Code Optimization Techniques to Reduce Energy Consumption of Multimedia Applications in Hybrid Memory

Thomas Haywood Dadzie, Seungpyo Cho, and Hyunok Oh

Department of Information Systems, Hanyang University, Seoul, South Korea
{ekowhaywood, seungpyo, hoh}@hanyang.ac.kr

***** Corresponding Author: Hyunok Oh

Received July 6, 2016; Revised August 23, 2016; Accepted August 24, 2016; Published August 30, 2016

** Regular Paper*

Abstract: This paper proposes code optimization techniques to reduce energy consumption of complex multimedia applications in a hybrid memory system with volatile dynamic random access memory (DRAM) and non-volatile spin-transfer torque magnetoresistive RAM (STT-MRAM). The proposed approach analyzes read/write operations for variables in an application. Based on the profile, variables with a high read operation are allocated to STT-MRAM, and variables with a high write operation are allocated to DRAM to reduce energy consumption. In this paper, to optimize code for real-life complicated applications, we develop a profiler, a code modifier, and compiler/link scripts. The proposed techniques are applied to a Fast Forward Motion Picture Experts Group (FFmpeg) application. The experiment reduces energy consumption by up to 22%.

Keywords: Compiler, Simulation, Hybrid memory, Energy consumption

1. Introduction

With the ongoing upsurge of mobile devices and embedded devices in general, one of the main concerns is to reduce the energy consumption of an entire system. A lot of research has been conducted into reducing energy consumed in many components of a system—the central processing unit (CPU), random access memory (RAM), storage, etc. Most research in recent years has been on dynamic management of CPU power consumption. Hence, we have CPU voltage/frequency scaling and multiple cores on one die.

In the area of primary memory (RAM), we have widely used dynamic RAM (DRAM). Recent research shows that DRAM has significant energy consumption and accounts for as much as 30% to 40% of energy consumed on a system-on-chip (SoC) [4]. This is mainly due to its high energy leakage and the need to refresh cells frequently. DRAM is also expensive in terms of cost per megabyte and due to its low density, which means it occupies a large amount of space on a SoC. These shortfalls in DRAM have presented an opportunity for non-volatile memories to replace it. This has been an active research topic focusing efforts on non-volatile memory, such as phase-change RAM (PRAM), resistive RAM (RRAM), ferroelectric RAM (FeRAM), and spin-transfer torque magnetoresistive RAM (STT-MRAM), to replace DRAM due to the commercial success of flash memory. These non-volatile memories have the advantages of comparable read latency with DRAM, low cost, high density, low energy for reads, and low leakage power. However, for the write operation, non-volatile RAM (NVRAM) requires long latency and high energy consumption.

In this paper, we present variable read/write analysis of the Fast Forward Motion Picture Experts Group (FFmpeg) audio/video application. We demonstrate a technique with a tool set developed for performing the analysis. We then introduce a variable declaration keyword and a compiler-level feature that enables variables to be assigned to an NVRAM module of hybrid memory with a code modifier helping to revise the code. Finally, we present a hybrid memory-aware FFmpeg application that reduces energy consumption by up to 22% for a set of frequently accessed variables.

Section 2 briefly explains the NVRAM technology and its read/write properties. Section 3 discusses related work to adopt NVRAM as primary memory. In Section 4, we present the motivation that drives our research, and in Section 5, we define the problems tackled to ensure maximum energy reduction. Section 6 proposes techniques and tools developed to build a hybrid memory-aware FFmpeg application. Finally, we show the experimental results in Section 7 and offer a conclusion in Section 8.

2. Background

Static random access memory (SRAM) and DRAM have been the dominant technologies for primary memory for years. These technologies have improved over the years to overcome or subdue their shortfalls, which are high energy consumption and low density. These problems can easily be ignored in PC and server environments. However, in mobile devices, the problems are more pronounced, as energy consumption becomes important when designing mobile devices. A lot of work has been done to reduce energy consumption in mobile devices, but so far has been in the area of enhancing the CPU and other co-processor designs to decrease the amount of energy consumed during program execution. In addition, processors have been enhanced greatly with the introduction of the multi-core CPU, where a job is shared among low-frequency multi-cores as opposed to one core with a high frequency, which consumes great amounts of energy. Aside from all these improvements, we still have a major challenge, because SRAM and DRAM used in mobile devices still account for about 30% to 40% of the processing unit, and the basic technology for these two memories has not changed much.

RAM vendors propose alternative solutions to develop non-volatile memory technology to replace conventional SRAM and DRAM. That includes PRAM and STT-MRAM, which are briefly explained in the following subsections to denote their strengths and limitations

2.1 PRAM

PRAM uses a special material called phase-change material to preserve a bit. The phase-change material (Ge2Sb2Te5, or GST) can be in at least two different structural states: amorphous or crystalline. These two states represent a 0 or 1, since they have different resistivity. In the amorphous state, the material has low electrical conductivity, and in the crystalline state, high electrical conductivity. The GST material is one type of alloy that can switch between two states with the application of heat. When heated above the crystallization temperature of 300°C (but below the melting temperature of 600°C) over time, GST changes to the crystalline state, which corresponds to the SET state, or logic 1. When heated above the melting point and quenched quickly, the GST changes to the amorphous state, which corresponds to the RESET state, or logic 0. The GST material's state is altered by injecting a large, but fast, current pulse for a few hundred nanoseconds to heat up the GST active region, as shown in Fig. 1. This means the write energy and latency is high. To read a PRAM cell, the power needed is very

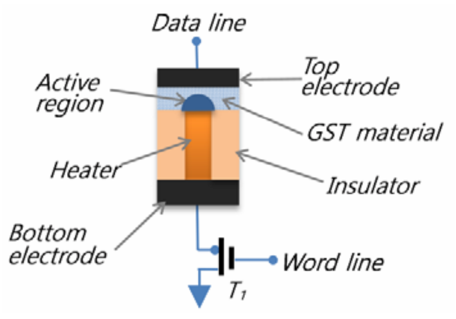


Fig. 1. PRAM cell.

