When Data Management Systems Meet Approximate Hardware: Challenges and Opportunities

Bingsheng He
Nanyang Technological University, Singapore

ABSTRACT

Recently, approximate hardware designs have got many research interests in the computer architecture community. The essential idea of approximate hardware is that the hardware components such as CPU, memory and storage can trade off the accuracy of results for increased performance, reduced energy consumption, or both. We propose a DBMS *ApproxiDB* with its design, implementation and optimization aware of the underlying approximate hardware. ApproxiDB will run on a hybrid machine consisting of both approximate hardware and precise hardware (i.e., the conventional hardware without sacrificing the accuracy). With approximate hardware, ApproxiDB can efficiently support the concept of approximate query processing, without the overhead of pre-computed synopses or sampling techniques. More importantly, ApproxiDB is also beneficial to precise query processing, by developing non-trivial hybrid execution mechanisms on both precise and approximate hardware. In this vision paper, we sketch the initial design of ApproxiDB, discuss the technical challenges in building this system and outline an agenda for future research.

1. INTRODUCTION

In the last few decades, data management systems have been significantly improved and re-designed for performance and energy consumption. Among various factors, hardware evolution is one of the key and the enabling factors for driving the evolution of data management systems. Therefore, hardware-conscious (or architecture-aware) databases have been a fruitful research area [2, 10, 7]. We have witnessed fruitful research results and performance improvement brought by hardware-conscious optimizations, from CPU cache optimizations on addressing the memory wall, emerging storage techniques (such as solid state memories) on easing the I/O bottleneck, to multi-core/many-core processing for parallel databases. We strongly believe that hardware evolution continues to be a major driving force for architectural evolutions of future data management systems. "Prediction is very difficult, especially if it's about the future". The radical vision that we make in this paper is: data management on **approximate hardware**.

Recently, computer architecture researchers have proposed various emerging designs on approximate hardware, such as CPU [15, 12, 4], main memory [9] and storage [13]. The essential idea of approximate hardware is that the hardware components such as CPU, memory and storage can trade off the accuracy of results for increased performance, reduced energy consumption, or both. To illustrate, consider the approximate CPU. One may reduce the energy consumption of floating point operations, by ignoring part of the mantissa in the operands. As observed in the previous study [15], a floating-point multiplier using 8-bit mantissas reduces over 78% of energy per operation than a full 24-bit multiplier. Another example is approximate storage [13], which improves write latencies by 1.7× on average by trading off less than 10% of accuracy. We note that, approximate hardware does not mean that the hardware is fault-prone or unreliable. In fact, the error distribution of approximate hardware conforms to certain distributions (e.g., normal), which represent a programmable tradeoff between the accuracy and the performance/energy consumption. We give a quick introduction on approximate hardware in Section 2.

Approximate hardware opens up various optimization opportunities for the performance and energy consumption of data management systems. Approximate hardware can not only efficiently support approximate query processing (AQP), where users do not need accurate answers or it is too costly to get accurate answers, but also enables new optimization dimensions for precise query processing, where users require precise query answers (details are presented in Section 3.1). We propose a new data management system, ApproxiDB, with its design, implementation and optimization aware of the underlying approximate hardware. ApproxiDB will run on a hybrid machine consisting of both approximate hardware and precise hardware. It supports both AQP, and precise query processing. The interesting interplay between data management and approximate hardware raises various challenging issues on hardware and software co-design. In this vision paper, we sketch our initial design for ApproxiDB, along with the underlying challenges.

2. APPROXIMATE HARDWARE BASICS

We use approximate storage as an example for a closer look at approximate hardware. Approximate storage has been designed for solid-state memories, particularly for multi-level cell (MLC) configurations [13]. Examples of

Proceedings of the VLDB Endowment, Vol. 7, No. 10

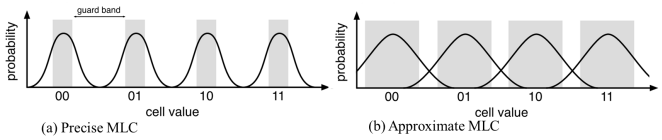

Figure 1: The probability of reading a given analog value after writing one of the levels in a precise (a) and approximate (b) four-level cell. The shaded areas are the target regions for writes to each level (figures are reproduced from [14]).

solid-state memories include flash and PCM (Phase Change Memory), which have been widely studied in databases (e.g., [8, 16]).

