Approximate Memory support for Linux Early Allocators in ARM architectures

Giulia Stazi, Antonio Mastrandrea, Mauro Olivieri, and Francesco Menichelli

Dept. of Information Engineering, Electronics and Telecommunications (DIET)
Sapienza University of Rome, Via Eudossiana, 18, 00184 Roma, Italy
{stazi,mastrandrea,olivieri,menichelli}@diet.uniroma1.it

Abstract. Keywords: Approximate Memory, Approximate computing, Linux OS, Low Power Embedded Systems

Approximate computing is a new paradigm for energy efficient design, based on the idea of designing digital systems that trade off computational accuracy for energy consumption. The paradigm can be applied to different units (i.e. internal units of the CPU, floating point coprocessors, memories). Considering the memory subsystem, approximate memories are physical memories where circuit-level or architecture-level techniques are implemented in order to reduce energy at the expense of errors occurring in bit cells. Supporting approximate memories at operating system level is required for managing them efficiently and for allowing user level applications to use it directly, but its implementation is subject to specific requirements and constraints, sometimes architecture dependent.

In this paper we describe the introduction of approximate memory support on ARM architectures, which are widely adopted in low power embedded systems. While Linux support for approximate memory has already been introduced for main allocators, porting it to ARM architectures required the introduction of its specific support in the Linux early allocators, that are a fundamental function of the Linux kernel startup phase, before instantiation of the main allocators.

1 Introduction

Approximate memories, as part of the *approximate computing* design paradigm, have been a particularly prolific source of research works in the late years. Depending on the technology (i.e. DRAM, SRAM) many architectural ideas and circuit design implementations have been proposed [1–4], demonstrating a large impact in reducing the energy consumption of the memory subsystem. In parallel with the introduction of them as a component in a computing platform, a second problem arises: the ability to efficiently managing and make them usable in software applications.While in simpler architectures memories are managed and allocated directly by software, larger and more complex embedded systems platforms (e.g. embedded systems for networked applications, graphics or multimedia [5, 6]) tend to have an operating system which provides fundamental

services as multiprocessing, virtual memory management, file system and network stack. These systems are also those that require larger array of physical memories, thus they benefit more from the reduction of energy consumption offered by approximate memories.

Since Linux is widely used as OS in embedded systems, due to its flexibility and availability of source code, approximate memory support for the main allocators has already been investigated and implemented [7]. However, for the complete support of approximate memories, especially for embedded platforms, such as ARM architectures, it is required that *Linux early allocators*, which are the fundamental allocators used during the startup of the operating system, are modified. *Linux early allocators* are responsible for allocating, among others, data structures required by kernel deeply internal functions, as the virtual memory management in the main allocator.

The following Sections are organized as follows: in Section 2 we describe the state of approximate memory support before this work, Section 3 reports the main contribution of the present work. Section 4 provides results, in the form of allocation statistics provided by the kernel and discusses the characteristics of the implementation.

2 Previous works

In this section we briefly describe our previous work regarding the introduction of approximate memory support in Linux kernel [7]. This extension, which relies on the internal concept of *physical zone*, involved the creation of a new Linux memory zone, called ZONE APPROXIMATE, where approximated pages containing non-critical data can be grouped, and the implementation of a custom system call to allow user space applications to dynamically request pages within this zone.

The advancements described in this paper regard the extension of approximate memory support for Linux kernel early allocators, along with boot time vector allocation. This further steps were required for porting approximate memory support to ARM architectures, but is valid for all architectures that make use of early boot allocators during the booting process.

3 Linux Early allocators and approximate memory

Linux early boot allocators are used during the boot process in order to allocate data structures in the initial phase of system startup, before the main allocators are instantiated. For ARM architectures, the initialization of all physical zones, including ZONE_APPROXIMATE, takes place in *bootmem_init* function. This routine determines the limits of all physical memory available (PFN limits) and sets up the early memory management subsystem. After this process, pages allocated by the boot allocator are freed and physical zone limits, including ZONE_APPROXIMATE, are determined.

