What Makes a Good Physical plan? — Experiencing Hardware-Conscious Query Optimization with Candomblé

ABSTRACT

Query optimization is hard and the current proliferation of “modern” hardware does not make it easier. In addition, the tools that are commonly used by performance engineers, such as compiler intrinsics, static analyzers or hardware performance counters are neither integrated with data management systems nor easy to learn. This fact makes it (unnecessarily) hard to educate engineers, prototype and optimize database query plan for modern hardware. To address this problem, we developed a system called Candomblé that lets database performance engineers interactively examine, optimize and evaluate query plans using a touch-based interface. *Candomblé* puts attendants in the place of a physical query optimizer that has to rewrite a physical query plan into a better equivalent plan. Attendants experience the challenges when ad-hoc optimizing a physical plan for processing devices such as GPUs and CPUs and capture their gained knowledge in rules for a rule-based optimizer.

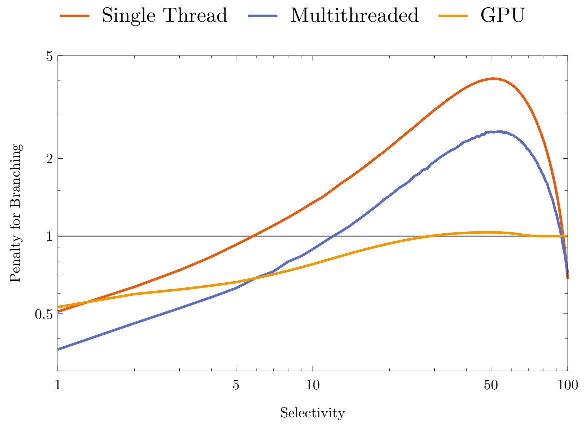


Figure 1: The penalty for branching selections is hardware as well as data dependent

1. INTRODUCTION

Hardware conscious database tuning is often considered a dark art – understood by few, difficult to learn and a discipline in which tiny details can have surprisingly large effects. Figure 1 illustrates how simple choices such as way the output of a selection is materialized can have significant effects on query performance. In the figure, we see the penalty on query runtime if we use a branching implementation of the select operator: When branches are least predictable (50% selectivity) the effect still is slight on a GPU (Nvidia Titan X) but significantly negative on CPUs (an Intel(R) Xeon(R) CPU E3-1270 v5).

While sensitivity to parameters seems an inherent feature of the field of hardware-conscious optimization, we address the first two problems (understanding and learning curve) in our demo. These problems are, to a large extent, due to the lack of proper tools: the available tools are usually not integrated with the data management system but inherited from the programming language and the computer architecture. In the light of the increasing importance of hardware-conscious optimization on database performance, many recent proposals have blurred the line between these fields: systems like HyPeR [3], Legobase [2] and TupleWare [1] compile declarative queries into imperative code at runtime and rely on the underlying compiler to optimize it. This way, the optimizer can make decisions with full knowledge about the hardware it is optimizing for. Unfortunately, this approach has a significant shortcoming: the low-level compiler is not aware of properties of the data that is to be processed by the generated code. This prevents a number of optimizations of which predication (the one applied for Figure 1) is one example. Other examples include the selection of appropriate parallelization strategies, materialization points, hash-functions and collision handling.

To address this problem, we developed *Candomblé* , a *virtual* database kernel based on a unified algebra for physical and logical optimization. The *Candomblé* kernel is “virtual” in the sense that it generates efficient executable machine code for variety of different hardware platforms, both sequential as well as parallel. In addition, *Candomblé* provides fine grained control over low-level tuning through the use of RISC-style[1] plan operators. Through these techniques, *Candomblé* achieves performance that is competitive and sometimes superior to that of existing “hard-optimized” systems such as HyPeR and Tupleware “on their home turf”, i.e., classic multicore processors. In addition, *Candomblé* code

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

[1] Reduced Instruction Set Computing [4]

