Energy Minimization on Heterogeneous Systems through Approximate Computing

Michalis Spyrou, Christos Kalogirou, Christos Konstantas, Panos Koutsovasilis, Manolis Maroudas, Christos D. Antonopoulos [1] and Nikolaos Bellas
Centre for Research and Technology Hellas - CERTH
Department of Electrical and Computer Engineering, University of Thessaly
e-mail: {mispyrou, hrkalogi, hriskons, pkoutsovasilis, emmmarou, cda, nbellas}@uth.gr

Abstract. Energy efficiency is a prime concern for both HPC and conventional workloads. Heterogeneous systems typically improve energy efficiency at the expense of increased programmer effort. A novel, complementary approach is approximating selected computations in order to minimize the energy footprint of applications. Not all applications or application components are amenable to this method, as approximations may be detrimental to the quality of the end result. Therefore the programmer should be able to express algorithmic wisdom on the importance of specific computations for the quality of the end-result and thus their tolerance to approximations.

We introduce a framework comprising of a parallel meta-programming model based on OpenCL, a compiler which supports this programming model, and a runtime system which serves as the compiler backend. The proposed framework: (a) allows the programmer to express the relative importance of different computations for the quality of the output, thus facilitating the dynamic exploration of energy / quality tradeoffs in a disciplined way, and (b) simplifies the development of parallel algorithms on heterogeneous systems, relieving the programmer from tasks such as work scheduling and data manipulation across address spaces.

We evaluate our approach using a number of real-world applications, beyond kernels, with diverse characteristics. Our results indicate that significant energy savings can be achieved by combining the execution on heterogeneous systems with approximations, with graceful degradation of output quality.

Keywords. Energy Saving, Approximate Computing, Programming Model, Controlled Quality Degradation, Heterogeneous Systems

Introduction

Energy efficiency is a primary concern when designing modern computing systems. Moving to the multicore era allowed architects to exploit increasing transis-

[1]Corresponding Author: Christos D. Antonopoulos, Assistant Professor, Department of Electrical and Computer Engineering, University of Thessaly, Greece; E-mail: cda@inf.uth.gr URL: http://www.inf.uth.gr/~cda

tor counts by implementing additional cores. However, the end of Dennard scaling [3] limits expectations for energy efficiency improvements in future devices by manufacturing processors in lower geometries and lowering supply voltage. Traditional hardware / system software techniques, such as DFS and DVFS also have their limitations when it comes to CPU intensive workloads.

Heterogeneous systems appeared as a promising alternative to multicores and multiprocessors and dominate the Top500 [10] and Green500 [4] HPC lists. They offer unprecedented performance and energy efficiency for certain classes of workloads, however at significantly increased development effort: programmers have to spend significant effort reasoning on code mapping and optimization, synchronization, and data transfers among different devices and address spaces.

One contributing factor to the energy footprint of current software is that all parts of the program are considered equally important for the quality of the final result, thus all are executed at full accuracy. However, as shown by previous work on approximate computing [2,16,19], several classes of applications include blocks of computations that do not affect the output quality significantly. Non-significant computations can often tolerate approximations or even substitution by a default value.

In this paper we introduce a directive-, task-based meta-programming model on top of OpenCL [20], which allows programmers to: (a) express their insight on the importance of different parts of the computation for the quality of the end-result, (b) provide alternative, approximate versions of selected computations, (c) control the ratio of approximate / accurate computations executed, and thus the energy / quality tradeoff using a single knob, and (d) exploit heterogeneous, accelerator-based systems, without many of the development overheads typically associated with such systems. To the best of our knowledge this is the first time a single programming model offers support for approximate execution on heterogeneous, accelerator-based systems. For our experiments we use a number of real-world applications from different domains, with diverse characteristics. We discuss approximation techniques that fit each application, and we apply appropriate metrics to evaluate the output quality. We find that, by exploiting application specific high level information among proper device and ratio selection, we can approximate the output with acceptable quality, while considerably reducing the energy footprint of the application.

