices is said to be Pareto efficient if we cannot find a reallocation of those choices such that the value of a single choice is improved without worsening values of others choices. As bi-objective query optimization is NP-hard, we need an approximate solution [12].

3 Heterogeneous Query Planer

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'.

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 Eq. 1
f_t, g_t	estimated algorithm run time and copy time see Sec. 3.2
f_c, g_c	estimated algorithm run cost and copy cost see Sec. 3.3
f_b, g_b	estimated algorithm run and copy bi-objective scalarization see Sec. 3.4
$F_x(QS_{het})$	sum of f_x and g_x over columns of QS_{het} where $x \in \{t, c, b\}$ see Eq. 2

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

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

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

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 - bioptimization } - 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  |  A, B, C = DiffSeq (QS_hybrid, Q_u, u, x);
10 |  val, start, end = MaxSubseq(A, B, C);
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.

For simplicity let us assume that $x = t$, $f_t(A_u^o, D_i) = t_{run}(A_u^o, D_i)$ and $g_t(M_{u,u'}, D) = t_{copy}(M_{u,u'}, D)$. Later in this 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. 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., $A_u^{o_i} \in AP_{o_i}$ 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

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 - bioptimization $\}$ - optimization type
Result: A - operations improvement array; B, C - copy to/from unit arrays

```
1 A, B, C = [], [], [];
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 A ;   /* e.g. f_t(A^o_u, D) */
6 |   append g_x(M_{u_f,u}, D_i) to B ;                     /* e.g. g_t(M_{u,u'}, D) */
7 |   append g_x(M_{u,u_t}, D_i) to C;
8 end
9 return A, B, C
```

corresponding segment of the original sequence). This is done by calculating the improvement and copy cost arrays in DiffSeq and finding 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 $f_x(A^o_{u_b}, D_i) - f_x(A^o_u, D_i)$ and copy costs from/to unit. The procedure MaxSubseq is based on Kadane's algorithm for maximum subarray problem [1]. 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 (B and C 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 Economics in Heterogeneous Environment

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:

Procedure MaxSubseq(A, B, C)

```
   Input: A - operations improvement array; B, C - copy to/from unit arrays
   Result: maximum_improvment, start, end
 1 max_ending_here = max_so_far = 0 ;
 2 begin = tbegin = end = 0 ;
 3 for i, x in enumerate(A) do
 4 |   max_ending_here = max(0, max_ending_here + x) ;
 5 |   if max_ending_here = 0 then
 6 |   |   tbegin = i ;
 7 |   |   max_ending_here -= B[i];
 8 |   end
 9 |   if max_ending_here - C[i] >= max_so_far then
10 |   |   begin = tbegin ;
11 |   |   end = i ;
12 |   |   max_so_far = max_ending_here ;
13 |   end
14 end
15 return max_so_far - C[end], begin, end
```

- the estimation of needed resources (the size of the data D, the performance of the task $A_{u_i}^{o_i}$),
- 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, pricing for using the resources of computational unit is established. This depends on the previous load of the device: the higher demand for computational unit, the higher price for using it. This is a periodic process which calculates prices every Δt_{up} seconds by calculating computational unit price P_u. Let $0 < L_{curr} < 1$, $0 < L_{prev} < 1$ be current and previous computational unit load factors. Additionally, let L_{th} be a threshold below which prices should decrease, and P_{min} be the minimal price. Then the price is calculated using the following formula[4]:

$$P_u := \begin{cases} max(P_{min}, P_u \cdot (1 + \frac{\Delta P_u}{(1-\Delta U)}) & \text{if } (\Delta P > 0 \wedge \Delta U > 0) \vee (\Delta P < 0), \\ P_u & \text{otherwise,} \end{cases} \tag{3}$$

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 exception to the pricing formula i.e., 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 price we need to define a function returning speedup factor between base device u^* (defined in the previous section)