Let us first describe how precise storage works. Analog reads and writes are inherently imprecise. Thus, some *guard bands* are defined to separate different analog values so that they can safely represent different digital values. Figure 1 illustrates the idea of guard bands for a four-level cell design. On the other hand, due to process variation and nondeterministic material behavior, a single programming pulse typically has poor precision of achieving the guard band. Therefore, current MLC designs for both flash and PCM adopt iterative program-and-verify (P&V) mechanisms [11, 14]. In each P&V iteration, the cell's resistance is adjusted and read back to check whether the correct value has been achieved. The process continues until an acceptable resistance value has been set. The performance and energy consumption of a write are almost proportional to the number of P&V iterations during the write.

We now describe the design of approximate solid state memory proposed by Sampson et al. [13]. Unlike precise storage with relatively strict and wide guard bands, approximate storage reduces the guard bands to speed up iterative P&V writes at the cost of occasional errors. Figure 1 illustrates the idea of a smaller guard band in the approximate storage. Since the guard band is reduced, we need a smaller number of P&V iterations to achieve the acceptable accuracy.

Formally speaking, the performance/energy consumption of a write on approximate storage satisfy certain *monotonically increasing* functions: $t = T(c)$ and $e = E(c)$, where t, e and c ($0 \leq c \leq 1$) denote the elapsed time, energy consumption and the accuracy of the write, respectively. The accuracy c is defined to be $c = 1 - \frac{|v-v'|}{v}$, where v and v' are the actual value and the value that we write in, respectively. We also note that the distribution of errors $(1 - c)$ conforms to certain distributions (such as normal distributions for approximate MLC). Thus, the errors are bounded for each cell, and also we can derive the error bound for an access unit of the approximate storage. For example, the access unit for external memory is a page. We assume that each access unit of the approximate storage can be programmed with different accuracy settings.

Finally, the accuracy-performance/energy tradeoff on approximate storage is mainly for writes, and reads are almost the same as the precise storage. Nevertheless, storage reads may exploit accuracy-performance/energy tradeoff of other approximate hardware components in the machine, e.g., approximate main memory and CPUs.

3. APPROXIDB: AN INITIAL DESIGN

We start with identifying some concrete opportunities on optimizing data management systems on approximate hardware, and then give an initial design of ApproxiDB.

3.1 Opportunities

The approximate hardware brings a new and important dimension for database system design, ranging from operator implementation to query optimizations. There are many opportunities for performance/energy optimizations, if we examine the space exposed by approximate hardware. Particularly, we identify the following three kinds of optimization opportunities (***O1***, ***O2*** and ***O3***). In a nutshell, *O1* and *O2* represent the opportunities that approximate hardware can efficiently support approximate query processing, whereas *O3* represents the opportunities for improving precise query processing (i.e., the query requires precise results).

O1. The data are inherently imprecise, and thus can tolerate loss of accuracy. There are many such data from real-world applications such as sensor data, image, video and audio. For example, the reading from a temperature sensor may not need the accuracy to "last decimal". One can improve the performance and/or the energy consumption of writes (especially for writes in high-speed streaming sensor data) by strong the data on approximate storage/main memory.

O2. The query processing itself can tolerate loss of accuracy, and the result can be imprecise/approximate. There are many queries such as aggregate queries, where exact answers are not always required [6]. Approximate hardware can offer better performance/energy consumption by trade off the accuracy. Query processing of those queries or operators can be performed on approximate hardware. For example, the query execution can be performed on approximate CPU, and intermediate/final output can be stored on approximate storage/main memory.

