Impact of Approximate Memory Data Allocation on a H.264 Software Video Encoder

Giulia Stazi[1], Lorenzo Adani[1], Antonio Mastrandrea[1], Mauro Olivieri[1], and Francesco Menichelli[1]

Sapienza University of Rome Dept. of Information Engineering, Electronics and Telecommunications (DIET) Rome, Italy
{stazi,menichelli,mastrandrea,olivieri}@diet.uniroma1.it
adani.1342114@studenti.uniroma1.it

Abstract. This paper describes the analysis, in terms of tolerance to errors on data, of a H.264 software video encoder; proposes a strategy to select data structures for approximate memory allocation and reports the impact on output video quality. Applications that tolerate errors on their data structures are known as ETA (Error Tolerant Applications) and have an important part in pushing interest on approximate computing research. We centered our study on H.264 video encoding, a video compression format developed for use in high definition systems, and today one of the most widespread video compression standard, used for broadcast, consumer and mobile applications. While data fault resilience of H.264 has already been studied considering unwanted and random faults due to unreliable hardware platforms, an analysis, considering controlled hardware faults and the corresponding energy quality tradeoff, has never been proposed.

1 Introduction

Reducing power consumption in digital architectures gained a prominent role in research since almost two decades, especially when technology shrinking of physical devices started to rise important design issues due to increased power density. The problem has been further amplified by application requirements demanding increasingly amounts of processing power and memory size (e.g. high definition multimedia, high speed communication, big data applications).

The contribution of this work is to propose a strategy for selecting error tolerant data structures and to study the impact of faults on the quality of the H.264 video stream. The results, as a case study, allow to find the relationship between video output quality and hardware fault rate, which is the final metric to guide the relaxation of hardware design constraints to save power (energy quality tradeoff [1]).

1.1 Approximate memory

In modern digital systems memory represents a significant contribution to system power consumption. Approximate memories are memory circuits where cells are

subject to hardware errors (bit flips) with controlled probability. From a conceptual point of view they are not different from standard memories (which are also affected by errors), but, in approximate memories, errors are allowed by design and are non-negligible for the software application. The presence of errors is the result, in general, of design implementations introduced to significantly reduce power consumption [2–4]. Different design strategies can be actively used in order to introduce approximation, depending on memory technology. Specifically for eDRAM/DRAMs, the refresh operation degrades performance and wastes energy; for example, when the system is in standby mode, it can reach up to 50% of total power consumption [5].

In exact DRAMs refresh time interval is set according to the worst case access-statistics of the most leaky cells. Commercial DRAM modules, for example, have a worst case retention time of 64ms determined by the leakiest cells in the entire array [3]. High refresh rate, which guarantees a storage without errors at the expense of power consumption, may not be necessary, especially considering low occurrence probability of the worst case.

In [6] the authors propose to abandon worst case design paradigm, showing the benefits that can be achieved by relaxing the refresh time interval at the expense of increasing error rates. In particular, tests on 8 chips of GC-eDRAMS show that, admitting an error rate of 10^{-3} and relaxing refresh rate from 11ms (worst case retention time) to 24 ms, 55% of energy can be saved; while an error rate of 10^{-2} guarantees energy savings up to 75%. These experimental results are just an example of the potential benefits that approximate memories can achieve, depending on memory technologies and approximation levels that the target applications can tolerate.

2 OS managed approximate memory and AppropinQuo emulator

In this section we briefly describe our previous work regarding the introduction of approximate memory support in Linux kernel and the development of a hardware emulator, AppropinQuo, for platforms containing approximate memory.

Linux kernel support for approximate memory [7] allows the OS to distinguish between exact memory banks and approximate memory banks. Approximate memory management has been integrated in the kernel memory management, relying on the internal concept of *physical zone*. In this way, the Linux kernel is aware of exact memory and approximate memory physical pages, managing them as a whole for the common part (e.g. optimization algorithms, page reuse) but distinguishing them in terms of allocation requests and page pools management.

AppropinQuo [8] is a platform emulator that supports approximate memory models (along with normal exact memory) and allows the execution of the operating system and applications while injecting faults at run time. The fault injection mechanism has different levels of insertion, tunable at byte level and at segment level, as well as configurable error rates and error injection models. In

particular, faults are based on models designed to reproduce the effects on memory cells of circuital and architectural techniques for approximate memories (e.g. errors on cells can be introduced by access operations or can occur randomly, even if the cell is not accessed).

3 H.264 video encoding

H.264, or MPEG-4 AVC, is a video compression format developed for use in high definition systems.