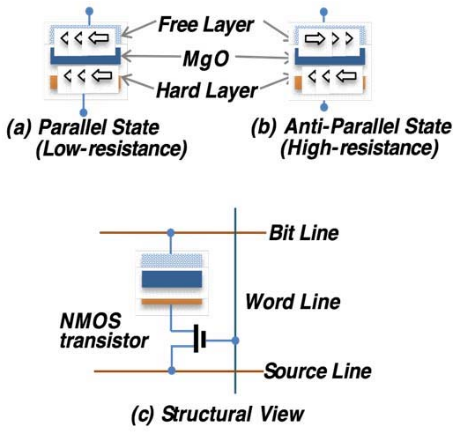


Fig. 2. STT-MRAM cell.

low, since no heating is involved. In addition, PRAM needs no refresh energy and has a low standby power, as it retains its state permanently. PRAM has limited write endurance of 109−1012 cycles, which poses a reliability problem.

2.2 STT-MRAM

STT-MRAM is a variant of magnetoresistive RAM (MRAM). MRAM is a non-volatile memory technology where a bit is stored by the magnetic orientation of the free layer of a magnetic tunnel junction (MTJ). Applying a small, fixed voltage to the MTJ results in a high or low current, depending on whether the free layer is parallel or anti-parallel to the magnetic orientation of the hard layer, as shown in Figs. 2(a) and (b), respectively. Fig. 2(c) shows the cell structure with a transistor. The different orientations represent a 0 or a 1 bit.

It uses spin-transfer torque to re-orient the free layer by passing a large and directional write current through the MTJ. The switching process is regulated by a thermally controlled stochastic process, which means that the free layer could change state at any time. However, the MTJ magnetic properties and sizing are selected to make this unlikely. Performing a write operation requires holding the write current for a sufficient amount of time, which we call the MTJ write time, to ensure the free layer has changed state. STT-MRAM presents a technology that is highly efficient for read access, but poor with write access. As the free layer needs no current to maintain its state, MTJs have no intrinsic leakage power.

Table 1. Comparison of different memory technologies.

Feature	SRAM	DRAM	STT-MRAM	PRAM
Density	Low	High	High	Very high
Speed	Very Fast	Fast	Fast read	Slow read
			Slow write	Very slow write
Dynamic Power	Low	Medium	Low read	Medium read
			High write	High write
Leak Power	High	Medium	Low	Low
Non-volatility	No	No	Yes	Yes
Scalability	Yes	Yes	Yes	Yes

2.3 Hybrid Volatile/Non-volatile RAM

The above-mentioned non-volatile memories have high density, low energy for read operations, and low leakage power. However, they exhibit obstacles from high energy and high latency for write operations or changing the state of the core material used in them. Table 1 shows a comparison of the general characteristics of memory technologies [13].

2.4 FFmpeg

FFmpeg is a popular open source audio/video codec engine for conversion and streaming [6]. It is written in the C language and compiled by the GNU GCC compiler. It is capable of handling a wide range of video/audio codecs and encapsulation. The capability is evident when considering its code base of over 500,000 lines of code. FFmpeg reads and writes a number of inputs and outputs, which may be regular files, pipes, network streams, audio/video grabbing devices, etc. It handles most of the well-known audio/video codecs.

3. Related Work

Based on the attractive advantages of non-volatile memory, there have been several proposals for, and a lot of research into, using it as primary memory. Dhiman et al. proposed hybrid DRAM/PRAM by attaching DRAM and PRAM to utilize the high performance in read and write operations from DRAM and the low standby energy consumption from PRAM [10]. Another study specified computation replication to reduce communication overhead in multiprocessor systems [4]. Other research proposed an architecture where each processor core has an NVRAM in addition to the traditional scratchpad memory (SRAM) and introduced a runtime memory manager for efficient sharing of distributed scratchpad memories (SPM) and NVRAM [20]. Several studies tried using hybrid SRAM/PRAM or DRAM/STT-MRAM that addresses the allocation of data to volatile RAM or non-volatile RAM.

Stancu et al. [21] proposed an annotation-based approach to reduce write-energy consumption. In their approach, a multimedia application is profiled to obtain the data access patterns and is used to minimize write operations. However, their approach determines the mapping onto DRAM and PRAM dynamically when a virtual page is mapped to a physical page. Therefore, if variables with high read frequencies and high write frequencies are placed in a page, then the mapping of the page into the memory module does not improve the performance. Since our approach is to decide the mapping of variables onto memory modules statically, it can divide variables based on the read/write frequencies. However, sometimes it is necessary to split structure-type variables, which makes the problem difficult.

4. Motivation

In a hybrid primary memory architecture, data that is strictly write-once (or a few times) but that is read-intensive is placed in a non-volatile module, like STT-MRAM, and other data may be conveniently placed in a volatile module, such as DRAM. In today's matured and highly complex software, certain portions of software can easily be identified as suitable for non-volatile RAM. Such read-intensive data could be constants, variables, and instruction code, which hardly ever changes. With this simple approach, a reasonable amount of energy is saved as write energy is minimized. In this case, the work involved is only a simple compiler-level modification.

But the bulk constituent of software data is its non-constant variable data, and the standard compilers will not be able to predict the runtime read/write access pattern to make any energy-saving decisions. For this reason, if a strategy is developed that assists the compiler by attaching information about read/write patterns of non-constant variables, then a huge amount of energy will be saved.

Multimedia-enabled embedded devices can now play, capture, and transcode audio/video content, and it is a widely used feature. This comes with high energy consumption from DRAM due to its large data footprint in memory, both in complex instruction code and data. FFmpeg is at the center of this feature, where several graphic user interface (GUI) applications employ it as the back-end for audio/video processing. If devices employ the hybrid memory approach, a huge energy reduction is achievable with energy-aware FFmpeg.