O3. Although the query processing requires precise final result, a hybrid execution on precise hardware and approximate hardware could have better performance/energy consumption than the execution with precise hardware only. We propose an non-trivial paradigm for the hybrid execution: *Approximate-and-refine*. The basic idea has two steps: 1) we use approximate hardware in some of the processing steps to get some intermediate results (to be precise, they must form the superset of the final results), and 2) we run on precise hardware to refine the intermediate results and obtain the final result. Since approximate executions in Step 1) can be faster or more energy efficient than precise executions, this hybrid scheme wins when the refinement overhead of Step 2) is low. We give the following two examples to illustrate this paradigm.

EXAMPLE 1: selection. We use the selection as an example to illustrate the benefit of approximate CPU. As shown in Figure 2(a), a selection is performed on the precise CPU. Assume a scan on the entire relation, and we have eight precise operations on key comparisons. In contrast, we may consider some approximate execution (we simply assume the approximate CPU supports some approximate operation for key comparison with comparing the integer part only). A hybrid execution is illustrated in Figure 2(b). First, we perform the approximate execution on the relation. We refine the filtering condition to be $R.x >= 4.0$ and $R.x < 6.0$, and get two candidate results. Second, we perform the precise key comparison with the original condition ($R.x > 4.5$ and $R.x < 5.9$) and get the final output. In this hybrid execution, we have eight approximate key comparisons

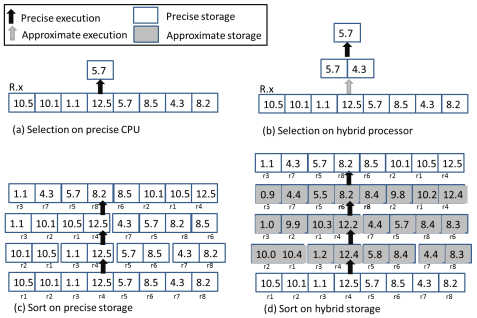

Figure 2: Examples on *O3*: (a, b) selection ($R.x > 4.5$ and $R.x < 5.9$), and (c, d) sort in increasing order.

and two precise key comparisons. If the cost of an approximate key comparison is 50% of the precise key comparison, the hybrid execution can be 33% faster than the precise execution in terms of key comparison costs. We note that the cost difference between the hybrid execution and the precise execution depends on several factors, including the accuracy of the approximate execution as well as the selectivities of the precise selection and the approximate selection.

EXAMPLE 2: sort. We use the merge sort as an example to illustrate the benefit of approximate storage. The idea is illustrated in Figures 2(c, d). Suppose the sort is performed on an input with each tuple consisting keys and record ids. The merge sort has two kinds of writes (writes for intermediate outputs and writes for the final output). In Figures 2(c), all writes go to the precise storage. In a hybrid execution, we consider writing the intermediate output to approximate storage (which can improve the performance/energy of those writes). Thus, the hybrid execution of merge sort has three stages: 1) in the first run of merge sort, we read from precise storage, sort the chunk and output the sorted chunk to approximate storage, 2) perform sorting entirely on approximate storage, and 3) refinement on precise storage. In Stage 2), the values have errors, and we assume that record ids are stored in the precise storage. After Stage 2), we get a nearly sorted output on approximate storage (in the example, only **r6** *and* **r8** *are out of order). In Stage 3), we apply the efficient adaptive sorting algorithm [5] and output the sorted data to precise storage. The hybrid execution can be faster, since reads are much faster than writes (reads are in the last stage of the hybrid execution) on current solid state memories, and writes on approximate storage are much faster than those on precise storage. Similar to the selection in Example 1, the accuracy is still a tuning parameter for the hybrid execution. We also note that, approximate CPU can be helpful in further reducing the cost of key comparisons.*

3.2 Initial Design

As a start, we focus on leveraging approximate storage and approximate CPU in ApproxiDB, and the main memory is precise. We discuss integration of other approximate hardware in Section 4. ApproxiDB runs on a hybrid machine consisting of both precise storage/CPU and approximate storage/CPU. This is because some data by design should be precise, for example, pointers and some critical data. Those data should be stored in the precise storage.

