RAM: array database management through relational mapping

van Ballegooij, A.R.

Citation for published version (APA):
Chapter 7

Conclusion and Future Work

This thesis set out to realize an extensible array-database architecture using relational mapping and existing relational database technology. To this end we have presented the Relational Array Mapping system (RAM) and discussed a variety of aspects regarding its mapping scheme throughout this thesis. This chapter concludes with a brief summary of the contributions made, the conclusions to be drawn, and a brief peek into the future of relational-array mapping.

7.1 Summary of Contributions

Chapter 3 presents a relational mapping scheme for array data and an associated declarative query language. The relational mapping evolved from the lessons learned from an early prototype [1] based on ideas outlined early on [2]. The query language, based on comprehension syntax and semantics, focuses solely on the array data-type. This explicit focus results in a system where the array paradigm can be studied without being side-tracked by irrelevant engineering problems. However, the deployment of this query language for real-world applications, explored in Chapter 6, is somewhat impaired precisely due to this limitation.

At the core of the system sits a query optimizer. This optimizer rewrites the internal algebraic representation of an array query based on equivalence rules, as discussed in Chapter 5. It is technologically inspired by the cost model driven query-rewriting approach to (relational) query optimization [3]. The performance experiments, presented in Chapter 6, show that this optimizer is effective [4]. Extensibility of the system has been shown [5] by extending the optimizer to distribute query processing over multiple sites.

The system includes modules to translate the intermediate array algebra to the native language of a variety of back-ends. Translators are provided for SQL, the industry standard for relational query languages; MIL, the native relational interface of MonetDB; scripts for Matlab, a widely used mathematical tool; X100, a fully vectorized next generation query processing engine for MonetDB; and low-level C++ programs.
The availability of mappings to different back-ends provides the opportunity to study the requirements imposed on the system by the characteristics of the platforms considered.

### 7.1.1 Conclusion

Chapter 6 shows that an array database system has the potential to make database technology interesting for a wide variety of computationally intensive problems. Integration of a multi-dimensional array data type and associated query facilities into an existing relational framework complements it with a suitable means to concisely express many computational problems. The relational mapping proposed in this thesis, the relational array mapping (RAM), is a viable approach to achieve this integration.

In this thesis we set out to achieve three goals, each contributing to the overall objective of an extensible array database architecture.

**The first goal was the specification of an efficient array-mapping scheme.** In Chapter 3 we presented such a scheme. The performance figures presented in Chapter 6 show that a level of performance competitive with native solutions is within reach: experimental evidence indicates that the RAM/MonetDB solution exhibits performance comparable to Matlab.

**The second goal was the exploration of query optimization at the array level.** Chapter 5 investigates array-query optimization based on (relational) query-optimization techniques and the performance evaluation in Chapter 6 shows its effectiveness. The key observation is that it makes sense to target optimization at the array level, rather than relying on the optimizer of the relational back-end.

Optimization at the array level has two advantages: First, the array domain allows for a simple yet effective cost model by providing exact (intermediate) result sizes. Second, optimization at the array level overcomes the inevitable loss of context that is a result form the translation of array queries to the relational domain. This loss of context impairs relational optimizers to recognize optimization opportunities easily recognized before the mapping. Examples of optimizations difficult to perform by a relational system without explicit knowledge of the array context are discussed in Chapter 5.

**The third goal was to show that translation of array operations directly into primitive relational operations allows for more efficient queries than high-level relational query languages.** Chapter 4 argues that the specific characteristics of a given back-end require special attention in the query-generation process of RAM. For example, the main-memory processing paradigm of MonetDB/MIL makes it essential to generate iterative query plans that reuse intermediate results and control memory usage, whereas the pipelined paradigm of MonetDB/X100 performs best on query plans that avoid intermediate materialization. This argument is supported by the experimental evidence in Chapter 6, which convincingly shows that these specialized query plans outperform the generic ones.
Back-end specific characteristics can be exploited only because RAM explicitly generates the query plan, which is, by design, not possible through a high-level declaratives query language such as SQL. Directly mapping into the relational layer allows the RAM optimizer to provide directly the context information about the query and the array domain. For this reason, it may be better equipped than a relational optimizer to generate a query-evaluation plan.

7.2 Future Work

The RAM system as presented in this thesis provides for the most part a positive answer to the research questions posed. As is, however, it has a few shortcomings that may interfere with its deployment for full-scale applications. In this section we briefly touch upon a number of these issues.

7.2.1 Set Integration

The RAM query language is explicitly limited to array structures, which means that it does not offer the means to express “selection” of elements based on their values. A lack of value-based selection does not seem problematic, at first, as we can manipulate values based on location. However, certain types of value-based operations are common and necessary.

For example, the case study recurring throughout this thesis is a retrieval application. While the RAM system allows a concise expression of the mathematics required to compute “scores” for documents in a collection, it lacks the means to sort these scores to produce a ranking of documents. It cannot sort because sorting is an operation that “selects” elements based on their values rather than their location in an existing array.

In order to improve usability in practice, value-based operations are essential. Naturally, a variety of such operations could be added to RAM as special functions, but that does not solve the real problem. The generic alternative is to integrate the array structures with sets, such that the value-based operations can be evaluated in the set domain.

The RAM systems is based on relational mapping and as such, array queries are evaluated by physically mapping them to the set domain. However, the details of this mapping are hidden from the user. In Chapter 3, “array-to-set” and “set-to-array” conversion operators were introduced to make the relational mapping process explicit. By providing these operations to the user, both an array-based and a set-based representation of the same data can be made available.
7.2.2 Control structures

Analysis tasks often consist of steps that are to be repeated a given number of times or until a certain condition is met. The query language of RAM does not offer any constructs to express such repetitions other than literally repeating the same query multiple times. However, given that a certain processing step (query) will be repeated multiple times, the optimal execution plan may differ from the non-repetitive case. For example, an optimizer could factorize out all parts of the query that are constant during the loop, thereby significantly improve performance. Hence a (conditional) loop construct may result in improved performance if the query optimizer is aware of it.

7.2.3 Sparse Storage

The relational mapping scheme presented in this thesis stores all values in a given array explicitly. Storing all elements explicitly is usually called a “dense” storage scheme. However in many application domains data can easily be compressed by using a “sparse” storage scheme. Many of these compressed storage schemes are known for arrays, the simplest variant is defining a default value (typically 0) and storing only those values that differ from it explicitly. This scheme is commonly used in linear-algebra applications.

Preliminary experiments using a sparse implementation of the RAM primitives in MIL have shown that results are promising. These results indicate that for arrays with up to 20% non-default values, the sparse implementation does not only reduce storage requirements, but is also more efficient.

The interesting aspect of using a “sparse” storage scheme is that it brings array query evaluation closer to the relational domain. In the “dense” case, all array elements are physically present and the optimal query plan is essentially that plan that scans through all that data quickest. In the “sparse” case, the optimization problem is suddenly back in the domain of relational systems: Efficient processing of “sparse” array queries requires efficient indexing schemes to retrieve data elements.
Bibliography