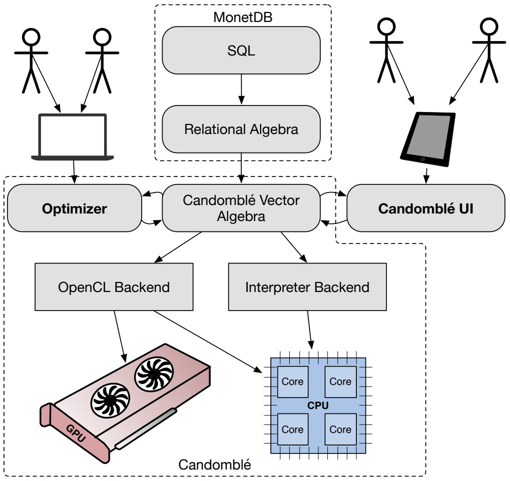


Figure 2: The *Candomblé* System Architecture

can also run on massively parallel processors and, thus, significantly outperform existing CPU-oriented systems.

While *Candomblé* allows expressing and executing such programs, it still relies on a hardware-conscious optimizer to generate an appropriate plan. Fortunately, we found that appropriate plans on these different devices are often similarly shaped and can, thus, be enumerated efficiently. So, while *Candomblé* makes it possible to express, enumerate and execute plans, it takes a hardware-conscious cost model to select an appropriate plan. In the absence of that, we currently rely on rule-based optimization. However, *Candomblé* 's RISC operator is an easily comprehensible way to represent virtually every static [2] aspect of the query evaluation process such as join or grouping strategies. We propose to use this illustrate the challenges of hardware-conscious optimization but also the superior tunability that is offered by a RISC-style plan algebra.

Purpose of the Demonstration

The purpose of the demonstration is for the attendants to experience the challenges of hardware-conscious optimization without the need to learn to use specialized tools. To that end, we demonstrate a touch-based visual query builder that allows users to examine, optimizer and evaluate the query plan by tuning operator parameters as well as by manipulating the plan structure itself. Through this UI, users will experience how a) local changes to operators can have broader impact on the plan and b) additional operators can improve performance by, e.g., increasing locality or enabling intermediate reuse. We also demonstrate that a compact RISC-style plan algebra is a useful intermediate stage on the way to executable machine code, allowing effective tuning without suffering from prohibitive interpretation overhead.

2. THE CANDOMBLÉ SYSTEM

Before presenting the specific demonstration, let us briefly discuss the architecture (see Figure 2). The demonstration is built on the *Candomblé* database kernel. While it only provides the query execution layer in the DBMS stack, *Candomblé* is designed to be a drop-in replacement for existing database kernels. For our prototype we replaced the execution layer of MonetDB with *Candomblé* . By reusing the MonetDB SQL compiler, we allow users to formulate ad-hoc queries in SQL. We extract the relational algebra plan that is generated by MonetDB and compile it to the *Candomblé* plan algebra (see Section 2.1). This plan is passed through an extensible rule based optimizer before being sent to the interactive plan optimization component that is the heart of our demonstration. At this point, the evaluation process blocks until the user finishes her interactive optimization pass and sends the plan back for execution. At that point, the *Candomblé* program is evaluated by either compiling and running it using the OpenCL Backend on a CPU, a GPU or using the Interpreter Backend (which provides interactive debugging but inferior performance).

To effectively optimize a *Candomblé* program, the attendants need a basic understanding of our operator model. For the demo, we will provide a poster that briefly explains the operators. Fortunately, as we illustrate in the next section, the operator model is very simple.

2.1 The Candomblé Algebra

The goal of the *Candomblé* Vector Algebra is to expose a maximum amount of control to the optimizers while abstracting away hardware specifics. To achieve this level of control, we follow a different approach than most “classic” database algebras: Traditional algebras start with a a high-level algebra (usually relational algebra) and carefully lower the level of abstraction by a) introducing multiple alternatives for a given logical operator, b) introducing designated physical operators such as partitioning or sorting and c) annotating operators with auxiliary information such as sizes of hash-buckets or the number of sorted runs to generate. The goal of these algebras is to introduce just as many physical concepts as necessary to perform the desired optimizations while maintaining a plan that is as high-level as possible.