The rest of this paper is organized as follows: Section 1 presents the key features of our programming model and Section 2 discusses the runtime implementation, highlighting its fundamental design decisions. Section 3 presents the applications used to evaluate our programming model and the heterogeneous platform we used for the evaluation, whereas in Section 4 we present and discuss evaluation results. Related work is discussed in Section 5 and Section 6 concludes the paper.

1. Programming Model

The programming model we introduce adopts a task-based paradigm, using *#pragma* directives to annotate parallelism and approximations. Tasks are implemented as OpenCL kernels [20], facilitating execution on heterogeneous systems.

The main objectives of the programming model are to allow (a) flexible execution on heterogeneous systems, without overwhelming the programmer with low-level concerns, such as inter-task synchronization, scheduling and data manipulation, and (b) flexible exploration by the user of the quality / energy tradeoff at run-time, using a single knob and exploiting developer wisdom on the importance of different parts of the code for the quality of the end-result.

Listing 1 summarizes the *#pragma* task and taskwait directives used for task manipulation. Listing 2 outlines the implementation of Discrete Cosine Transform (DCT), which serves as a minimal example to illustrate the use of the main programming model concepts. Below we explain each directive and clause referring to this DCT example as appropriate.

```
#pragma acl task [approxfun( function )] [significant( expr )] [label("name")] \
                 [in( varlist )] [out( varlist )] [inout( varlist )] \
                 [device_in( varlist )] [device_out( varlist )] \
                 [device_inout( varlist )] [bind( device_type )] \
                 [workers( int_expr_list )] [groups( int_expr_list)] \
accurate_task_impl( ... );
#pragma acl taskwait [label("name")] [ratio( double )]
```

Listing 1 Pragma directives for task creation/completion.

```
 1 __kernel void dctAccurate(double *image,double *result,int subblock) { }
 2 __kernel void dctApprox(double *image, double *result, int subblock) { }
 3
 4 int subblocks=2*4, subblockSize=4*2, blockSize=32, imgW=1920, imgH=1080;
 5 /*DCT block to 2x4 subblocks with different significance, image dimensions*/
 6 double sgnf_lut[] = {  1,   .9,  .7,  .3,
 7                       .8,   .4,  .3,  .1};
 8 void DCT(double *image, double *result, double sgnf_ratio) {/* entry point */
 9     for (int id = 0; id < subblocks; id++) {  /*spawn dct task group*/
10         #pragma acl task in(image) out(&result[id*subblockSize]) \
11                          label("dct") \
12                          significant(sgnf_lut[id]) approxfun(dctApprox) \
13                          workers(blockSize, blockSize) groups(imgW, imgH)
14         dctAccurate(image, result, id);
15     }
16     #pragma acl taskwait ratio(sgnf_ratio) label("dct") /*execution barrier*/
17 }
```

Listing 2 Programming model use case: Discrete Cosine Transform (DCT) on blocks of an image.

The *task* directive (lines 10-13) defines a new task. It annotates the following function call (line 14) as the task body which corresponds to an OpenCL kernel (*dctAccurate()*, line 1) and specifies the accurate implementation of the task.

The *approxfun()* clause (line 12) allows the programmer to provide an alternative, approximate implementation of the task. This is generally simpler and less accurate (may even return a default value in the extreme case), however has a lower energy footprint than its accurate counterpart. For example, *dctApprox()*, also defined as OpenCL kernel at line 2, sets all coefficient values equal to zero. The actual call to the accurate version may be replaced at execution time by a call to the approximate version, if present.

The *significant()* clause quantifies the relative significance of the computation implemented by the task for the quality of the output, with a value (or expression) in the range [0.0, 1.0]. If set to 1.0 or omitted, the runtime will always execute the task accurately. If set to 0.0, the runtime will execute the task approximately, or even discard it if an approximation is not available.

