D5.5.1 Preliminary Report on Architectural Techniques for power-efficiency

Seventh Framework Programme

Document Information

No. & Title of Deliverable : D5.5.1 Preliminary Report on Architectural Techniques for power-efficiency

Grant Agreement No. : 288653

Project Website : lpgpu.org

Date : 3rd August 2012

Delivery Date : 31st August 2012

Nature : P

Authors: Stefanos Kaxiras, Georgios Keramidas, Konstantinos Koukos, Iakovos Stamoulis

Contributors : Ben Juurlink (TUB)

Reviewers : Jan Lucas (TUB) and Mauricio Alvarez-Mesa (TUB)

Acknowledgement

The research leading to these results has received funding from the EU's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 288653.

1. Slack

1.1 Summary

In this work we are presenting the impact of slack by exploring state of the art GPU memory hierarchies under different Dynamic Voltage-Frequency Scaling (DVFS) configurations. DVFS is one of the most beneficial techniques for CPU's in terms of power. GPUs however do not gain significant benefit from that technique. In this report we are analyzing the impact of core DVFS for different memory frequencies into state of the art GPUs. For the evaluation we are using part of the NVIDIA-CUDA toolkit and some custom micro-benchmarks. Our analysis shows that DVFS can give significant energy benefit at architectures with restricted memory bandwidth, such as embedded or mobile GPUs. However in high-end discrete GPUs we observe that DVFS capabilities are not of the same importance as on mobile. As part of our work we discuss the limitations on the state of the art GPUs, we analyze power efficiency issues and suggest novel solutions. In this preliminary report

we are presenting an exhaustive exploration of the DVFS capabilities for state of the art GPUs. We also present the basic ideas of GPU cache coherency protocols for data race free (DRF) applications. We design novel cache coherency protocols capable of reducing the power consumption on-chip caused by the current write-through protocols used in modern GPUs.

1.2 Description of Task

This task involves the following work:
• Modeling and estimation of Slack. Modeling of memory access behavior slack and load balance slack in GPUs and the stalls it can create. These models will be used to predict performance, energy, and optimal voltage frequency settings (DVFS) to maximize power efficiency in WP5.
• Design and development of techniques for exploiting load balancing slack: Develop dynamic (run-time) and static (profile-based) techniques necessary to DVFS or power down idle cores in load-imbalance situations.
• Design and development of techniques for exploiting memory access slack: Develop run-time mechanisms for GPU DVFS to determine optimal power-efficiency operational points according to memory behavior.

1.3 Relevance of the task to the project

Slack exploration and modeling will provide a strong fund for both runtime and architectural improvements. Reducing slack where it exists will improve power efficiency without any performance degradation. Our models will be used as a guide to optimize the frequency of the core with respect to program behavior and memory hierarchy. Because of the heavily parallel nature of GPU workloads, we focus on implementing that support on the architecture. DVFS can become a major factor of power saving. Our previous work on CPU's shows energy saving of up to 50% [12].

1.4 Work Performed

1.4.1 Infrastructure

We develop an infrastructure to measure power consumption of the different components comprising the total system. With our setup we can measure the power consumed by the CPU, the main memory and the GPU. To achieve this, we place current sensor boards between each component and the power supply. For each cable that supplies current to a component of interest, we develop a small board that contains a current sensor. These small boards are then placed at the top of a main board. The output of the power supply is connected to the input of the main board, and the output of the main board is connected to the motherboard. By measuring the voltage of each cable and the current that flows through the cable, we are able to determine the power consumed be the corresponding component. Modern GPUs are supplied with power: i) directly from the system power supply through dedicated cables and ii) through PCI-express pins. For the first source of power consumption we use one of the boards described above. There is a total of 6 cables supplying the GPU with power, so we use 6 current sensors. For the power supplied through the PCI express we use a PCI-express extender. This is a board that is placed between the motherboard and the GPU and allows us to have access to each of the PCI-express pins. Two voltage rails (of 12V and 3.3V) supply power to the GPU through PCI-express. By placing current sense resistors along these two voltage rails we are capable of measuring the current and consequently the power consumed in the GPU. With our current setup we are able to measure concurrently the power consumed by the CPU, the main memory and the GPU of two separate systems.

1.4.2 Slack and power efficiency on modern GPU architectures

Modern GPU architectures can be classified as high-end (discrete) and embedded or mobile GPUs. High-end GPUs targets gaming market and scientific community where they can be used as accelerators for parallel scientific workloads. These GPUs are characterized by the enormous memory bandwidth they have which is typically 15-20 times higher than the main memory. Architecturally they do not feature memory coherency, branch prediction or any other sophisticated components used in CPU's. Even caches are newly introduced into such architectures. The design mainly focuses on performance and uses power constraints only to the degree of working TDP. These cards typically exceed 250W operational power. On the other hand embedded or mobile GPUs are designed with respect to power consumption either because they are on the same package with a

CPU and can afford only a portion of the die TDP or because of battery autonomy. Our work is extended on both classes of GPUs. The power efficiency of GPUs comes from the enormously high bandwidth they have from device memory. When an application is stalled at a cache miss (waiting for that to be resolved from the main memory) the clock rate of the core becomes irrelevant to the real time required for the miss to be resolved. Reducing the core frequency results in less idle core cycles while the total execution time remains the same. Slack is the ability of a program (on a specific system) to reduce the executing frequency without any severe performance degradation.

1.4.3 Analyzing the power consumption of components (NVIDIA fermi)

In this paragraph we a re discussing the power consumption measurements taken from a state of the art discrete GPU (NVIDIA fermi - GTX580). The card power consumption is 90W idle (without driving a screen) and goes up to 250W when running scientific workloads. It is roughly impossible to measure the power consumption of each component inside the GPU without interference to the voltage regulators. Each modern GPU is supplied by 4 main resources as listed:

- PCI-Express (12V input)
- PCI-Express (3.3V input)
- 8-Pin PSU adapter
- 6-Pin PSU adapter

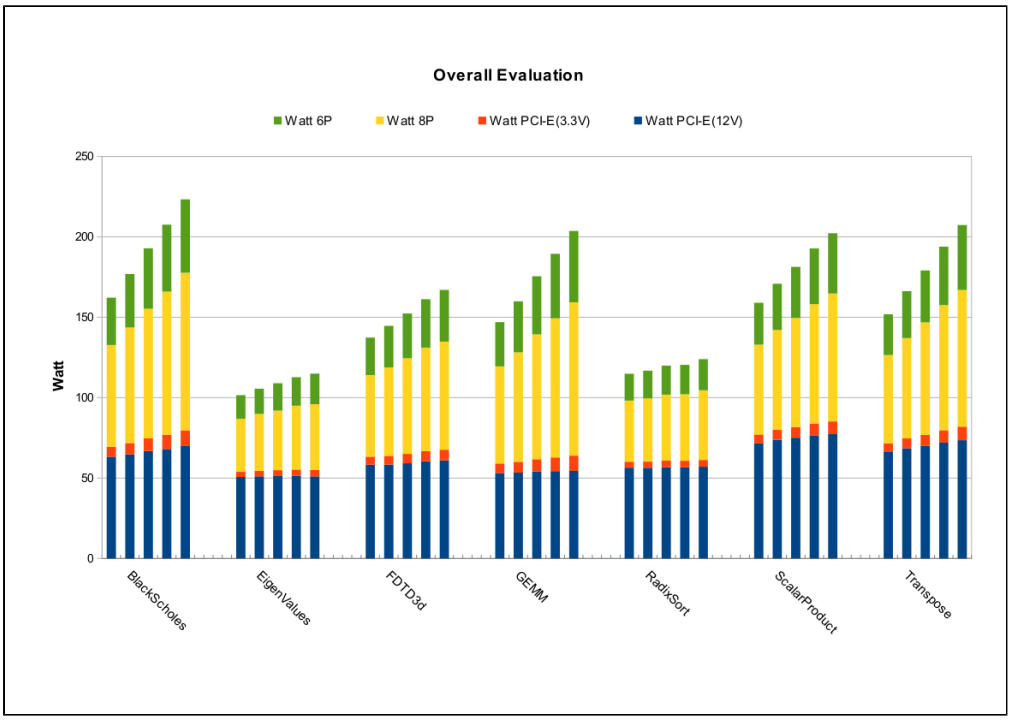

FIGURE 1.

As shown in FIGURE 1 almost 80% of the power comes from 12V PCI-E and 8-Pin PSU adapter. Each cluster of bars in the graph shows the power for different core frequency from 800MHz - 1600MHz at a step of 200MHz. The power consumption is load balanced across all available input rails to the card and is increased proportionally. Our study is further complicated by the fact that memory frequency does not only affect the power consumed by the memory itself but can also increase the power consumption of the GPU core. To verify that, we run our benchmark suite at different core and memory frequencies. This effect is caused from the thread scheduler of the card. Doubling the memory frequency almost doubles the bandwidth and that allows the thread scheduler to better utilize the GPU core (increasing both performance and power consumption). The performance impact of reducing frequency is severe and in all cases results to worst Energy-Delay Product (EDP).

1.4.4 Analyzing the bandwidth (NVIDIA fermi)

In this section we are discussing the impact of core DVFS to the GPU bandwidth. We observe that only at the lowest core frequency there is a significant bandwidth decrease of 20% as shown in FIGURE 2. Further study on power consumption using bandwidth micro-benchmarks, shows that memory is responsible only for a small portion of the total power spent by the GPU. FIGURE 3 shows the power consumption of a typical bandwidth micro benchmark. Doubling the core frequency increase the power by a maximum of 5% and leads to a maximum of 20% bandwidth improvement. For that benchmark the best EDP is at 1GHz core frequency 4.92% better than running at highest frequency. In the next section we further discuss DVFS impact on real applications.

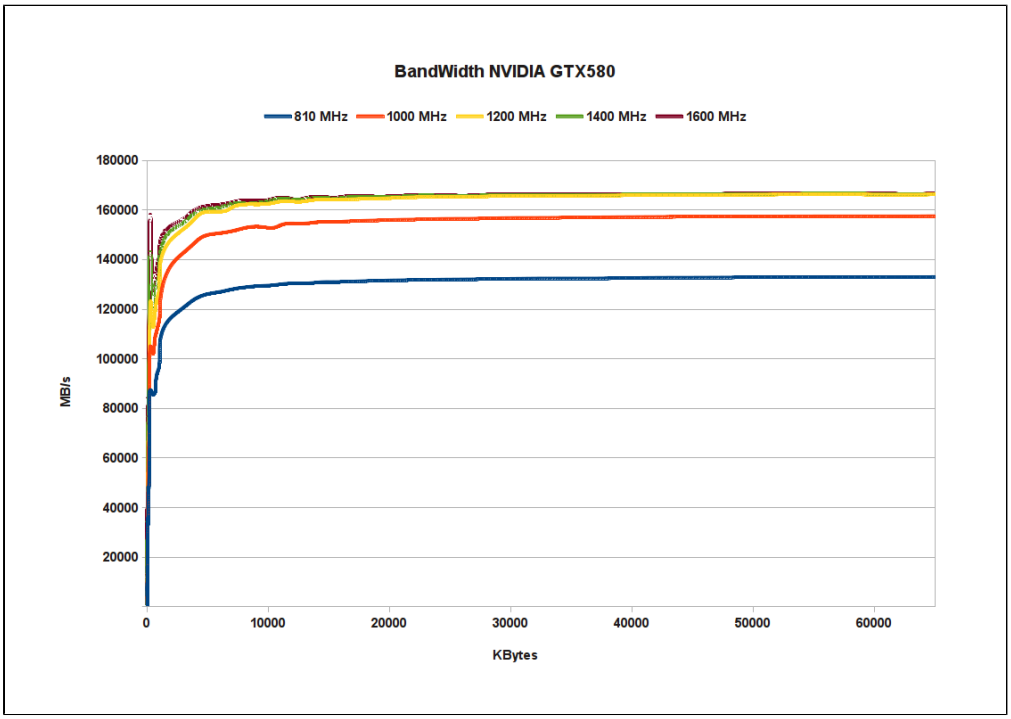

FIGURE 2.

