Quality Aware Approximate Memory in RISC-V Linux Kernel

Giulia Stazi, Antonio Mastrandrea, Mauro Olivieri, Francesco Menichelli
Sapienza University of Rome
Dept. of Information Engineering, Electronics and Telecommunications (DIET)
Rome, Italy
Email: {g.stazi,antonio.mastrandrea,mauro.olivieri,francesco.menichelli}@uniroma1.it

***Abstract*—Improving power consumption and performance of error tolerant applications is the target of the design paradigm known as approximate computing. The memory subsystem is one of the units of a computational architecture where approximations can be introduced, leveraging on the resilience of an application to maintain an acceptable output quality even if its input data are subject to imprecision and errors.**

This paper proposes and implements the management, in the Linux kernel, of multiple approximate memory banks. Applications can then allocate approximate memory for their data structures selecting between different levels of approximation, depending on the requirements on output quality. This allows to design an architecture where approximate physical memory, instead of being composed of a unit intercepting a single point in the energy-quality tradeoff curve, can be split into multiple banks trading off levels of approximation and energy savings.

We finally show a case study in the results, where we explore the allocation of different data structures of a signal processing application, depending on sensitivity to errors and desired output quality.

***Index Terms*—Approximate Computing, Energy Quality Tradeoff, Low Power**

I. Introduction

The increasing requirements on energy consumptions and performance in modern digital systems have led to the research of new design approaches that were able to go beyond the established energy-performance tradeoff. A viable proposal, known as approximate computing, has demonstrated to be particularly prolific. Many applications in the domain of signal processing, multimedia, computer vision, machine learning, etc. (ETAs, Error Tolerant Applications) are known to be particularly resilient on errors occurring on their input data and during computation, producing an output that, although degraded, is still largely acceptable from the point of view of quality. Approximate computing design paradigm leverages on ETAs characteristics to develop circuits, architectures, algorithms that perform their computations in an approximate or inexact manner, saving on energy consumption.

According to this paradigm, approximate memories are memory circuits designed with relaxed constraints on data integrity in exchange of large energy savings during operations. Depending on the technology (i.e SRAM, DRAM), many implementations have been proposed at circuit and architectural level [1]–[4].

In general ETAs, albeit being error-resilient on a large portion of their input data, will require also exact storage for part of their data structures (e.g. see the distinction between critical and non-critical data in [5]). Moreover considering that operating systems are not error tolerant, keeping a portion of exact memory is always required.

Once asserted the possible energy benefits of systems with approximate memory, their management has been considered and implemented in lightweight [2] and full operating systems [6]. In this way, after characterizing and profiling their memory allocation requests, applications and algorithms can be converted to straightforwardly use approximate memory for portions of their data structures [7].

This paper proposes a new approach to introduce, in the Linux OS, the ability to allocate approximate data in separate memory zones according to quality requirements. The potential of having quality aware memory zones opens the way to further investigations, tailoring allocations of approximate memory depending on, for example: (a) different sensitivity of output quality to errors in input data structures; (b) variable-time output quality requirements; (c) requirements of different applications in a multitasking environment.

The new kernel can run in architectures containing several physical memory banks with different levels of approximation. In this context, we will refer often to the term *level of approximation* of a memory in order to classify different approximate memories. This definition is related to error rate (i.e. higher error rate corresponds to higher level of approximation), but also, depending on approximate memory circuits, to the weight of bits affected by errors (i.e. on equal conditions, a memory with approximate cells limited to the least significant bits of a word has a lower level of approximation [1]). The multiple levels of approximation could be realized, for example, using several DRAM banks with different refresh rates, lower than required by specifications.

In particular, this work makes the following contributions:

- we introduced approximate memory support for the first time in 64-bit Linux kernel, specifically for RISC-V architectures [8];
- we introduced the capability of configuring up to four approximate memory zones (in addition to standard Linux

978-1-7281-3549-6/19/$31.00 ©2019 European Union

memory zones), where each of these zones corresponds to physical memory with a certain level of approximation;
- we implemented an internal data allocation scheme, capable of handling separately the allocation requests in quality aware approximate zones;
- we developed a user space data allocation mechanism and support library.

We tested and evaluated our implementation on the emulator AppropinQuo [9], after integrating the latter with models of quality-configurable approximate memory regions.

The rest of this paper is organized as follows. Section II presents the current state of the art. Section III illustrates the implementation of the quality aware approximate memory zones in 64-bit Linux kernel. Section IV describes the simulation setup in AppropinQuo and the results on a case study.

II. Related Work

Approximate computing techniques have been explored for arithmetic units and memory units. Concerning the former, authors in [10] present the implementation of variable bit approximate addition and multiplication on a RISC-V platform, exploring the effects on a filter application.

