Multi-Objective, Energy-Aware GPGPU Design Space Exploration for Medical or Industrial Applications

Pascal Libuschewski, Peter Marwedel
Department of Computer Science XII
TU Dortmund University
Dortmund, Germany
{forename}.{surname}@tu-dortmund.de

Dominic Siedhoff, Heinrich Müller
Department of Computer Science VII
TU Dortmund University
Dortmund, Germany
{forename}.{surname}@tu-dortmund.de

***Abstract*—This work presents a multi-objective design space exploration for Graphics Processing Units (GPUs). For any given GPGPU application, a Pareto front of best suited GPUs can be calculated. The objectives can be chosen according to the demands of the system, for example energy efficiency, run time and real-time capability. The simulated GPUs can be desktop, high performance or mobile versions. Also GPUs that do not yet exist can be modeled and simulated. The main application area for the presented approach is the identification of suitable GPU hardware for given medical or industrial applications, e.g. for real-time process control or in healthcare sensor environments. As use case a real-time capable medical biosensor program for an automatic detection of pathogens and a wide variety of industrial, biological and physical applications were evaluated.**

***Keywords*-GPGPU; energy-awareness; design space exploration; medical image processing; biosensor; simulation;**

I. Introduction

In the field of medical and industrial applications the use of Graphics Processing Units (GPUs) has become more and more important in the recent years. Rofouei et al. [1] have shown that the use of GPUs can result in less energy consumption for some tasks and concluded that there is "a huge potential of research in the field of energy-aware high performance computing with GPUs".

When a system for a specific medical or industrial application is to be equipped with GPUs to increase the processing power of the system and/or to keep the energy consumption low, it is beneficial to know in advance which GPUs meet the requirements of the system. Just choosing the most powerful or energy-efficient GPUs is often not sufficient as various conflicting requirements have to be met.

This work presents a novel multi-objective design space exploration method for GPUs. For a selected set of objectives, a selected set of programs and a selected set of desired GPUs, an automatic design space exploration is conducted. GPUs or entire GPU architectures are simulated using a parametric GPU model. In conclusion a Pareto front of best suited GPUs is identified. For this set of GPUs the tradeoff between the selected objectives like energy consumption, number of core cycles, run time or cycles per Watt, is shown. It is possible to identify GPUs that fulfill the requirements of a system without the need to switch hardware and before the GPUs are bought or even exist.

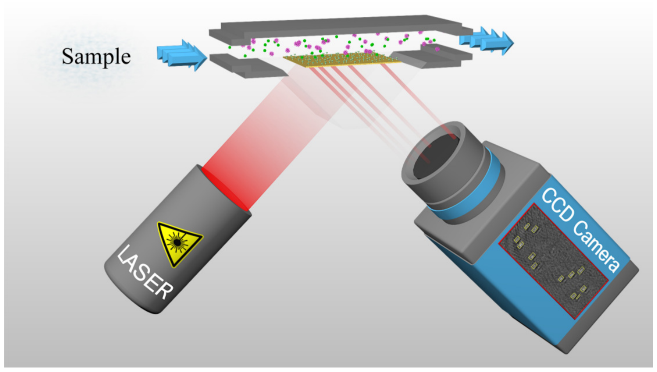


Figure 1. PAMONO biosensor. A blood or saliva sample with viruses is inserted into a flow cell. A gold layer with antibodies on top is illuminated by a laser. The reflected light is recorded by a camera. Individual viruses can be made visible indirectly as they influence the reflected light in micrometer scale, as soon as they attach to the antibodies.

The method is evaluated with a wide variety of industrial, biological and physical applications. As a real-world application the medical biosensor GPU program *virusDetectionCL* [2] was chosen. This application detects biological viruses in real-time, while they are pumped through a sensor and interact with the antibodies on the sensor surface. The PAMONO (Plasmon-Assisted Microscopy of Nano-Objects) biosensor is shown in Figure 1 and works as follows. A blood or saliva sample are pumped into a flow cell, which contains a gold layer that is coated with antibodies. As soon as the viruses in the sample interact with the antibodies the reflectivity properties of the laser light on the bottom side change and a small increase in intensity can be detected in real-time on the recorded images. Detailed information about the PAMONO sensor, which is a modified surface plasmon resonance sensor, can be found in [3].