[4]Slightly abusing notation we will also denote the new price by P_u.

and current device: $speedup(A_u^o, D_i) = t_{run}(A_{u^*}^o, D_i)/t_{run}(A_u^o, D_i)$. Then we define a cost function as $c_{run}(A_u^o, D_i) = \frac{\#D_i}{speedup(A_u^o, D_i)} \cdot P_u$, where $\frac{\#D_i}{speedup(A_u^o, D_i)}$ part combines the estimation of needed resources and the preference of the device. A computational unit with high speedup on 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 often speedup depends on the size of processed data (usually low speed-up on small datasets) so discount depends on data size.

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

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

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

Now let $f_c(A_u^o, D_i) = c_{run}(A_u^o, D_i)$ and $g_c(M_{u,u'}, D) = c_{copy}(M_{u,u'}, D)$. A solution minimizing the cost may be found under the previous assumptions and using procedure *OptimalSeq*.

3.4 Bi-objective Heterogeneous Query Planer

As finding Pareto optimal bi-objective query plan is NP-hard (bi-objective shortest path problem) [12], we will use previously described *OptimalSeq* single objective approximation algorithm and extend it to bi-objective case.

We will use *a priori* articulation of preference approach which is often applied to multi-objective optimization problems. It may be realized as the scalarization of objectives, i.e., all objective functions are combined to form a single function. In this work we will use *weighted product* method, where weights express user preference [11]. Let us define:

$$f_b(A_u^o, D) = c_{run}(A_u^o, D)^{w_c} \cdot t_{run}(A_u^o, D)^{w_t},$$
$$g_b(M_{u,u'}, D) = c_{copy}(M_{u,u'}, D)^{w_c} \cdot t_{copy}(M_{u,u'}, D)^{w_t}.$$

where w_t and w_c are weights which reflect how important cost and time is (the bigger the weight the more important the feature – values of f_b, g_b are higher than 1). It is worth to mention that a special case with $w_t = w_c = 1$ (i.e., without any preferences) is equivalent to Nash arbitration method (or objective product method) [11].

4 Preliminary Experimental Results

4.1 Simulation settings

In order to evaluate this model we prepared a *proof of concept* and evaluated it using custom developed simulation environment. Simulation environment was de-

Device	o_1	o_2	o_3	o_4	o_5	o_6
GPU1	20	11	6	0.38	14	15
GPU2	5	11	6	0.33	4.66	5
CPU2	1	1	1.09	1	1.27	1.36

(a) Average speedup of operation o_i on given device compared to CPU1

Option	CPU1	CPU2	GPU1	GPU2
Unit threshold	0.75	0.75	0.4	0.4
Unit minimal price	5	5	70	70

(b) Pricing model configuration

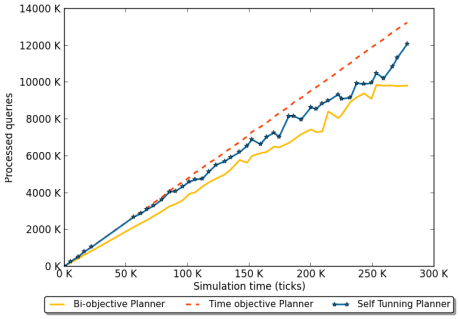
Table 2: Simulation environment configuration

veloped using Python and SimPy framework. All presented experiments are derived from simulation. There where four devices defined in environment: CPU1, CPU2, GPU1, GPU2. Data transfer bandwidth between CPU* $\leftrightarrow$ GPU* was measured on real system, bandwidth of GPU1 $\leftrightarrow$ GPU2 was calculated using CPU1 as transit device. Following weights in bi-objective scalarization were used $w_t = w_c = 1$ (i.e., without any preferences setting). Other settings of the simulation environment are gathered in Tables 2b and 2a. Simulation environment generates new query sequences, when spawn event occurs. Spawn event is generated randomly in a fixed interval and generates randomly set of query sequences (with fixed maximum). Every query sequence consists of maximally six operations and operates on random data volume. In the simulation the processed data size has a direct (linear) influence on processing speed. Each device has got a limited number of resources; a database operation can be performed only if needed resources are available. In other cases the operation is waiting. After generating desired number of query sequences the simulation stops spawning of new tasks and waits until all generated query sequences are processed.