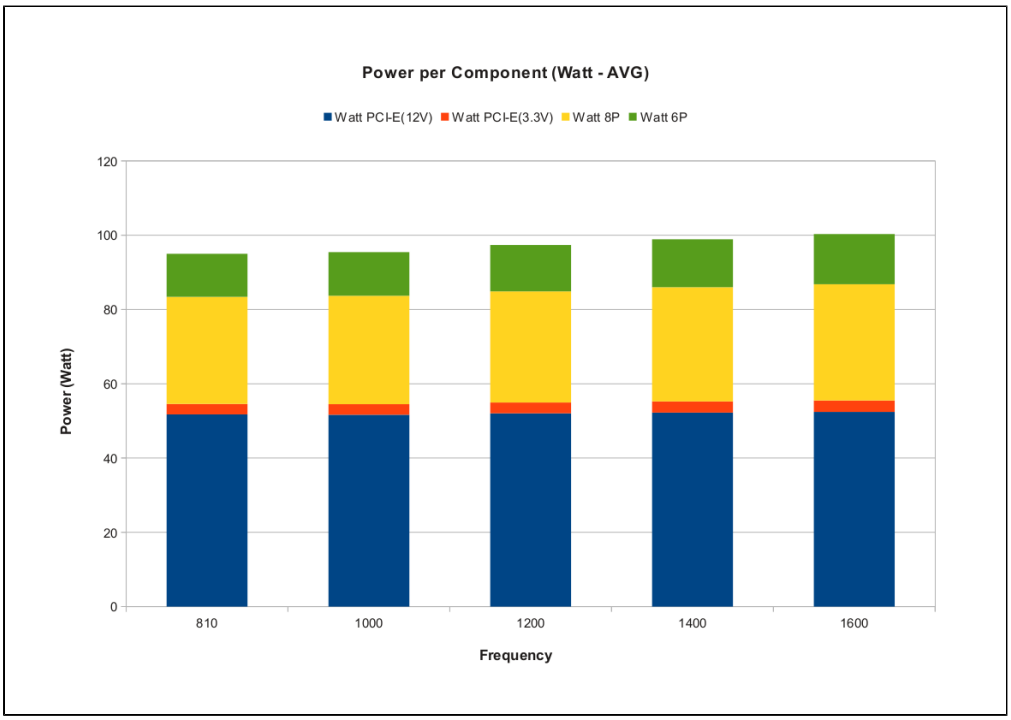

FIGURE 3.

1.5 Evaluation

1.5.1 Evaluating DVFS (NVIDIA fermi)

As noticed on the previous section high end discrete GPUs have an enormous bandwidth compared to main memory. This becomes the main restriction when using traditional DVFS techniques to improve power efficiency. DVFS is a very useful technique to improve power efficiency for applications that are highly memory bound. In section 1.4.2 we give a definition of slack. Many heuristics have been used as a criterion for reducing frequency such as miss rate and IPC in CPUs. These ideas can be used successfully in GPUs when necessary. FIGURE 4 depicts the EDP for a subset of the CUDA toolkit benchmarks for different core frequencies. We observe that all applications have a best EDP at the highest frequency. There are two reasons for that:

- The enormous device memory bandwidth on the card (up to 170 GB/s total)
- The enormous amount of outstanding threads per core (up to 1536 threads)

The NVIDIA fermi architecture uses giga-thread scheduling engine that allows up to 1536 threads to be scheduled on the same core. When some thread is stalled the scheduler is responsible to schedule the next immediate ready thread on the core. The L1 access latency is 200 core cycles, while the memory access latency is 800 core cycles [6]. With that configuration it is impossible for some core to become idle even by excluding scheduling overhead. The impact of DVFS for real applications varies from 30% for the case of radix sort till 3.5x for heavily computational bound applications. The observation that the best EDP is observed always at the highest frequency, defines complete lack of slack for high end GPUs. For our study we are using an NVIDIA GTX580, the architecture and specifications of which is described in [14]. For the evaluation workload we are using CUDA SDK benchmarks [15].

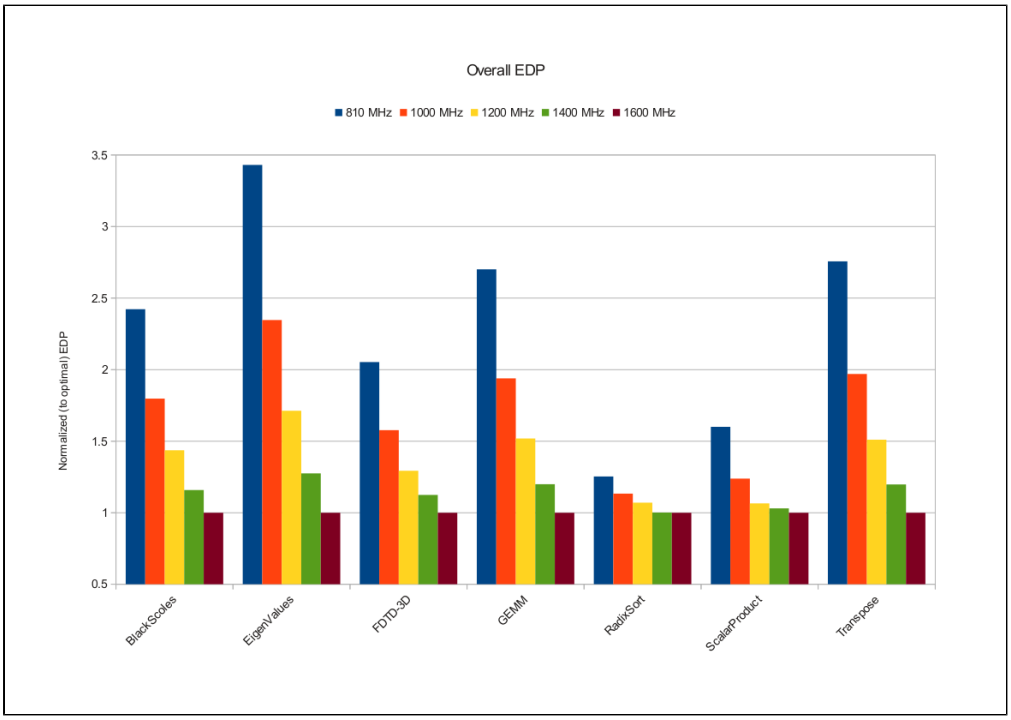

FIGURE 4.

1.5.2 Evaluating on restricted bandwidth

The conclusions from previous section show that the best EDP is achieved by running applications at the maximum core frequency. Although this observation is orthogonal for discrete cards it does not apply to embedded or mobile GPUs. In the latter the GPU has a restricted memory bandwidth depending on the implementation. Usually it can access the same L3 cache as the processor, have direct access to the main memory or use some dedicated link to the CPU cores. Even in the best case that the selected memory hierarchy is on chip (L3) the total available bandwidth is less than the bandwidth available on discrete cards. In this section we are evaluating the performance and DVFS impact on a memory frequency close to that of the main memory. We are reducing the memory frequency close to that of modern DDR3 modules at 2GHz which is half of the maximum card frequency. At this memory frequency the bandwidth is measured at 90GB/s which is 5x-6x higher

than high-end DDR3 memories and even higher than state of the art L3 caches. FIGURE 5 shows DVFS impact when running at restricted bandwidth. In most of the cases the best EDP is observed when running at the lowest frequency. Eigen-Values application is a proof that a very computational bound application will benefit from higher core speed. Memory bound applications tend to have best EDP close to the lowest frequency resulting to a maximum 40% EDP reduction. With that setup we emulate less than half of the theoretical throughput 18Gpixels/second at core frequencies varying from 800MHz to 1.6GHz. Compared with a low power GPU such as the ARM mali 450MP which operates at 480 MHz providing roughly 3.8Gpixels/second our setup can be characterized as efficient to describe in terms of bandwidth and cores many generations of low power GPUs.

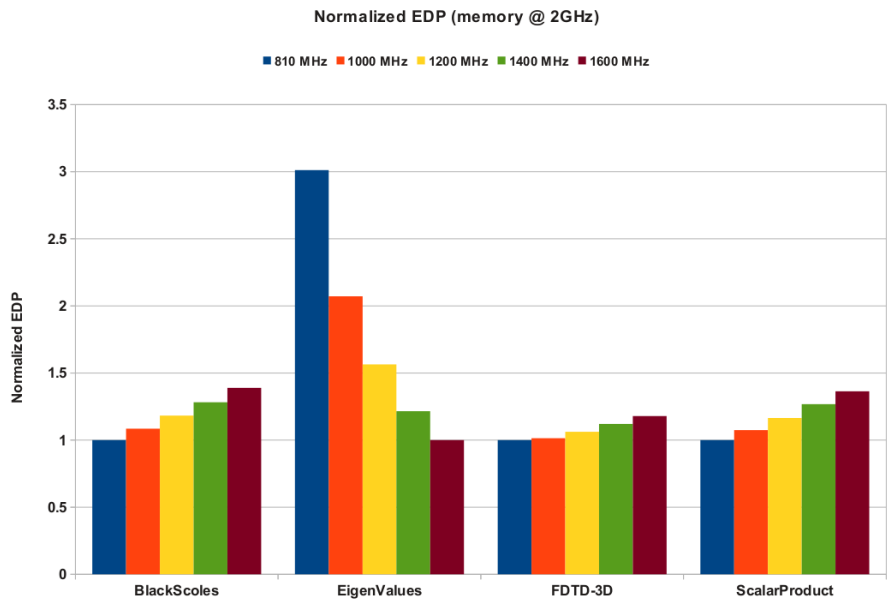

FIGURE 5.

1.6 Work status

We evaluate our current work on slack as: ***meet project requirements***. In this work we give a definition of slack. We analyze applications behavior under different core / memory frequencies. We simulate future low power embedded GPUs using state of the art HW and throttling the memory bandwidth. This knowledge can be used at all development stages of the project emphasizing to the final architecture deliverable. In the current phase we didn't exceed the state of the art because we depend on state of the art HW for our models. Despite that we detect the restrictions on the current state of the art HW and suggest mechanisms to solve them in our final architecture. Further that we need to analyze OpenCL programming model and state of the art applications for that in term of slack. We also need to compare SIMT vs VLIW architectures.

1.7 Conclusion

Our current work on slack is mainly focused on DVFS techniques for GPUs. We conclude that DVFS can be beneficial only in terms of restricted bandwidth as in the case of mobile or embedded GPUs. As part of the current work we are using offline profiling to gather performance statistics and correlate them with power measurements. We are currently restricted by the architecture and the driver on the HW events we can monitor. State of the art GPUs do not have performance counters for power which makes it impossible to create interactive interfaces and dynamic policies. In addition to that on embedded GPUs, DVFS support is very limited. Our future work on slack exploitation is mainly focused on analyzing in terms of performance and power the following:

- Number of threads per warp in terms of power efficiency.
- Exploit slack for reducing leakage power.
- Modelling optimal number of threads / cores for a given bandwidth

- Analyze die area distribution (cache vs cores) in terms of power efficiency
- Replace write through policies with more power efficient coherency protocols

1.8 Future work

1.8.1 HW accelerated GPU coherency

Our current work is mainly focused on DVFS techniques to improve power efficiency for the state of the art GPUs. Our current experience shows that it is really hard to apply such techniques without proper HW support. For the near future we focus on extending DVFS and power measuring support in our prototypes. Beyond that we are exploring the power efficiency of internal GPU components. Most of the power on a GPU is consumed by the core while most of the performance is gained by the memory hierarchy and the available bandwidth as shown in previous section. That leads to the conclusion that our future effort should be spent on the architectural improvements inside the core. One of the most power consuming components in the state of the art GPUs and CPU's is the Network on Chip (NoC). Modern GPUs independent of the programming model (CUDA or OpenCL) or manufacturer (NVIDIA, AMD, Intel, etc) have the following common features.