The DCT example defines each task's significance at line 12, using values from a lookup table (array *sgnf_lut[]*, line 6). Notice that tasks calculating coefficients near the upper left corner of each block (low spatial frequencies) are more significant (values in the *sgnf_lut[]* array which are used to assign significance to tasks) than those calculating coefficients which correspond to higher spatial frequencies. This is due to the fact that the human eye is less sensitive to higher frequencies.

The programmer explicitly specifies the input and output arguments of each task with the *in()*, *out()* and *inout()* data clauses (line 10). The corresponding information is exploited by the runtime system for dependence analysis and scheduling, as well as for data management, as explained in Section 2. The *device_in()*, *device_out()* and *device_inout()* data clauses extend the above clauses by forcing data transfers from/to device. We also support a subarray notation to express data dependencies, in the form of *array[i:i+size]* in the spirit of OpenACC [14]. Expressing arguments in data clauses as subarrays further reduces unnecessary data transfers.

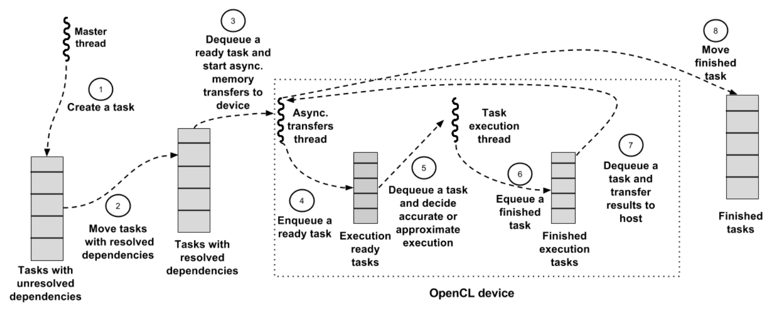
The programmer can explicitly annotate a task for execution on a specific device, using the *bind()* clause. This limits the flexibility of the programming model, however it proves useful in case an implementation is optimized for a specific device. To specify the work-items and work-groups geometry for kernel execution, the programmer uses the *workers()* and *groups()* clauses as shown at line 13, which follow the semantics of local- and global workgroup size of OpenCL, respectively. Finally, the *label()* clause associates tasks with named task groups. Each group is characterized by a unique string identifier. In our DCT example, line 11 adds the newly created task to the *"dct"* task group.

The *taskwait* directive specifies an explicit synchronization point, acting as a computation and memory barrier. By default, taskwait waits on all issued tasks so far, unless the *label()* clause is present, which limits the barrier to tasks of the specific task group.

The *ratio()* clause accepts a value (or expression) ranging in [0.0, 1.0] as an argument. It specifies the minimum percentage of tasks of the specific group that the runtime should execute accurately. *ratio* is a single knob which allows the programmer or the user to control the energy footprint / quality tradeoff. If ratio is 0.0, the runtime does not need to execute any task accurately. Similarly, if ratio is 1.0, all tasks need to be executed accurately regardless of their significance. In our DCT example, line 16 waits for all tasks inside the *"dct"* task group with a user defined ratio.

The programming model is implemented in the context of a source-to-source compiler, based on LLVM/Clang [8] and the LibTooling library. The compiler lowers the *#pragma* directives to the corresponding runtime API calls.

2. Runtime Support



Master thread
1
Create a task
Tasks with unresolved dependencies
2
Move tasks with resolved dependencies
Tasks with resolved dependencies
3
Dequeue a ready task and start async. memory transfers to device
Async. transfers thread
4
Enqueue a ready task
Execution ready tasks
5
Dequeue a task and decide accurate or approximate execution
Task execution thread
6
Equeue a finished task
Finished execution tasks
7
Dequeue a task and transfer results to host
8
Move finished task
Finished tasks
OpenCL device

Figure 1. Runtime system architecture and typical task life cycle

Fig. 1 outlines the architecture of our runtime system. It is organized as a master/slave work-sharing scheduler. For each device on the system, two threads are created: (a) a memory transfers thread, responsible for transparent data transfers between the host and the device, and (b) a task issue thread, responsible for issuing tasks (implemented as OpenCL kernels) for execution to the corresponding device. Our runtime reuses the underlying vendor OpenCL implementation for each device for data transfers, code execution, as well as to extract system configuration information. The master thread executes the main program sequentially and every task created is stored into a global pool (Fig. 1, step 1).