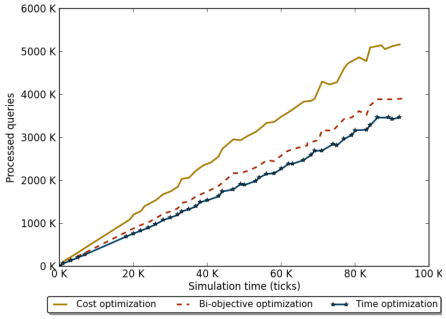
4.2 Simulation Results

Figure 1a presents simulated execution time of three scheduling frameworks processing a pool of generated sequences of queries. To each generated query sequence an optimization criterion (time, cost or bi-optimization) was assigned with equal probability 1/3. Optimization criteria are only used if query scheduling framework supports it, otherwise default criteria is used. All scheduling frameworks process exactly the same pool of generated query sequences.

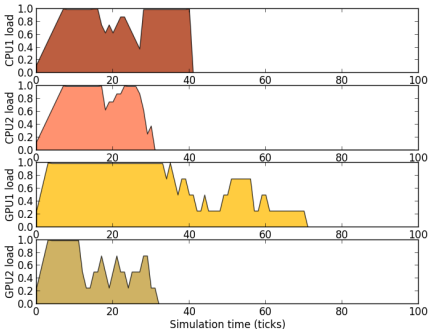
Compared frameworks are based on *OptimalSeq* algorithm but use different objective function. *Time objective planner* uses only f_t and g_t functions as optimization criteria; this means that it has no idea on load of each of devices. *Self Tuning Planner* is based on idea presented in Breß et. al. [3], i.e. it maintains a list of observed execution times on data D for each algorithm $A_{u_j}^{o_i}$. Observations are interpolated (using e.q. cubic splines) to form new estimated execution time function. And finally *Bi-objective planner* is a proof of concept implementation of the model described in this work.



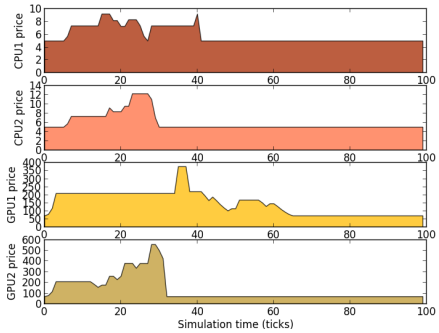
(a) Simulated efficiency of Bi-objective Heterogeneous Query Planer, Time based Query Planer and Self-Tuning Query Planner



(b) Efficiency of Bi-objective Heterogeneous Query Planer for various optimization tasks



(c) Simulated load of devices ($0 < load < 1$)



(d) Pricing of device

Fig. 1: Simulation results. Note that time is in simulation ticks.

As expected *Time Objective Planner* is the slowest one since it has no knowledge on the current load of devices. A solution suggested in [3] performs better. However, there are two problems with this approach: first it adds an additional overhead due to the interpolation of the observed execution times [3]; Secondly as may be observed in 1a it takes some time before it adapts to a new situation (in early stage it performs similarly to the *Time Objective Planner*). This is due the fact that it does not immediately react to load change of the device. Instead, it has to gather enough observations before adapting. The best performance is gained when using *Bi-objective planner*, this is due to the three types of optimization and the cost model which assures proper load balancing.