Because of its widespread use and computational requirements, advanced platforms for its efficient implementation in terms of cost, power, quality, have been proposed. Research has been conducted on processing units and memory subsystems [9].

Data fault resilience of H.264 algorithm has already been studied [10, 11], however the approaches consider unwanted and random faults due to unreliable hardware platforms. These faults, manifested as spurious bit flips, can be characterized in terms of statistical probability, but cannot be controlled at data level. In our work, faults are intentionally allowed on selected data structures and with controlled and higher probability than the former works.

The x264 encoder is a free software library and application for encoding video streams into H.264 [12]. We characterized heap memory usage for different video resolutions and encoding options. Heap memory is commonly used by applications for dynamic allocation of large memory buffers during data processing, which, for the x264 encoder and in general for ETAs, are good candidates for approximate memory storage.

We expected larger memory requirements for higher resolution video, but also for different encoding options. Encoding options in x264 set a tradeoff between encoding speed and output quality (considering the same bitrate) and are another source of increasing memory requirements. For practical use these options are grouped in presets ranging from high speed/low quality (*ultrafast* preset) to extremely low speed/high quality (*slow* preset).

Table 1 reports memory usage for different input video resolutions and encoding options, showing the expected dependency on them. Peak heap represents memory peak allocation, while useful heap is the actual memory used for application data; the difference being memory consumed by allocation size rounding and administrative byte associated with each allocation. We note that not all heap can be allocated in approximate memory, since part of its data, typically called *critical data*, are not tolerant to errors. A strategy for selecting candidates for approximate memory allocations is then required, and it is described in the following section.

3.1 Approximate memory data allocation for the x264 encoder

In order to select candidate data structures for approximate memory allocation, we analyzed the x264 memory usage traces during execution (memory profiling).

Table 1. Heap memory usage

video resolution	x264 option (preset)	peak heap[MB]	peak usefulheap [MB]
176x144	medium	15.6	15.4
704x576	veryfast	57.2	49.6
1920x1080	ultrafast	90.1	77.8
1920x1080	superfast	216.0	192.1
1920x1080	veryfast	269.0	238.6

We traced all functions called to allocate heap memory and then determined which data are not critical for program execution. The profiling has been performed using the Valgrind debug and profiling suite [13]; in particular, the heap profiler tool called Massif. In Fig. 1 we report an extract (peak memory sample) of Massif output for the encoding of a 1920x1080 resolution video and *veryfast* option setting. The following analysis is valid for other preset options and resolutions since, apart from absolute memory usage, relative percentages remain similar.

```
88.69% (250,198,526B) (heap allocation functions)
malloc/new/new[], --alloc-fns, etc.
->88.03% (248,347,652B): x264_malloc
| ->70.33% (198,419,648B): x264_frame_new
| | ->70.33% (198,419,648B): x264_frame_pop_unused
| | ->41.92% (118,246,016B): x264_encoder_encode
| | | ->41.92% (118,246,016B):encode_frame
| | | ->41.92% (118,246,016B): main
| | ->12.18% (34,360,128B): x264_encoder_open_152
| | ->12.18% (34,360,128B): x264_encoder_encode
| | ->04.06% (11,453,376B): x264_encoder_encode
| ->08.82% (24,883,200B):x264_encoder_open_152
| ->04.47% (12,603,136B):
x264_macroblock_cache_allocate
| ->04.41% (12,441,668B): x264_encoder_open_152
```

Fig. 1. Memory allocation profiling: Massif output

From the profiling reported we deduce that the total amount of *useful heap* memory is about 239MB. The largest part of heap memory allocation is indeed handled by function *x264_malloc*, which covers about 88.03% of total allocated heap memory. The function *x264_frame_new*, which in turn calls *x264_malloc*, covers 70.33% of heap allocations.

