technische universität
dortmund

department of
computer science

TECHNICAL REPORTS IN COMPUTER SCIENCE

Technische Universität Dortmund

Reducing the Energy Consumption of Embedded Systems by Integrating General Purpose GPUs

Constantin Timm, Andrej Gelenberg, Peter Marwedel,Frank Weichert

Computer Science 7 & 12

Number: 829
June 2010

Technische Universität Dortmund — Fakultät für Informatik
Otto-Hahn-Str. 16, 44227 Dortmund

Constantin Timm, Andrej Gelenberg, Peter Marwedel,Frank Weichert: *Reducing the Energy Consumption of Embedded Systems by Integrating General Purpose GPUs,* Technical Report, Department of Computer Science, TU Dortmund University. © June 2010

ABSTRACT

Nowadays, General Purpose Computing on GPUs (GPGPU) accelerates many industrial and scientific applications in the high performance computing (HPC) domain. Recently, GPU vendors, such as Nvidia and AMD, promoted the utilization of high end GPUs in embedded systems. The intention of the GPU vendors is the acceleration of traditional graphics computations, but in analogy to the HPC desktop domain, GPUs could also be used as GPGPU in the embedded domain. However, energy constraints are omnipresent in the embedded world and therefore, one central question for embedded system designers is: Can energy be saved by using an additional GPGPU-equipped graphics card to accelerate general purpose applications?

This paper firstly discusses the theoretical background of an energy aware embedded system design including a GPGPU-equipped graphics card. In order to support these theoretical considerations, secondly an energy and runtime evaluation of a low power GPU/CPU system is presented. We demonstrate that a profitable GPU integration, seen from an energy perspective, strongly depends on the structure and the features of an application such as a high parallelizability and the utilization level of the graphics card. The evaluation of several real world benchmarks shows that increasing the system's power consumption by integrating a GPU can lead to a reduced overall energy consumption of a system.

CONTENTS

1 Introduction 1

2 GPGPU Programming and Energy Saving on GPUs 3
2.1 GPGPU Programming 3
2.2 Energy Saving on GPUs 3

3 Related Work 5

4 Energy Reduction for GPGPU-accelerated Applications 7
4.1 Profiling GPGPU Applications 7
4.2 Profit Analysis 10
4.2.1 Evaluation Results 12
4.2.2 Decision for Integration 12

5 Conclusion 15

1 INTRODUCTION

Using a GPGPU-capable GPU for scientific and industrial general purpose applications is widely accepted for HPC (High Performance Computing) at the desktop or server level [19]. On the other hand, GPGPUs (General Purpose Computing on Graphics Processing Units) can also be used in embedded systems without a graphical interface in order to accelerate parallel general purpose applications. The trend towards GPGPU computing can be noticed through the fact that GPU vendors, such as Nvidia and ATI, provide developers with dedicated GPGPU frameworks like CUDA [17] and Stream [1] to create applications for their GPUs. Moreover, initiatives such as OpenCL [13], make programming of GPUs easier and several programming frameworks such as StreaMIT [26] and BrookGPU [4] were developed in the scientific community. GPGPU-implemented applications comprise a wide area [16] including video and image processing, simulation, cryptography or machine-learning.
The GPUCV project offers, for example, GPU-accelerated replacement routines for the image processing library OpenCV [8]. Another field of applications are solvers for parallel Sequential Minimal Optimization (SMO) within Support Vector Machine training [5]. Both of the last examples are especially important for medical and biological problems, which require rapid evaluation of complex and extensive data sets. Published GPGPU applications achieved runtime reductions of 50% and more in comparison to a CPU execution. This makes the GPGPU computing extremely interesting for embedded system design. E.g. small optical biosensors with integrated GPU-based analytical units for mass screening at airports to curtail the global spreading in terms of virus infections can be pointed out. Further, every additional computing resource in a system increases the power consumption which adversely affects the objective of saving energy in the embedded system domain, e.g., in mobile systems. A question that arises for the embedded system design is based on the high speedup potential on GPUs. Namely, whether energy can be saved by integrating an additional graphics card/processor for GPGPU computing.

The main intention of this paper is to provide an embedded system designer with the theoretical background and practical advices to reduce total system energy consumption employing an additional GPGPU-equipped accelerator. The reduction of the energy consumption can be achieved under the assumption that the applications that are running on the system are known at system design time. The paper shows that this reduction is only possible for certain types of applications, namely highly parallelizable applications [3]. An overview about parallelizable applications implemented in CUDA can be found at [17]. The evaluation of several benchmarks demonstrates that even doubling the system's power consumption by integrating a GPU can lead to an energy reduction of the overall system.