5. Problem Formulation

In a hybrid DRAM/STT-MRAM memory architecture, there is varying energy consumption for read/write operations. The main challenge is to construct a runtime read/write access pattern of non-constant variables, and subsequently develop an allocation strategy to minimize energy consumption.

The energy requirement for each variable when placed in a hybrid DRAM/STT-MRAM memory architecture, as opposed to a DRAM-only memory architecture, is formulated as follows:

$$EV = (C_{DR}*Dyn_{Dr}) + (C_{DW}*Dyn_{Dw}) + (C_{SR}*Dyn_{Sr}) + (C_{SW}*Dyn_{Sw})$$

where
EV = energy per variable
C_{DR} = variable read count from DRAM
C_{DW} = variable write count from DRAM
C_{SR} = variable read count from STT-MRAM
C_{SW} = variable write count from STT-MRAM
Dyn_{Dr} = dynamic energy per read operation in DRAM
Dyn_{Dw} = dynamic energy per write operation in DRAM
Dyn_{Sr} = dynamic energy per read operation in STT-MRAM, and
Dyn_{Sw} = dynamic energy per write operation in STT-MRAM

Suppose there is a variable with 100 and 10 runtime read and write accesses, respectively. The proposed formulae show that when this variable is placed in STT-MRAM, the energy consumption becomes 63.00nJ, as opposed to 79.20nJ when placed in a DRAM-only memory architecture.

6. Proposed Approach

Considering the highly complex and mature nature of the FFmpeg software, re-engineering the source code may unknowingly break continuity and contributions from other developers. For this reason, focus is placed on making compiler-level modifications. Current compilers can easily identify and separate instruction code as well as constant variables. In this case, the modification will be making the compiler allocate such variables to a non-volatile module, which mostly happens once during the loading stage of program execution.

For other variables, we need to develop techniques and tools to determine the allocation of the dynamic variables to memory modules in order to reduce power consumption. In this paper, we develop a read/write profiler based on the Qemu emulator, a data structure revision editor, and a compiler supporting placement of a variable into a specific memory module, and introduction of a keyword (e.g. persist) to a variable declaration syntax. These proposed tools are utilized together to produce a hybrid memory-aware application. The overall flow is shown in Fig. 3. In Step (a), we first use the Qemu emulator to collect the runtime access count for variables of interest. Based on the results, we determine which variable is placed in either of the memory modules. In Step (b), we revise data structures using the code editor. In Step (c), we declare the separated members of the revised data structure with the persist keyword. In Step (d), the compiler compiles the source code into an object file and passes it to the linker, along with the link script. For the final Step (e), a hybrid memory-aware FFmpeg application is constructed.

Secondly, we have to analyze the usage of data structures in the FFmpeg application. The data structures

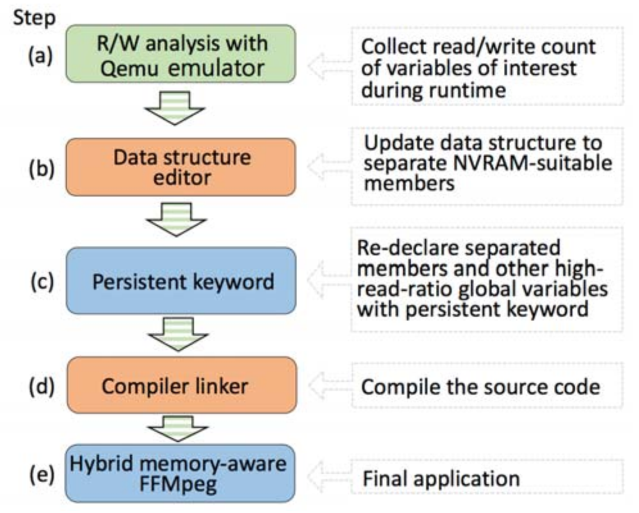

Step
(a)
R/W analysis with Qemu emulator
Collect read/write count of variables of interest during runtime
(b)
Data structure editor
Update data structure to separate NVRAM-suitable members
(c)
Persistent keyword
Re-declare separated members and other high-read-ratio global variables with persistent keyword
(d)
Compiler linker
Compile the source code
(e)
Hybrid memory-aware FFMpeg
Final application

Fig. 3. The overall technique for creating a hybrid memory-aware application.

and control flow built into FFmpeg is centered on decoding one macroblock (MB) at a time. The decoder maintains a tree of elements and structures rooted in a data structure (MPEGContext), which holds the current state of the decoder, including modes and data associated with the current MB, arrays for accessing information about neighboring MBs, references to the current output picture, any reference pictures, etc. This structure is passed from one function to the next, conveying the state of the decoder and the MB. The MPEGContext structure is made up of about 250 members.

With such a highly complex application, it is almost impossible to obtain read/write access patterns of the MPEGContext structure members, since the application includes 500,000 lines, and some members are accessed by pointer variables declared outside the structure. A runtime analysis is the only approach to log all access to members of this structure. With the read/write access analysis information, read-intensive members are mapped into the non-volatile module of the hybrid memory architecture.