The PAMONO sensor will be used for a rapid detection of pathogens. The scope of use will be a fast virus detector at

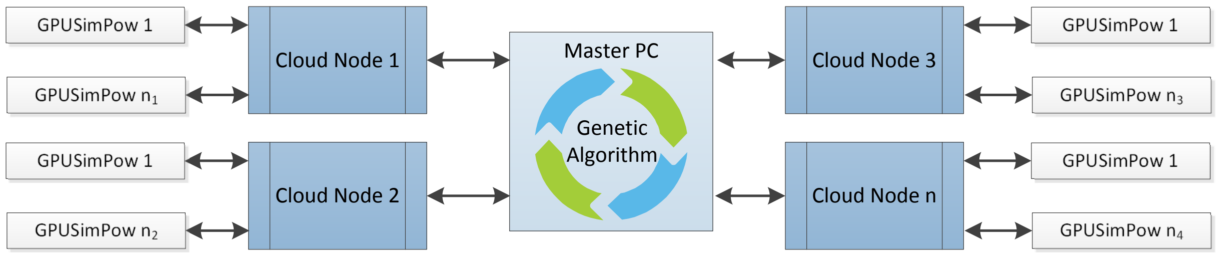


Figure 2. Distributed evaluation process.

places with a high passenger volume (e.g. airports), a system for the development of new drugs and an epidemic early warning system. The presented multi-objective design space exploration can be used to automatically identify suitable GPUs for each of the systems.

The structure of the paper is as follows: Section II gives an overview of the related work. Section III presents the methods for design space exploration of GPUs. In Section IV the experimental setup and the results are described. Finally, the Sections V and VI provide discussion and future work.

II. Related Work

In the field of design space exploration of GPUs and energy-aware computing, different approaches have been published. Basically, they divide into simulation and measurement of GPUs. A survey of different simulation-based and measurement-based approaches can be found in [5]. The presented approach is simulation-based and is hence compared to this class of related work.

A regression-based GPU design space exploration was performed in [6]. Randomly selected GPUs from the design space are sampled, and from these samples a model of the run time is constructed using stepwise regression. Mirsoleimani et al. [7] also employ a linear regression model, which was used to find the most important GPU parameters affecting the run time of different GPGPU programs, where GPGPU is the acronym for General Purpose Computation on Graphics Processing Units. Jooya et al. [8] presented an approach to find the GPU configuration which delivers the best run time under a given transistor budget. It uses the Plackett-Burman scheme to set up the experiments and solves a knapsack problem. An evaluation of energy consumption was not in the focus of these three publications.

In [9], the GPGPU-Sim simulator was used to explore different GPU design parameters but only targeting run time, not energy consumption and not in an automatic fashion. In [10] and [11], GPGPU-Sim was extended to also simulate the energy- and power-consumption of single GPUs. In [12] an evolutionary algorithm was used to find the most power-efficient GPUs for given tasks.

In contrast, the presented novel approach uses multi-objective optimization to automatically find the most efficient GPUs for given tasks. The objectives include, but are not limited to, the energy consumption, energy delay product, number of cycles, cycles per Watt and the run time of a program. In addition, the computationally expensive evaluation can be automatically distributed to a cloud of heterogeneous PCs.

III. Design Space Exploration of GPUs

In this section, a novel approach for the design space exploration of GPUs is presented. In contrast to existing approaches, multiple objectives can be optimized and the design space can easily be pruned to keep the run time of an evaluation low. Furthermore, this approach is designed for parallel simulation and optimized for a remote evaluation with an automatic deployment of all needed files.

To demonstrate the difficulties and solutions on a specific problem the *virusDetectionCL* program [2] is employed as the main use case. As described in Section I, *virusDetectionCL* is a real-time-capable streaming computer vision application, which processes sensor images. The streams of surface plasmon resonance sensor images are inspected to automatically detect nanometer-sized viruses that are indirectly made visible as bright spots that appear in the images.

For different fields of application, the GPU in the system needs to fulfill different requirements. If the virus detection should be done at places with a high passenger volume, a fast and reliable sample analysis is essential, the energy consumption is of only secondary importance. For the development of new drugs, energy and run time are both negligible as long as the analysis can be performed in a reasonable time. And if the sensor should be used as an epidemic early warning system the major objective is the energy consumption for a long battery lifetime when no electricity is available. Also the analysis time should be kept low to analyze as many samples as possible in a short period of time when necessary.

To clarify the terminology of General Purpose Computation on Graphics Processing Units (GPGPU) a short introduction should be given: If a graphics card is used to execute general purpose programs, numerous lightweight threads are deployed on the streaming processors (SPs) of a GPU. Groups of streaming processors are organized in so-called streaming multiprocessors (SMs), which also

Table I
EVALUATED GPU ARCHITECTURES. FOR EACH ARCHITECTURE THE NUMBER OF STREAMING MULTIPROCESSORS (#SMS), NUMBER OF STREAMING PROCESSORS (#SPS), CLOCK RATES, DRAM TYPE, NUMBER OF TEXTURE MAPPING UNITS (#TMUS) AND RASTER OPERATION PROCESSORS (#ROPS) ARE LISTED. THE ARCHITECTURE MARKED WITH AN ASTERISK (*) IS DEVISED AND DOES NOT YET EXIST.