The remainder of this paper is structured as follows. Section 2 will make the

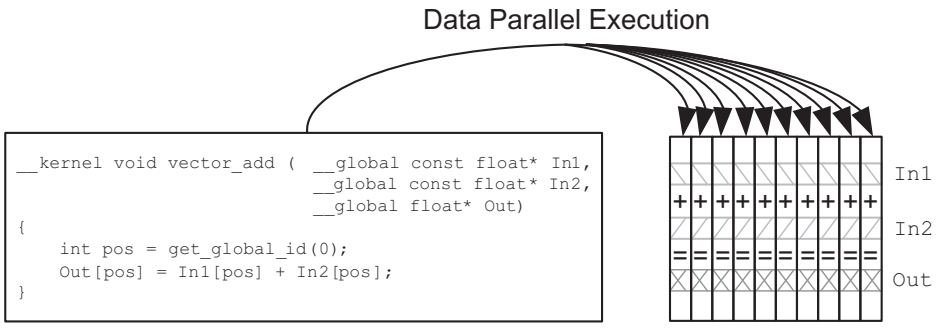


Figure 1: Schematic View of a Vector Addition with OpenCL

reader of this paper familiar with GPGPU programming and presents energy saving techniques for graphics cards. The related work is described in Section 3. Section 4 introduces the steps towards energy reduction of GPGPU-accelerated applications for energy aware embedded systems. After discussing the results of our practical evaluation, the work is concluded in the last section and future work is presented.

2 GPGPU PROGRAMMING AND ENERGY SAVING ON GPUS

This section will introduce the programming for GPGPUs and techniques which can be used to save energy on graphics cards.

2.1 GPGPU PROGRAMMING

GPUs can work in parallel. But what does this mean? It can be distinguish between at least two types of parallelism [7]. Task or function parallelism described the concept, when different parts of an application (short: tasks) can work on different data sets in parallel. The second type of parallelism is data parallelism, i.e., several tasks can work on the data items of one data set in parallel. Therefore, dedicated programming languages such as CUDA, Stream or OpenCL are tailored towards using the GPU-parallelism capabilities for general purpose applications by the possibility to express task and data parallelism. These previous mentioned languages have in common that so called "kernels" have to be specified for the code running on the GPU. The code running on the devices hosting, e.g. a GPU, can be normally written in C, C++ etc. One kernel works on a shared data set in parallel on the different stream processors of the GPU, and several kernels can run on these processors in parallel on different data sets. An exemplary kernel for vector addition with CUDA is depicted in Figure 1. It can be seen, the kernel `vector_add` works on two input arrays and one output array. The qualifier `__global` indicates that these arrays are available in the main memory of the graphics card. The function `get_global_id(0)` provides the kernel with information on which index of the arrays it has to work.

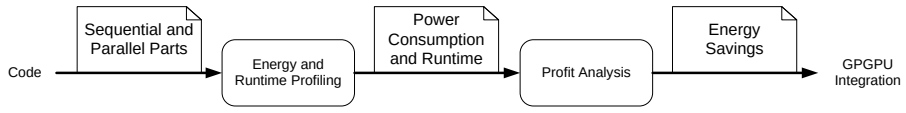


Figure 2: Workflow towards Energy Reduction for GPU-accelerated Applications

2.2 ENERGY SAVING ON GPUS

An interesting feature, provided by up-to-date graphics cards supporting energy efficient design, is, that these cards are capable of saving power at runtime when the complete computational power of the GPU is not required. Analog to the CPU domain, GPU vendors equip their GPUs with several power/frequency domains for different load settings. A low power and/or frequency domain is chosen when only two dimensional graphics computations are performed, otherwise a higher power/frequency domain is selected. In addition to that, GPU vendors allow their GPUs to power down certain units if they are not utilized. Under different loads these techniques can save energy and

make GPUs more interesting for energy aware designs. This feature can also be used for GPGPU computing in order to save energy and to support the idea of reducing the total energy consumption of systems by integrating a GPGPU-equipped graphics card. Additionally, Nvidia's HybridPower as well as ATI's PowerXpress integrated an interesting mechanism which could enable the usage of GPGPUs in embedded systems. They are equipped with two GPUs. One basic and low power GPU and one powerful GPU which could also be used for GPGPU computing. The powerful GPU is only activated on demand otherwise it remains in sleep mode.