- Are programmed on relaxed coherency models
- Most of the workloads are data race free (DRF)
- They do not feature cache coherency across cores
- They mainly use write through policies for the on-chip caches (L1)

The use of write through policies is commonly accepted to be power inefficient. In this chapter we are discussing a novel protocol that comes to full accordance with the programming interfaces of OpenCL and CUDA and is capable of improving the performance and power efficiency of future GPUs and manycores. We suggest a novel protocol for relaxed memory consistency models named VIPS: Valid-Invalid / Private-Shared. The protocol has only valid 3 states Invalid, Private, Shared for the classification of each cache line. The simplicity of the protocol over MOESI or MESI is critical to reduce NoC traffic and ease the verification process. The protocol is discussed extensively in section 1.8.2.

1.8.2 GPU Coherency (VIPS Protocol)

FIGURE 6 gives the overall picture of our proposed coherency protocol for GPUs. The idea of the protocol is very simple and is based on dynamic selecting the write policy. For the protocol to work properly each cache line is marked either as shared or private on both L1 and LLC. In case of a private cache line the LLC keeps track of the sole owner of the cache line. There is no dictionary required on our architecture. We are using write-back policy for the private cache lines and write-through policy for the shared lines. The clean-dirty bit is used to keep track of the state of a private cache line. If a private cache line is dirty then we need to update the data to the higher level cache upon eviction. Private cache lines don't need to be flushed on synchronization points. Private cache lines are only updated upon program termination and replacement. For the shared cache lines we are using self invalidation upon synchronization points. The only transition in our protocol is "private to shared". When a cache line is requested by a remote core the LLC already has a track of the cache line and its owner if is private. Upon a miss resolve the LLC sends a share request to the owner, that has to update the data if modified to the LLC and mark the cache line as shared. After that the LLC replies with the data and the shared state. The LLC also updates the state to shared and ignore the previous owner. All requests to that cache line after this point will be replied as shared from the LLC. There is no shared to private transition in the protocol. For a cache line to be retrieved as private all cores need to invalidate that cache line. For the caches we are using the following setup:

- Write allocate L1, non-write allocate LLC
- Non-inclusive, non-blocking

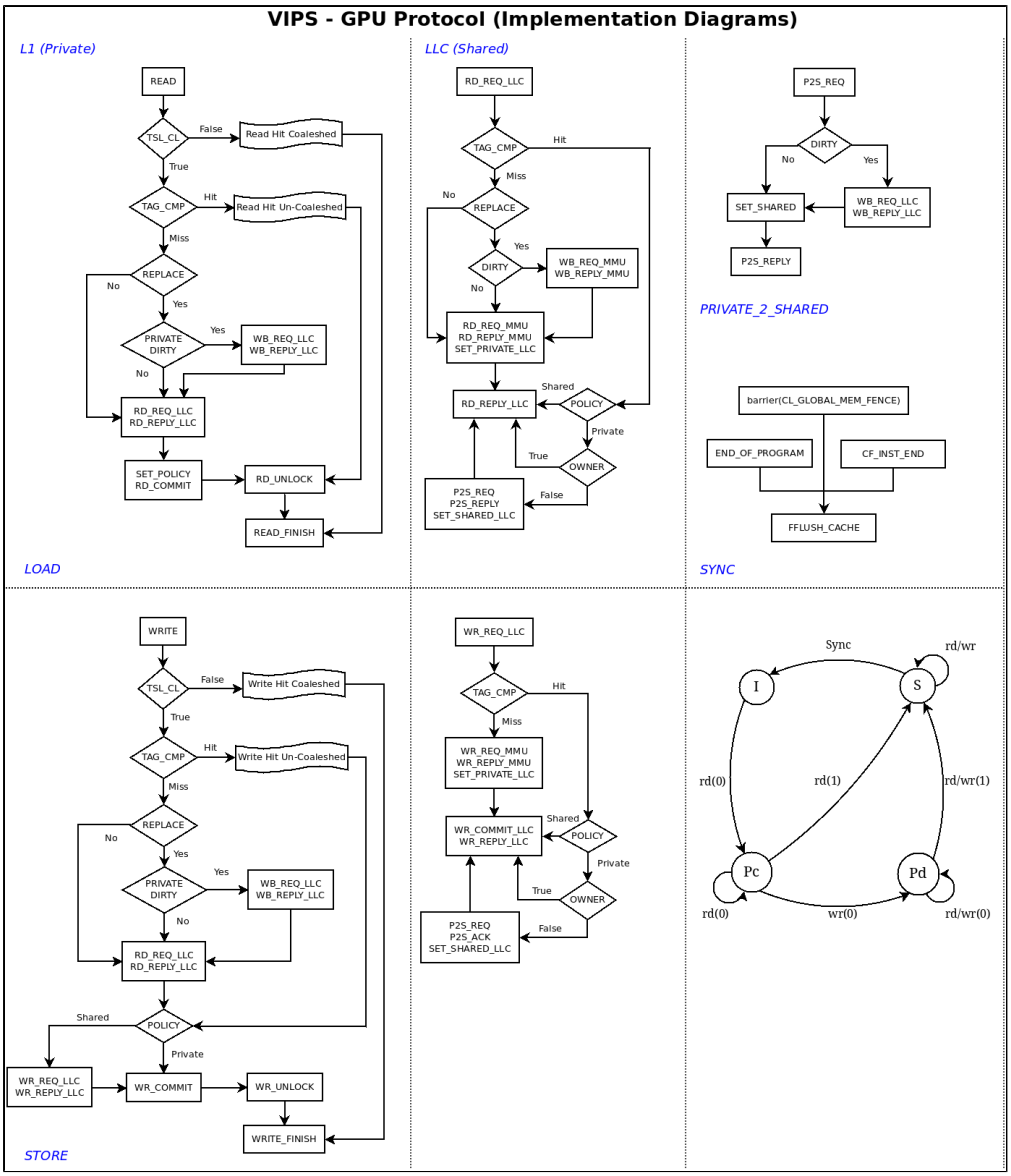
VIPS - GPU Protocol (Implementation Diagrams)
L1 (Private)
READ
TSL_CL
False
Read Hit Coaleshed
True
TAG_CMP
Hit
Read Hit Un-Coaleshed
Miss
REPLACE
No
Yes
PRIVATE DIRTY
Yes
WB_REQ_LLC WB_REPLY_LLC
No
RD_REQ_LLC RD_REPLY_LLC
SET_POLICY RD_COMMIT
RD_UNLOCK
READ_FINISH
LOAD
LLC (Shared)
RD_REQ_LLC
TAG_CMP
Hit
Miss
No
REPLACE
Yes
DIRTY
WB_REQ_MMU WB_REPLY_MMU
No
RD_REQ_MMU RD_REPLY_MMU SET_PRIVATE_LLC
Shared
RD_REPLY_LLC
POLICY
Private
True
OWNER
False
P2S_REQ P2S_REPLY SET_SHARED_LLC
P2S_REQ
DIRTY
No
Yes
SET_SHARED
WB_REQ_LLC WB_REPLY_LLC
P2S_REPLY
PRIVATE_2_SHARED
barrier(CL_GLOBAL_MEM_FENCE)
END_OF_PROGRAM
CF_INST_END
FFLUSH_CACHE
SYNC
WRITE
TSL_CL
False
Write Hit Coaleshed
True
TAG_CMP
Hit
Write Hit Un-Coaleshed
Miss
REPLACE
No
Yes
PRIVATE DIRTY
Yes
WB_REQ_LLC WB_REPLY_LLC
No
RD_REQ_LLC RD_REPLY_LLC
Shared
POLICY
Private
WR_REQ_LLC WR_REPLY_LLC
WR_COMMIT
WR_UNLOCK
WRITE_FINISH
STORE
WR_REQ_LLC
TAG_CMP
Hit
Miss
WR_REQ_MMU WR_REPLY_MMU SET_PRIVATE_LLC
Shared
WR_COMMIT_LLC WR_REPLY_LLC
POLICY
Private
True
OWNER
False
P2S_REQ P2S_ACK SET_SHARED_LLC
Sync
rd/wr
I
S
rd(0)
rd(1)
rd/wr(1)
Pc
Pd
rd(0)
wr(0)
rd/wr(0)

FIGURE 6.

2. Redundancy and Accuracy

2.1 Summary

This section presents the work conducted in Task T5.2 (Redundancy) and Task T5.4: (Accuracy) during the first year of the project as defined in the LPGPU work plan. The target in Task T5.2 is to identify and exploit redundancy, either in calculations or in (off-chip) memory accesses, in various levels of a graphics processing system. The target in Task T5.4 is to devise informed performance or power efficient policies in which controllable errors are allowed to occur. More specifically, our goal is to dynamically decrease the accuracy of the computations or the memory accesses (e.g., by ignoring few low order bits) without significantly hurting the quality of the rendered images. Of course, by decreasing the accuracy of specific operations (calculations or memory accesses) noteworthy power and performance benefits can be reported. As a result, our aim in T5.4 is to establish a relation between QoS and accuracy and at a later step to exploit this tradeoff in order to create power saving policies with respect to the battery life of a handheld devise or the QoS requirements posed by the user or the application itself.
However, in the course of this work, we realized that the concept of redundancy and the concept of accuracy are tightly coupled and should not be studied separately. Consider for example that someone (as we did in this work) tries to eliminate some calculations (e.g., the calculations of the RGBA values of a new pixel) by remembering the results of the previous calculations to avoid re-calculations. If an exact match occurs then the concept of redundancy is exposed, but by ignoring a few bits from the input (source pixels) or the output (generated pixels), the possibility for a match (partial match in this case) is significantly increased. The question in this case is to what extent the quality of the rendered image will be affected.
In this section, we present techniques and methodologies to take advantage of the abovementioned concepts in order to increase the performance and the power efficiency of a typical graphics processing system. More specifically, the contributions of this work are as follows:

- Exploiting redundancy/accuracy in calculations by relying on memoization or work reuse techniques (those techniques can be applied either at the hardware or the application level). We choose to apply our methodology to two common functionalities of a graphics processing system and more specifically to typical image color transformations and to image blending transformations. Both techniques require a significant amount of complex floating point operations per image pixel.
- Exploiting redundancy/accuracy in off-chip memory accesses. The target is to substitute, to the extent possible, the expensive off-chip memory accesses with on-chip memory accesses. We opt to apply this technique to a resource (in terms of power, time and I/O bandwidth) consuming memory operation which is the writing/storing of a rendered image to the so-called framebuffer (without loss of generality we assume that the framebuffer is located in the system main memory or in a separate off-chip memory connected to the system bus).

In the rest of this section, we present our preliminary results in each of those directions and we will draw specific directions for future work.

2.2 Redundancy/Accuracy in Calculations

It is well-known that, low-power is an imperative requirement for portable multimedia devices employing various signal and image processing algorithms. However, in most multimedia applications, the final output is interpreted by human senses, which are not perfect. This fact obviates the need to produce exactly correct numerical outputs (accuracy). Previous research in this context exploits error-resiliency primarily through voltage over-scaling, utilizing algorithmic and architectural techniques to mitigate the resulting errors [3][4][11].
In the rest of this subsection, we propose a different direction to tackle the problem through a memorization or work reuse approach. The proposed technique offers significant benefits over the previous techniques since i) it can be applied at the software or at the hardware level, and ii) if applied at the hardware level, minimal modifications in the underlying hardware are required.