6.1 Read/write Access Pattern Extraction

The Qemu emulator is a module-based application, and is a good environment in which to run all kinds of software analysis at any level for ARM processors. The Qemu project integrates the well-known GDB debugger stub, which enables all sorts of debugging capabilities, such as setting breakpoints. The emulated machine can be paused, resumed, and stopped manually and automatically if a condition is met. In this environment, we can set and trigger a breakpoint when the CPU accesses specific variables or memory regions registered in the Qemu emulator. Although setting breakpoints is supported in Qemu with its integrated GDB debugger stub, there is some difficulty as to which variables should trigger the break event. And when the emulated machine pauses, there will be the need for a manual recording of the event and issuing a command to resume execution. This is practically impossible, since there are too many variables in FFmpeg, and the CPU touches each variable more than 150,000

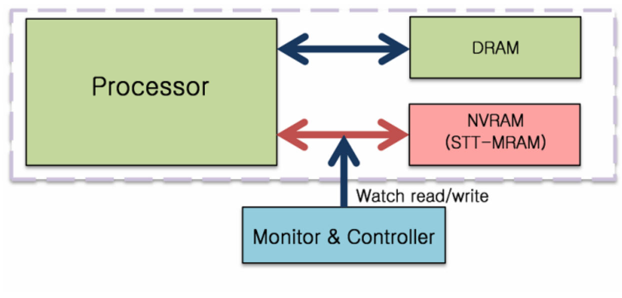
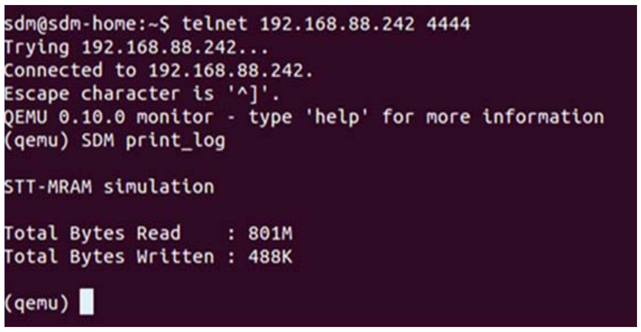
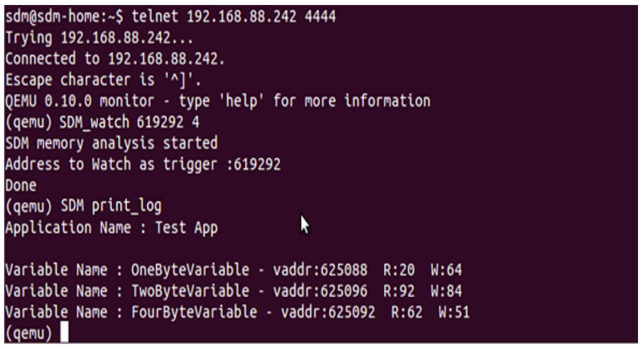


Fig. 4. Extended Qemu emulator for profiling read/write operations.



```
sdm@sdm-home:~$ telnet 192.168.88.242 4444
Trying 192.168.88.242...
Connected to 192.168.88.242.
Escape character is '^]'.
QEMU 0.10.0 monitor - type 'help' for more information
(qemu) SDM print_log

STT-MRAM simulation

Total Bytes Read    : 801M
Total Bytes Written : 488K

(qemu)
```

Fig. 5. The Qemu monitor console displaying read/write information of simulated NVRAM.



```
sdm@sdm-home:~$ telnet 192.168.88.242 4444
Trying 192.168.88.242...
Connected to 192.168.88.242.
Escape character is '^]'.
QEMU 0.10.0 monitor - type 'help' for more information
(qemu) SDM_watch 619292 4
SDM memory analysis started
Address to Watch as trigger :619292
Done
(qemu) SDM print_log
Application Name : Test App

Variable Name : OneByteVariable - vaddr:625088  R:20  W:64
Variable Name : TwoByteVariable - vaddr:625096  R:92  W:84
Variable Name : FourByteVariable - vaddr:625092  R:62  W:51
(qemu)
```

Fig. 6. The Qemu monitor console displaying read/write information of individual variables in an application.

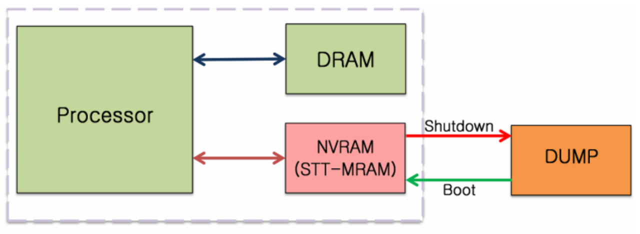


Fig. 7. The modified Qemu with the non-volatility simulation mechanism.

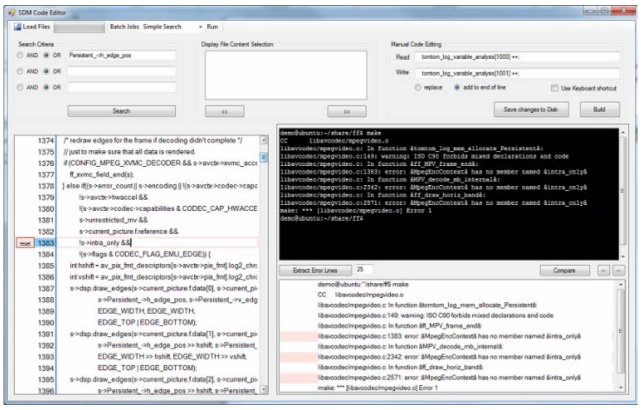
times, on average, when 30 video frames are decoded.

We extended the Qemu emulator to overcome the above-mentioned difficulty. Fig. 4 shows our modified Qemu with the mechanism for achieving the automated profiling of read/write accesses and simulation of hybrid memory. Qemu includes an integrated monitor console for interacting with a running virtual machine instance. The monitor exhibits several commands for controlling and reviewing the state of the virtual machine instance. We append new commands for hybrid memory simulation. The new commands allow the setting of breakpoints, logging processes, and reviewing the collected log data. The following procedures are implemented in the extended Qemu.