Architecture	GPU	#SMs	#SPs	Core clock	Shader clock	DRAM	DRAM clock	#TMUs	#ROPs
Fermi	GF-108	1-2	48-96	700-810 MHz	1400-1620 MHz	GDDR-3	400-450 MHz	4-8	3-6
Fermi	GF-106	3-4	114-192	590-790 MHz	1180-1580 MHz	GDDR-5	450-1000 MHz	12-16	9-12
Fermi	GF-104	6-7	288-336	650-675 MHz	1300-1350 MHz	GDDR-5	850-950 MHz	24-28	18-21
Fermi	GF-100	11-15	352-480	610-780 MHz	1220-1560 MHz	GDDR-5	800-1000 MHz	44-60	33-45
Mobile Fermi	GF-108-M	1-2	48-96	600-740 MHz	1200-1480 MHz	GDDR-3	800-900 MHz	8-16	4-8
Mobile Fermi	GF-116-M	3-4	144-192	590-775 MHz	1180-1550 MHz	GDDR-5	900-1250 MHz	24-32	12-16
Mobile Fermi	GF-114-M	7-8	336-384	575-620 MHz	1150-1240 MHz	GDDR-5	1500 MHz	56-64	21-32
Pascal*	GP-107	4-5	1024-1280	1200 MHz	2400 MHz	GDDR-5	1250-1450 MHz	64-80	16-20
Pascal*	GP-104	6-8	2304-3072	820-1040 MHz	1640-2080 MHz	GDDR-6	1500-1750 MHz	192-256	48-64
Pascal*	GP-110-a	12-15	4608-5760	860-880 MHz	1720-1760 MHz	GDDR-6	1500-1750 MHz	384-480	96-120
Pascal*	GP-110-b	14-15	5376-5760	840-890 MHz	1680-1780 MHz	GDDR-6	1500-1750 MHz	448-480	112-120

contain some shared memory used by the threads. Usually, a relatively large number of streaming multiprocessors is present on a GPU, so that multiple groups of threads (warps) can be scheduled to the SMs and executed in parallel. As a result hundreds or thousands of threads can be run at the same time on a modern GPU. The flexible scheduling of threads, the complex thread cooperation and many hard- and software dependencies make it difficult to predict the execution behavior of GPUs.

One challenge for every GPU design space exploration is to evaluate the highly nonlinear influence of GPU parameters on the run time and energy consumption of programs. The presented method makes use of a genetic algorithm which generates different GPU configurations that are inspected. The individual GPU configurations are used to set up a GPU simulator, which simulates GPGPU programs in a cycle-accurate manner.

The evaluation was designed to run on a heterogeneous compute cloud. A master PC is distributing all program dependencies to the different nodes in the network, as is shown in Figure 2. It also generates new individuals in the genetic algorithm and distributes the work to the available CPU cores. Every node can evaluate numerous GPU configurations at the same time using multiple instances of the GPUSimPow [11] GPU simulator. The presented genetic algorithm uses SPEA2 [13] for the multi-objective evaluation and is configured with a vector mutation pipeline with mutation probability 0.01 and a tournament selection of size two.

Particularly (but not exclusively) in real-time applications, run time has to be considered in addition to energy consumption. If deadlines have to be met, the most energy efficient solution might not be feasible. The run time can be taken as a constraint to the optimization process or as a further objective function, naturally leading to multi-objective optimization. More precisely, the objectives energy consumption, energy delay product, number of cycles, cycles per Watt and the run time of a program can be used.

Another challenge is to keep evaluation run time low, which is closely related to keeping the design space small without sacrificing accuracy in the objective-function. If several types of GPUs or GPU architectures are to be examined, the number of parameters that need to be varied is large. More specifically in this paper the following parameters are varied: The number of streaming multiprocessors (SMs), the number of streaming processors (SPs), the core-/shader-/interconnect-/DRAM- and cache-clock-rate, the DRAM bus width, the number of raster operation processors (ROPs) and the number of texture mapping units (TMUs). Most of these parameters heavily depend on each other. For example, the number of streaming multiprocessors in a GPU affects the total number of SPs, ROPs and TMUs. Current approaches do not make use of these dependencies.

A way to model these dependencies is to map an invalid configuration to the nearest valid configuration in the parameter space, as done in [12]. The problem with this method is, that it does not keep the overall design space small. As simple genes can not model the dependencies of the values, the genes can express all valid but also some invalid configurations. This gene expression is then mapped to a valid configuration. This results in a design space which is only effectively smaller.

One solution to this problem would be to enhance the process of creating new individuals in a way that only valid values are generated in the fitness function. But this would require that the fitness function needs to be rewritten and recompiled if the parameter dependencies change. However, here, a different approach was chosen for search space pruning. The main objective was to keep the design space small. Therefore the number of genes was reduced to a number of necessary genes and the rest of the parameters was derived from these genes.