In contrast, *Candomblé* is designed using a different approach: we start with low-level design concepts that are similar to those from computer architecture design (RISC, SIMD, ...) and carefully raise the level of abstraction. The design goal is to abstract the specifics of the hardware to allow efficient reasoning about the plan without obstructing the potential for hardware-specific optimization. By that we make every intermediate step of the query execution process explicit and, thus, tunable. This is captured in three guiding principles in our design of *Candomblé* :

No Implicit State The key design decision in *Candomblé* is the lack of hidden operator state. Where existing dataflow models, such as Spark RDDs and most database kernels, have hidden per-operator state, *Candomblé* exposes all generated data structures. This makes *Candomblé* both portable and tunable. It is tunable because if the optimizer decides that a certain step in a complex operator is unnecessary or shared with other operators, it can simply remove or share the code that creates it. It is portable, because all state needs to be declared up front, which allows us to run in execution environments that prevent dynamic memory (re-)allocation such as GPUs.

Leanness *Candomblé* follows the design principle of *Reduced Instruction Set (RISC)* [4]. This decision is

[2] as opposed to runtime aspects such as aspects like load-balancing or memory allocation

strongly tied to the previous: complex operators are prone to maintain implicit state which violates the first principle. We also aspired to keep the language free of redundancy: there should not be more than one way of expressing the same algorithm.

Static Critical Path The third distinguishing property of *Candomblé* is a *Static Critical Path.* In *Candomblé* , runtime decisions about operator execution (such as scheduling or load balancing) is outside the control of the optimizer. For our purposes the main advantage of this approach is that it provides portability and performance: by making the critical path of the program entirely static, we keep it free from hazards such as function calls or preemption. This allows good (and predictable) performance on classic CPUs as well as the efficient execution of *Candomblé* code on massively parallel processors that suffer significant performance degradation when execution paths diverge at runtime.

2.1.1 Primitive Operators

The *Candomblé* primitive operators behave much like SIMD CPU operations albeit on variable sized vectors: like CPUs, *Candomblé* implements operators that only create a single output *of known size* and no auxiliary data structures. Additionally, *Candomblé* operators have no side effects and variables can only be assigned once. *Candomblé* contains two kinds of data processing operations: the first kind are *data-parallel operations* such as arithmetics and logics but also `scatter` (write data values to specified positions) and `gather` (read values from specified positions). The second kind are *fold operations* that are not fully data parallel. This covers all operations in which an output value depends on more than one value of the input vector. Naturally, this includes aggregations but also others such as order preserving selections or partitioning. While not entirely data-parallel, *Fold operations* can still be parallelized by logically partitioning the input. To designate logical partitions, *Candomblé* allows the creation of *Control Vectors*: virtual vectors that are merely used as partition ids when operating on data vectors. When compiling the plan, *Candomblé* maintains metadata about the *Control Vectors* and uses it as tuning hints (much like `#pragma`s in imperative languages). *Control Vectors* are never translated into executable code.

2.1.2 Macros

In addition to the primitive operators, *Candomblé* allows the definition of macros: complex operators that are comprised of primitives. Macros can be expanded in the UI to perform optimization of their implementation. While users can define their own macro operators, *Candomblé* comes with a library of standard operators for common data processing operations such as nested-loop joins, hash-building and -probing as well as partitioning. Since these operators are expandable, attendees are free to investigate and even optimize the Candomblé *Standard Library.*

3. DEMONSTRATION

A *Candomblé* query plan can be interpreted as a dataflow DAG where the vertices represent operators and edges represent data dependencies between operators.

We believe that the most approachable way to interact with such query plan DAGs is through a graphic interface. Consequently, Candomblé *UI* implements a plan visualizer (see Figure 3) that allows both the manipulation of operator parameters as well as the manipulation of the plan shape by adding, removing and reconnecting operators. Operators can be added to the plan by dragging an edge from an existing operator to an operator-class in the roster (on the right of the screens in Figure 3). Since most operators take more than one input, the user may have to draw more than one line before an operator is instantiated.