3 RELATED WORK

General purpose application support for GPUs is widely utilized for scientific and industrial applications (see e.g. [17]). One paper which evaluates the performance of general purpose applications implemented on GPUs was written by the Trancoso and Charalambous [27]. The authors evaluated the speedup for several applications and application types on a GPU compared to a high-end CPU. Important parameters such as the memory access patterns for data, different data types and input data size were investigated. However, not energy consumption reduction but the speedup was the goal of this work.
The energy consumption of a generalized many-core system was considered in [29]. The authors of the paper describe how many-core architectures could be constructed and how this affects the speedup of parallelizable applications and the power consumption of such a system. The first many-core architecture analyzed in this paper comprises a couple of full-blown processors such as the Intel hexa-core Westmere [12]. Another architecture style that was evaluated, consists of many small energy-efficient processors in the system. The last described architecture style is to reconcile a full-blown processor with many small energy-efficient processors. This last architecture style generalizes systems such as GPU-accelerated systems or the IBM Cell Broadband Engine [11]. The energy considerations were done in this paper from the hardware and not from the software perspective. For example, only a fixed worst-case power consumption for the different processors for executing an application was assumed.
In the HPC world, energy constraints arise from the field of green computing. In this area, parallel computing GPUs are used for a wide range of applications [17]. Several papers [21, 23, 6] show that choosing the right processing unit (CPU vs. GPU) for a problem can lead to energy savings and performance increases of the considered applications. In [21], the authors take the idle power consumption of the parallel parts of an application into account and analyze whether it is beneficial to integrate a GPU into a high performance computing system. Their analysis is based on the fact that nowadays, high performance CPUs are also quite powerful for parallel computing. However, their paper is based on the HPC perspective and the authors worked with different constraints concerning GPU and CPU power consumptions and processing speeds compared to the embedded system domain.
An extension of the CUDA framework with the capability of choosing at runtime whether a kernel should be executed on a CPU or on a GPU in order to save energy was proposed in [23]. This is especially important if the developer does not know the actual platform in advance. It was shown that an efficient runtime management for choosing the execution location can reduce the energy consumption.
The authors of [6] presented a pure theoretical work on the dependencies between parallel processing and energy savings. The work also includes DVFS (dynamic voltage and frequency scheduling). In [20] the multiplication of large matrices was optimized the with regards to energy consumption. Overall, it

can be summarized that energy-aware embedded system design with the help of GPUs/graphics cards is not exploited in completeness yet.

4 ENERGY REDUCTION FOR GPGPU-ACCELERATED APPLICATIONS

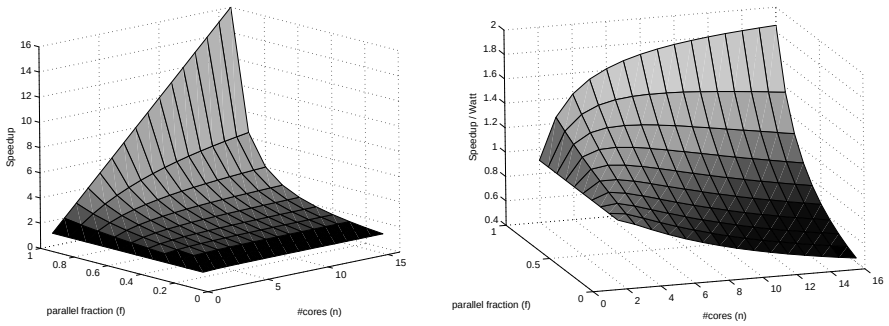
The workflow of integrating a GPGPU-equipped graphics card (depicted in Figure 2) comprises two steps and starts with an application which includes GPGPU accelerated code. First, the runtimes and the power consumption of the system with and without the GPGPU-equipped graphics card (including GPU, memory and busses) have to be measured for the application. In a profiler framework, which is described in Section 4.1, the power consumption during the runtime of an application is traced. If a real system is not available, a power model can be used [14] to estimate the power consumption based on the GPU utilization degree, in order to evaluate the power consumption for the code. The decision whether the integration of an additional graphics card into the system is beneficial is based on the considerations that are discussed in Section 4.2. In this section, the energy profit of integrating a GPU is analyzed and a formula is presented for the GPU integration decision.