For example, if the NVIDIA architectures GF-108 to GF-100 (as listed in Table I) are to be evaluated, the architecture is uniquely identified by the number of streaming multiprocessors (SMs). All other parameters that vary with

Table II
EVALUATED PROGRAMS

Suite	Program	Language	Description
none	virusDetectionCL	OpenCL	Real-time computer vision application on biological sensor data
CUDA-SDK	blackScholes	OpenCL	Price prediction of European options
CUDA-SDK	histogram	OpenCL	Histogram calculation
CUDA-SDK	quasiRandomGen	OpenCL	Random number generator
CUDA-SDK	reduction	OpenCL	Parallel sum reduction on large arrays
CUDA-SDK	matrixMul	OpenCL	Parallel matrix multiplication
CUDA-SDK	mersenneTwister	OpenCL	Random number generator
CUDA-SDK	vectorAdd	OpenCL	Parallel vector addition
GPGPU-Sim	AES	CUDA	AES encryption algorithm
GPGPU-Sim	BFS	CUDA	Breadth first search graph traversal
GPGPU-Sim	CP	CUDA	Calculation of the coulomb potential physics simulation
GPGPU-Sim	LPS	CUDA	Laplace discretization on a 3D structured grid
GPGPU-Sim	NN	CUDA	A Neural Network on GPU
GPGPU-Sim	NQU	CUDA	N-Queens Solver
GPGPU-Sim	RAY	CUDA	Raytracing simulation for rendering of light effects
GPGPU-Sim	STO	CUDA	MD5 hash calculation
Rodinia	backProp	OpenCL	Back propagation algorithm in neural networks
Rodinia	hotSpot	OpenCL	Processor temperature physics simulation
Rodinia	gaussian	OpenCL	Gaussian elimination for solving systems of linear equations
Rodinia	nn	OpenCL	Nearest neighbor location calculation on hurricane data
Rodinia	nw	OpenCL	Needleman-Wunsch nonlinear global optimization on DNA sequences

the selected architecture can be derived from the number of SMs. As a result the design space and therefore the evaluation time can be smaller as in [12].

For the simulation of complex GPU programs another critical problem is the required time to evaluate a single run of the program, especially if hundreds of runs need to be evaluated to sample the overall design space. Simply reducing the size of the input is not permitted or has to be done very carefully because the utilization of the GPU streaming multiprocessors heavily depends on the input size. Utilization is a crucial factor in the simulation: Simulating unrealistic utilization due to too small input sizes can have unwanted side effects on the evaluation results.

For example, if a program processes a matrix of size 1024×1024 and uses local work group sizes of 16×16 threads, the input is partitioned into 4096 sub matrices each of size 16×16. Each sub matrix is then processed by 256 threads on one streaming multiprocessor on the GPU. If a GPU has e.g. 15 multiprocessors, it will need 273 steps to process 15 sub matrices (4095 in total) with full utilization of the multiprocessors and then one last step for the one missing sub matrix with about 6.7% utilization as 14 streaming multiprocessors are idle. In result the overall utilization of the GPU will be 99.7%.

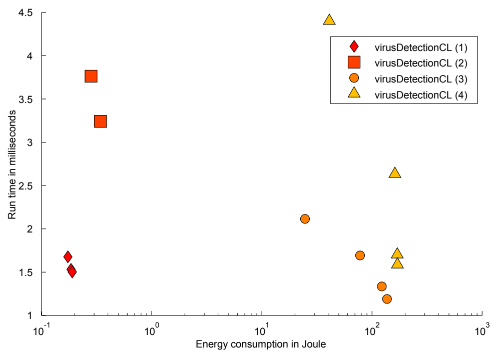
In comparison, if the input size in the example is simply decreased to e.g. 128×128 instead of 1024×1024 to speed up the evaluation, the GPU can be fully utilized for 4 steps and in the last step 11 of the 15 multiprocessors are active. The overall utilization of the GPU will drop to 85.3%. With the reduced input set it is possible that now, wrongly, a GPU with only 8 instead of 15 streaming multiprocessors is identified as the best, as these 8 multiprocessors will be fully utilized during the entire processing time.

To circumvent this problem, a *checkpointing approach* was used to reduce the evaluation time, without a reduction of the input size. In one single GPU run (either on hardware or simulated), all memory buffers are stored to the hard drive at a certain point to create a checkpoint. During the simulation this pre-calculated memory state is restored. Afterward, the remaining data can be processed. The restoring introduces an overhead to the measurement. By extracting the unwanted energy portion the evaluation time can be reduced without losing accuracy. The *checkpointing approach* is a good choice in the presence of data dependencies in the GPU code, necessitating perfectly initialized buffers.

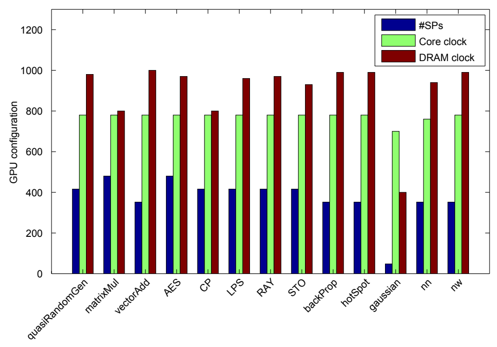
The *checkpointing approach* can be used to reduce the evaluation time for many types of programs. A good example is the *virusDetectionCL* program. This program processes hundreds or thousands of sensor images for the analysis of one probe. The checkpoint approach can be used, as the run time and energy consumption of the initialization phase is of no interest in this context. The initialization is done only once in the beginning and is insignificant for the overall energy consumption.