2.1. Data Flow Analysis - Scheduling & Memory Transfers

The runtime system can perform automatic data flow analysis at the granularity of tasks, exploiting the information provided by the programmer via the data clauses of each task. More specifically, upon task creation the runtime system tracks the memory ranges read and written by each task. The results of this analysis are exploited in two different ways: (a) to detect data dependencies among tasks and enforce execution in the correct order, and (b) to automate memory transfers among different address spaces of the heterogeneous system.

All task scheduling and data manipulation are transparent to the programmer. Overlaps between data ranges from data clauses of different tasks coexisting in the system at any time, indicate potential WaW, RaW or WaR data dependencies. The runtime identifies these dependencies and enforces execution of inter-dependent tasks in the order they were spawned. Once all its dependencies are resolved, a task is transferred to the ready queue (Fig. 1, step 2) and can be selected for execution.

Devices can execute tasks from the global pool whenever they have resources (execution units and memory) available, even if there are other tasks concurrently executing on the device (Fig. 1, step 3). The runtime respects potential limitations for task execution on specific devices specified by the programmer. If input data for the task does not already reside in the device address space, they have to be transferred before the task can be executed. The runtime system includes a simple memory manager which tracks the location of each named object used as an argument in any data clause, as well as the amount of available memory on each device. The corresponding data structures are updated by the memory transfers threads. When input data for a task are on the device memory, they can be issued for execution (Fig. 1, step 5). Similarly, *out()* and *device_out()* data are transferred to the respective device when they are required as input for another task, or to the host either at synchronization points, or whenever the memory manager needs to reuse memory on the device (Fig. 1, step 7). The runtime tries to overlap data transfers with computations when possible by prefetching data for tasks to be executed while other tasks still keep the computational units of the device busy. Given that data transfers typically incur significant overhead, the scheduler associates tasks with devices according to data locality (beyond resource availability). If a device, despite data affinity, has no free memory or computational resources, the next available device is used.

2.2. Accurate / Approximate Task Execution

A newly created task includes the binaries of both accurate and approximate versions of the OpenCL kernels. When issuing the task for execution, the runtime decides whether it will execute the approximate or the accurate version. More specifically, the runtime observes the distribution of significance values assigned to spawned tasks and heuristically and dynamically adjusts the significance threshold beyond which tasks are executed accurately, with the target of achieving the user-specified ratio of accurate/approximate tasks. Whenever a task is issued for execution the runtime compares its significance to the current threshold and executes the appropriate implementation (accurate/approximate) accordingly.

2.3. Performance and Energy Monitoring

Information about energy, power and performance is collected during execution by periodically polling the interfaces (hardware counters and libraries) offered by each device. For example, for Intel CPUs, power is calculated by sampling energy measurements using the RAPL [7] interface whereas for NVIDIA GPUs we use the NVML [13] library. In order to monitor task execution times and the overhead of memory transfers we exploit OpenCL events.

3. Applications - Approximation and Quality Estimation Case Studies

In order to validate our framework and quantify the quality / energy footprint tradeoff on heterogeneous systems using realistic codes, we ported and evaluated a number of real-world applications to our programming model. In the following paragraphs we introduce these applications.

PBPI [5] is a high performance implementation of the Bayesian phylogenetic inference method for DNA sequence data. It starts from random phylogenetic trees and estimates the likelihood of them being realistic. The trees are then modified and re-evaluated in an iterative evolutionary process. The tree with the maximum likelihood is the output of the application. PBPI is quite sensitive to errors and applying approximations is not a straightforward task. We introduce an implementation where we randomly drop calculations for mutations with low probabilities. We validate the approximate version by comparing the similarity of the produced trees to those of the accurate version, using an algorithm for tree comparison [12]. This quality metric takes into consideration both tree topology and branch lengths.