2.2.1 Color Transformations

2.2.1.1 Basic Functionality

This is a typical operation used in many graphics applications. The target is to transform or filter an input image using a user-defined matrix. The OpenVG standard defines this transformation as follows (called vgColorMatrix function) [10]:

$$\begin{bmatrix} R_{dst} \\ G_{dst} \\ B_{dst} \\ \alpha_{dst} \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \cdot \begin{bmatrix} R_{src} \\ G_{src} \\ B_{src} \\ \alpha_{src} \end{bmatrix} + \begin{bmatrix} m_{04} \\ m_{14} \\ m_{24} \\ m_{34} \end{bmatrix}$$

where R_{src}, G_{src}, B_{src}, and A_{src} are the RGBA values of each pixel of the input (source) image, while the R_{dst}, G_{dst}, B_{dst}, and A_{dst} are the new (generated) RGBA values of the transformed or filtered image. All the other matrix entries (starting with *m*) are set by the user and they actually formulate the exact functionality of the transformation. The above matrix calculations can be also transformed as follows:

$$R_{dst} = m_{00} R_{src} + m_{01} G_{src} + m_{02} B_{src} + m_{03} \alpha_{src} + m_{04}$$
$$G_{dst} = m_{10} R_{src} + m_{11} G_{src} + m_{12} B_{src} + m_{13} \alpha_{src} + m_{14}$$
$$B_{dst} = m_{20} R_{src} + m_{21} G_{src} + m_{22} B_{src} + m_{23} \alpha_{src} + m_{24}$$
$$\alpha_{dst} = m_{30} R_{src} + m_{31} G_{src} + m_{32} B_{src} + m_{33} \alpha_{src} + m_{34}$$

So it is clear, that for each color of each pixel of the input image four floating point multiplications and four floating point additions are required.
As an example, if we set the matrix entries using the following values (we assume that those values are used for the rest of this subsection):
m00 = 0.5, m01 = 0.311, m02 = 0.1, m03 = 0 , m04 = 0.1
m10 = 0.3, m11 = 0.5, m12 = 0.1, m13 = 0 , m14 = 0.1
m20 = 0.3, m21 = 0.3, m22 = 0.522, m23 = 0 , m24 = 0.1
m30 = 0, m31 = 0, m32 = 0, m33 = 0 , m34 = 255

The effect of the transformation can be seen in the following figure:

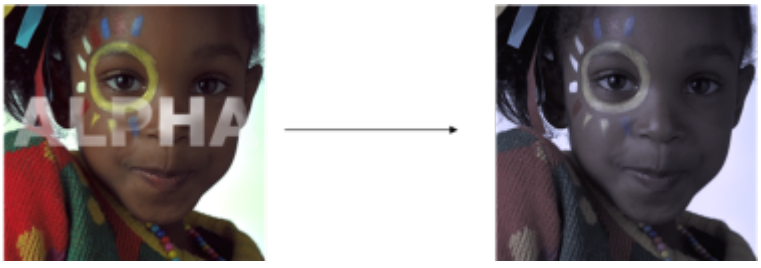

FIGURE 7: Image transformation example.

2.2.1.2 Result Memorization

Ignoring for the moment the implementation details, the following figure (FIGURE 8) depicts a high-level view of the proposed value memorization or work reuse technique. The top part of the figure shows the transformation of the input RGBA values to the output ones using four floating points multiplications and additions (as already explained in the previous paragraph), while the bottom part of the figure illustrates our approach. A specialized

memory structure is inserted in the design. This memory structure, called Value Cache or VC, is fed with input color values while for each input a corresponding output color value exists. The input-output pairs are created using the result of previous color transformations (previous pixels).

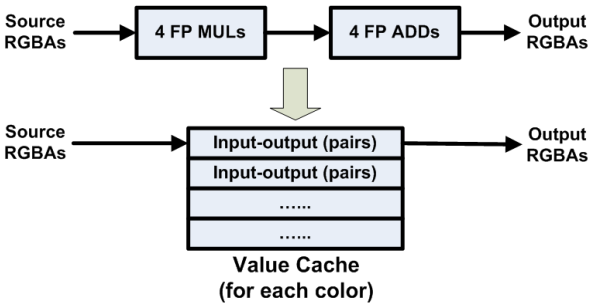

FIGURE 8: The Value Cache (VC) approach.

So given the above scheme, when a transformation of a new pixel color is requested, the following steps are taken place:

- We initially feed the VC with the new pixel color. If a match occurs (the same pixel color appeared previously in the pixel color stream), then the corresponding output is immediately retrieved bypassing the expensive (in power and performance) computations shown in the top part of FIGURE 8.
- If a mismatch occurs, then the input pixel color is redirected to the computational data path and when the new output is produced, the new input-output pair is stored to VC for future reuses.

2.2.1.3 Exploiting Accuracy

As already mentioned, the concept of exploiting redundancy and the concept of exploiting accuracy are tightly coupled. The design illustrated in FIGURE 8 fits very well within this scenario. Upon the arrival of a new pixel color (input), instead of trying to perform full matches between the new input and the VC contexts (inputs stored in VC from previous calculations), partial matches can be assumed. For example, if the new input and the already stored input in VC differ only in the last significant bits, then the possibility to avoid the re-calculations through the data-path pipeline will be significantly increased. The drawback of this technique is that we alleviate the "correctness tax" imposed in traditional computing systems since we may introduce errors in the output screen reducing the quality of the rendered image. The advantage of this technique is that it is possible to further reduce the energy consumption of the rendering process. As a result, a trade-off between QoS (what you loss) and energy consumption (what you get) is coming into the table. Finally, we should mention that the application domain of the LPGPU project is a perfect "fit" towards the "relaxed correctness" direction. The results presented in the Results Section showcase this tradeoff revealing that ample room for improvement exists.

2.2.1.4 Implementation

One additional advantage of the proposed value caching technique is that it can be applied either at the software or at the hardware level. While the design presented in FIGURE 8 advocates for a hardware implementation of the Value Cache, software alternatives are also possible. For example, the input-output pairs can be stored in the on-chip local memory (e.g., a scratchpad) and accessed/retrieved from the local memory at the software/application level. The following figure shows a possible embodiment of the value caching technique at the software level.

```
// original code
int pixelcolor_filter
(int input)
{
 int src_color = input ;
 int dst_color ;

     // computational part
     // 4 FP MULs & 4 FP ADDs

 return dst_color ;
}
```

```
// transformed code
int pixelcolor_filter
(int input)
{
 int src_color = input;
 int dst_color ;

 if ( src_color ==
local_mem[src_color] )
 {
  return local_mem[src_color+1];

 }
 // assuming the input-output
 // pairs are stored in
 // subsequent positions in the
 // local/scratchpad memory

 // computational part
 // 4 FP MULs & 4 FP ADDs

 // update local mem. with
 // the new input-output pair

 return dst_color ;
}
```

FIGURE 9: Value cache software implementation.

The results presented in this deliverable do not pertain to a particular implementation of the Value Cache, since our target in this step of the project is to exploit the potential of the redundancy and accuracy concepts without having in mind a specific implementation.

2.2.1.5 Benchmarks

To evaluate the effectiveness of our proposal, we rely on a wide range of source images from different application domains. More specifically, our benchmark suite is divided into three main categories: photos, games, and GUIs (Graphical User Interfaces).
The selected images are of different sizes, while in all cases 8-bit palette size is used for each RGBA value and the images are stored using 24-bit PNG lossless format. The following table shows the details for each category.

Category	Images	Size in pixels
GAMES (7 images)	*002hn4.png, Angry_birds2.png, Crysis_Aliens.png, crysis.png, NFSU11.png, Skyrim.png Uschtenheim.png.*	800x600
GUIs (4 images)	*Android1.png, android2215.png, bbcnews_800.png, OOo.png.*	800x600
PHOTOS (27 images)	*girl_alpha.png*	256x256
	lena.png, lorikeet.png	512x512
	kodim04.png, kodim09.png, kodim10.png, kodim17.png, kodim18.png, kodim19.png.	512x768
	kodim01.png, kodim02.png, kodim03.png, kodim05.png, kodim06.png, kodim07.png, kodim08.png, kodim11.png,kodim12.png, kodim13.png, kodim14.png, kodim15.png, kodim16.png, kodim20.png, kodim21.png, kodim22.png, kodim23.png, kodim24.png.	768x512

TABLE 1: Benchmarks.

The photos starting with *kodim* have been acquired by the Eastman Kodak Company and they actually represent an image suite, called Kodak Lossless True Color Image Suite [8], which is a standard test suite used by many researchers.

2.2.1.6 Timing and Energy Models

To obtain a high-level estimation of the power benefits achieved by the proposed approach we rely on the McPAT tool [5] which is the primary power modeling tool used in the LPGPU project. Since in our approach, only three operations are involved (i.e., multiplications, additions, and cache/scratchpad accesses), we normalized the time and energy numbers to those of the most lightweight operation which is the ADD operation in the current setup. In addition, we build our own simulator to analyze the input images and extract the required statistics (in terms of redundancy and accuracy).

Note that our target in this phase of the project is not to report the exact energy and timing figures but to gain a first estimation about the potential of the redundancy/accuracy concepts in typical graphics applications. The following table shows the timing and energy numbers assumed in our simulations.

Operation	Energy (norm.)	Latency (norm.)
ADD	2	1
MUL	8	3
Cache or scratchpad access	2.5	1

TABLE 2: Energy and performance estimates.

2.2.1.7 Error Metrics

When comparing two techniques or methodologies defining the notion of "better" is necessary. However when comparing two digital images, deriving the degree of similarity between the two images is a difficult process. Of course, if the two images are identical, it is a trivial task. But if those images differ, then appropriate error metrics are needed to measure the distance between images.

In this work, we used the root mean square error or RMS error to decide about the similarity of two images. Although the RMS error is a well established generic error measure, its main drawback, when used to compare images, is that it does not take into account the human perception. Using more appropriate error measures, like the Structural Similarity factor or SSIM [13], is left for future work.
Moreover, since those metrics are useful to get information for the whole image, we also used the *compare* instr uction (provided by [9]) in order to indentify the spatial characteristics of the errors with respect to the coordinates of the source/destination image.

2.2.1.8 Results

This section presents the results of our evaluation. It is divided into two main parts. The first part (FIGURE 10, FIGURE 11, and FIGURE 12) contains the simulation statistics when only the concept of redundancy is exploited. The second part (FIGURE 13 and FIGURE 14) depicts the gathered results when the concept of redundancy and the concept of accuracy are utilized at the same time (as mentioned those two concepts are orthogonal to each other).
So in this context, FIGURE 10 shows the potential (coverage) of exploiting the redundancy in calculations for the specific color transformation process. The vertical axis corresponds to the number of times in which the Value Cache reported a match or a hit (only full matches are considered in this case) normalized to the total number of VC accesses (number of pixels times the 4 RGBA colors). The blue and red bars illustrate the simulation results when the size of the VC is equal to one and two respectively. In the case of the 2-entry value cache, a LRU replacement algorithm is assumed, while other replacement policies may be promising as well like LFU or FIFO, but we left this analysis for future work (note that a similar eviction policy may be performed even for the 1-entry VC). Finally, the horizontal axis of FIGURE 10 shows the images that we study in this work grouped in three categories as explained in the Benchmarks Section.
As it can be seen from FIGURE 10, the number of redundant calculations (coverage) that can be captured by the proposed VC approach is significant in all cases. Even for the 1-entry VC, the reported coverage is well-above 30% in the majority of the applications. This means that at least 30% of the color calculations can completely bypass the data path (computation pipeline) achieving proportional timing and power benefits as it shown in FIGURE 11 and FIGURE 12 respectively. The minimum coverage is appeared in the *kodim16.png* ima ge (29.8%) while the maximum one in the *Ooo.png* image (96.53%). Exactly the same images show the minimum and maximum coverage for the 2-entry VC (31.74% and 97.56% respectively). Of course, the gathered coverage is also a function of the image category. As it is expected, the high definition images (photos) have the lower coverage (but above 30% in almost all cases), whereas the GUIs images offer the largest coverage numbers. Finally, by comparing the statistics of the 1-entry VC to those of the 2-entry VC, it is obvious that the 2-entry configuration is always superior to the 1-entry one. The distance between the two alternatives is image dependant starting from 1.03% (*Ooo.png*) up to 30.09% (*Angry_birds2.png*).