A second example for the *checkpointing approach* is the neural network program *NN*. If the neural network is used e.g. in an industrial process control this program will be initialized and then repeatedly be used to classify several requests. Here also the initialization phase is of no interest for the overall energy consumption.

To even further reduce the evaluation time the *checkpointing approach* was extended to a *quick start approach.* If it is known that some parts of the GPU program do not depend on the actual data in the buffers, these buffers can be



(a) Multi-objective evaluation. Detailed view of 4 Pareto fronts for the virusDetectionCL program for the he symbols indicate different target design spaces.



(b) Single-objective evaluation. The most energy efficient GPU configurations for different programs. Evaluated for the architectures GF-108, GF-106, GF-104 and GF-100.

Figure 3. Single- and multi-objective evaluation for different programs (cf. Table I and III).

just kept uninitialized. The program *virusDetectionCL* was modified with the *quick start approach* to keep some of the buffers uninitialized. For example, the program filters the input images to reduce noise. As the run time and energy consumption of this filter does not depend on the data that is processed, the calculation can be done on random values in the memory without losing accuracy. The evaluation time for the *checkpointing* step or a full initialization can be saved.

IV. Results

The proposed method has been evaluated for one real-world biosensor application and a wide field of applications from three different benchmark suites. In Section IV-A the experimental setup is described, followed by the evaluation in Section IV-B.

A. Experimental Setup

The evaluation of the GPUs was done by using a extensively modified version of ECJ (A Java-based Evolutionary Computation Research System) [14] as a framework for the evolutionary algorithm. The multi-objective evaluation was done with SPEA2 [13]. To calculate the number of cycles and the energy consumption, a modified version of GPUSimPow [11] was used, which is based on the well known cycle-accurate GPGPU simulator GPGPU-Sim [9]. Instead of using GPUSimPow it is also possible to use GPGPU-Sim in conjunction with GPUWattch [10] or virtually every other simulator that can provide the energy and cycles for a given GPGPU program.

To run the evaluation, a PC with four AMD Opteron 6272 CPUs with 16 cores (64 cores in total) and 256 GB RAM with Ubuntu 12.10 server Linux as operating system was used. Each GPGPU program was evaluated in 25 generations with a population size of 62 individuals. It has been shown that the number of total samples provides a good compromise between the evaluation time and the quality of the results. The evaluation time for a GPU program varied from 20 minutes to five days, with an average evaluation time of 19.8 hours.

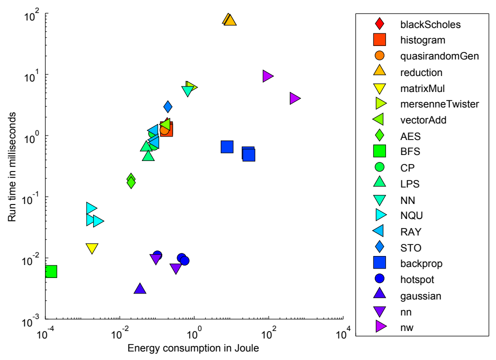
B. Evaluation

The evaluation of the presented approach has been done for a variety of GPU architectures and a variety of GPGPU programs from different benchmark suites, with the main focus on the evaluation of the *virusDetectionCL* use case.

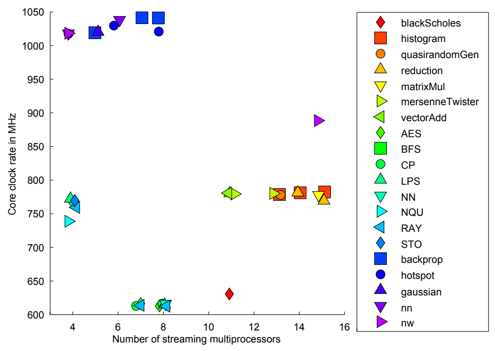
The evaluated GPU architectures are listed in Table I. Three major types of NVIDIA architectures were modeled: The Fermi architecture with the GPUs GF-108, GF-106, GF-104 and GF-100, the mobile Fermi architecture with GF-108-M, GF-116-M and GF-114-M and the high performance Pascal architecture with GP-107, GP-104 and GP-110-a/b, that currently does not exist yet. The Pascal architecture shows what a new high performance architecture might look like in the near future. It is assumed that the number of available streaming processors (SPs), raster operation processors (ROPs) and texture mapping units (TMUs) will double, compared to todays GM-107, GK-104 and GK-110 architectures. And it is also assumed that the current GDDR-5 RAM will be replaced by a GDDR-6 RAM with doubled memory bandwidth.