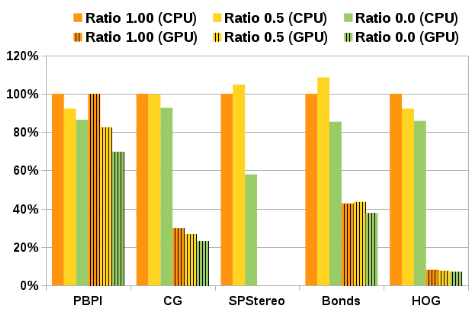
Conjugate gradient (CG) is an iterative numerical solver for systems of linear equations. The matrix form of these systems has to be symmetric and positive-definite. The algorithm stops when it reaches convergence within a tolerance value, or executes the maximum number of iterations requested by the user. In order to approximate CG we used mixed precision: we perform computations of low significance in single precision, while the rest are executed in double precision. Given that the application is iterative, the number of iterations to convergence depends on the method and degree of approximation. To quantify the quality of the solution we use the relative error w.r.t. the result of the fully accurate execution.

The SPStereo Disparity [22] application calculates a dense depth estimate image from a stereo pair camera input. It consists of two parts: the first produces an initial disparity image; the second exploits shape regularization in the form of boundary length, while preserving connectivity of image segments to produce the depth estimate image. The hotspot of the algorithm is the computation of the initial disparity image. We approximate the disparity image computation by relaxing synchronization between consecutive rows of pixels in the image. We compare the image quality using the PSNR metric, with respect to the disparity image produced by a fully accurate execution.

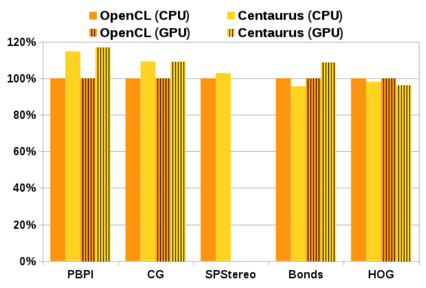
Bonds [6] is part of the QuantLib [1] library used in computational finance. The application calculates the dirty price, clean price, accrued amount on a date and the forward value of a bond[2]. We apply two different approximation techniques. The first one uses mixed precision and fast-math functions for the calculation of exponentials, while the second drops computations of the iterative algorithm that computes the bond yield. The quality metric we use is the relative error of the computed forward price of the bond with respect to the value computed by a fully accurate execution.

HOG [15] is a computer vision application for pedestrian detection using machine learning techniques. The input image is divided into independent blocks that can be analyzed in parallel. A set of kernels is applied iteratively, in a pipeline manner on each block. The first kernel creates a histogram of the gradients orientation. Then it combines them into a descriptor and finally feeds it on a Support Vector Machine (SVM) which classifies each block. We use an approximation approach that skips the histogram and SVM computations on some image blocks in a round robin manner. We ensure that neighboring tasks have different significance values, allowing the runtime to apply approximations uniformly to the image. To assess quality, we calculate the percentage of overlap between bounding

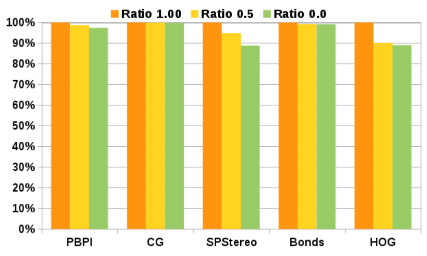
[2]A bond is a loan that exists between an issuer and a holder. The issuer is obligated to pay the holder the initial loan augmented by an interest.



(**a**) Relative energy consumption w.r.t. the accurate CPU run.



(**b**) Performance overhead w.r.t. the corresponding accurate OpenCL implementation.



(**c**) Relative quality of output w.r.t the fully accurate execution.