4.1 PROFILING GPGPU APPLICATIONS

One of the important tasks when optimizing the energy consumption of an application is to evaluate how much energy is consumed by the system components during its execution. A very fine-grained model for energy consumption per access for memories [28] or per instruction for processors [22] can be used to be as precise as possible. Nevertheless there are some obstacles: For example, this requires detailed information on the architecture which is normally not available to application developers. Moreover, if working on such fine-grained level, the complexity of the system can be too high since one has to create power models for every component and their interactions. In order to solve these problems, we modeled and measured on a very coarse-grained level and considered the system components as black boxes. This section therefore introduces the testbed utilized to trace the power consumption of the system and the graphics card for CUDA applications.

Hardware/Software Testbed

The hardware used for this testbed is listed in Table 1. As the main processor, a quite up-to-date low power CPU, the Intel Atom 270, was selected. This processor can be found in many embedded systems such as routers, small size NAS etc. For the GPGPU computations, a low power GPU, the Nvidia 8400 GS was selected. This GPU has one stream multiprocessor with eight stream processors which can be used for GPGPU computing. The power consumption for the system (without the GPGPU-equipped graphics card) plus the power consumption of the GPGPU-equipped graphics card under high load was 25 Watt and 14 Watt under low load. For this paper, we measured the overall power consumption at two different points during the execution of the benchmarks. The



(a) (b)

Figure 3: (a) Speedup Estimation and (b) Estimation of an Energy Efficient Speedup , both with Amdahls Law (mod. after [29])

Table 1: System Configuration

Component	Configuration	Power Consumption [W] (High Load)	Power Consumption [W] (Idle)
System	Intel Atom 270 2GB-DDR2-Memory Linux OS	15	10
Graphics Card	Nvidia 8400 GS 512MB-DDR-Memory	10	4

power consumption of the GPGPU-equipped graphics card (including GPU, memory, busses) was measured by inserting two probes between the supply lines (12V and 3.3V) of the PCI Express bus. The system power consumption (without the GPGPU-equipped graphics card) was measured at the main power supply of the system. In our case there is only one 19V supply line. For the measurements a 0.1Ω resistor is embedded into the 12V and 19V supply lines and a 0.01Ω resistor into the 3.3V supply line. The voltage drop at these resistors with is proportional to current was then measured by an oscilloscope. From these current values, power values P_{sys},P_{gpu} were calculated. The energy consumption E_{sys} over the runtime of a benchmark $T_{runtime}$ was then approximated (using a Riemann sum) for the system (without the GPGPU-equipped graphics card) by

$$E_{sys} = \sum_{t \in [0, T_{runtime}]} \frac{P_{sys}(t)}{f_s} \tag{1}$$

and energy consumption $E_{sys+gpu}$ was approximated for the system plus a GPGPU-equipped graphics card by

$$E_{sys+gpu} = \sum_{t \in [0, T_{runtime}]} \frac{P_{sys}(t)}{f_s} + \sum_{t \in [0, T_{runtime}]} \frac{P_{gpu}(t)}{f_s}, \quad (2)$$

in which f_s is the sampling frequency $[s^{-1}]$ of the measurements, P is a power measured in Watt $[W]$, E is the energy in Joule $[J]$ and P_{sys} does not include the power consumption of the GPGPU-equipped graphics card.

Benchmarks

The benchmark suite that was used in this paper consists of four benchmarks that were all written with Nvidias CUDA SDK for GPGPU execution and with C/C++ for the CPU execution. The benchmarks were optimized towards their runtime environment (CPU or GPGPU) and were chosen from scientific projects, standard libraries and from the CUDA examples as follows:

- Fast Fourier Transform:
  - Description: This is the well-known runtime-efficient version of the Fourier transform which is often used to analyze signals and images.
  - GPU version: cufft [17]
  - CPU version: fftw3 [9]
- Matrix Multiplication
  - Description: Multiplication of 3 large matrices.
  - GPU version: cublas [17]
  - CPU version: goto2 [24]
- Range-Doppler Algorithm
  - Description: Creation of images from radar data with the Range-Doppler algorithm.
  - GPU version: gpu-vsipl [10]
  - CPU version: vsipl [10]
- Air Pollution Simulation
  - Description: Simulating the air pollution with the help of a stochastic Lagrangian particle model
  - GPU version: CUDA implementation [15]
  - CPU version: C++ implementation [15].