In Table II the evaluated programs are shown. The real-world application *virusDetectionCL* [2] and programs of the following three benchmark suites were evaluated, to show that the method generalizes: The NVIDIA CUDA SDK [15], GPGPU-Sim [9] and Rodinia [16]. The programs make use of both OpenCL [17] and NVIDIA CUDA [15] as programming languages and cover a wide variety of industrial, physical or biological applications like AES encryption, denoising of radar data, solving systems of linear equations, simulation of temperature distributions on processors or ray tracing simulations.

The results for the *virusDetectionCL* use case are shown in Figure 3a and in Table III. The *virusDetectionCL* program was evaluated with a reduced pipeline configuration against



(a) Tradeoff between energy and run time. Points from different Pareto fronts, the symbols indicate different programs.



(b) The best GPU configurations with respect to the objectives energy and run time. Values are slightly jittered to separate the points.

Figure 4. Multi-objective evaluation for different programs

all architectures (indicated by (1) to (3) in the table) and also in the most complex pipeline configuration against the high performance architecture (indicated by (4) in the table). The result shows that for each GPU architecture the most powerful GPU (GF-100, GF-114-M and GP-110) was chosen. This indicates that the program can utilize a large number of processors, even the 5376 streaming processors of the GP-110. Figure 3a shows a comparison between different GPU architectures for the *virusDetectionCL* program. The GF-1xx Fermi architecture (indicated by (1) in the figure) is the best choice with respect to energy consumption. The GP-1xx high performance architecture (indicated by (3) in the figure) is the best choice with respect to the run time, as expected. A result which may be surprising is that the mobile Fermi architecture (indicated by (2) in the figure) is dominated in both objectives by the Fermi architecture.

The GF-100 Fermi architecture could be identified as a good compromise between energy consumption and run-time. In detail the Geforce GTX 465, 470 and 480 were identified for the *virusDetectionCL* program, with the GTX 465 as the most energy efficient GPU. Only if the fastest run-time is needed a high performance architecture has to be chosen.

The results for the benchmark programs are shown as bar plot with the configurations of the single-objective evaluation in Figure 3b, as Pareto fronts for the multi-objective evaluation in Figure 4a and as configurations for the multi-objective evaluation in Figure 4b. In Table III the architectures from the Pareto fronts are listed in detail. A single-objective optimization was done for energy-consumption as objective to identify the most energy efficient Fermi architecture. The GF-100 architecture with eleven to fifteen streaming multiprocessors shows to be the best for all but one inspected program, as shown in Figure 3b. For the *gaussian* program the GF-108 with only one streaming multiprocessor and a low core clock of 700 MHz was the most energy-efficient GPU.

For the multi-objective evaluation, the programs (*blackScholes* to *vectorAdd*) were evaluated for the architectures GF-108, GF-106, GF-104 and GF-100. Although different GPUs were chosen, the GF-100 architecture is dominating all other inspected GPUs. Only the configurations of the GF-100 architecture, like the number of streaming processors, differs depending on which program was evaluated.

The next eight programs (*AES* to *STO*) in the table were evaluated for the mobile Fermi architecture. For seven out of eight the GF-114-M or GF-116-M architecture was best. For *BFS* the GF-108-M was the dominating architecture, indicating that this program does not benefit much from an increased number of streaming multiprocessors.

Finally, the programs (*backProp* to *nw*) in the table, were evaluated for the high performance architecture. An interesting result is that the GP-110 GPU with the most streaming processors was only for the *nw* program part of the Pareto front, whereas the GP-107 was part of every Pareto front. This result contrasts to the evaluation of the Fermi architecture, where the most powerful GPU was dominating all the other GPUs.

In order to investigate the validity of the *quick start approach* in estimating the run time and energy consumption of complex programs, three different OpenCL kernels from the *virusDetectionCL* program were inspected for different numbers of frames (1, 5, 10, 20 and 40 frames). On the basis of each of these runs, energy consumption was predicted for a run with 80 frames and compared to the actual simulated result. For the *Gauss filter*, the actual simulated energy consumption is 14.28 Joule for 80 frames. The average predicted energy consumption is 14.28 Joule with a relative error of 0.019%. For the *preprocessing* that converts the raw input to an internally used format, the simulated energy consumption is 0.159 Joule and the average predicted energy consumption is 0.157 a relative error of 1.37%. Even if only one instead of 80 frames is processed, the relative errors are merely 0.02% for the *Gauss filter* and 3.6% for the

Table III
RESULTS FOR THE MULTI-OBJECTIVE EVALUATION WITH THE OBJECTIVES ENERGY CONSUMPTION AND RUN TIME. COMMA SEPARATED BEST GPU CONFIGURATIONS FOR EVERY PROGRAM. SEE TABLE I AND II FOR A DESCRIPTION OF THE ARCHITECTURES AND THE PROGRAMS. ARCHITECTURES MARKED WITH AN ASTERISK (*) ARE DEVISED AND DO NOT YET EXIST.