Our initial design for ApproxiDB extends an existing column-based DBMS (such as MonetDB). We choose a column-based DBMS other than a row-based DBMS, because column stores are more efficient for approximate storage. Figure 3 shows the architecture of ApproxiDB in our initial design. The storage and CPU have the

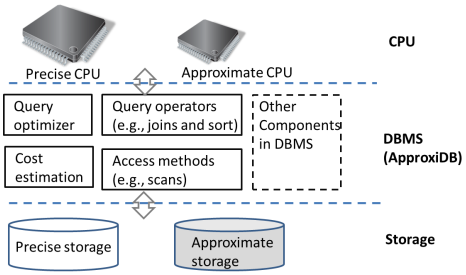

Figure 3: The architecture of our initial design of ApproxiDB

hybrid design. For either CPU or storage layer, precise hardware and approximate hardware can be loosely or tightly integrated. For example, the precise CPU and the approximate CPU could be integrated on a single die or different dies.

The following extensions are made to the existing DBMS.

We extend the physical design (e.g., CREATE TABLE/VIEW) by allowing users to specify which column can be stored in approximate storage as well as the accuracy requirement. Given the accuracy requirement on each column, we are able to figure out the accuracy requirement on each access unit of approximate storage. The calculation depends on the data types, and we can adopt the existing technique [12].

With hybrid CPUs and storage, we have four query processing modes: *M1*) precise query execution on precise data, which is the conventional processing without approximate hardware, *M2*) precise query execution on approximate data, *M3*) approximate query execution on precise data, and *M4*) approximate query execution on approximate data. To exploit optimization opportunities (*O1–O3*), ApproxiDB chooses suitable modes to support both approximate query processing and precise query processing.

Particularly, the access methods (such as table scan and indexes) and relational operators in the existing DBMS can be extended to support these four query processing modes. Each method and each operator need to support those four modes. We adopt the overloading concept to select the suitable mode according to the accuracy requirement, in order to keep the interfaces of access methods and operators unchanged. Basically, before performing any of those operations, we need to first determine the data source (either precise storage or approximate storage), and next to select the suitable CPU instructions according to the accuracy requirement (e.g., choose adders with different accuracy levels), and then execute them in the same way as the original algorithm (either on precise CPU or on approximate CPU). That means, we *pretend* that the data stored in approximate storage as accurate in query processing, which is quite different from existing probabilistic query processing [3]. The intermediate outputs are stored in the approximate storage, if 1) the input data to the operation is already stored in the approximate storage, and/or 2) the operator can be executed in the approximate manner, such as aggregate queries. We allow users to annotate an SQL query to specify the query operator that can execute on approximate storage/CPU.

For the query optimizer and the cost model in the DBMS, the major change is that the cost model needs to consider the performance/energy consumption tradeoff of approximate hardware and the statistics (updated by the error generated

from approximate hardware).

Still, we will develop new query processing schemes or query optimization mechanisms for *O3*. Particularly, we will revisit the physical operator implementation as well as query processing executions to see how we can exploit the third kind of optimization opportunities (*O3*). As we can see from the examples, there are tuning knobs on the accuracy as well as other database statistics that affect the effectiveness of the approximate-and-refine paradigm. We plan to use the extended cost model to determine the suitable accuracy so that the performance/energy consumption is optimized.

4. EXTENSIONS AND OPEN PROBLEMS

Automated physical design: We may wish to reduce the burden of physical design from users (e.g., deciding whether a column can be approximate or not), and also to enable more fine-grained design (for example, we may store only some frequently written tuples to approximate storage, instead of the entire column). ApproxiDB might be able to automate the physical design by extending the existing automated physical design techniques in databases.

Multi-level approximate hardware: There have been designs on different kinds of approximate hardware, including CPUs, main memory and storage. If we apply more than one kind of approximate hardware, we may have various challenging and interesting problems. We give two examples. First, errors are propagated along the execution flow among different kinds of approximate hardware. Second, there can be data consistency issues. Consider the buffer management if we read data from the approximate storage and store the data to the buffer in the approximate memory. The data are inconsistent among the storage and the buffer, which makes the data flow among different approximate hardware complicated.