The next step involved the analysis of source code in order to identify the actual data allocated by these functions. By this analysis we discovered that the first one, *x264_malloc*, is too generic, handling also allocation of critical data structures. We could classify as critical in x264, for example, data regarding encoder behavior, frames analysis, color space bits depth setting and encoding bitrate control. These data are critical because they are responsible of program control flow, which cannot be altered randomly by faults without completely compromising the encoding algorithm. The analysis of the function *x264_frame_new* revealed that this routine is used to create and allocate frames for encoding or decoding the video, in the form of *frame structures*. For each of these frames, x264 allocates a heap space large enough to contain the whole picture buffer and other information, depending on encoder options. In particular the *frame structure*, among others, stores data concerning frame encoding options, colors space information, buffers for frame pixels, motion vector buffers

Table 2. Test videos from derf's collection

name	resolution	lenght [frames]
ducks_take_off	1080p	500
dinner	1080p	950
crowd_run	1080p	500
blue_sky	1080p	217

and rate control; some of this information is involved in the encoding control flow and must be still kept exact. Conversely, buffers for frame pixels are optimal candidates for approximate memory, because introducing errors in them does not alter program execution flow. Further analysis showed that image pixels are grouped into three different buffers, containing the pixel values for each color component, each 1-byte large (8-bit per pixel). This data representation analysis is important for the optimization of approximate memory techniques, as will be discussed in the following section.

4 Results

The approximated x264 encoder was compiled and executed in the AppropinQuo emulator, running Linux kernel version 4.3 with support for approximate memory management and built for the x86 architecture.

Input test files were selected from the Xiph.org Video Test Media (derf's collection) [14]. Given the large number of choices available, we selected videos in raw format (no compression applied), color and characterized by moving and still parts. A list of them is present in Table 2.

Tests were executed configuring the approximate memory model in AppropinQuo for DRAM memories using slower refresh rate [3, 5], considering different fault rates and bit-level error masking (looseness level). The range of fault rates was chosen according to a refresh rate increase ranging from 8x (256ms) up to 400x (25s), while bit-level error masking allows to take into account more advanced approximate techniques that distinguish between bit weights (the quality of the user experience in multimedia application is mainly defined by the most significant bits [15]).

Results are provided in terms of user perceived video quality, comparing original and coded frames. In particular, as quality metric, we used peak signal-to-noise ratio (PSNR), defined as the ratio between the maximum pixel value and rms of corrupting noise that affects the fidelity of its representation.

All tests were executed with the x264 *veryfast* preset option, since this setting provides a good balance between encoding processing time and quality.

In order to produce reference values, we first run the original x264 encoder, with buffers allocated in exact memory. The average results are a global PSNR of 29.696 dB and an output bitrate of 17537 kbps. We note that the global PSNR value on output videos for exact compression should be considered an upper bound to evaluate x264 with the present settings.

4.1 Output with approximate memory and energy saving considerations

Table 3 shows the results of the same encoding using approximate memory. Global PSNR values are reported for each fault rate/looseness mask combination. According to AppropinQuo simulation parameters [8], looseness mask is a 32-bit configurable mask (constant for the whole memory array) that is applied to every 32-bit location in memory in order to allow selective fault protection at bit level (i.e. the MSBs); considering that pixel values stored by approximated buffers have 8-bit size and are packed on 32-bit locations by the compiler (see Section 3.1), we set the looseness mask as repetitions of an 8-bit submask.

Table 3. Video Output PSNR [dB]

Looseness mask	**Fault rate** $[errors/(bit \times s)]$		
	10^{-2}	10^{-3}	10^{-4}
0x3F3F3F3F	19.97	25.18	28.84
0x1F1F1F1F	24.47	28.01	29.43
0x0F0F0F0F	27.35	29.13	29.59
0x07070707	28.96	29.52	29.63
0x03030303	29.47	29.61	29.64
0x01010101	29.61	29.64	29.64

Fig. 2. Extract from HD frame: exact (left), 0x0F0F0F0F mask (center), 0x3F3F3F3F mask (right), fault rate 10^{-3} $errors/(bit \times s)$

We note that for a fault rate of 10^{-3} $errors/(bit \times s)$ and a looseness mask set to 0x0F0F0F0F (i.e. error allowed on the four LSBs of each byte), PSNR is 29.13 dB, or about 0.5 dB under the exact case, confirming good tolerance to errors. The table shows also that, with the same fault rate, all masks more protective than 0x0F0F0F0F (i.e 0x07...07, 0x03...03, 0x01...01) produce very close outputs, but would result in larger energy consumption (since they imply a larger portion of exact bits). Figure 2 shows the visible effects, for 0x0F0F0F0F and 0x3F3F3F3F bit masks, on a portion of a frame.

Simulations with fault rate set to 10^{-2} $errors/(bit \times s)$ illustrate that the 0x0F0F0F0F mask produces a PSNR value about 2 dB under the exact case, resulting in more visible effects of corruption on the output. Figure 3 plots output PSNR for an extended fault rate range.