As our framework support different types of optimization in the Figure 1b, we present an impact of optimization type on processing performance. As it may be observed, time optimization is the most appropriate for query processing with high priority or with execution time constraint (like interactive queries or ad hoc data mining). Cost optimization is appropriate for operations with low priority or without time constraint (like batch processing or periodic jobs). Optimization

of both cost and time (without preferences) leads to moderate processing speed but with better load balancing which is discussed later.

As proposed economic model is an important part of presented framework, in Figure 1 interaction between device load 1c and established device pricing 1d is illustrated. Notice how increased load influences unit pricing according to the formula 3. It is worth noting that pricing model may be tuned for specific applications (see Table 2b for this simulation settings).

4.3 Discussion

In Section 2.1 we cite three challenges of Hybrid Query Processing initially presented in Breß et.al. [3]. As our bi-objective optimization framework was designed in order to address this challenges, an evaluation in the context of the former mentioned problems is needed. We address *Critical Query* problem by allowing different optimization targets for queries. Choosing time optimisation allows to a priori articulate importance of a query. 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 3.3). The problem of *Execution Time Prediction* is addressed indirectly with bi-objective optimisation. This is because the bi-objective optimisation combines the cost objective function, which uses a current device load when pricing a device, with the execution time objective. So in most cases it is preferred to optimize both cost and time (without preferences towards any) through time/cost trade-off. Lastly different types of optimization apply also to *Optimization Impact* challenge. Choosing optimization criteria specifies a possible impact on other queries. Although, the preliminary results are promising and seem to confirm this, an extended evaluation is needed in future.

5 Related Work

Multiobjective query optimization was considered i.a. 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. proposed a new approach where an algorithm for finding ϵ-Pareto optimal solutions was presented. The solution was that a user would manually choose one of presented solutions. This work differs both in an optimisation method and an 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 optimisation of 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 optimisation of hybrid CPU/GPU query plans and present two algorithms for

constructing hybrid query sequences. The first algorithm selected the fastest algorithm for every element of a query sequence (including the cost of transfer between devices) with complexity $O(n)$. Unfortunately, this algorithm had two flaws [3]: the constructed plan could generate too frequent data transfers between devices, which may significantly affect the performance of data processing and also an optimal plan was not always generated. To overcome those problems they proposed the second algorithm. It searched 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: $O(n^2)$. Our work extends this approach by allowing possible many various co-processing devices (Heterogeneous Query Planer in Section 3.1). Secondly our work incorporates commodity market model as well as bi-objective optimisation for better performance overcoming problems mentioned in 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 $O(n)$).

It is worth to mention two surveys: the first one describing economic models in grid computing [6] and the second one describing methods for multi-objective optimisation [11].

6 Conclusions and Future Work

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

The preliminary experiments are very promising. We achieved good load balancing of the simulated devices combined with better optimization results.

In future work, an extended evaluation of the presented framework is needed, including; examination of parameters' influence on the model behaviour, careful assessment against Hybrid Query challenges. Another interesting field is extension of this model beyond CPU/GPU co-processing. Finally, the framework will be evaluated in a prototype time-series database [14,13].

References

1. J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, Sept. 1984.
2. S. Breß, F. Beier, H. Rauhe, E. Schallehn, K.-U. Sattler, and G. Saake. Automatic selection of processing units for coprocessing in databases. In *Advances in Databases and Information Systems*, pages 57–70. Springer, 2012.

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, S. Mohammad, and E. Schallehn. Self-tuning distribution of db-operations on hybrid cpu/gpu platforms. *Grundlagen von Datenbanken, CEUR-WS*, pages 89–94, 2012.
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. M. J. Franklin, B. T. Jónsson, and D. Kossmann. Performance tradeoffs for client-server query processing. In *ACM SIGMOD Record*, volume 25, pages 149–160. ACM, 1996.
10. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.
11. R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.
12. 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.
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. A. Raith and M. Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, 2009.
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.
18. T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The implementation and performance of compressed databases. *ACM SIGMOD Record*, 29(3):55–67, 2000.