3.1 Bootmem allocator

At start-up Linux kernel gains access to all physical memory available in the system. Before memory zone allocator is set up and running, it can be necessary to preallocate some initial memory areas for kernel data structures and system-wide use, taking them from available RAM. To address this requirement, a special allocator called *bootmem allocator* or *memblock allocator*, is introduced. The initialization of this early allocator is architecture dependent and it is set up in *setup_arch* routine.

Once the boot memory management is available, it can allocate areas from low memory (memory directly mapped in Kernel space), with page granularity. The early allocator is used only at boot time to reserve and to allocate pages for internal kernel use. For example, *page tables* are built from this pool of physical memory pages, allowing the MMU to be turned on and Linux kernel to switch to virtual memory management.

The whole mechanism requires that the kernel must be aware of approximate memory in the early boot phases: approximate memory must be visible in order to properly instantiate paging and main allocators, but must not be used for kernel data structures, which contains critical data that do not allow any form of corruption. In order to exclude physical memory pages mapped as approximate from bootmem allocation, we modified the allocation algorithm of the *memblock* interface. A *memblock* is a structure that stores information of physical memory regions reserved by Linux kernel during the early bootstrap period. The core function of this early allocation is *memblock_virt_alloc_internal*, which in turn calls *memblock_find_in_range_node*. This routine receives as parameters, among others, the requested memory size and the lower and upper bounds of the physical region where the memory block will be allocated. At first, the allocation starts from the lower bound and the following allocation requests will proceed to lowest available address starting from the lower bound. The upper bound instead corresponds to the end of the candidate physical memory range and it is set to the value of the global parameter *memblock_current_limit*, which is set to the end of *low memory* region, forcing early boot allocation within the *low memory* region, that is the only region the kernel can directly access.

In order to include and support the presence of approximate memory, we modified the algorithm for computing *memblock_current_limit*, forcing it to be always below the lower limit of the approximate memory physical region. In this way it is ensured that the bootmem allocator gets free pages only from physical exact memory.

3.2 Vectors

In order to boot the primary core, the kernel allocates a single 4KB page as vector page, mapping it to the location of ARM exception vectors at virtual address 0xFFFF0000 or 0x00000000. When this step is completed, the *trap_init* function copies the exception vector table, exception stubs, and helpers from *entry-arm.S* into the vector page.

In particular, the allocation of the ARM vectors page is performed by the *early_alloc* function allocator. This allocator cannot exclude approximate memory, since it does not allow to specify the memory zone. In order to ensure that the vectors page is never allocated in approximate memory, the implementation of a new *early_alloc* was required. The new *early_alloc* uses the *memblock* interface and allow to explicitly indicate an address limit for the allocation request, in order to exclude approximate memory.

3.3 Approximate memory and DTB

During the boot process, a "Device Tree Blob" (DTB) file is loaded into memory by the bootloader and passed to the Linux kernel. This DTB file is a tree data structure containing nodes that describe the system hardware layout to the Linux kernel, allowing for platform-specific code to be moved out of kernel sources and replaced with generic code that can parse the DTB and configure the entire system as required. Each physical device is indeed described inside the device tree, in particular it is represented as a node and all its properties are defined under that node.

In order to support approximate memory management in ARM architectures, we defined a new DTS file (from which the DTB is generated), specific for each ARM platform, with a special node for approximate memory (Fig.1 on left). In particular this node, called *approx_mem*, collects the information about the physical address range and the size of zone approximate.

```
memory {
reg = <(baseaddr1) (size1)
(baseaddr2) (size2)
...
(baseaddrN) (sizeN)>;
};
```

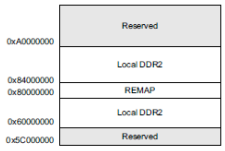

Fig. 1. On left: Memory node in device tree. On right: Vexpress Cortex A9 board memory map (extract)

4 Experimental results

In this section we describe the results and the architecture setup used to evaluate the introduction of approximate memory support for early boot allocators in Linux kernel on ARM architectures.