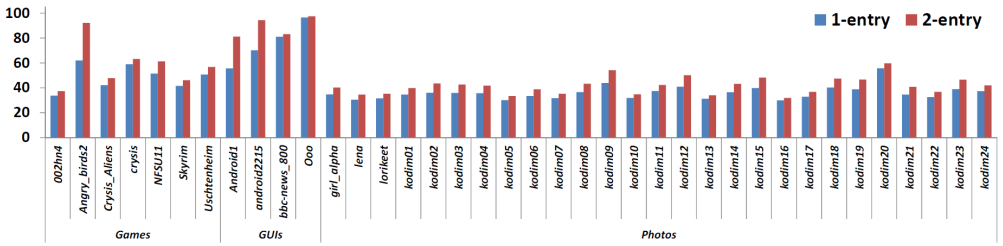

FIGURE 10: Coverage of redundant operations using 1 and 2-entry VC for all the images (only redundancy is exploited).

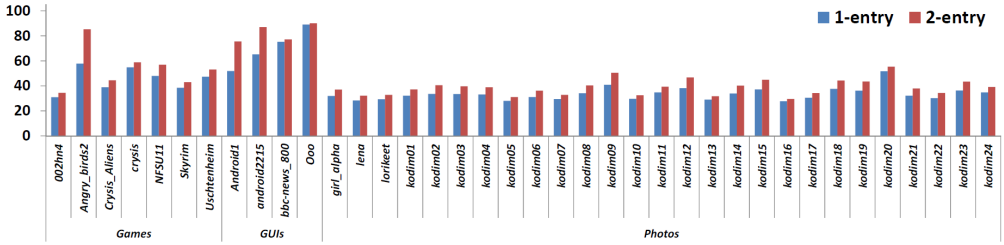

FIGURE 11: Power savings using 1 and 2-entry VC for all the images (only redundancy is exploited).

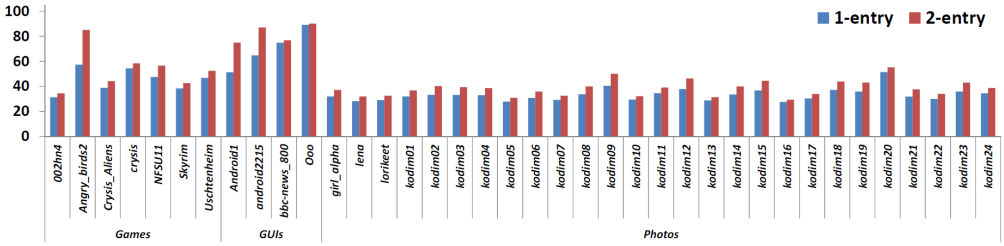

FIGURE 12: Execution time reduction using 1 and 2-entry VC for all the images (only redundancy is exploited).

Moreover, FIGURE 11 and FIGURE 12 depict the benefits, in terms of execution time and power reduction respectively, achieved by the 1-entry and the 2-entry VC. As it is expected, in both cases, the results strictly follow the results presented in FIGURE 10 (coverage). It is noteworthy to mention that our power savings approach is unique in the sense that it offers a reduction in power and in execution time simultaneously (compared to the baseline system), which means that if we choose to represent our results with a combined power metric (like Energy-Delay-Product or EDP), the achieved power benefits will be accordingly increased. Finally, we should mention at this point, that all the results presented in this section target only the implementation of the specific color transformation function, i.e., the power and time numbers are normalized to the power and time reported by the baseline system (no VC exists in the system) when executing only the specific image transformation.

Having verified the potential of removing the redundant calculations in the studied color transformation function, our next step is to examine the effect of ignoring a few low bits in the calculations (moving from full matches to partial matches when a new input color is directed to VC). FIGURE 13 quantifies this approach by illustrating the reporting coverage for the 1-entry (top graph in FIGURE 13) and the 2-entry (bottom graph) VC respectively. In both graphs, each bar depicts the resulting coverage for the image shown in the bottom of the graph. The bottom part of each bar (tagged as "0-bit") corresponds to the coverage when full matches are taken place (i.e., only the effect of redundancy is exploited), while every additional part on top shows the increase in coverage when 1, 2, 3, or 4 bits are excluded from the matching process (exploiting accuracy). Note that full matches assumed that the whole input color (8-bits long) is used. Of course, when partial matches occur, the output of the value cache will not necessarily be identical with the output produced through the data path decreasing accordingly the quality of the transformed output image (as mentioned we choose to quantify this error using the root square mean or RMS error).

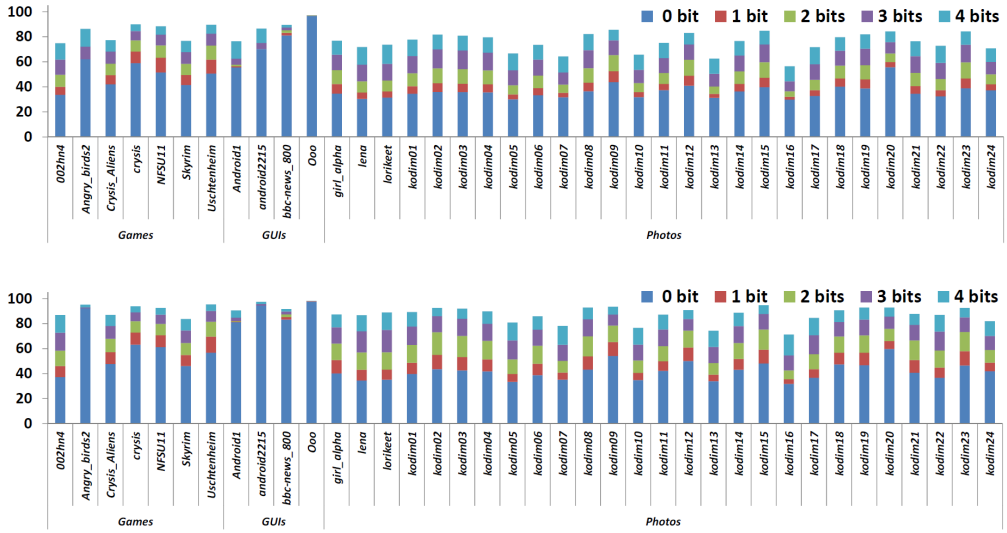

FIGURE 13: Coverage of redundant operations using 1 (top graph) and 2 entry (bottom) Value Cache and partial matching (exploiting accuracy) for all the images.

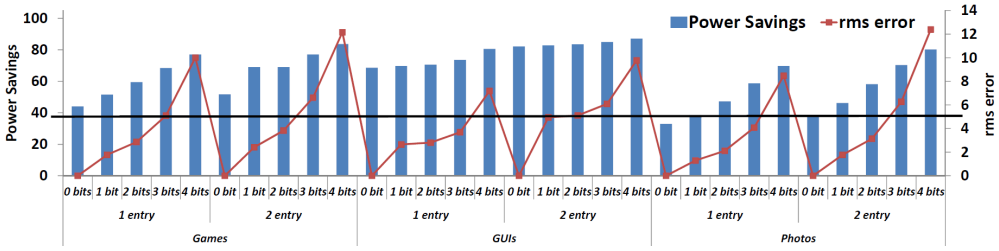

FIGURE 14: Power savings and RMS error for various configurations as averaged values for the three image categories.

As FIGURE 13 indicates, exploiting accuracy manages to further increase the strength of the proposed approach. For example, by ignoring two low order bits of the input, up to 22% (9% on average) increase in coverage is achieved for the 1-entry cache (29.58% and 11.52% for the two-entry cache respectively) compared to the "0-bits" VC configuration revealing the potential of the proposed technique. Of course, the more the number of bits that are ignored, the more the increase in coverage. More specifically, the average increase in coverage between the case of ignoring 0 bits (redundancy) and ignoring 4-bits (half of the input) is 37.11% for the 1-entry and 41.13% for the 2-entry VC.

However, in order to get the full picture of the resulting benefits, the RMS error in each case must be also taken into account. This is because a tradeoff between what you loss in image quality (by disregarding specific input bits) and what you get in power (which is proportional to coverage) exists. FIGURE 14 quantifies this view. In the interest of clarity, FIGURE 14 shows the averaged values for each of the three image categories. For each VC configuration (shown in the horizontal axis), the resulting energy savings (blue bars) and the RMS errors (red lines) are illustrated.

As we can see, a clear tradeoff between the resulting image quality and the power savings is appeared in all image categories. As in the redundancy case, the power savings numbers strictly follow the coverage numbers. As for the RMS error, it is clear that the resulting RMS error is application or category dependant. If we consider that a RMS error smaller that 5% (shown by the black line in FIGURE 14) is a safe error limit (see also the Error Metric section), then for the 1-entry VC, it is possible to ignore three bits from the input without violating the 5% limit in all image categories. For the 2-entry VC, the same safe limit is not violated if two bits from the input are ignored during the matching process. For the given safe limit, between the two design points (1-entry/3-bits and 2-entry/2-bits) and as FIGURE 14 indicates, the first design point provides larger power savings for images belonging in the Games and in the Photos categories, while the second design point seems more appropriate in the GUIs category. Of course, this analysis is application dependant and pertains to a specific safe limit (5%). As

a result, it becomes obvious that more design points can be put into the table and more importantly the selection of the appropriate points can be done dynamically (e.g., based on a feedback loop driven by the run-time RMS error), however we omitted this analysis for future work.

2.2.2 Image Blending

In this section we showcase the effectiveness of our value caching approach on a slightly different scenario. The process under optimization in this case is to blend or mix two separate images (often called layers) in a single output image. While many algorithms and methodologies have been proposed to blend images, for the purposes of this work, we rely again on the basic blending mode provided by the OpenVG standard [10]. More specifically, the OpenVG standard defines the basic blending transformation as follows (according to OpenVG, there are many blend mode transformations; however in this work and without loss of generality, we rely on the most basic one):

$R_{dst} = R_{src1} * (A_{src2}/255) + R_{src2} * ((255- A_{src2})/255)$
$G_{dst} = G_{src1} * (A_{src2}/255) + G_{src2} * ((255- A_{src2})/255)$
$B_{dst} = B_{src1} * (A_{src2}/255) + B_{src2} * ((255- A_{src2})/255)$
$A_{dst} = B_{src1} * (A_{src2}/255) + A_{src2} * ((255- A_{src2})/255)$

where (R_{src1}, G_{src1}, B_{src1}, A_{src1}) and (R_{src2}, G_{src2}, B_{src2}, A_{src2}) correspond to the RGBA values of the two input images respectively, whereas the (R_{dst}, G_{dst}, B_{dst}, A_{dst}) refer to the new (generated) RGBA values of the output image.
So it is clear, that for each color of each pixel of the transformed image two floating point multiplications, two floating point divisions, one floating point addition and one floating point subtraction are required.
The following figure shows an example of blending two images according to the above formula:

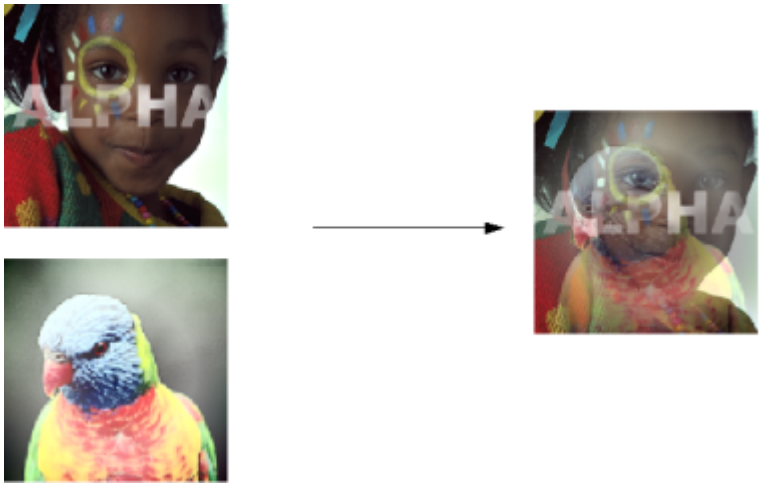

FIGURE 15: Image blending example.