Probabilistic query processing. Probabilistic databases [3] are relevant to ApproxiDB, because the data in the approximate hardware are uncertain. Approximate hardware usually offers the error distributions [13], which make the techniques in probabilistic databases feasible to query processing on approximate hardware. That makes an interesting case for hardware and software co-design.

Query-level tradeoff between accuracy and performance/energy: Instead of letting users specify the accuracy requirement on each column and operator, we may expose the tradeoff between accuracy and performance/energy to users. Particularly, when a user submits a query, she may specify the tradeoff as well (e.g., minimizing the execution time when the accuracy of the query output is higher than 80%). This adds a multi-criteria optimization requirement on query optimizer. We will develop a new DBMS optimizer that incorporates the user inputs on error tolerance and the tradeoff on accuracy and performance/energy consumption of approximate hardware, and generates the optimized query plan given the accuracy requirement. As a result, the query processing output of ApproxiDB will comprise not only the result but also the accuracy of the output.

5. RELATED WORK

Approximate query processing (AQP) has been widely studied in the literatures [6, 1]. AQP mainly leverages some pre-computed synopses or sampling techniques to answer queries. Unlike AQP, ApproxiDB does not rely on any pre-computed synopses or sampling. With little storage overhead and precomputation overhead, ApproxiDB may be an ideal candidate for mobile platforms or sensors. More importantly, ApproxiDB can also improve the performance/energy of precise query processing.

Along the line of architecture-aware databases [2, 10, 7], ApproxiDB investigates a new dimension of hardware features – the tradeoff on the accuracy and performance/energy consumption. More broadly, we not only need to deal with a certain kind of hardware (either approximate or precise), but also consider non-trivial interactions and collaborations between precise hardware and approximate hardware.

6. CONCLUSIONS

We outline our radical vision for ApproxiDB: a data management system that runs on hybrid hardware with both approximate hardware and precise hardware. We demonstrate the optimization opportunities for both approximate and precise query processing. We have defined a number of concrete problems in architecting ApproxiDB, ranging from physical design and query processing to multi-criteria optimization. We believe that approximate hardware will bring various research challenges and opportunities to database management research as well as interesting problems on hardware-software co-design. Also, we conjecture that the promising result on *faster* and *greener* databases on approximate hardware can drive the development and implementation of other data processing (e.g., key/value stores and transaction processing) on approximate hardware.

Acknowledgement

The author would like to thank Yinan Li, Qiong Luo, Saurabh Jha, Mian Lu and anonymous reviewers for their insightful comments.

7. REFERENCES

[1] S. Acharya and et al. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[2] J. Cieslewicz and K. A. Ross. Architecture-conscious database system. In *Encyclopedia of Database Systems*. 2009.

[3] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 2007.

[4] H. Esmaeilzadeh and et al. Architecture support for disciplined approximate programming. In *ASPLOS*, 2012.

[5] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 1992.

[6] M. N. Garofalakis and P. B. Gibbon. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[7] B. He and et al. Relational query coprocessing on graphics processors. *ACM TODS*, 2009.

[8] Y. Li and et al. Tree indexing on solid state drives. *Proc. VLDB Endow.*, pages 1195–1206, 2010.

[9] S. Liu and et al. Flikker: Saving dram refresh-power through critical data partitioning. In *ASPLOS*, 2011.

[10] S. Manegold and et al. Database architecture evolution: Mammals flourished long before dinosaurs became extinct. *PVLDB*, 2009.

[11] M. K. Qureshi and et al. Morphable memory system: A robust architecture for exploiting multi-level phase change memories. In *ISCA*, 2010.

[12] A. Sampson and et al. Enerj: Approximate data types for safe and general low-power computation. In *PLDI*, 2011.

[13] A. Sampson and et al. Approximate storage in solid-state memories. In *MICRO*, 2013.

[14] K. Takeuchi and et al. A multipage cell architecture for high-speed programming multilevel nand flash memories. *Solid-State Circuits, IEEE Journal of*, 1998.

[15] J. Y. F. Tong and et al. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE TVLSI*, 8(3), 2000.

[16] S. Viglas. Write-limited sorts and joins for persistent memory. *Proc. VLDB Endow.*, 2014.