1. Create a virtual NVRAM region on the quest memory. All CPU accesses to the region are trapped and logged automatically without halting the virtual machine. Fig. 5 shows the Qemu monitor console displaying the collected log data for energy consumption analysis.
2. Watch and automatically log the CPU read/write accesses to variable memory locations of interest without halting the emulated machine. The gathered data are displayed on the monitor console in a human-readable format, and each variable is displayed with its read and write count. Fig. 6 shows the Qemu monitor console displaying the read/write count for each variable. This example shows collected data for three variables. For cases where the number of variables is huge, a command is provided to export the data to a text file in CSV format.
3. Automatically register variable memory addresses of interest to be watched, as most applications will have many variables. Registering a large number of variables in the Qemu emulator through the monitor console manually can be extremely tedious. For this reason, we provide an automation technique. We provide functions that can be integrated into a subject application's source code. These functions are intended to enable the subject application to communicate with the Qemu emulator. The extended Qemu emulator is designed to receive communication from the subject application. The subject application calls the function with the variables to watch as a parameter.
4. Simulate the non-volatility capability of NVRAM. One of the promising advantages of NVRAM is the ability to retain its data between power off and power on, which allows the design of new operating systems for the hybrid memory system. We implement this feature by dumping the content of the region in the virtual machine memory into a file in the host file system. Then, the virtual machine can be powered off. The next important stage is when a new instance of the virtual machine is started; the data in the file will be reloaded into memory during initialization of the virtual machine instance. This process will complete before the guest operating system is loaded and started. Fig. 7 shows the simulation technique

6.2 Data Structure Editor

A data structure editor revises the source code utilizing a search-and-replace functionality. After gathering the



Search Criteria: ( ) AND (●) OR Persistent_->h_edge_pos

Read: tomtom_log_variable_analysis[1000] ++;

Write: tomtom_log_variable_analysis[1001] ++;

( ) replace (●) add to end of line ☐ Use Keyboard shortcut

```
/* redraw edges for the frame if decoding didn't complete */
// just to make sure that all data is rendered.
if (CONFIG_MPEG_XVMC_DECODER && s->avctx->xvmc_acce
    ff_xvmc_field_end(s);
} else if((s->error_count || s->encoding || !(s->avctx->codec->capa
        !s->avctx->hwaccel &&
        !(s->avctx->codec->capabilities & CODEC_CAP_HWACCE
        s->unrestricted_mv &&
        s->current_picture.f.reference &&
        !s->intra_only &&
        !(s->flags & CODEC_FLAG_EMU_EDGE)) {
    int hshift = av_pix_fmt_descriptors[s->avctx->pix_fmt].log2_chro
    int vshift = av_pix_fmt_descriptors[s->avctx->pix_fmt].log2_chro
    s->dsp.draw_edges(s->current_picture.f.data[0], s->current_pi
            s->Persistent_->h_edge_pos, s->Persistent_->v_edg
            EDGE_WIDTH, EDGE_WIDTH,
            EDGE_TOP | EDGE_BOTTOM);
    s->dsp.draw_edges(s->current_picture.f.data[1], s->current_pi
            s->Persistent_->h_edge_pos >> hshift, s->Persistent_
            EDGE_WIDTH >> hshift, EDGE_WIDTH >> vshift,
            EDGE_TOP | EDGE_BOTTOM);
    s->dsp.draw_edges(s->current_picture.f.data[2], s->current_pi
            s->Persistent_->h_edge_pos >> hshift, s->Persistent_
```

```
demo@ubuntu:~/share/ff4 make
CC      libavcodec/mpegvideo.o
libavcodec/mpegvideo.c: In function âtomtom_log_mem_allocate_Persistentâ:
libavcodec/mpegvideo.c:149: warning: ISO C90 forbids mixed declarations and code
libavcodec/mpegvideo.c: In function âff_MPV_frame_endâ:
libavcodec/mpegvideo.c:1383: error: âMpegEncContextâ has no member named âintra_onlyâ
libavcodec/mpegvideo.c: In function âMPV_decode_mb_internalâ:
libavcodec/mpegvideo.c:2342: error: âMpegEncContextâ has no member named âintra_onlyâ
libavcodec/mpegvideo.c: In function âff_draw_horiz_bandâ:
libavcodec/mpegvideo.c:2571: error: âMpegEncContextâ has no member named âintra_onlyâ
make: *** [libavcodec/mpegvideo.o] Error 1
demo@ubuntu:~/share/ff4
```

Extract Error Lines 25

```
demo@ubuntu:~/share/ff5 make
CC      libavcodec/mpegvideo.o
libavcodec/mpegvideo.c: In function âtomtom_log_mem_allocate_Persistentâ:
libavcodec/mpegvideo.c:149: warning: ISO C90 forbids mixed declarations and code
libavcodec/mpegvideo.c: In function âff_MPV_frame_endâ:
libavcodec/mpegvideo.c:1383: error: âMpegEncContextâ has no member named âintra_onlyâ
libavcodec/mpegvideo.c: In function âMPV_decode_mb_internalâ:
libavcodec/mpegvideo.c:2342: error: âMpegEncContextâ has no member named âintra_onlyâ
libavcodec/mpegvideo.c: In function âff_draw_horiz_bandâ:
libavcodec/mpegvideo.c:2571: error: âMpegEncContextâ has no member named âintra_onlyâ
make: *** [libavcodec/mpegvideo.o] Error 1
```

Fig. 8. The code editor GUI helping with the search and renaming of suitable NVRAM variables.

```
#include "persist.h"

int a = 15;
persist int b = 25;
float persist c = 10;
persist struct
        {
          int x, y;
        } d;
```

Fig. 9. An example declaration with persist keyword.

profiling information, we determine the mapping of variables to memory modules. To realize the mapping in the code, we may need to change the structure of the code. The data structure editor accelerates the code changes. In the revision of the source code, variables may be re-declared in a different way, or renamed to enable the compiler to assign it to NVRAM. Also, in some cases, variables may be a member of a structure datatype, and not all members will be suitable for NVRAM assignment. In such a situation, we cannot achieve the desired result, since structure datatype members are stored in memory in a continuous format. To solve the continuity problem, some structures should be split, and variables suitable for NVRAM will be contained in a second data structure. This modification requires the revision of the entire source code accessing the original structure. Making such a large-scale modification to source code can be aided with the code editor. Fig. 8 shows the code editor interface and FFmpeg application source under modification.