2.2.2.1 Benchmarks

To test our approach in this context (image blending), we randomly created pairs of input images. All the source images are described in TABLE 1, while TABLE 2 shows the generated image mixes. Each image mix consists of two input images of equal size (although it is possible to blend images of different sizes).

Id	Image pairs	Size (in Pixels)
00	*girl_alp.png - stencil1.png*	256 x 256
01	*lena.png - lorikeet.png*	512 x 512
02	*002hn4.png - Crysis_Aliens.png*	800 x 600
03	*Angry_birds2.png - crysis.png*	800 x 600
04	*Crysis_Aliens.png - NFSU11.png*	800 x 600
05	*crysis.png - Skyrim.png*	800 x 600
06	*Skyrim.png - Android1.png*	800 x 600
07	*Uschtenheim.png - android2215.png*	800 x 600
08	*Android1.png - bbc-news_800.png*	800 x 600
09	*android2215.png - OOo.png*	800 x 600
10	*bbc-news_800.png - 002hn4.png*	800 x 600
11	*OOo.png - Angry_birds2.png*	800 x 600
12	*002hn4.png - NFSU11.png*	800 x 600
13	*Angry_birds2.png - Skyrim.png*	800 x 600
14	*Crysis_Aliens.png - Uschtenheim.png*	800 x 600
15	*crysis.png - Android1.png*	800 x 600
16	*kodim04.png - kodim10.png*	512 x 768
17	*kodim09.png - kodim17.png*	512 x 768
18	*kodim10.png - kodim18.png*	512 x 768
19	*kodim17.png - kodim19.png*	512 x 768
20	*kodim18.png - kodim04.png*	512 x 768
21	*kodim19.png - kodim09.png*	512 x 768
22	*kodim04.png - kodim17.png*	512 x 768
23	*kodim09.pn g- kodim19.png*	512 x 768
24	*kodim01.png - kodim03.png*	768 x 512
25	*kodim02.png - kodim05.png*	768 x 512
26	*kodim03.png - kodim06.png*	768 x 512
27	*kodim05.png - kodim07.png*	768 x 512
28	*kodim06.png - kodim08.png*	768 x 512
29	*kodim07.png - kodim11.png*	768 x 512
30	*kodim08.png - kodim12.png*	768 x 512
31	*kodim11.png - kodim13.png*	768 x 512
32	*kodim12.png - kodim14.png*	768 x 512
33	*kodim13.png - kodim15.png*	768 x 512
34	*kodim14.png - kodim16.png*	768 x 512
35	*kodim15.png - kodim20.png*	768 x 512
36	*kodim16.png - kodim21.png*	768 x 512
37	*kodim20.png - kodim22.png*	768 x 512
38	*kodim21.png - kodim23.png*	768 x 512
39	*kodim22.png - kodim24.png*	768 x 512
40	*kodim23.png - kodim01.png*	768 x 512
41	*kodim24.png - kodim02.png*	768 x 512

TABLE 3: Image pairs.

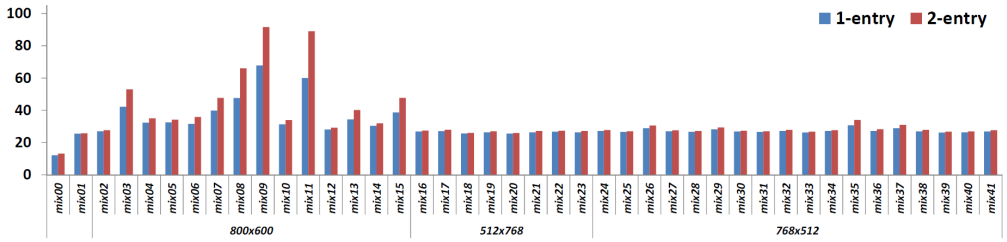

FIGURE 16: Coverage of redundant operations using 1 and 2-entry VC for all the images (only redundancy is exploited).

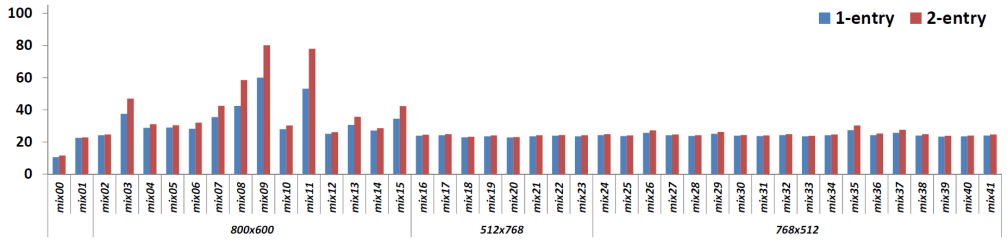

FIGURE 17: Power savings using 1 and 2-entry VC for all the images (only redundancy is exploited).

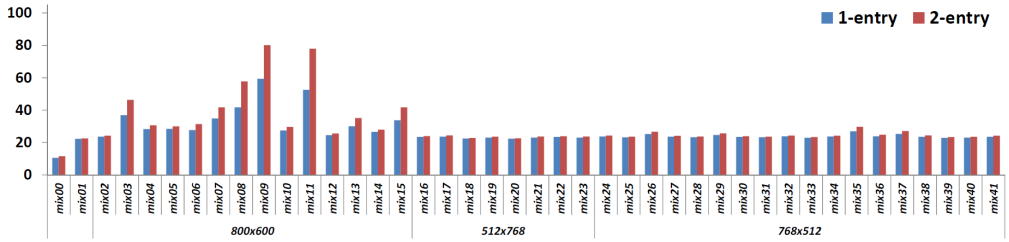

FIGURE 18: Execution time reduction using 1 and 2-entry VC for all the images (only redundancy is exploited).

2.2.2.2 Result

As in the previous section (color transformation), this section illustrates our evaluation results for the image blending transformation. Compared to the color transformation section, the main difference in this case is that two pixel colors are used as inputs in the Value Cache (one from each input image). All the other system parameters are similar to the previous section. FIGURE 16 depicts the coverage numbers reported in this case. Since the benchmark mixes used in this section have been created using images from different categories, we selected to categorize the mixes according to their size.

As FIGURE 16 indicates, the coverage in this case is smaller than the color transformation case. This is the result of doubling (two input colors) the number of input bits (a larger input must be compared in this case). However even in this case, the average coverage (over all mixes) is 29.47% for the 1-entry VC and 31.76% for the 2-entry. One more difference between FIGURE 10 and FIGURE 16 is that the 2-entry configuration does not seem to provide a significant benefit over the one-entry counterpart (only in 7 mixes the difference between the two configurations is more than 5%). The same conclusions can be drawn from the results presented in FIGURE 17 and FIGURE 18 (power savings and execution time reduction). As mentioned the power and timing benefits offered by our technique strictly follow the resulting coverage of predicting the redundant operations (FIGURE 16).
Moving one step forward, FIGURE 19 shows the resulting coverage when the concept of accuracy is also employed in our predictions for the 1-entry (top graph) and the 2-entry (bottom) VC. Similar to FIGURE 13, the bottom part of each bar (tagged as "0-bit") in FIGURE 19 shows the coverage of the predictions when full matches are taken place (i.e., only the effect of redundancy is exploited), while every additional part on top shows the increase in coverage when 1, 2, 3, or 4 bits are excluded from the matching process (exploiting accuracy). Note that in our setup we assume that the same number of bits is ignored in both input colors (although this is not necessary). For example, the "4-bits" bars show the coverage when 4 (out of 8) most significant bits are utilized in each of the two inputs colors. As we can see, even in this case exploiting accuracy manages to further increase the strength of the proposed approach. For example, by ignoring two low order bits of the input, up to 25.77% (7.13% on average) increase in coverage is achieved for the 1-entry VC (31.16% and

18.4% for the 2-entry respectively) compared to the "0-bits" VC configuration.
Finally, FIGURE 20 shows our overall range of results, in terms of power savings (blue bars) and RMS error (red line), as averaged values for each category. As we can see, the "3-bit" configuration offers an RMS error well below 5% for both the 1-entry and 2-entry VC. Moreover, the "2-bit" configuration exhibits an average error 0.37%, while the maximum error observed over all the images is 1.94%. As a final remark, we can say that the best VC configuration in this case is the 2-entry/3-bits configuration which reports an average RMS error of 2.79% offering 51.24% reduction in power.

FIGURE 19: Coverage of redundant operations using 1 (top graph) and 2-entry (bottom) VC and partial matching (exploiting accuracy) for all the images.

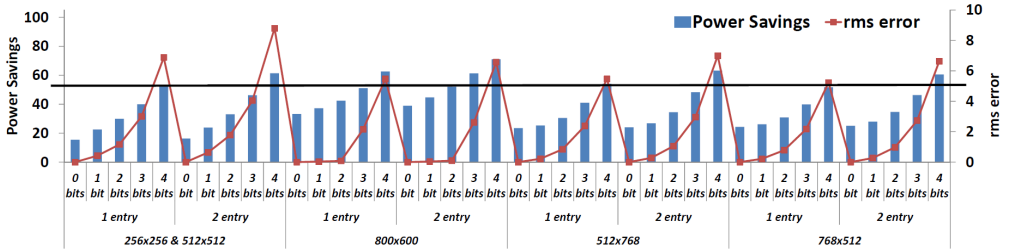

FIGURE 20: Power savings and RMS error for various configurations as averaged values for the three image categories.

2.2.3 Discussion

In this section we presented the work conducted in Task T5.2 (Redundancy) and Task T5.4 (Accuracy) during the first year of the project as defined in the LPGPU work plan. The target of this section was to override complex arithmetic calculations via simple value memorization techniques. A simple hardware design was presented (Value Cache), while an alternative software implementation was also illustrated. In addition, the same hardware design (Value Cache) was used for exploiting the concept of redundancy and accuracy at the same time. The benefits of the proposed techniques are twofold: benefits in terms of power and benefits in terms of execution time.
Our preliminary results are promising revealing and validating the strength of the proposed ideas. In our future work we will proceed with a more detailed elaboration of the proposed ideas, while a more concrete decision if the proposed idea should be implemented in the hardware or in the software (application) level will be taken (a hybrid implementation is also possible). In addition, we will investigate if those ideas can be applied in other (software or hardware) components of a typical graphics processing system.

2.3 Redundancy/Accuracy in off-chip Memory Accesses

This section presents our preliminary results in exploiting redundancy and accuracy under a different scenario: to minimize the off-chip (redundant) memory accesses. As it is explained later in this section, off-chip memory

accesses are responsible for a large portion of the total power budget. According to [2], transferring data between the processing chip and the main memory approximately consumes a factor of 40x more energy than accesses to the last level system cache, which is 2 to 3 orders of magnitude more than the energy dissipated by floating point operations and 16-bit MAC operations.
In this work, we choose to apply the concept of accuracy and the concept of redundancy in order to minimize a really resource greedy (in terms of power, time and I/O bandwidth) memory operation which is the writing/storing of a rendered image to the so-called framebuffer (without loss of generality we assume that the framebuffer is located in the system main memory or in a separate off-chip memory connected to the system bus).

2.3.1 General Description of the Target System

FIGURE 21 depicts schematically a high-level diagram of a typical graphics processing system. As we can see, the graphics generation logic generates the output frame that is to be displayed on a display device (LCD or a TFT LCD screen). Upon a portion or a whole image is produced by the graphics hardware, it would then normally be written to a framebuffer in the off-chip memory through an interconnection network (write path in FIGURE 21). In this example and without loss of generality, we assume that the framebuffer is hosted in the system main memory, however different arrangements may be assumed, i.e., the framebuffer can be a separate off-chip memory or can be a separate on-chip memory residing in the display controller.