The chosen benchmarks take computationally intensive applications and data intensive applications into account. The runtime of the benchmarks was measured for a parallel GPU implementation and for a CPU implementation. The GPU measurements for a benchmark include the initialization process of the GPU, the transfer and allocation of data from/to/on the graphics card and the runtime of the kernels itself. The communication costs were often neglected in the literature but this is dangerous especially when large data sets must be transferred to the graphics card memory.

4.2 PROFIT ANALYSIS

The central point of this paper is the question whether integrating a GPU for general purpose applications is beneficial from the energy point of view. Imagine a typical application for data processing, where there are sequential parts and parts which can be parallelized. The parallelizable parts can potentially be executed on a GPU. If these parts are executed in parallel on a GPU, the overall application runtime is shortened. But is this also profitable for energy optimization? With energy constraints in mind, it is beneficial to integrate a GPGPU-equipped graphics card into a system if the acceleration is large enough to compensate the power overhead created by the graphics card.

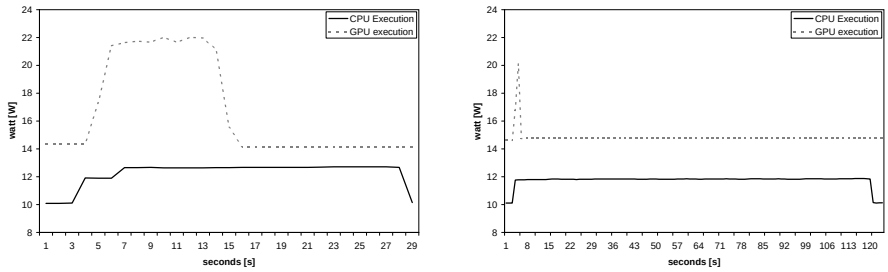
A more analytical view on the problem can start with Amdahls Law [2]:

$$Speedup = \frac{1}{(1-f) + \frac{f}{n}}. \tag{3}$$

Amdahl separates the application in parts (represented by application fraction f) which can be computed in parallel and parts that must be computed sequentially (represented by application fraction $(1-f)$) as depicted in Figure 3a. The acceleration is then expressed by $\frac{f}{n}$ if the parallel part can be independently computed on the n processors. Amdahls Law is used to estimate the performance speedup that can be achieved by parallelizing an application and running it on a parallel processor like a multi core architecture or a SIMD machine. This theoretical bound for speeding up an application is restricted in reality by many more factors such as communication costs, memory access patterns, the processing power of the multiprocessors, as evaluated later in Section 4.2.1. In Figure 3a it can be observed that the higher the fraction of parallel parts (Y-Axis) is, the better the speedup (Z-Axis) scales with the number of processors (X-Axis). Amdahls Law was created with having the performance speedup in mind but it was shown that Amdahls Law applies for energy consumption of multi-processors as well. The authors of [29] established a formula for describing the relative speedup per Watt based on Amdahls Law:

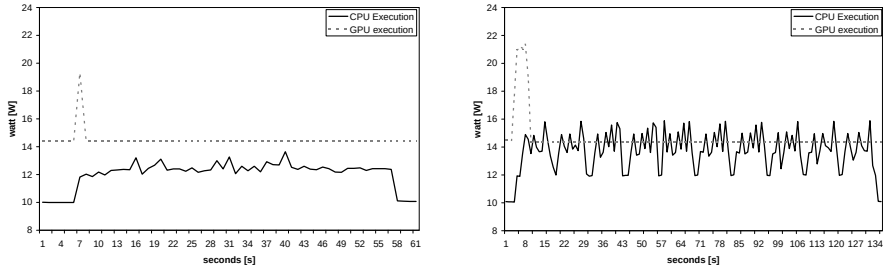
$$\frac{Speedup}{Watt} = \frac{1}{(1-f)\left(1 + (n-1)w_c k_c\right) + \frac{f}{s_c}\left(\frac{k}{n-1} + w_c\right)}. \tag{4}$$

Equation (4) describes a multi processor system comprising one powerful full-blown processor and $n-1$ small efficient cores with a realtive processing performance s_c compared to the main processor. The small efficient cores have a



(a) (b)

Figure 4: Power Consumption of (a) `Matrix Multiplication` and (b) `AirPollution` benchmark



(a) (b)

Figure 5: Power Consumption of (a) `FFT` and (b) `Range-Doppler-Algorithm` benchmark

relative power consumption of w_c for one core and one small efficient core consumes the fraction k_c over all small efficient cores. This is basically the principle of a system containing a main CPU and a GPU. From Figure 3b, one can see that in order to maximize the energy profit the application must be highly parallel (parallel fraction above 0.5).
A comparison of equation (3) to the real world measurements will be analyzed in Section 4.2.1 but obviously, only an average (or worst case) power consumption is the basis of the equation. Otherwise different power consumptions must be taken in account for different points in time.