6.3 Persistent Keyword

In order to map variables to a specific memory module, we introduce a persistent keyword that is integrated into the compiler syntax. This keyword informs the compiler whether the variable declared is a read-intensive variable and suitable for NVRAM module assignment. To use the keyword in an application, it is necessary to include header file *"persist.h"* in the application. This header file contains the implementation mechanism for making the compiler

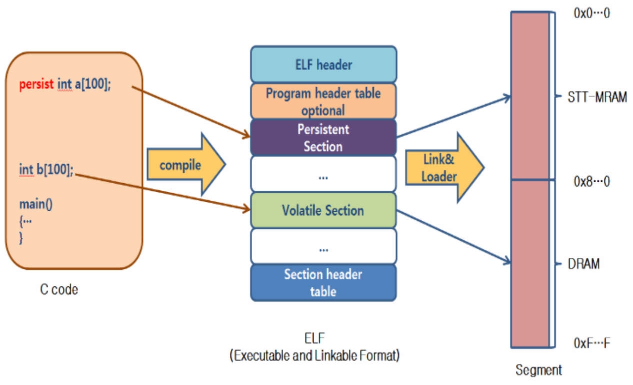


Fig. 10. Persist keyword support mechanism.

aware of the new keyword. The implementation mechanism places variables in a *"persist"* section using the *#define* keyword, which is applicable to only global variables. Then, the compiler assigns the variables to the "persist" section, and a linker maps the section to the memory address for NVRAM using a link script developed in this paper. The link script controls the memory layout using the "MEMORY" command. The command indicates the location and the size of the memory blocks for each memory module. The memory area is defined by the *">region"* command, and places memory sections into specific memory areas. For example, if there is a memory area named "persist", the ">persist" command will place variables in that area.

6.4 Case Study: Hybrid Memory-aware FFmpeg Application

We applied the techniques mentioned in the previous subsections to revise the FFmpeg application by assigning read-intensive or NVRAM-suitable variables to the NVRAM module. The read/write access pattern profiler collects the read/write frequency for each variable in FFmpeg. In FFmpeg, the MPEGContext structure is the main structure, and the profiling concentrates on analyzing this structure. The structure includes members for the main components: status information, previous and current frame being decoded, etc. Its member count is more than 250, with different data types and arrays of different sizes. Therefore, the profiling examines the memory access for all members in the structure. Results shows that some members are read-intensive and others, write-intensive. So, we split the structure and create a second structure; all read-intensive members are moved to the second structure, which the intern mapped into the NVRAM memory region. After this structure-splitting modification, all instructions in the entire source code referencing members that have been moved to the second structure need to be updated. The code editor is employed to aid in this task. Fig. 11 shows the divided MPEGContext structure, and Fig. 12 shows the modification of a code line with reference to a member in the second structure, which will be mapped into NVRAM (STT-MRAM). Fig. 13 shows the Qemu emulator running FFmpeg with the modified MPEGContext structure.

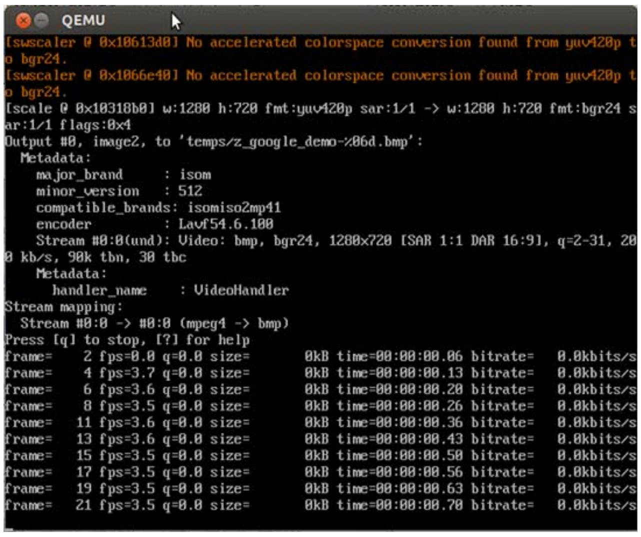
```
typedef  struct  MpegEncContext
          {
            AVClass *class;
            struct AVCodecContext *avctx;
            .
            .
            .
            MpegEncContext_Persist_members  *persist ;
          } MpegEncContext ;


persist struct MpegEncContext_Persist_members
          {
           int  qscale;
           int  mb_stride;
            .
            .
            .
          } MpegEncContext_Persist_members ;
```

Fig. 11. Split of structure data in FFmpeg.

```
xy = s->mb_x + s->mb_y * s->mb_stride;
```

```
xy = s->mb_x + s->mb_y * s->persist->mb_stride;
```