In [2], [11] the authors introduce a methodology for constructing a quality aware approximate memory system based on DRAM. The core idea is to refresh DRAM with a single but reduced rate, characterizing portions of the memory array and splitting them in several *quality bins*, based on the frequency, location and nature of bit errors in each physical page. During program execution, non-critical data can be allocated to bins sorted in descending order of quality. The setup included the use of the lightweight operating system μC/OS-II for memory management and task creation. However, the paper proposes to use the *quality bins* in a descending order, ensuring that lower quality bins (i.e. having a higher level of approximation) are always used as last resource. The work does not explore the possibility of selecting the *quality bins* at program level, depending on data. In this paper we take the notion of quality bins in approximate memory and implement their management in the Linux Kernel OS.

In [6] the approximate memory support on 32-bit Linux OS is described. Approximate memory management has been integrated in the kernel memory management, relying on the internal concept of Linux *physical zone*. In this way the Linux kernel is aware of exact physical memory pages (grouped in ZONE_DMA, ZONE_NORMAL and ZONE_MOVABLE) and approximate physical memory pages (grouped in a new ZONE_APPROXIMATE), managing them as a whole for the common part (e.g. optimization algorithms, page reuse, defragmentation) but distinguishing them in terms of allocation requests and page pools management. Compared to that work, this paper includes the approximate memory support to 64-bit architectures, that can manage larger memory sizes, but also has the ability to support up to 8 memory zones (32-bit kernel is limited to four zones, including the always active ZONE_NORMAL and ZONE_MOVABLE). We also inserted the instantiation and management of up to four approximate zones, each one corresponding to physical memory pages with different levels of approximation. The whole system was finalized and compiled for RISC-V 64-bit architecture and executed on AppropinQuo emulator [9].

III. Quality Aware Approximate Memory Zones in Linux OS

A. Memory Zones

Introducing multiple approximate memory zones within the Linux OS is mainly architecture independent, while a reduced number of modifications (as the definition of the memory map) is required in architecture dependent source files. For this first implementation, we chose as target the RISC-V 64-bit architecture.

The architecture independent part includes the creation of new memory zones and the implementation of the corresponding data allocation policy. Both should be consistent with the requirements for the approximate memory management already defined in [6]. We should note that, due to the Linux kernel memory management implementation, on 32-bit architectures it is possible to define and create up to 4 memory zones, while on 64-bit architectures this limit is extended up to 8. Considering that ZONE_NORMAL and ZONE_MOVABLE are always enabled and required, while ZONE_DMA and ZONE_DMA32 could be enabled depending on architecture requirements for managing DMA devices, on 64-bit architectures, with the current implementation, it is possible to create up to 4 zones for approximate memory (that we called ZONE_APPROXIMATEx, $x = 1...4$). The rationale behind these multiple zones is that each ZONE_APPROXIMATEx is filled with pages backed by physical memories with different, and *decreasing*, level of approximation (i.e. the approximate memory zone with the lowest index corresponds to memory with the highest level of approximation).

Defining an order is important since it has an impact on internal allocation policies. The former organization is compliant with the fallback mechanism of the Linux OS, which is activated if a memory zone is not able to satisfy an allocation request. In other words, if an allocation request for memory with a certain level of approximation cannot be satisfied (e.g. because the requested size is not available), the allocator will fallback to a hierarchically higher approximate zone, characterized with a lower level of approximation, up to the exact zones (ZONE_NORMAL, ZONE_DMA, etc.).

The mapping of the layout of approximate memory zones into physical RAM is architecture dependent. A function inside the kernel computes the available physical memory; this information is then used in the initialization phase to group physical memory pages into memory zones. Each zone must be characterized by its *start pfn* and *end pfn* (page frame number), corresponding to the physical address bounds of each memory (expressed in as $page\_number = physical\_address/page\_size$). On the 64-bit RISC-V architecture two memory zones are present by

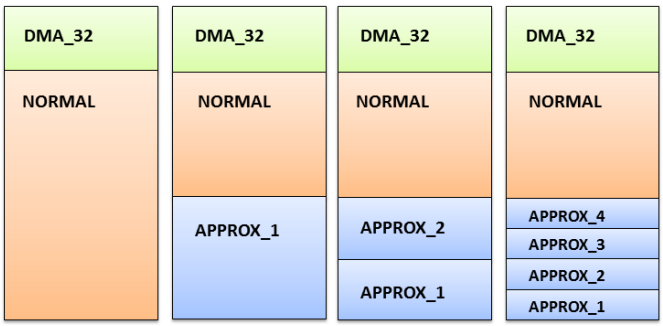

Fig. 1. Configuration of physical memory layout

default: ZONE_DMA32 and ZONE_NORMAL. In order to introduce quality aware approximate zones, we partitioned the physical memory into five parts. The first one corresponds to exact memory and it will be used for ZONE_DMA32 and ZONE_NORMAL; the others are used for approximate memory zones (1 to 4). The *pfn* bounds of these zones are assigned statically, depending on the number of quality aware memory zones that are present (Fig. 1). Moreover, the *start pfn* of the highest enabled ZONE_APPROXIMATE is used as the current limit inside the allocation algorithm of the Linux *bootmem allocator*. This implementation ensures that pages belonging to ZONE_APPROXIMATEx are never selected by the kernel to initialize page allocators data structures.