Table 2: Speedup

Benchmark	CPU Runtime [s]	GPU Runtime [s]	Speed up
Matrix Multiplication	22.46	13.66	1.6
Air Pollution	117.32	0.89	132
FFT	50.64	0.74	68
Range-Doppler-Algorithm	129.43	4.67	28

4.2.1 *Evaluation Results*

Figures 4 and 5 depict the power consumption of the system. The solid lines show the power consumption of the system without the GPGPU-equipped graphics card whereas the dashed lines represent a system equipped with an additional graphics card. It can be seen from Figures 4 and 5 that the idle power consumption of the system is 10 Watt without the GPGPU-equipped graphics card and over 14 Watt with the graphics card included. The benchmarks can be identified in these figures by the increased power consumption for a certain amount of time. Another observation on Figures 4 and 5 is that the power consumption is nearly doubled which means that even with the shortened runtime an graphics card integration can be profitable only under certain circumstances as shown in Section 4.2.2. But due to the enormous runtime difference between the execution of the benchmarks on the GPU and on the CPU, it can be expected that GPU executions need less energy.

It can be observed from Figures 4 and 5 that the runtime is shortened for each of the benchmarks. The speedup factors are between 1.6 and 131 as one can tell from Table 2 and they are comparable to the results reported by the authors [17, 10, 15]. The speedup factors from Table 2 indicate that Amdahls Law provides no optimal prediction for GPGPU-accelerated applications. From the Nvidia Visual Profiler [17] we can derive that 3 percent of the `Range-Doppler Algorithm` benchmark can not be executed in parallel (2%: gpu idle, 0.5% transfer time from and to the graphics card memory, 0.5% application initialization). Since we have 8 stream processors in this GPU the calculated speedup factor would be:

$$\frac{1}{(1-0.97)+\frac{0.97}{8}} = 6.61. \tag{5}$$

The measurements showed that the speedup factor is much higher, because the runtime reduction is influenced by factors such as memory access patterns or the processing power of the multiprocessors compared to the CPU.

4.2.2 *Decision for Integration*

The results clearly show that additional graphics cards for GPGPU computing should be integrated into an embedded system for speeding up the execution of applications. Based on the benchmarks, equation 6 is proposed in order to support an embedded system designer to decide whether it is profitable to integrate an additional graphics card for GPGPU computing into an embedded system:

$$\sum_{t\in[0,T_{app}]} \frac{P_{cpu}(t)}{f_s} > \sum_{t\in[0,T_{cpu}]} \frac{P_{cpu}(t)}{f_s} + \sum_{t\in[0,T_{gpu}]} \frac{P_{gpu}(t)}{f_s} + \sum_{t\in[0,T_{com}]} \frac{P_{com}(t)}{f_s} + \sum_{t\in[0,T_{idle}]} \frac{P_{idle}(t)}{f_s}. \tag{6}$$

Equation 6 checks if the energy consumption of a CPU execution is higher than the energy consumption of the same application when a GPU is employed to accelerate the application. The inequation therefore comprises the following parts:

- T_{app}: Complete benchmark runtime
- T_{cpu}: Runtime on CPU
- T_{gpu}: Runtime on GPU
- T_{com}: Runtime for communication time between embedded system and graphics card
- T_{idle}: Time until the next invocation of parallel application part
- $P_{cpu}(t)$: Power consumption for running code at point of time t
- $P_{gpu}(t)$: Power consumption for running application code on GPU at point of time t
- $P_{idle}(t)$: Idle power consumption (system + graphics card) at point of time t
- $P_{com}(t)$: Power consumption for communication between system and the graphics at point of time t.
- f_s: Sampling frequency of the multimeter or oscilloscope.