Fig. 12. Modification of code referring to the revised data structure in FFmpeg.



```
[swscaler @ 0x10613d0] No accelerated colorspace conversion found from yuv420p t
o bgr24.
[swscaler @ 0x1066e40] No accelerated colorspace conversion found from yuv420p t
o bgr24.
[scale @ 0x10318b0] w:1280 h:720 fmt:yuv420p sar:1/1 -> w:1280 h:720 fmt:bgr24 s
ar:1/1 flags:0x4
Output #0, image2, to 'temps/z_google_demo-%06d.bmp':
  Metadata:
    major_brand     : isom
    minor_version   : 512
    compatible_brands: isomiso2mp41
    encoder         : Lavf54.6.100
    Stream #0:0(und): Video: bmp, bgr24, 1280x720 [SAR 1:1 DAR 16:9], q=2-31, 20
0 kb/s, 90k tbn, 30 tbc
    Metadata:
      handler_name    : VideoHandler
Stream mapping:
  Stream #0:0 -> #0:0 (mpeg4 -> bmp)
Press [q] to stop, [?] for help
frame=    2 fps=0.0 q=0.0 size=       0kB time=00:00:00.06 bitrate=   0.0kbits/s
frame=    4 fps=3.7 q=0.0 size=       0kB time=00:00:00.13 bitrate=   0.0kbits/s
frame=    6 fps=3.6 q=0.0 size=       0kB time=00:00:00.20 bitrate=   0.0kbits/s
frame=    8 fps=3.5 q=0.0 size=       0kB time=00:00:00.26 bitrate=   0.0kbits/s
frame=   11 fps=3.6 q=0.0 size=       0kB time=00:00:00.36 bitrate=   0.0kbits/s
frame=   13 fps=3.6 q=0.0 size=       0kB time=00:00:00.43 bitrate=   0.0kbits/s
frame=   15 fps=3.5 q=0.0 size=       0kB time=00:00:00.50 bitrate=   0.0kbits/s
frame=   17 fps=3.5 q=0.0 size=       0kB time=00:00:00.56 bitrate=   0.0kbits/s
frame=   19 fps=3.5 q=0.0 size=       0kB time=00:00:00.63 bitrate=   0.0kbits/s
frame=   21 fps=3.5 q=0.0 size=       0kB time=00:00:00.70 bitrate=   0.0kbits/s
```

Fig. 13. Execution of FFmpeg decoding video frames.

7. Experiment

In this paper, we analyze the read/write access patterns of members in the MPEGContext structure found in the FFmpeg application. The structure is at the core of the FFmpeg internal decoding process. FFmpeg is executed in a virtual machine environment by the extended Qemu emulator. We decode frames for 12 video sequences shown in Table 4. During execution, every CPU access to members of the MPEGContext structure is automatically trapped and logged. Table 2 represents the virtual machine environment. To compute the energy consumption, we use the energy properties from Wu et al. [13], as shown in Table 3.

Table 4 shows the experimental results and compares a DRAM-only architecture to the proposed hybrid memory architecture for the 12 video sequences. The read-energy consumption in the hybrid memory system is much less than the DRAM-only system, while the write-energy in hybrid memory is slightly larger. On average, the proposed hybrid memory saves 21.89% in total energy consumption, compared with the DRAM-only architecture.

For most of the video sequences, energy reduction is always attained by the proposed hybrid memory system.

Table 2. Experiment Environment.

Emulator	R/W extraction-enabled Qemu Emulator
Emulated Board	ARM Versatile/PB (ARM926EJ-S)
CPU Architecture	ARMv5 (L1 cache 32K,L2 cache 128K)
RAM	256MB
Operating System	Linux (Debian)

Table 3. Parameters of memory technologies (45nm) [13].

RAM Technology	Latency (cycles)	Dynamic Energy (nJ)	Static Power (W)
SRAM	8	0.388	1.36
DRAM	24	0.72	0.4
MRAM (STT-MRAM)	Read: 20 Write: 60	Read: 0.4 Write: 2.3	0.15

8. Conclusion

This paper proposes optimal memory allocation in a hybrid DRAM/STT-MRAM main memory architecture. Memory is allocated through a compiler modification, in which data is placed in STT-MRAM in an orderly way: instruction code, constant variables, and then variables with the *"persist"* declaration keyword. The widely used FFmpeg audio/video library is analyzed to extract read-intensive variables that are best suited to STT-MRAM memory allocation. And we present a modified FFmpeg version (without any added complexity) that is aware of hybrid memory by declaring read-intensive variables separately. Our experiment shows an average 21.89% reduction in energy consumption.

Our future work aims to bring automation to all steps in migrating existing applications to be hybrid memory–aware. We aim to fully automate the revision of application code at the compiler level, rather than the source code level. Our work involves leveraging flexibility found in the LLVM compiler framework. We will develop a compiler pass for altering applications in the intermediate format. This will take care of data structure splitting, variable redefinition, assigning the appropriate memory segment attribute to variables, etc.

Table 4. Experiment results.

				Read Energy		Write Energy		Total Energy Consumption	
Sequence Name	**Format**	**Size**	**Frames**	**DRAM**	**Hybrid**	**DRAM**	**Hybrid**	**DRAM**	**Hybrid**
Akiyo	H.264 lossless	352x288	300	4098.88	3443.64	111.91	115.86	4210.79	3559.50
Akiyo	Mpeg2video	352x288	300	4362.78	3843.13	482.72	494.49	4846.50	4337.63
Foreman	X264, high profile	352x288	300	4534.16	3817.07	301.78	345.49	4835.94	4162.56
Foreman	H.264 lossless	352x288	300	4275.63	3617.51	11.91	115.86	4387.54	3733.37
BigBuckBunny	Xvid (Mpeg4)	352x288	2160	46751.84	40763.16	3874.03	3908.95	50625.87	44672.11
BigBuckBunny	X264, high profile	352x288	2160	33807.85	28611.88	2201.10	2621.18	26008.69	31233.05
Pedestrian_area	Xvid (Mpeg4)	640x360	375	19379.35	16951.07	1441.16	1447.27	20820.51	18398.33
Pedestrian_area	X264, high profile	352x288	300	13139.92	11098.20	865.41	1051.24	14005.33	12149.44
Crowd_run	Xvid (Mpeg4)	1920x1080	300	19379.35	17249.30	1441.16	1443.70	20820.51	18693.01
Mother_daughter	lossless H.264	352x288	300	4098.88	3442.86	111.91	116.34	4210.79	3559.20
Crowd_run	lossless H.264	640x360	500	19379.35	16598.64	1441.16	1462.18	20820.51	18060.83
Elephants_dreams	lossless H.264	352x288	1065	46751.84	40565.35	3874.03	3912.37	50625.87	44477.72

Acknowledgement

This work was supported by Basic Science Research Programs through the National Research Foundation of Korea(NRF) funded by the MSIP (2013R1A1A1013384) and by IT R&D program MKE/KEIT (No. 10041608, Embedded system Software for New-memory based Smart Device).