Actual energy saving related to the application of our test cases can be extracted assuming as reference the results showed in [3]. Refresh power depends on refresh rate, if we assume a 10^{-3} error rate, a 60x increase in refresh period

can be allowed. A looseness mask set to 0x0F0F0F0F means that half the cells must be exact while the other can be approximate memory cells. In our tests, data structures selected to be allocated in approximate memory are about 60% of total data, resulting in a system where, globally, about 30% are approximate memory cells while 70% are exact memory cells.

According to this partition, and considering only refresh power, we can expect a normalized refresh power in the range of 0.3-0.5 [3] with respect to the original exact implementation.

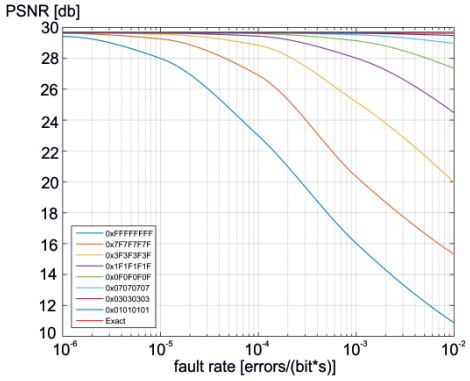

Fig. 3. Video Output PSNR graph [dB]

5 Conclusion

In this work we presented the analysis of the x264 video encoder and the impact of using approximate memory for storing its error tolerant data structures. We started by profiling memory usage and finding a strategy for selecting error tolerant data buffers. We then run the modified application on AppropinQuo emulator, for several combination of fault rates (derived from actual refresh rate reduction strategies) and fault masking at bit level (looseness level).

Results show the importance of exploring the relation between these parameters and output quality. For example, leaving some of the MSBs exact demonstrated to be an effective way of allowing error probabilities up to 100x higher with the same output quality and a refresh period increase in the order of 60x.

Since leaving exact a portion of memory cells reduces global energy savings, this knowledge is also fundamental in order to drive research on hardware techniques specifically tailored to the application, revealing the tradeoff between designing more aggressive approximate circuits and the number of bit cells that must be kept exact.

Future works can consider better allocation strategies and more advanced DRAM architectures for embedded systems, as DRAMs chips with integrated ECC units. Another important aspect is a more accurate quantification of power savings, which could be obtained by integrating a power consumption model in the approximate DRAM memory model.

References

1. J. Huang, J. Lach, and G. Robins, "A methodology for energy-quality tradeoff using imprecise hardware," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 504–509.
2. S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: saving dram refresh-power through critical data partitioning," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 213–224, 2012.
3. 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.
4. H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ACM SIGPLAN Notices*, vol. 47, no. 4. ACM, 2012, pp. 301–312.
5. 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.
6. A. Teman, G. Karakonstantis, R. Giterman, P. Meinerzhagen, and A. Burg, "Energy versus data integrity trade-offs in embedded high-density logic compatible dynamic memories," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 489–494.
7. 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.
8. F. Menichelli, G. Stazi, A. Mastrandrea, and M. Olivieri, "An emulator for approximate memory platforms based on qemu," in *International Conference on Applications in Electronics Pervading Industry, Environment and Society*. Springer, 2016, pp. 153–159.
9. B. H. Asma, N. Jarray, and Z. Abdelkrim, "Low-power hardware design of binary arithmetic encoder in h. 264," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 8, no. 7, pp. 412–416, 2017.
10. S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Revc: Computationally reliable video coding on unreliable hardware platforms: A case study on error-tolerant h. 264/avc cavlc entropy coding," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE, 2011, pp. 397–400.
11. M. Shafique, S. Rehman, F. Kriebel, M. U. K. Khan, B. Zatt, A. Subramaniyan, B. B. Vizzotto, and J. Henkel, "Application-guided power-efficient fault tolerance for h. 264 context adaptive variable length coding," *IEEE Transactions on Computers*, vol. 66, no. 4, pp. 560–574, 2017.
12. L. Merritt and R. Vanam, "x264: A high performance h. 264/avc encoder," *online] http://neuron2. net/library/avc/overview_x264_v8_5. pdf*, 2006.
13. N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in *ACM Sigplan notices*, vol. 42, no. 6. ACM, 2007, pp. 89–100.
14. C. Montgomery *et al.*, "Xiph. org video test media (derf's collection), the xiph open source community, 1994," *Online, https://media. xiph. org/video/derf*.
15. J. Kwon, I. J. Chang, I. Lee, H. Park, and J. Park, "Heterogeneous sram cell sizing for low-power h. 264 applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2275–2284, 2012.