In order to introduce equation 6 in more detail, we take a closer look to the `Range-Doppler Algorithm` benchmark for which the following values for inequation (6) apply:

- T_{app}: 129.43 seconds
- T_{cpu}: 4.67 seconds
- T_{gpu}: 4.4832 seconds
- T_{com}: 0.02335 seconds
- T_{idle}: 0.02335 seconds + λ seconds,

Table 3: Benchmark Energy Consumption

Benchmark	$E_{sys}[J]$	$E_{sys+gpu}[J]$
Matrix Multiplication	344	209
Air Pollution Simulation	1394	20
FFT	655	16
Range-Doppler-Algorithm	1833	108

in which λ is a variable parameter.

The power values for a particular moment, measured in Watt [W], can be derived from Figure 5b and the energy values, measured in Joule [J], for the following inequation

$$1833 > 108 + \sum_{t \in [0, T_{idle}]} \frac{P_{idle}(t)}{f_s} \tag{7}$$

from Table 3. The energy values in Table 3 are based on the runtime of a benchmark on the different platforms (CPU/GPU). The idle time have to be considered separately. Equation 7 is true as long as $\sum_{t \in [0, T_{idle}]} \frac{P_{idle}(t)}{f_s}$ is below 1724 Joule. If the GPU idle time (including λ) is too long and the idle power consumption of a system with a GPGPU-equipped graphics card is too high a reduction of the energy consumption can only be achieved if the idle power consumption is also reduced. One possibility to manage this, the techniques described in section 2.2 such as shutting down the graphics card or setting it to a lower power/frequency mode can easily be used.

5 CONCLUSION

This paper showed that integrating a GPGPU-equipped graphics card into an embedded system can be profitable in order to reduce the energy consumption of the complete system. Several real world benchmarks where evaluated in terms of energy and the results showed speedups from 1.6 to 132 and a reduction of the energy consumption of up to 83 percent.

Based on the tests conducted and on many other CUDA examples, the following theses have been proposed for reducing the energy consumption of an embedded system by integrating an additional graphics card:

1. The higher the idle time of the GPU is, the less the integration of a graphics card for GPGPU computations into the system is useful in terms of energy.
2. The communication overhead from the system memory to the graphics card memory is more important than discussed in literature.
3. Techniques for saving power on a GPU can enable the integration of this GPU for reducing the energy consumption of the total system.

The last thesis is especially important if a large amount of data such as images have to be transferred to the graphics card memory.

Overall, the integration of an additional GPGPU-capable graphics cards/processor into a system is a low-cost way of integrating high-performance computing resources and a promise way of saving energy by accelerating applications, in particular data parallel applications. Based on the tests conducted, on the computing components described above, future work will include the creation of more fine-grained resource models for GPGPU-capable embedded systems such as the Texas Instrument OMAP4 platform [25] or the Nvidia Tegra II platform [18]. These systems are small embedded systems which support OpenCL computations and they are tailored for the mobile market. From the results of this paper it can be concluded that an energy consumption reduction will also be achievable on these small embedded systems with the help of GPGPU computing. Moreover, optimizations of the energy consumption by using different power reduction techniques such as DVFS or DPM will be applied on these systems. These techniques will make the GPGPU computing more interesting in terms of energy consumption for a broader field of applications.

ACKNOWLEDGMENT

The authors would like to thank Dr. Da-Qi Ren (Department of Computer Science, University of Tokyo), Dr. Michael Engel, Matthias Meier (both: Embedded System Software Group, TU Dortmund) and Sascha Plazar (Design Automation of Embedded Systems Group, TU Dortmund) for giving advice on the power measurements at the PCI Express bus.

BIBLIOGRAPHY

[1] AMD Corporation. ATI Stream SDK, 2010. (Cited on page 1.)

[2] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the 1967 Spring Joint Computer Conference (AFIPS)*, pages 483–485, 1967. (Cited on page 10.)

[3] P. Boulet, A. Darte, G.-A. Silber, and F. Vivien. Loop parallelization algorithms: from parallelism extraction to code generation. *Parallel Computing*, 24(3-4):421–444, 1998. (Cited on page 1.)

[4] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: stream computing on graphics hardware. In *Proceedings of the 2004 SIGGRAPH*, pages 777–786, 2004. (Cited on page 1.)

[5] B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, 2008. (Cited on page 1.)

[6] S. Cho and R. Melhem. Corollaries to amdahl's law for energy. *IEEE Computer Architecture Letters*, 7(1):25–28, 2008. (Cited on page 5.)