References

[1] M. H. Kryder, C. S. Kim, "After Hard Drives—What Comes Next? ", *IEEE Transactions on Magnetics*, vol. 45, no. 10, pp. 3406- 3413, Oct. 2009. Article (CrossRef Link)

[2] P. Mangalagiri, A. Yanamandra Y. Xie, N. Vijaykrishnan, M. J. Irwin, K. Sarpatwari, O. O. A. Karim, "A Low-Power Phase Change Memory Based Hybrid Cache Architecture", GLSVLSI 08, May 2008. Article (CrossRef Link)

[3] K. Lee, A. Orailoglu, "Application specific non-volatile primary memory for embedded systems", CODES+ISSS 08, pp 31-36. Article (CrossRef Link)

[4] M. Kandemir, G. Chen, F. Li, I. Demirkiran, "Using data replication to reduce communication energy on chip multiprocessors", ASP-DAC 05, pp.769-772. Article (CrossRef Link)

[5] J.Hu, C. J. Xue, W. Tseng, Y. He, M. Qiu, E.H.-M. Sha, "Reducing Write Activities on Non-volatile Memories in Embedded CMPs via Data Migration and Recomputation", DAC 10, June 2010. Article (CrossRef Link)

[6] The FFmpeg website. http://ffmpeg.org/. Article (CrossRef Link)

[7] Video Sequence. http://trace.eas.asu.edu/yuv//. Article (CrossRef Link)

[8] Video Sequence. http://media.xiph.org/video/derf//. Article (CrossRef Link)

[9] The Qemu website. http://wiki.qemu.org/Main_Page. Article (CrossRef Link)

[10] G. Dhiman, R. Ayoub, and T. Rosing, “PDRAM: A hybrid PRAM and DRAM main memory system” In Proceedings of the 46th Annu-al Design Automation Conference (DAC09), 2009. Article (CrossRef Link)

[11] Hyunsun Park, Sungjoo Yoo, Sunggu Lee, “Power Management of Hybrid DRAM/PRAM-Based Main Memory” In Proceedings of the 48th Design Automation Conference (DAC2011), 2011. Article (CrossRef Link)

[12] Hyunchul Seok, Youngwoo Park, Kyu Ho Park “Migration Based Page Caching Algorithm for a Hybrid Main Memory of DRAM and PRAM” In Proceedings of the 2011 ACM Symposium on Applied Computing (SAC 2011), 2011. Article (CrossRef Link)

[13] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, Yuan Xie “Hybrid Cache Architecture with Disparate Memory Technologies“ In Proceedings of the 36th annual international sym-posium on Computer architecture (ISCA 2009), 2009. Article (CrossRef Link)

[14] R. F. Freitas and W. W. Wilcke, "Storage-class memory: The next storage system technology," IBM Journal of Research and Development, vol.52, no.4/5, pp.439-447,2008. Article (CrossRef Link)

[15] G.W. Burr, B.N. Kurdi, J.C. Scott, C.H. Lam,K. G., and R.S. Shenoy, "Overview of candidate device technologies for storage-class memory," IBM Journal of Research and Development, vol.52, no.4/5, pp.449-464, 2008. Article (CrossRef Link)

[16] Clinton W. Smullen, IV, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi , Mircea R. Stan. Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAMCaches. Department of Computer

Science and Department of Electrical and Computer Engineering University of Virginia. cws3k@cs.virginia.edu, vm9u@virginia.edu, an2z@virginia.edu, gurumurthi@virginia.edu, mrs8n@virginia.edu. Article (CrossRef Link)

[17] Radu Cornea, Alex Nicolau, Nikil Dutt. “Video Stream Annotations for Energy Trade-offs in Multimedia Applications” In Proceedings of The Fifth International Symposium on Parallel and Distributed Computing (ISPDC'06). Article (CrossRef Link)

[18] Radu Cornea, Alex Nicolau, Nikil Dutt. “Software Annotations for Power Optimization on Mobile Devices” In Proceedings of the con-ference on Design, automation and test in Europe, 2006. Article (CrossRef Link)

[19] Tiantian Liu, Yingchao Zhao, Chun Jason Xue, Minming Li” Power-Aware Variable Partitioning for DSPs with Hybrid PRAM and DRAM Main Memory” Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE. Article (CrossRef Link)

[20] Luis Angel Bathen, Nikil Dutt ” HaVOC: A Hybrid Memory-aware Virtualization Layer for On-Chip Distributed ScratchPad and Non-Volatile Memories” DAC 2012 June 3-7, 2012, San Francisco, Cali-fornia, USA. Article (CrossRef Link)

[21] C. Stancu, L. Bathen, N. Dutt, A. Nicolau, "AVid: Annotation Driven Video Decoding for Hybrid Memories", ESTImedia, 2012. Article (CrossRef Link)

Thomas Haywood Dadzie received a B.S. degree in Computer Science from Wisconsin University, Ghana in December 2006, and received his M.S. in Information Systems from Hanyang University, Seoul, South Korea, in December 2013. He is currently pursuing a Ph.D. in Information Systems at Hanyang University.

Seungpyo Cho received the B.S. degree in Computer Engineering in 2011 and the M.S. in Information Systems from Hanyang University, Seoul, South Korea, in 2013. He is currently pursuing a Ph.D. in Information Systems at Hanyang University.

Hyunok Oh received the M.S. and B.S. degree in Computer Engineering from Seoul National University, Korea, in 1998 and 1996, respectively. and He received the Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University, Korea, 2003. He is currently an associate professor in Hanyang University. His research interests include Cryptography, Non-volatile memory, Dataflow model and Real-time analysis.