4.1 Hardware setup

ARM Versatile Express (Vexpress) Cortex A9 has been chosen as the architecture for performing tests. These tests were run on the emulation platform AppropinQuo [8], that contains specific approximate memory models for the chosen architecture. Fig. 1 on right shows the memory map of Vexpress Cortex A9, in particular RAM memory can be present from 0x60000000 to 0x80000000 and

from 0x84000000 to 0xA000000. We chose to map 128+128 MB of RAM: the first 128MB part is exact, starting from address 0x60000000 to address 0x67FFFFFF and the second 128MB part, from address 0x68000000 to 0x6FFFFFFF,is approximate memory. This map was used to configure the emulator and also to set the corresponding kernel dts file.

4.2 Results

Fig. 2 (left) shows the statistics for the ZONE_NORMAL region, obtained through the *zoneinfo* system command. The region contains exact memory, is composed of 32768 pages; considering that every page in the ARM architecture is 4KB large, it confirms the availability of 128MB of exact memory. The start_pfn number indicates the start address of exact memory in physical pages ($393219 \times 4096 = 0x60000000$). Fig. 2 (right) shows the same statistics for the ZONE_APPROXIMATE. The approximate area has 32768 pages; again this confirms that the approximate region area is 128MB large. The start_pfn number indicates the start address of approximate at address 0x68000000 ($425984 \times 4096 = 0x68000000$). Particular importance comes from information regarding 'present' and 'managed' lines. The latter corresponds to pages managed by the buddy system (the main allocator); they are computed as the number of present pages minus the number of reserved pages, including those allocated by the bootmem allocator. Since the number of present pages matches the number of managed pages, we have the demonstration that during the boot phase no pages belonging to the zone_approximate were allocated.

```
cat /proc/zoneinfo
Node 0, zone Normal
pages free 28036
min 167
low 208
high 250
scanned 0
spanned 32768
present 32768
managed 30487
...
start_pfn: 393216
```

```
cat /proc/zoneinfo
Node 0, zone Approximate pages free 32768
min 180
low 225
high 270
scanned 0
spanned 32768
present 32768
managed 32768
...
start_pfn: 425984
```

Fig. 2. On left: kernel boot logs. On right: zone_approximate statistics

Moreover, in order to evaluate the correct memory allocation of apprroximate physical pages in ZONE_APPROXIMATE, we run a testbench that requests a block of 1000 memory pages (about 4MB) from ZONE_APPROXIMATE. Using the *zoneinfo* system command we get the information printed in Fig. 3 before (left) and after (right) the allocation request. We can see that ZONE_APPROXIMATE has 31768 free pages after allocation, confirming that the application allocated exactly 1000 pages.

5 Conclusions

In this work we analyzed early allocators in Linux kernel, proposed and implemented an extension of their function in order to introduce the support of

```
cat /proc/zoneinfo
Node 0, zone Approximate pages free 32768
min 180
low 225
high 270
scanned 0
spanned 32768
present 32768
managed 32768
...
start_pfn: 425984
```

```
cat /proc/zoneinfo
Node 0, zone Approximate pages free 31768
min 180
low 225
high 270
scanned 0
spanned 32768
present 32768
managed 32768
...
start_pfn: 425984
```

Fig. 3. On left: zone_approximate statistics before allocations. On right: zone_approximate statistics after allocation

approximate memory in the architecture. This work was done in the specific context of porting approximate memory support on ARM architectures, a widely adopted architecture in low power embedded systems, but it is valid also for other architectures that make use of early allocators during the boot process. After completing the porting we run the kernel on an ARM platform and demonstrated the correctness of the boot process and of the main allocators, that can now see and manage the correct number and set of physical pages for the approximate memory.

References

1. 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.
2. J. Lucas, M. Alvarez-Mesa, M. Andersch, and B. Juurlink, "Sparkk: Quality-scalable approximate storage in dram," in *The memory forum*, 2014, pp. 1–9.
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. 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.
5. D. T. Nguyen, H. Kim, H.-J. Lee, and I.-J. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018, pp. 1–5.
6. 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.
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.