[7] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture - A Hardware/Software Approach*. Morgan Kaufmann Publishers, San Francisco, California, 1999. (Cited on page 3.)

[8] J.-P. Farrugia, P. Horain, E. Guehenneux, and Y. Allusse. GPUCV: A framework for image processing acceleration with graphics processors. In *IEEE International Conference on Multimedia & Expo*, 2006. (Cited on page 1.)

[9] Frigo, M. and Johnson, S.G. FFTW3, 2010. (Cited on page 9.)

[10] Georgia Tech Research. Vector Signal Image Processing Library, 2010. (Cited on pages 9 and 12.)

[11] IBM Coporation. Product Description of the IBM CELL Broadband Engine, 2010. (Cited on page 5.)

[12] Intel Coporation. Product Description of the Westmere Product Line, 2010. (Cited on page 5.)

[13] Khronos Group. OpenCL Specification, 2010. (Cited on page 1.)

[14] Xiaohan Ma, Mian Dong, Lin Zhong, and Zhigang Deng. Statistical Power Consumption Analysis and Modeling for GPU-based Computing. In *Proceedings of the 2009 Workshop on Power Aware Computing and Systems (HotPower)*, 2009. (Cited on page 7.)

[15] F. Molnár Jr., T. Szakály, R. Mészáros, and I. Lagzi. Air pollution modelling using a Graphics Processing Unit with CUDA. *Computer Physics Communications*, 181(1):105 – 112, 2010. (Cited on pages 9 and 12.)

[16] Hubert Nguyen, editor. *GPU Gems 2*. Addison-Wesley, 2007. (Cited on page 1.)

[17] Nvidia Corporation. Collection of CUDA Applications and SDK. (Cited on pages 1, 5, 9, and 12.)

[18] Nvidia Corporation. Next Generation NVIDIA Tegra. (Cited on page 15.)

[19] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, 2007. (Cited on page 1.)

[20] D. Ren and R. Suda. Power Efficient Large Matrices Multiplication by Load Scheduling on Multi-core and GPU Platform with CUDA. *Proceedings of the 2009 IEEE International Conference on Computational Science and Engineering*, 1:424–429, 2009. (Cited on page 5.)

[21] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh. Energy-Aware High Performance Computing with Graphic Processing Units. In *Proceedings of the 2008 Workshop on Power Aware Computing and Systems (HotPower)*, 2008. (Cited on page 5.)

[22] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel. An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations. In *Proceedings of the 2001 International Workshop on Power And Timing Modeling, Optimization and Simulation*, 2001. (Cited on page 7.)

[23] H. Takizawa, K. Sato, and H. Kobayashi. SPRAT: Runtime processor selection for energy-aware computing. In *Proceedings of the 2008 IEEE International Conference on Cluster Computing*, pages 386–393, 2008. (Cited on page 5.)

[24] Texas Advanced Computing Center. BLAS, 2010. (Cited on page 9.)

[25] Texas Instruments. OMAP 4 Platform. (Cited on page 15.)

[26] W. Thies, M.I. Gordon, M. Karczmarek, J. Lin, D. Maze, R.M. Rabbah, and S. Amarasinghe. Language and Compiler Design for Streaming Applications. *IEEE International Parallel and Distributed Processing Symposium*, 11(2):261 – 278, 2004. (Cited on page 1.)

[27] P. Trancoso and Ma. Charalambous. Exploring Graphics Processor Performance for General Purpose Applications. In *Proceedings of the 8th Euromicro Conference on Digital System Design (DSD)*, Washington, USA, 2005. IEEE Computer Society. (Cited on page 5.)

[28] M. Verma, L. Wehmeyer, and P. Marwedel. Efficient Scratchpad Allocation Algorithms for Energy Constrained Embedded Systems. In *Proceedings of the 3th International Workshop on Power-Aware Computer Systems*, pages 41–56, San Diego, USA, 2003. (Cited on page 7.)

[29] D.H. Woo and H.-H.S. Lee. Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era. *IEEE Computer*, 41(12):24–31, 2008. (Cited on pages 5, 8, and 10.)

Forschungsberichte
der Fakultät für Informatik
der Technischen Universität Dortmund

ISSN 0933-6192

Anforderungen an:
Dekanat Informatik | TU Dortmund
D-44221 Dortmund