B. Data Allocation

The Linux OS manages each allocation request using a set of internal flags, called *GFP flags*, in order to drive the allocation algorithm. These flags are used, among others, to define which memory zone should be selected for the current request. More specifically, the zone requested is not completely binding since there are policies (fallbacks) that allow to go back in the memory zone hierarchy (e.g. in case a memory zone is full).

To correctly manage allocation requests in multiple approximate memory zones, it was necessary to define new *GFP flags*, one for each approximate memory zone (*GFP_APPROXIMATE*x, $x = 1...4$). According to the requirements described in [6], the priorities and fallback mechanism were configured to ensures that (a) the allocation in approximate zones can only take place on explicit request; (b) the memory zone hierarchy guarantees that the fallback mechanism will always move from a higher to a lower level of approximation.

To allow user space applications to request data allocation in different approximate memory zones, we implemented a new *approx_malloc* library function. This function takes as input parameters the *size* of data that should be allocated and the *level* of approximation required. The *level* parameter is propagated inside the kernel and then it is associated to the GFP flag of the corresponding approximate memory zone.

```
...
Zone ranges:
DMA32 [mem 0x0000000080200000-0x0000000080ffffff]
Normal [mem 0x0000000081000000-0x0000000087ffffff]
Approximate2 [mem 0x0000000088000000-0x000000008bffffff]
Approximate [mem 0x000000008c000000-0x000000008fffffff]
Movable zone start for each node
Early memory node ranges
node 0: [mem 0x0000000080200000-0x000000008fffffff]
...
```

Fig. 2. Boot messages printing the physical memory layout

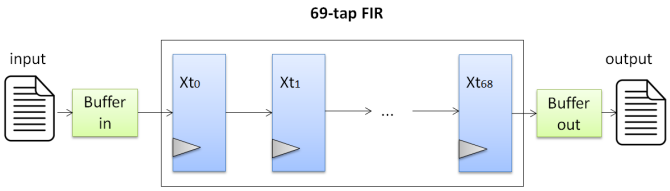

Fig. 3. Digital FIR architecture

IV. Experimental Results

A. Evaluation setup

We analyzed the usage of quality aware approximate memory in an application, setting up an emulation of a hardware platform with the following characteristics:

- RISC-V 64-bit CPU, for which the kernel was compiled;
- RISC-V *SiFiveU* platform, with 256MB RAM memory. The 256MB are partitioned into 128MB exact RAM, 64MB approximate memory (level 1), 64MB approximate memory (level 2), as shown by the boot messages in Fig. 2;
- destructive error on read (EOR, see [9] for the definition of error on access for approximate RAM).
- destructive error on write (EOW, see [9] for the definition of error on access for approximate RAM).

The AppropinQuo emulator [9] was used for the whole setup.

B. Case study

As case study we present the application of quality aware memory allocation on a digital filter, as a typical ETA, working on audio signals. We chose a software FIR filter, composed of 69 taps and implemented in C language using 32 bit integer arithmetic.

In particular, Fig. 3 shows the diagram of data (audio samples) flowing from the input to the output. The application has been implemented in order to allocate the input and output buffers (indicated in green in Fig. 3) in ZONE_APPROXIMATE1 and the tap registers (showed in blue) in ZONE_APPROXIMATE2. This choice is due to the fact that, in the implementation, the input and output buffer locations are accessed less frequently with respect to the tap registers.

Table I reports the relevant data. The input and output buffers size is 100Kbyte (containing 25,000 32-bit samples), while the tap registers size is 276 bytes (for 69 32-bit registers). Access tracing reveals that, as expected, the tap registers

array is accessed about two orders of magnitude more than the input and output buffers.

We analyzed different combinations of levels of approximation for ZONE_APPROXIMATE1 and ZONE_APPROXIMATE2. Considering the order showed in Fig. 1, fault rate of ZONE_APPROXIMATE1 is higher or equal than ZONE_APPROXIMATE2. An interesting point is when fault rate is equal, since it corresponds to the case of having just one ZONE_APPROXIMATE for all non critical data.