Program	Architecture(s)	#SMs	#SPs	Core clock	DRAM
virusDetectionCL (1)	GF-110	11,14,15	352,448,480	780,780,780	GDDR-5
virusDetectionCL (2)	GF-114-M	4,7	192,336	760,590	GDDR-5
virusDetectionCL (3)	GP-107*/104*/110*	5,6,12,15	1280,2304,4608,5760	1020,1040,870,890	GDDR-5/6
virusDetectionCL (4)	GP-107*/104*/110*	4,8,13,14	1024,3072,4992,5376	1020,870,870,890	GDDR-5/6
blackScholes	GF-100	11	352	630	GDDR-5
histogram	GF-100	13,14,15	416,448,480	780,780,780	GDDR-5
quasiRandomGen	GF-100	13	416	780	GDDR-5
reduction	GF-100	11,14,15	352,448,480	780,780,770	GDDR-5
matrixMul	GF-100	15	480	780	GDDR-5
mersenneTwister	GF-100	11,13	352,416	780,780	GDDR-5
vectorAdd	GF-100	11	352	780	GDDR-5
AES	GF-114-M	7,8	336,384	615,615	GDDR-5
BFS	GF-108-M	1	48	740	GDDR-4
CP	GF-116/114-M	4,7,8	192,336,384	770,615,615	GDDR-5
LPS	GF-116/114-M	4,8	192,384	770,615	GDDR-5
NN	GF-114-M	8	384	615	GDDR-5
NQU	GF-108/116-M	1,2,4	48,96,192	740,740,740	GDDR-4/5
RAY	GF-116/114-M	4,7,8	192,336,384	760,615,615	GDDR-5
STO	GF-116-M	4	192	770	GDDR-5
backProp	GP-107*/104*	5,7,8	1280,2688,3072	1020,1040,1040	GDDR-5/6
gaussian	GP-107*	5	1280	1020	GDDR-5
hotSpot	GP-107*/104*	4,6,8	1024,2304,3072	1020,1030,1020	GDDR-5/6
nn	GP-107*/104*	4,6	1024,2304	1020,1040	GDDR-5/6
nw	GP-107*/110*	4,15	1024,5760	1020,890	GDDR-5/6

preprocessing. Errors can be reduced further by processing more frames. The third GPU program was a *sliding median* calculation with heavy data dependencies and therefore not well suited for the *quick start approach*. The simulated energy consumption was 2.37 Joule and the predicted energy 0.88 Joule with a relative error of 62.8%.

The *checkpointing approach* was validated by predicting the energy consumption for 50 frames based on simulating the processing of 1, 5, 10 and 20 processed frames. For the *Gauss filter* the the simulated energy consumption was 14.26 Joule and the prediction was 14.26 Joule with 0.0006% relative error. For the *preprocessing* 0.159 Joule was simulated and 0.162 was predicted with an relative error of 1.37%. Finally the *sliding median* took 3.188 Joule in the simulation while 3.187 Joule were predicted. In result the relative error is 0.047% instead of the 62.8% for the *quick start approach*.

V. DISCUSSION

As more and more medical or industrial applications are using GPUs for processing, it is necessary to identify real-time capable and/or energy-efficient GPUs for these systems. Several questions can arise if a new GPU system should be designed: Which is the most energy efficient GPU for a mobile device? Which is the fastest GPU for a high performance system? Is the chosen system capable of real-time processing?

The proposed framework can be used to determine efficient GPUs for various given problems and under various objectives. It has been shown that the design space can be kept small by making use of parameter dependencies, leading to a reduced evaluation time without losing accuracy in the objective functions. By reducing the input size of complex GPGPU programs, the evaluation times was reduced further.

The energy prediction results show that for programs which have no data dependencies, the *quick start approach* is a promising method to reduce the evaluation time. The energy consumption was predicted accurately for the *Gauss filter* and the *preprocessing* with good error rates of 0.019% and 1.37%. As the *sliding median* calculation exhibits heavy data dependencies, realistic input data is needed. An initialization with random data is not sufficient which is also reflected in the error rate of 62% with the *quick start approach*. For this type of programs, the *checkpointing approach* is perfectly suited. The computation can be done on initialized buffers which are loaded from the hard disk, without the initialization overhead. In result the relative error dropped from 62% to 0.047%. The evaluation time was reduced by the *checkpointing approach* from one day and seven hours to six hours with a very low error.

The multi-objective evaluation has shown that although different programs are evaluated, some architectures tend to be more energy and run time efficient than others. For the Fermi architecture surprisingly the GF-100 dominated all other GPUs. For the mobile Fermi architecture, the GF-116-M or GF-114-M is best for most of the programs. And for the Pascal architecture the GP-107 was always one solution in the Pareto fronts. The huge amount of cores of the GP-110 architecture could only be fully utilized by the programs *nw* and *virusDetectionCL*. Fully utilizing the streaming processors of upcoming GPU architectures seems to be a challenge for the developers, which needs to be solved to keep the energy consumption on a reasonable level.