To familiarize the attendee with the *Candomblé* algebra before letting him explore freely, we divide our demonstration into two parts: a guided tour through the user interface by means of an example followed by a challenge.

3.1 The Guided Tour

To demonstrate the usage of the *Candomblé* UI for hardware-conscious optimization, we start by walking the user through the steps our rule-based optimizer takes.

Starting with an initial sequential plan that is generated from an SQL query we guide the user to speed up an aggregation through parallelization. The work flow in Figure 3 shows the steps the user needs to take. The initial plan contains a selecttion and a global sum operator over the selected values. To speed up the computation of this sum, we can split the sum into two hierarchical phases: The first will partition the input into multiple independent chunks (introduced Fig. 3a), and compute their subtotals (Fig. 3b) in parallel. In the stage a global sum operator will sequentially merge the results from the separate chunks into one final answer(introduced in Fig. 3c). However, since the optimal performance of the plan usually depends on size of the partitions, the user can drill into the partition-step (Fig. 3d), and tune the partition size (Fig. 3e). The system will provide live update on the performance (see Fig. 3d) as well as result correctness.

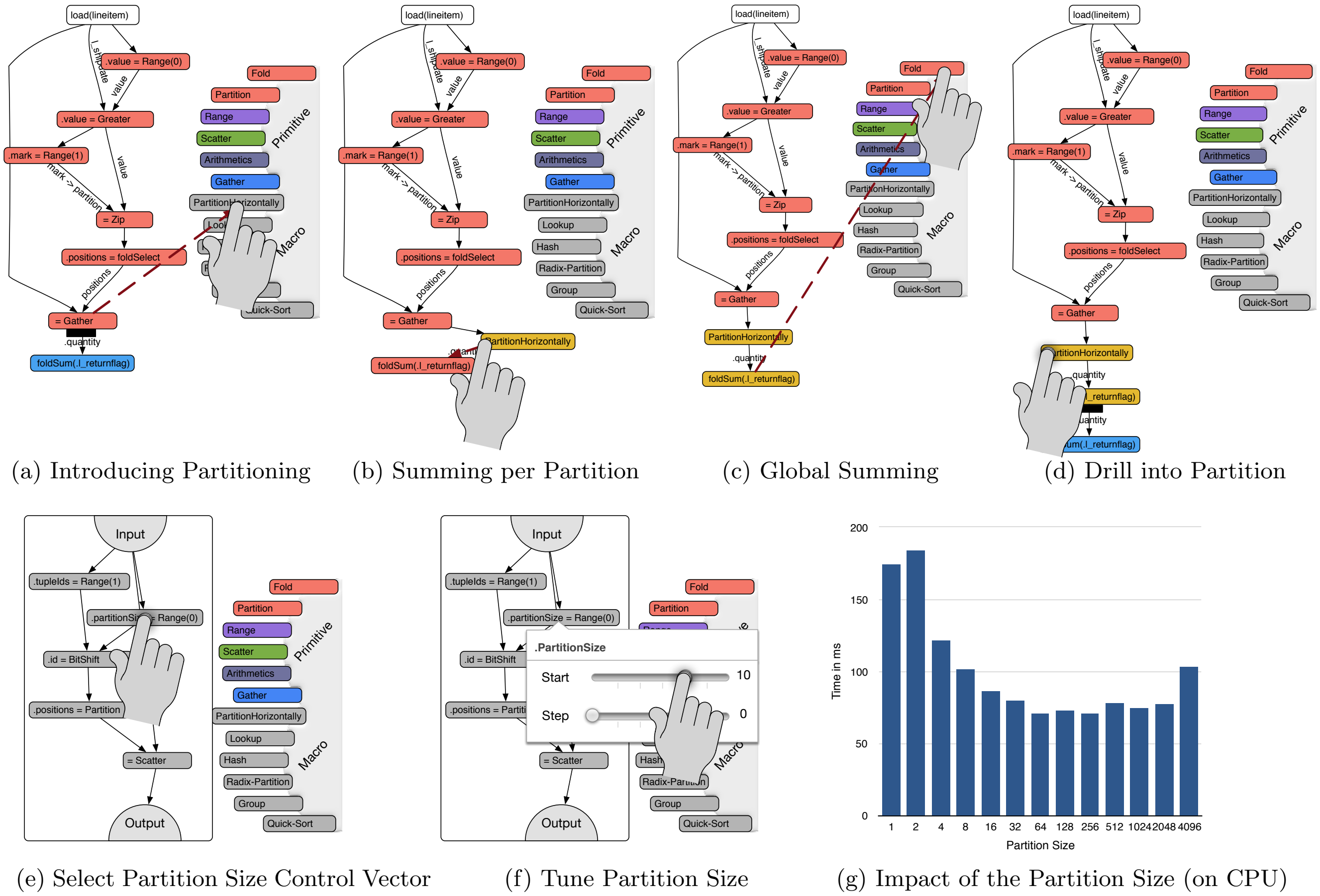
3.2 The Challenge

While we believe that the guided tour provides an insight into the difficulties of hardware-conscious optimization, we hope to entice attendants to try to beat our rule-based optimizer by tuning plans using our UI. However, the challenge is not as simple as it sounds: since *Candomblé* has the ability to run queries on different hardware platforms with varying costs for synchronization, materialization and even arithmetics on different datatypes[3], an optimal query plan for one platform may not translate to an optimal one for a different one. The attendants can experiment with tuning techniques such as a) eliminate redundant computation in an existing plan by sharing intermediates, b) replacing a multiplications with a bit shift, c) pushing aggregations through joins (in *Candomblé* -terms: replacing Add(Gather(A,X), Gather(B,X)) with Gather(Add(A,B),X)) or d) switch a selection from branching to branchfree or vice-versa.

3.3 Further exploration of rule based optimizations

