A Tunable Cache for Approximate Computing

Magnus Själander, Nina Shariati Nilsson, and Stefanos Kaxiras
Uppsala University, Department of Information Technology
P.O. Box 337, SE-751 05 Uppsala, Sweden
{magnus.sjalander, nina.shariati-nilsson, stefanos.kaxiras}@it.uu.se

***Abstract*—CMOS scaling is near its end but new emerging devices are being developed to replace CMOS. These devices have different features than CMOS, such as the possibility for multi-value logic, which present new opportunities when designing computer systems. In this work we investigate the use of multi-value devices to design a cache that can tune the amount of resources used to store application data. We leverage work on approximate computing to store data that are not application critical in a compact quaternary format while critical data is stored in a more error resilient binary format.**

I. Introduction

CMOS scaling is already facing many challenges and is likely to reach its end in the near future. New emerging technologies, e.g., single atom transistor (SAT) [1] and single electron transistor (SET) [2], are promising CMOS alternatives for the continuation of Moore's law. However, shrinking the semiconductor devices down to a single atom or electron raises concerns regarding their accuracy.

Approximate computing is an approach where accuracy is traded against other factors like performance, power, and energy [3]. Previous work has shown that approximate computing can be used to improve performance and endurance for multi-level phase change memories [4]. In this work we trade accuracy against storage resource requirements for on-chip caches. When an application provides opportunities to relax the accuracy then data are stored as quaternary words in multi-value cells, while precise data are stored as binary words in twice the number of multi-value cells. We present a cache organization where a cache line can either hold approximate quaternary or precise binary words. This provides a tunable memory hierarchy where the amount of resources can be allocated depending on the accuracy needed by the application.

II. Device Characteristics

We consider any device where the output can be described as a continuous transfer function of its inputs. For multi-value logic this transfer function contains plateaus that presents a relatively stable output even if the input value varies slightly [5]. These plateaus appear as discrete ranges that can be used to represent logic states. Such devices have higher probability that the output of a device will drift, due to interference, to a nearby value than to a value far away from the expected value. Here we use the term interference for any physical property that affects the output such that it deviates from the ideal transfer function, e.g., input noise or manufacturing variances. We take advantage of this property to create two logic representations for representing approximate and precise data.

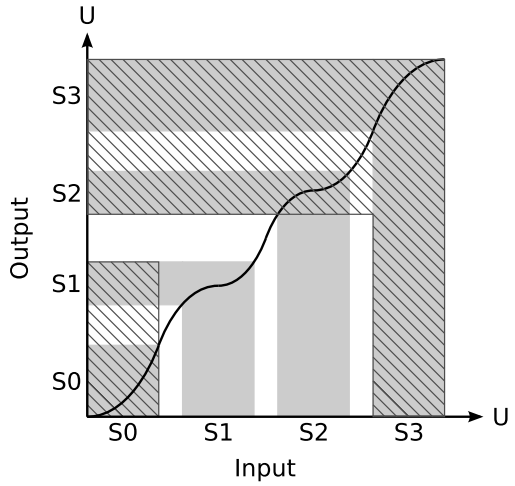

Fig. 1. Illustration of an input to output transfer function of a fictitious quaternary device. Gray regions represent the input and output ranges when working with quaternary data. Striped regions represent input and output ranges when working with binary data.

Fig. 1 illustrates a transfer function for a fictitious quaternary device in some unit (U) that could be, e.g., voltage or current. The figure shows in light gray the four different states (S0, S1, S2, and S3) and how they are represented as ranges on the input and output of the device. When representing approximate data we use all logic states of the device. This maximizes the amount of data that can be represented per device, but it also increases the probability that a state drifts to a nearby state (e.g., S2 gets interpreted as S1 or S3). Precise data is represented in binary form by only using the maximum (S3) and minimum (S0) states as input to the device. When the output is read nearby states are also considered as part of the maximum (S2 and S3) and minimum (S0 and S1) states (see striped regions in the figure). This increases the reliability of the device as the probability is low that S0 would drift all the way to S2 or that S3 would drift to S1. However, it also requires twice the number of devices compared to representing the data approximately.

III. A Tunable Approximate Cache

We use the device characteristics presented in the previous section to design a cache memory that can store both approximate and precise data. This enables a tunable memory hierarchy where (1) more memory resources can be allocated to increase the reliability of precise data or (2) approximate data can be stored compactly.

We assume a system where we known what data can be stored approximately and what data must be stored precisely. Such information can be extracted through compile time

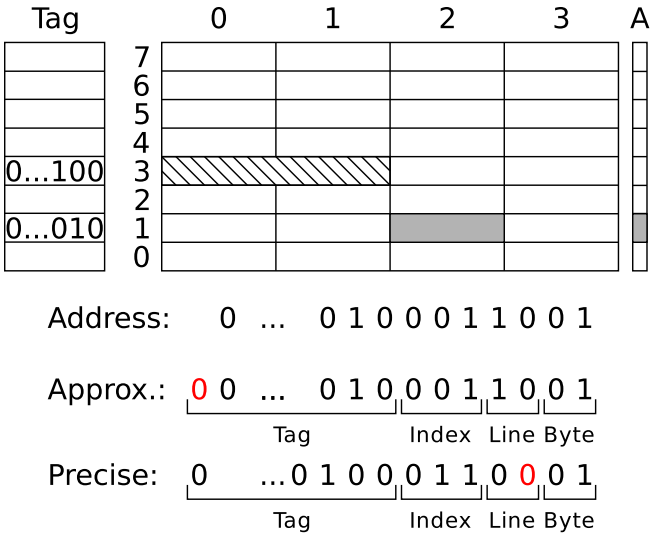

Fig. 2. A tunable approximate direct mapped cache, with eight sets, four approximate words per cache line ($N = 4$), and four bytes per word.

analysis [6], the use of approximate frameworks [7], or a combination of both. The information can then be provided to the hardware either by special load and store instructions or by placing approximate data in separate memory pages.