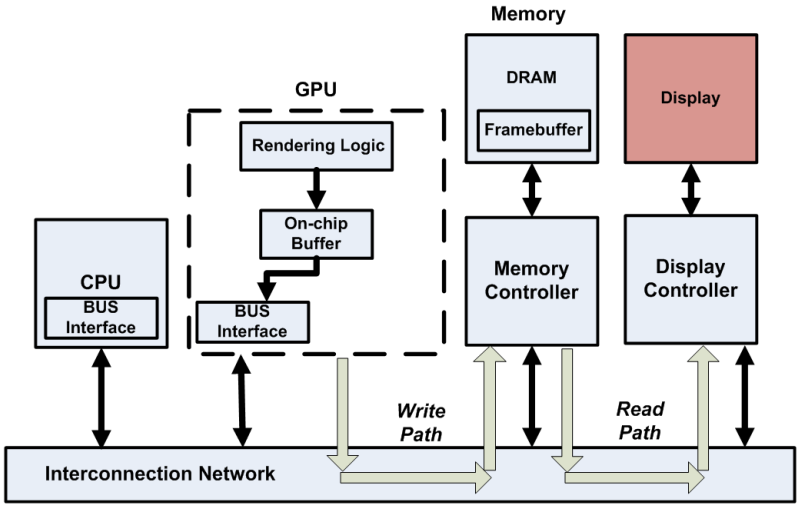

FIGURE 21. A typical graphics processing system.

Sometime later, the framebuffer will be read by the display controller in order to output the generated frame (read path in FIGURE 21). For the sake of completeness, FIGURE 21 contains also a host CPU.

2.3.2 Potential Benefits

Based on the power and area estimates provided by Bill Dally from NVIDIA [2] and considering first order effect of the framebuffer and ignoring, for the purposes of this example, the display control power, the power consumed by the LCD panel, the video output power consumption, etc., a 32-bit mobile DDR-SDRAM transfer consumes about 1nJ per 32-bit transfer. Thus assuming a graphics processor frame output rate of 60 Hz and considering first order effects only, graphics processor framebuffer writing traffic consume about (1920x1080)x(1nJ)x60=125 mW (and 4748MB/s) for high-defintion (HD) graphics at 60 fps and (1024x768x4)x(1nJ)x60=47 mW and 188 MB/s for 1024x768 graphics at 60 fps. If one is able to eliminate 20% of the framebuffer writing traffic (by exploring the redundancy between the frames as we propose in this work) for HD graphics, that would save about 25mW (and 95MB/s). For HD composition framebuffer, removing 90% of the framebuffer traffic would save 113mW (and 427 MB/s). Therefore, the power savings, by exploiting the redundancy between subsequent frames, can be significantly high which actually proves that there is ample room for optimization.

2.3.3 Description of our Approach

As already mentioned, the framebuffer is accessed by the graphics processing hardware via write operations when a new generated portion (called block) of the output screen (stored in the on-chip buffer) is ready. FIGURE 22 shows a high level view of the proposed approach. As FIGURE 22 indicates, the abovementioned writing process is modified by the use of a specialized hardware unit called redundancy prediction unit or RPU. The role of the RPU is to predict if the new generated block is identical or almost identical with the block already stored in the framebuffer.

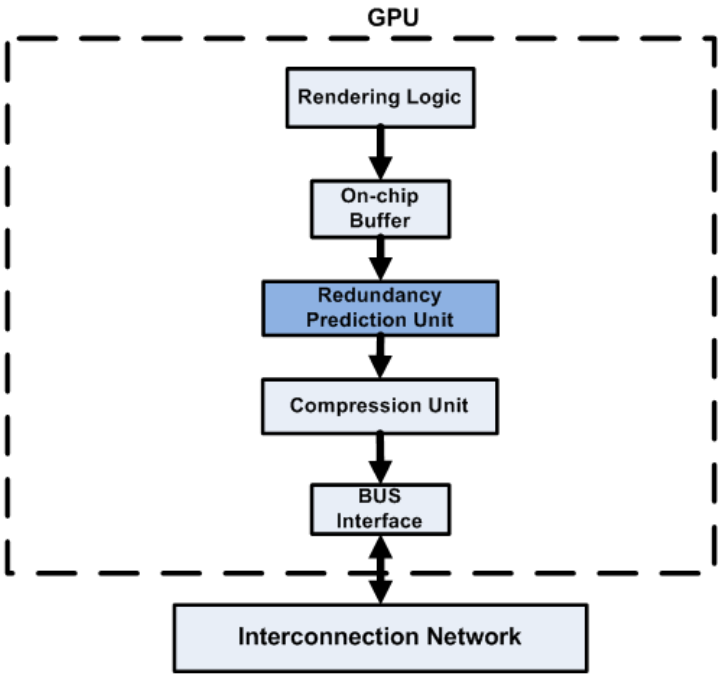

FIGURE 22: Proposed scheme.

Before describing the operation of the RPU, we first have to define the notion of the block. In this work we assume that the two-dimensional frame or screen is divided or partitioned into a number of smaller two-dimensional regions (we call them blocks). Each block is rendered separately (typically one after another or even in parallel to the other blocks in the case of a multiprocessor or a multithreaded system). The rendered blocks are then compiled to provide the complete output frame or screen. The blocks generated by the rendering logic are saved in the framebuffer in a block-by-block basis. The blocks can be of any desired and suitable size or shape. However in this work, we assume that the rendered blocks are all of the same size and shape and more specifically that the shape of each block is a square (although this is not necessary). Finally, we have to mention that in the context of this work, we have not analyzed how this block-based approach will complicate the operation of the graphics processing logic (typically a graphics rendering system generates the pixels in the form of scanlines) and most importantly how the proposed approach should be adjusted in the case of a tile-based rendering system. All those issues are left for future work.
So given the above assumptions, FIGURE 24 details the functionality of the proposed RPU, while in FIGURE 23 we can see how the framebuffer should be organized to support the proposed block-based rendering technique. The target of RPU is to indentify at run-time the blocks that are not changed between subsequent frames in graphics and video operations. By indentifying those blocks and accordingly eliminate the redundant framebuffer accesses, significant power and memory savings can be achieved. Consider for example, a graphics application in which for most of time only the mouse-pointer moves, the rate of unchanged blocks can be very high in this case.

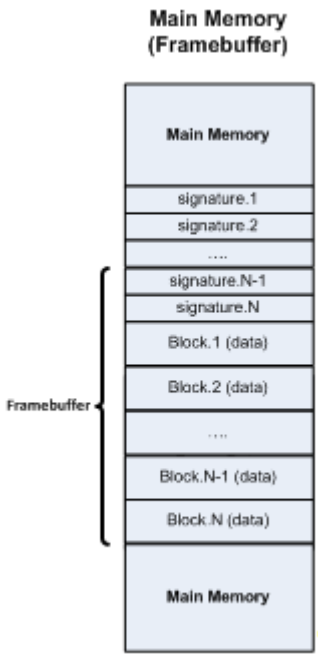

FIGURE 23: Framebuffer organization.

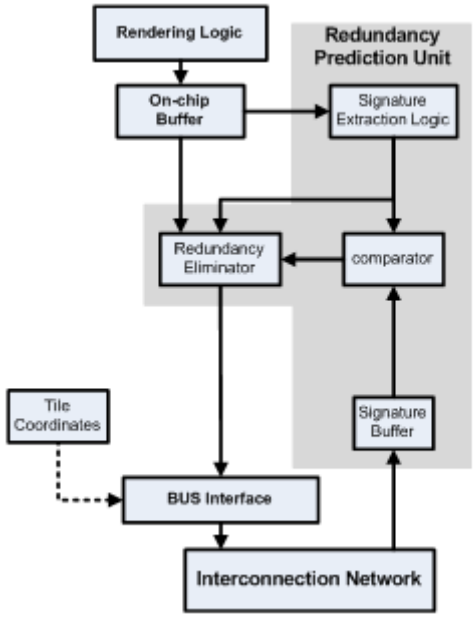

FIGURE 24: Redundancy prediction unit.

The RPU consists of the following hardware units (marked by the grey background color in FIGURE 24): the signature extraction logic, the redundancy eliminator component, a signature buffer and a comparator. When a new block is generated by the graphic processing system, the signature extraction logic operates to generate for the new block a hash value (a.k.a. signature) representative of the contents of the block. The next subsection describes the exact functionality of the signature extraction logic.
The next step is to fetch from the framebuffer the signature of the block with the same coordinates that is generated and stored in the framebuffer during a previous generation of the block. Upon the signature stored in the framebuffer arrives, it is inserted in the signature buffer (shown in FIGURE 24) and it is compared with the new generated signature (via the comparator shown in FIGURE 24). If a match occurs then the redundancy eliminator hardware decides not to store the new generated block in the framebuffer (the contents of the on-chip buffer are immediately thrown away). In case of a mismatch, the new block is transferred to the framebuffer, as it would normally be happened in a typical graphic rendering system (e.g., the one shown in FIGURE 21). If we assume that the size of a block is equal to 16x16 pixels (16 x 16 x 4 = 1024 bytes) and the size of the signature is 32-bits long, then a signature match will save 1020 bytes from storing them to framebuffer, while a mismatch will increase the traffic in the framebuffer by only eight bytes (four to read the old signature and another four to store the new signature) which is insignificant.
However, there is one more important aspect and this is where the concept of accuracy comes into the table. Two blocks with different contents may have the same signature, thus if the comparator indicates that two blocks have the same signature, then the blocks may be similar, we call them true similar blocks, or totally different (still with the same signature), we call them false similar blocks. Of course, the larger the size of the signature, the smaller the possibility to end up with false similar blocks. In any case, the size of the block and the size of the signature formulate a tradeoff between accuracy and bandwidth/power savings which is elaborated in the Result section.

2.3.3.1 Signature extraction

The extraction of the signatures as well as the size of the signatures is a critical design parameter of the proposed approach for two main reasons. First, it defines the quality of the predictions. Second, the calculation of the new signatures (for the new generated blocks) resides in the critical path of the execution. This is because in order to proceed with the rendering of the new block, the signature of the previously generated block must be calculated (the time/power overhead introduced by the comparison logic is negligible). Of course, the latter problem can be overridden if we use a double buffering approach, but in this case we double the size of the

on-chip buffer.
Another important parameter is that the power consumed (which is proportional to the number of required calculation) to extract the signatures should be minimal otherwise, the premise for a truly power efficient solution will be invalidated. For the extraction of the signatures, many hashing algorithms exist in the literature starting from simple ones like the checksum and CRC family of algorithms all the way up to more complex ones such as MD5, SHA-1, etc. In the context of this work we rely on four hashing algorithms offered by the standard CRC family, namely CRC16, CRC32, CRC64, and CRC128 because they combine simplicity and efficiency at the same time. Investigating more sophisticated algorithms is left for future work.

2.3.4 Benchmarks

To test the effectiveness of the proposed techniques, we selected five scenes from the Qt framework [6] which are typical of modern graphical user interface such as those on embedded devices or tablets. In our future work, we plan to extract and use more input scenes characterized by a larger number of input frames of various sizes. The following table depicts the details of the input scenes used in our current setup.

Id	Description	Number of snapshots	Size in pixels
Scene 1	This is a vector based animation which is typical of cartoon-like flash based applications.	101	640x480
Scene 2	This scene has moving buttons that trigger actions. This is typical of an animated menu screen where user is expected to press a button to start an activity.	98	640x480
Scene 3	It is a solid color animation like those in simple games.	98	640x480
Scene 4	This scene is a modern 3D projected button menu application that is common in user interfaces with semi transparent windows and extensive use of blending.	98	640x480
Scene 5	This scene is a scene typical of a presentation/diagram editing applications.	98	640x480

TABLE 4: Benchmarks.

2.3.5 Results

This section presents our evaluation results in utilizing the concept of redundancy and the concept of accuracy to reduce the writing activity to the framebuffer. The section contains two groups of graphs. Each group shows the gathered statistics for each of the five scenes that we consider in this work. The first group of graphs (FIGURE 25) quantifies the quality of the prediction performed by the RPU hardware for various block sizes and CRC configurations. The second group of graphs (FIGURE 26) illustrates the resulting benefits (in terms of bandwidth savings) and the overall prediction error for the same configurations.