	Device	Time (s)	Energy (J)	Quality metric
PBPI	CPU	72.20	11811	Tree similarity (topology & branch length)
	GPU	104.00	11065	
CG	CPU	83.00	16157	Relative error wrt correct execution (%)
	GPU	48.00	4814	
BONDS	CPU	2.02	471	Relative error wrt correct execution (%)
	GPU	1.97	201	
HOG	CPU	15.43	3820	Windows overlapping (%)
	GPU	3.80	312	
SPStereo	MIXED	5.12	935	PSNR (dB)

(**d**) Baseline execution time and energy consumption of fully accurate executions. We also report the quality metric used for each application.

Figure 2.

windows of recognized pedestrians produced by the accurate and the approximate versions.

4. Experimental Evaluation

The experimental evaluation was carried out on a dual-socket system equipped with two Intel XEON E5 2695 processors, clocked at 2.3 GHz, with 128 GB DRAM and an NVIDIA Tesla K80 GPU. The operating system is Ubuntu 14.04 server, using the 3.16 Linux kernel. The combination of two CPUs results to more energy consumption than a single GPU in some applications despite comparable execution times. The GPU power monitoring interface returns the instantaneous power consumption polled every 2 ms. To estimate energy on GPU, we calculate the integral of power in this time window of 2 ms, assuming constant power within the window. On CPU, in order to calculate power, we monitor energy consumption using the Running Average Power Limit (RAPL) [7] interface. Although our evaluation system and runtime can handle execution on multiple GPUs, we limit our profiling benchmarks only to one GPU chip of the K80, as most of the applications we use do not offer enough parallelism to exploit both GPU chips.

We evaluate each application on different devices (CPU/GPU) and degrees of approximation (fully accurate (ratio 1.0), fully approximate (ratio 0.0) and mixed (ratio 0.5)). In the literature, approximation techniques have been typically evaluated at the granularity of computational kernels, and not on end-to-end applications. We focus on the energy efficiency and the total execution time of each application, including data transfers between host and device. Fig. 2a depicts the relative energy consumption of all cases compared with the energy consumption of the accurate CPU execution for each application. Fig. 2c shows the quality loss due to approximations. Note that the quality between different device types for the same approximation degree remains the same for all applications in our evaluation. Also the energy consumption of the fully accurate, pure OpenCL implementation and the accurate run using our programming model is the same for all applications, for both CPU and GPU executions. Table 2d outlines CPU/GPU execution time and energy consumption for a fully accurate execution of each application.

All applications exploit the features of our programming model such as dependency analysis and automated data manipulation. We observe measurable energy gains in all applications ranging from 12% up to 43% due to approximations only. Considering also the proper device selection, these percentages become 30% and 90% respectively. We also observe that properly adjusting the ratio that controls the degree of approximation, we can control the energy/quality tradeoff in a controlled and straightforward manner. Both observations validate our approach. On average our runtime adds a minimal performance overhead about 0.28% on all applications except for PBPI and CG which are discussed below.

We notice that our runtime introduces an overhead of 15.7% (Fig. 2b) on average for PBPI when compared with the pure OpenCL version. The reason is the fine task granularity: PBPI creates about 100,000 tasks with an average execution time of 1 ms each. The overhead is due to both the latency of the underlying OpenCL implementation which notifies our runtime for task completion, and the frequent calls to the data dependency resolver. Another interesting observation for PBPI is that although GPU is slower than the CPU execution, the approximate version consumes 30.2% and 19.2% less energy than the accurate and approximate CPU execution respectively.

Our runtime also introduces an overhead of 9.1% on average in CG, again due to the number and granularity of tasks (which are however coarser than in the case of PBPI). GPU seems to gain more performance and saves more energy because these devices are designed to execute single precision calculations – as those used by the approximate version of the implementation – efficiently. CG has an energy gain of 76.9% on the approximate GPU execution.