In addition to manually changing sections of the graph, we allow the user to define new simple, i.e., wild-card free, rule-based optimizations and apply them to the full plan DAG as well as other queries. He does so by selecting a set of adjacent operators through a circling gesture which is then used

[3] On GPUs, for example, floating point arithmetic is usually cheaper than integer arithmetic



(a) Introducing Partitioning (b) Summing per Partition (c) Global Summing (d) Drill into Partition

(e) Select Partition Size Control Vector (f) Tune Partition Size (g) Impact of the Partition Size (on CPU)

Figure 3: The Interaction Flow: Introducing a 2 level hierarchical aggregation

as an example for a more generic pattern. The sytem hightlights other matches of this pattern throughout the DAG. The user can then continue to refine the pattern or move on to specify a transformation for that pattern through the *Candomblé* UI similar to Figure 3e. In the latter case, the UI compiles the proposed transformation into a pass over the expression DAG, executes it and displays the transformed graph as well as the new performance results.

4. CONCLUSION & FUTURE BENEFIT

We believe that *Candomblé* is a good abstraction of executable machine code for the purpose of implementing and tuning data-intensive algorithms. It provides a simple operator model that allow efficient examination, optimization and evaluation of a physical query plan. However, it also gives fine grained control over virtually every (static) aspect of the query plan.

In addition to these benefits, we feel that *Candomblé* is well suited to educate students, engineers and even non-experts about the challenges and benefits of hardware-conscious optimization. To facilitate this, both *Candomblé* and Candomblé *UI* will be available in open source upon demonstration. We hope and invite others to use *Candomblé* for teaching and prototyping.

5. REFERENCES

[1] Crotty, A., Galakatos, A., Dursun, K., Kraska, T., Cetintemel, U., and Zdoni, S. Tupleware:" big" data, big analytics, small clusters. In *CIDR* (2015).

[2] Klonatos, Y., Koch, C., Rompf, T., and Chafi, H. Building efficient query engines in a high-level language. *PVLDB* (2014).

[3] Neumann, T. Efficiently compiling efficient query plans for modern hardware. *PVLDB* (2011).

[4] Patterson, D. A., and Ditzel, D. R. The case for the reduced instruction set computer. *SIGARCH Comput. Archit. News* (October 1980).