In conclusion, it has been shown that the presented GPGPU design space exploration can be used to identify the best GPU architectures for a large variety of programs. Also GPUs that do not yet exist can be evaluated, which is beneficial for choosing the right hardware and also to optimize software for upcoming GPUs and identify bottlenecks early. The design space can be efficiently pruned to keep the evaluation time low, even for complex programs.

VI. Future Work

The presented approach will be extended towards a hardware/software co-design. As software parameters, e.g. the size of the work groups of an OpenCL/CUDA program, significantly influence the run time on the GPU, it is recommendable to optimize software and hardware parameters at the same time.

Moreover a quality of service prediction will be conducted for different software parameters: As some software does not need to produce 100% accurate results, a tradeoff can be calculated between the accuracy and the required energy for the calculation. This can be used to automatically adapt the software or hardware settings in order to save energy when there is no need for fully accurate results.

In addition, it will be examined how distributed embedded systems can make use of compute servers to offload parts of a computational task in order to improve objectives like energy consumption or run time. The software on the distributed embedded system will predict the saved energy based on environmental information, like the available bandwidth in the mobile network. Besides the energy and transfer costs on the embedded system, the costs to calculate the results on the server and the current server system load can be taken into account. This enables taking the decision of whether it is beneficial to offload the complete processing task, to offload only parts of it, or to process it locally.

Acknowledgment

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis”, project B2. URL: http://sfb876.tu-dortmund.de

References

[1] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, “Energy-aware high performance computing with graphic processing units,” in *Proceedings of ACM SOSP Workshop on Power Aware Computing and Systems*, 2008.

[2] P. Libuschewski, D. Siedhoff, C. Timm, A. Gelenberg, and F. Weichert, “Fuzzy-enhanced, real-time capable detection of biological viruses using a portable biosensor,” in *Proceedings of the International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSIGNALS)*, 2013.

[3] A. Zybin and et al., “Real-time detection of single immobilized nanoparticles by surface plasmon resonance imaging,” *Plasmonics*, vol. 5, pp. 31–35, 2010.

[4] P. Libuschewski, P. Marwedel, D. Siedhoff, and H. Mller, “Multi-objective energy-aware GPGPU design space exploration for medical or industrial applications,” in *Signal-Image Technology and Internet-Based Systems (SITIS), 2014 Tenth International Conference on*, Nov 2014, pp. 637–644, doi10.1109/SITIS.2014.11.

[5] I. Ahmad and S. Ranka, *Handbook of Energy-Aware and Green Computing*. Chapman & Hall/CRC, 2012.

[6] W. Jia, K. Shaw, and M. Martonosi, “Stargazer: Automated regression-based GPU design space exploration,” in *Performance Analysis of Systems and Software (ISPASS), IEEE International Symposium on*, 2012, pp. 2–13.

[7] S. Mirsoleimani, A. Karami, and F. Khunjush, “A two-tier design space exploration algorithm to construct a GPU performance predictor,” in *Architecture of Computing Systems (ARCS)*. Springer, 2014.

[8] A. Jooya, A. Baniasadi, and N. Dimopoulos, “Efficient design space exploration of GPGPU architectures,” in *Parallel Processing Workshops (Euro-Par)*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 7640, pp. 518–527.

[9] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, “Analyzing CUDA workloads using a detailed GPU simulator,” in *Performance Analysis of Systems and Software (ISPASS). IEEE Int. Symposium on*, 2009, pp. 163–174.

[10] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, “GPUWattch: Enabling energy optimizations in GPGPUs,” *International Symposium on Computer Architecture*, 2013.

[11] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, “How a single chip causes massive power bills GPUSimPow: A GPGPU power simulator,” in *Performance Analysis of Systems and Software (ISPASS), IEEE International Symposium on*. IEEE, 2013, pp. 97–106.

[12] P. Libuschewski, D. Siedhoff, and F. Weichert, “Energy-aware design space exploration for GPGPUs,” *Computer Science - Research and Development*, pp. 1–6, 2013.

[13] E. Zitzler, K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailiou, T. Fogarty, E. Z. Ler, M. Laumanns, and L. Thiele, “SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization,” in *Proceedings of the International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems (EUROGEN)*, 2001.

[14] S. Luke and L. Panait, “A survey and comparison of tree generation algorithms,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2001, pp. 81–88.

[15] NVIDIA Corporation, “CUDA Architecture,” 2014, URL: http://nvidia.com/object/cuda_home_new.html.

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in *Workload Characterization (IISWC). IEEE International Symposium on*, 2009, pp. 44–54.

[17] Khronos Group, “OpenCL Specification,” 2014, URL: http://khronos.org/registry/cl/.