There are five graphs in FIGURE 25; one for each studied scene. Each bar in the graphs of FIGURE 25

correspond to a specific configuration (block size and signature size) shown in the bottom of the graph. Each bar is divided into three parts. The blue part (tagged as true_similar_blocks) corresponds to the cases in which the comparisons of the respective signatures (the signature of the newly generated block and the signature produced during a previous generation of the block) indicates a match and the respective blocks (new and old one) are indeed similar (*true positive answer*). In this case, the generated block is not written to the framebuffer. The red part (tagged as false_similar_blocks) corresponds to the cases in which the comparison indicates a match and the respective blocks are not similar (*false positive answer*). Again, the generated block is not written to the framebuffer, although in this case this decision is not correct (the two blocks differ). Finally, the green part (tagged as true_different_blocks) corresponds to the cases in which a mismatch is reported (*true negative answer*). In this case, the generated block is written to the framebuffer. In other words, the blue and the green parts of the graphs correspond to the correct decisions indicated by the comparison process (the blocks are indeed similar or not similar), whereas the red part shows the incorrect or false decisions (the blocks are not similar, while the signatures are similar).

As we can see from FIGURE 25, the percentage of the false answers is at a first glance application dependant. In scene 2 and scene 4, the false answers are well below 1%, while the percentage of the true positive answers is above 58% (up to 76.88%) in all cases. This means that for those two scenes, a significant reduction in bandwidth/execution time/power can be achieved without hurting the image quality (less than 1% reduction in all cases, 0.35% on average). An important observation for those two scenes is that the accuracy of the predictions does not depend (in a significant way) from the block/signature configuration, which is not the case for the other three scenes (scene 1, scene 3, scene 5).

In those three scenes, the accuracy of the predictions follows very different trends with respect to the size of the blocks and the size of the signatures. For example, in scene 1 the accuracy of the predictions is improved (less false positive decisions) when the size of the blocks and the size of the signatures are increasing. Exactly the opposite happens in scene 3, while in scene 5 (if we ignore the CRC16 case), the accuracy of the predictions remains almost stable independently of the underlying block/signature configuration. Overall (FIGURE 25, bottom-right graph), the average ratio (across all scenes) of the false positive answers is 3.28% for the CRC16 case, 1.91% for the CRC32, 1.72% for the CRC64, and 1.41% for the CRC128. However, as it is explained in the following paragraph, those ratios are actually an overestimation of the actual error rate since we do not take into account the human perception.

FIGURE 25: Accuracy of the predictions.

Finally, FIGURE 26 shows the resulting bandwidth savings achieved by each configuration for the five scenes that we examine in this work. In the sake of competences we also show in the same graph the percentages of the false positive answers reported by each configuration. It is important to mention that we consider a false positive answer even in the case that the new generated block and the corresponding block stored in the framebuffer differ even in a single bit. Of course, since minor errors in a moving screen cannot be captured by human eyes, the error rate shown in FIGURE 26 is actually an upper bound of the error that can be interpreted by human senses. We plan to further elaborate this issue in our future work.

As it is depicted in FIGURE 26, the resulting bandwidth savings are significantly high in all cases. Consider for example, the 32x32/CRC64 configuration, the saved bandwidth in scene 1 for this configuration point is 77.04%, 70.74% in scene 2, 88.79% in scene 3, 63.71% in scene 4, and 90.57% in scene 5, while the rates of the false positive answers are 7.31%, 0.76%, 7.01%, 0.14%, 3.11% respectively.

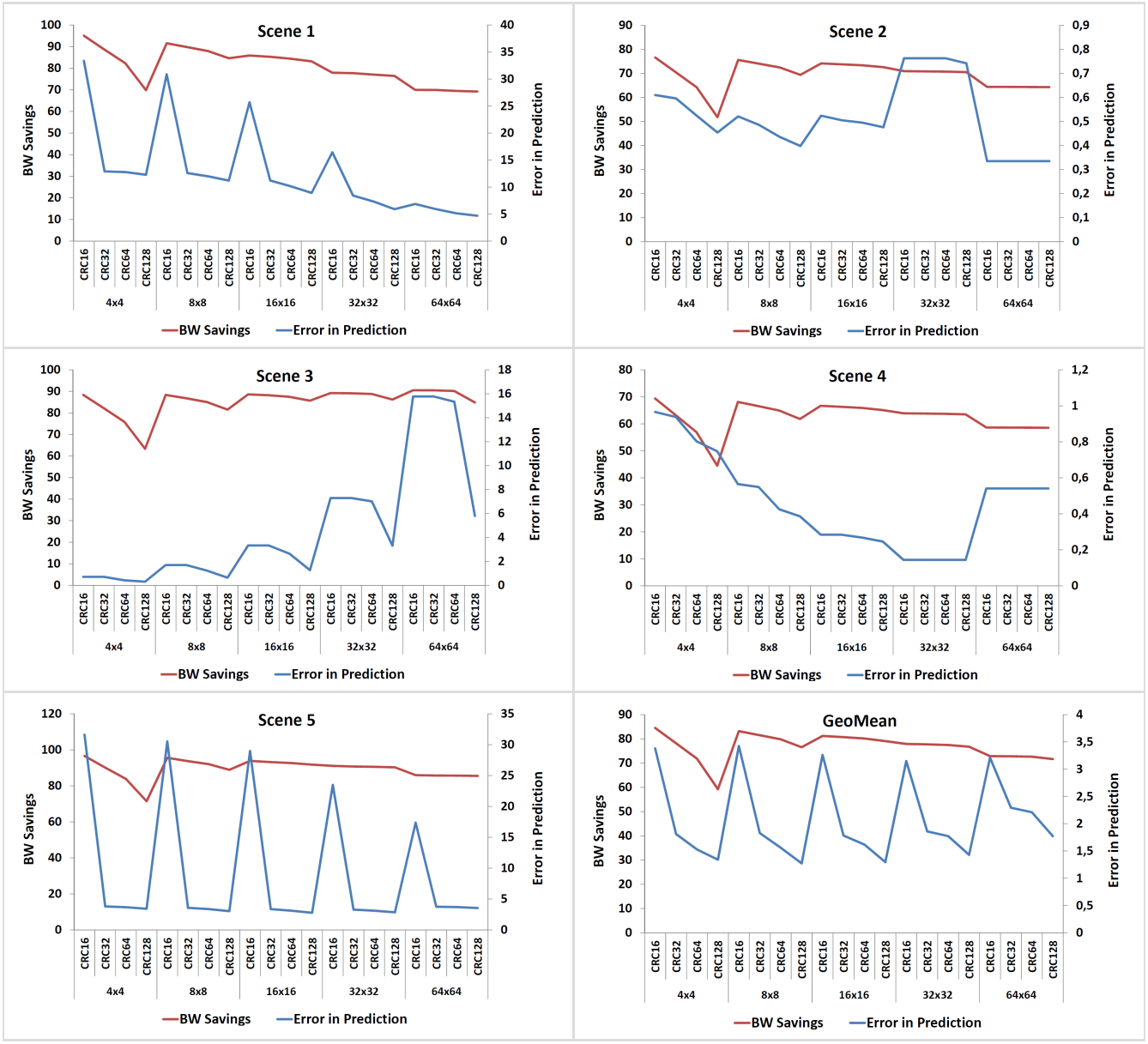

FIGURE 26: Bandwidth savings and prediction error.

2.3.6 Overheads

As we have seen, the proposed approach is able to offer a significant reduction in the framebuffer activity (smaller number of framebuffer updates), lowering the required memory bandwidth at the cost of a small reduction in the quality of the rendered images. One more overhead that we must take into account is the extra memory required to store the signatures. The size of this memory is a function of the size of the signatures and the size of the blocks. FIGURE 27 presents the results of our exploration for various block sizes and signature sizes shown in the bottom of the graph. The vertical axis shows the extra memory required to store the block signatures normalized to the size of the initial framebuffer (only the blocks are stored in the framebuffer). Note that a 16x16 block is equal to 16 x 16 x 4 = 1024 bytes (assuming 24-bits truecolor framebuffer format enhanced with 8-bits alpha channel information). In addition the size of the CRC16 is 2 bytes, 4 bytes of the CRC32, 8 bytes of the CRC64, and 16 bytes of the CRC128 code.

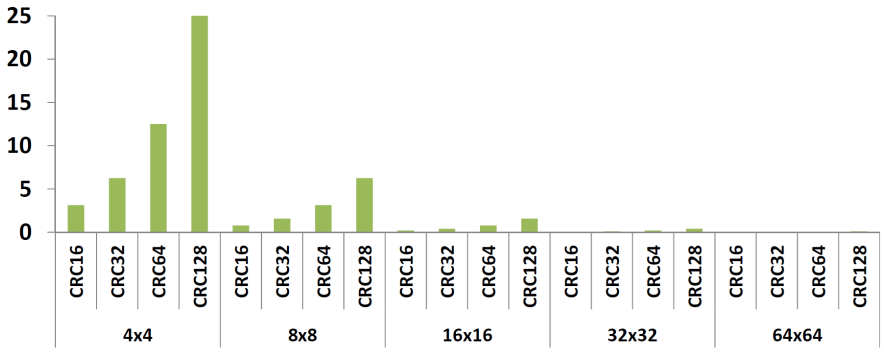

FIGURE 27: Hardware overheads.

As FIGURE 27 illustrates, expect from four cases (4x4/CRC32, 4x4/CRC64, 4x4/CRC128, and 8x8/CRC128) the memory overheads for supporting the signatures required by the proposed technique is below 5%. Furthermore, for the 16x16, 32x32, and 64x64 block configurations the memory overheads are below 1.6% in all cases which is negligible. Finally, consider for example the 32x32/CRC64 configuration and assuming a 640x480 graphics system, the total memory size required to store the signatures for all frame blocks is 9.375KB which can be also located on-chip (instead in the off-chip main memory as we assume so far). However, in order to answer this design question, a more detailed elaboration of the system is needed (taking into account all the system parameters, i.e., the graphics rendering logic, the display, the display controller, the required QoS level, etc.) which is left for future work.

2.3.7 Discussion

In this section we presented our preliminary results in exploring the concept of accuracy and the concept of redundancy to reduce the resource greedy (in terms of time, power and bandwidth) off-chip memory accesses. We selected to apply our techniques in order to reduce the frequent framebuffer updates occurring every time a new portion of the rendered image is generated by the graphics rendering logic. Our results indicated that there is a lot of potential in the area.
However, more work is needed in order to indentify the correct design parameters of the target system and many other issues have to be investigated. Our target is to end up with a highly dynamic system in which the refresh rate of the framebuffer will be based on the rate that the output frame is changing. Finally, we plan to include in the design a controlled compression technique which will be able to further reduce the amount of data that has to be transferred between the graphics rendering logic and the framebuffer.

References

1. A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In Performance Analysis of Systems and Software, 2009.
2. B. Dally, NVIDIA. Invited Talk at the International Conference on Supercomputing. 2010.
3. A. Kahng, S. Kang, R. Kumar, and J. Sartori. Designing a processor from the ground up to allow voltage/reliability tradeoffs. In International Conference on High Performance Computer Architecture, 2010.
4. L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra. ERSA: Error resilient system architecture for probabilistic applications. In Design, Automation, and Test in Europe Conference, 2010.
5. S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In International Symposium on Microarchitecture, 2009.
6. http://developer.download.nvidia.com/CUDA/training/cuda_webinars_GlobalMemory.pdf
7. http://qt.nokia.com/
8. http://r0k.us/graphics/kodak/
9. http://www.imagemagick.org
10. OpenVG Specification, Khronos Group Inc., 2008.
11. N. R. Shanbhag, A. Rami, R. Kumar, and J. Douglas. Stochastic computation. In International Design Automation Conference, 2010.
12. V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive

dvfs. In International Green Computing Conference, 2011.
13. Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing, 2004.
14. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580/architecture
15. NVIDIA, NVIDIA CUDA SDK 2.1, 2nd ed., NVIDIA Corporation, Santa Clara, California, October 2008.