Table II and Table III show the results considering the two opposite corners of an approximate SRAMs, EOR and EOW. As quality metric, we used SNR, measured considering noise as the difference between the output of the exact filter and the output of the approximated filter. The diagonal values correspond to the case of a single ZONE_APPROXIMATE for all approximate data; on a row, moving from the diagonal to the adiacent element reveals that if tap registers are allocated in memory with a fault rate 10 times lower, a gain of 7 to 8 dB in SNR is obtained for the EOR case. This gain is quite repeatable across all cases, while further reducing fault rate for factors of 100, 1000, etc. produces minor advantages. Table III shows how the same concept is valid in the EOW corner, but, since in this case study tap register reads are about twice than write accesses, SNR gain is 6 to 7 dB.

TABLE I
FIR, ACCESS COUNT ON APPROXIMATE DATA STRUCTURES

	buffer_in	**buffer_out**	**tap regs**
size [bytes]	100,000	100,000	276
#read/location	196	196	10,035,200
#write/location	196	196	5,017,601
#total_reads	4,900,000	4,900,000	692,428,800
#total_writes	4,900,000	4,900,000	346,214,469

TABLE II
FIR, OUTPUT SNR $[dB]$ FOR SRAM, EOR

	Fault rate (buffers) $[errors/access]$	**Fault rate** (taps) $[errors/access]$				
		10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
EOR	10^{-2}	32.9	37.52	43.64	44.17	44.19
EOR	10^{-3}	_	42.8	50.92	53.57	53.96
EOR	10^{-4}	_	_	53.4	60.85	63.84
EOR	10^{-5}	_	_	_	63.09	71.01
EOR	10^{-6}	_	_	_	_	72.92

TABLE III
FIR, OUTPUT SNR $[dB]$ FOR SRAM, EOW

	Fault rate (buffers) $[errors/access]$	**Fault rate** (taps) $[errors/access]$				
		10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
EOW	10^{-2}	35.3	41.18	43.94	44.12	44.1
EOW	10^{-3}	_	45.5	52.17	53.34	53.79
EOW	10^{-4}	_	_	56.4	61.58	63.19
EOW	10^{-5}	_	_	_	65.69	69.84
EOW	10^{-6}	_	_	_	_	78.11

V. CONCLUSION

In the present paper we describe the implementation, inside the Linux kernel, of the management of approximate memory, including support for multiple zone with different levels of approximation. Applications can select the level of approximation of their data structures, trading off more efficiently the approximation level of data (and, hence, energy consumption) with quality of the output.

An example case study was analyzed, showing how, even with just two levels of approximation and manual allocation strategy, output quality can be risen by moving a reduced number of more sensitive data to memory with lower level of approximation, while leaving the large buffers on memory with higher level of approximation.

Future works will target more complex applications requiring larger memory size, exploring the use of multiple approximate zones. Automatic allocation strategies will also be considered as a way of reaching more significant savings.

REFERENCES

[1] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, "Approximate srams with dynamic energy-quality management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2128–2141, 2016.

[2] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality configurable approximate dram," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1172–1187, 2017.

[3] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3. IEEE Computer Society, 2012, pp. 1–12.

[4] M. Jung, D. M. Mathew, C. Weis, and N. Wehn, "Efficient reliability management in socs-an approximate dram perspective," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016, pp. 390–394.

[5] 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 *ACM SIGPLAN Notices*, vol. 46. ACM, 2011, pp. 164–174.

[6] G. Stazi, F. Menichelli, A. Mastrandrea, and M. Olivieri, "Introducing approximate memory support in linux kernel," in *Ph. D. Research in Microelectronics and Electronics (PRIME), 2017 13th Conference on*. IEEE, 2017, pp. 97–100.

[7] G. Stazi, L. Adani, A. Mastrandrea, M. Olivieri, and F. Menichelli, "Impact of approximate memory data allocation on a h.264 software video encoder," in *Approximate and Transprecision Computing on Emerging Technologies ATCET2018, Workshop on*, 2018.

[8] A. S. Waterman, "Design of the risc-v instruction set architecture," Ph.D. dissertation, UC Berkeley, 2016.

[9] G. Stazi, A. Mastrandrea, M. Olivieri, and F. Menichelli, "Appropinquo: A platform emulator for exploring the approximate memory design space," in *2018 New Generation of CAS (NGCAS)*. IEEE, 2018, pp. 66–69.

[10] G. Ndour, T. T. Jost, A. Molnos, Y. Durand, and A. Tisserand, "Evaluation of approximate operators case study: sobel filter application executed on an approximate risc-v platform," in *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. ACM, 2018, pp. 146–149.

[11] A. Raha, H. Jayakumar, S. Sutar, and V. Raghunathan, "Quality-aware data allocation in approximate dram," in *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. IEEE Press, 2015, pp. 89–98.