The cache is constructed out of quaternary memory cells [8]. Each memory cell can store either one out of four approximate values (0, 1, 2, 3) or one out of two precise values (0, 1). The cache is organized such that each cache line can store N approximate words. A cache line can also store precise data by combining pairs of approximate words into precise word (see striped word in Fig. 2). Each cache line has a bit (A) that is set if the data in that cache line is stored approximately. Thus, a cache line can only store approximate or precise data, not a combination of both.

As the size of the data depends on if the data are stored approximately or precise a new addressing scheme needs to be devised. Fig. 2 shows how a conventional address produced by an application is treated by the cache depending on if the data are approximate or precise. The illustration shows a simple direct mapped cache with four bytes per word, four approximate words per cache line, and eight sets. The cache is also assumed to be word addressable.

When referencing approximate data the address is treated the same way as in a conventional cache. The set of least significant bits constitutes the *Byte* offset, which is used to address specific bytes within a word (the illustration depicts a word size of four bytes). The next set of bits is the *Line* offset, which is used to address the word to read/write from/to the cache line. The following set of bits is the line *Index*, which identifies the row in the cache. The remaining bits are used as a *Tag* to identify a particular cache line stored in the cache. The tag is appended with a zero at its most significant bit (depicted in red). This is required to create a tag that is of the same size as the tag needed to store precise data.

When referencing precise data a cache line can only hold half the number of words as a cache line with approximate data. Therefore, the address needs to be treated differently than for the approximate case. Each precise word consists of the same number of bytes as an approximate word so the same set of least significant bits is used as the byte offset. The line offset has to be handled differently though. Each precise word takes up the space of two approximate words and has to be placed on a double word boundary (starting at offset 0 or offset 2 for the case illustrated in Fig. 2). The address is therefore shifted left one step and a zero is inserted (depicted in red). The shifted address is used to select the bits representing the line offset and index. As the address has been shifted one step to the left the portion for the tag is now one bit larger than for a conventional cache with the same cache configuration. This is the reason why the tag of approximate addresses has to be appended with a zero.

This addressing scheme assures that precise words are aligned correctly. It also assures that the address mapping for precise data is without any holes or overlaps, which is also true for the approximate address mapping. It does have the effect that the same address can get mapped to different cache lines depending on if the referenced data are approximate or precise (see marked words in Fig. 2). It is therefore preferable that the compiler keeps precise and approximate data separated and aligned at cache line boundaries.

A cache line can only store approximate or precise data. When a reference to a cache line is made the approximate bit for that cache line is checked (A in Fig. 2). If the type of the reference being made and the stored data differ then the access is treated as a miss. A conventional replacement policy can be used to evict a cache line and replace it with the correct data.

IV. Conclusion

The combination of approximate computing and emerging multi-value devices enables the design of a tunable memory hierarchy. In our proposed system approximate data is compactly stored in a quaternary format while precise data is stored in a binary format that requires twice the number of memory cells but significantly decreases the probability for errors.

References

[1] J. Verduijn, G. C. Tettamanzi, and S. Rogge, "Wave function control over a single donor atom," *Nano Letters*, vol. 13, no. 4, 2013.

[2] V. Deshpande, R. Wacquez, M. Vinet, X. Jehl, S. Barraud, R. Coquand, B. Roche, B. Voisin, C. Vizioz, B. Previtali, L. Tosti, P. Perreau, T. Poiroux, M. Sanquer, B. De Salvo, and O. Faynot, "300 K operating full-CMOS integrated single electron transistor (SET)-FET circuits," in *Electron Devices Meeting*, Dec. 2012, pp. 8.7.1–8.7.4.

[3] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Test Symposium*, 2013, pp. 1–6.

[4] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *International Symposium on Microarchitecture*, Dec. 2013, pp. 25–36.

[5] M. Seo, C. Hong, S.-Y. Lee, H. K. Choi, N. Kim, Y. Chung, V. Umansky, and D. Mahalu, "Multi-valued logic gates based on ballistic transport in quantum point contacts," *Scientific Reports*, vol. 4, no. 3806, Jan. 2014.

[6] M. Carbin and M. C. Rinard, "Automatically identifying critical input regions and code in applications," in *International Symposium on Software Testing and Analysis*, Jul. 2010, pp. 37–48.

[7] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Programming Language Design and Implementation*, Jun. 2011, pp. 164–174.

[8] H. Inokawa, A. Fujiwara, and Y. Takahashi, "A multiple-valued SRAM with combined single-electron and mos transistors," in *Device Research Conferenc*, Jun. 2001, pp. 129–130.