The SPStereo Disparity accurate version suffers from sequential dependencies across consecutive rows of the image, limiting parallelism. In contrast, the approximate version due to the alleviation of dependencies is highly parallel and can easily benefit from running on GPU. Thus this application makes appropriate use of both GPU and CPU task bindings, a feature our programming model offers out of the box. The PSNR of the images produced by the approximate execution is in the range of 36 dB, indicating extremely high quality. Therefore, the approximate version offers an excellent trade-off between performance, energy efficiency (up to 43%) and quality of output. Although the PSNR value of the accurate run is – by definition – infinity, for visualization purposes in Fig. 2c we limit it to 41 dB.

In Bonds, we run a number of experiments with different input sets, using combinations of the available input variables such as issue date, maturity data and coupon rate for the bond. The application appears to benefit from execution on a GPU, facilitated by our programming model, although the difference with CPU is quite small. Bonds uses demanding operations (like exponential) in which CPU thrives against GPU. Using both approximation techniques, we introduce a quality loss of 1%, in favor of energy gains up to 14.7% in CPU and 11.8% in GPU w.r.t the accurate CPU and GPU executions respectively.

In HOG approximations result to some quality loss, which is mainly due to unrecognized pedestrians when they are smaller than the block size. Energy gains correlate with image sizes and not with the content. The highest energy gain with respect to the CPU accurate version is 90% and comes from the GPU approximate version. Our programming model results to a better execution time on both CPU and GPU than pure OpenCL, because it automatically exploits all opportunities for minimization of data transfers and their overlap with computations. The energy gain from running the approximate versions is 13% for GPU and 14% for CPU compared with the corresponding GPU and CPU accurate executions.

5. Related work

Green [2] supports energy-conscious programming using controlled approximation while providing guaranteed QoS. Ringenburg et al. [17] propose an architecture and tools for autotuning applications, that enable trading quality of results and energy efficiency. They assume, however, approximations at the hardware level. EnerJ [19] introduces an approximate type system using code annotations without defining a specific programming and execution model. ApproxIt [23] approximates iterative methods at the granularity of one solver iteration. Variability-aware OpenMP [16] also follows a *#pragma*-based notation and correlates parallelism with approximate computing. Quickstep [11] is a tool that parallelizes sequential code. It approximates the semantics of the code by altering data and control dependencies. SAGE [18] is a domain-specific environment with a compiler and a runtime component that automatically generates approximate kernels for image processing and machine learning applications. GreenGPU [9] dynamically splits and distributes workloads on a CPU-GPU heterogeneous system, aiming to keep busy both sides all the time, thus minimizing idle energy consumption. It also applies DFS for the GPU core and memory for maximizing energy savings. Tsoi and Luk [21] estimate performance and power efficiency tradeoffs to identify optimal workload distribution on a heterogeneous system.

Our work introduces the concept of computation significance as a means to express programmer wisdom and facilitate the controlled, graceful quality degradation of results in the interest of energy efficiency. We support approximate computing in a unified, straightforward way on different devices of accelerator-based systems, thus exploiting and combining energy efficiency benefits from both heterogeneity and approximation. Our approach does not require hardware support apart from what is already available on commodity processors and accelerators.

6. Conclusions

We introduced a framework which allows the programmer to express her wisdom on the importance of different computations for the quality of the end result, to provide approximate, more energy efficient implementations of computations and to control the quality / energy efficiency tradeoff at execution time, using a single, simple knob. In addition, our framework allows execution on heterogeneous systems and alleviates some technical concerns, such as computation scheduling and data management, which limit the programmer productivity. We evaluated our approach using a number of real-world applications and found that exploiting the concept of significance at the application level enables measurable energy gains through approximations, while the programmer maintains control of the quality of the output.

It should be noted that software-level approximate computing, as discussed in this paper, is orthogonal to energy efficiency optimizations at the hardware-level. Therefore, our approach can be applied on a wide range of existing and future systems, spanning the range from HPC architectures to embedded systems.

Acknowledgements

This work has been supported by the "Aristeia II" action (Project "Centaurus") of the operational program Education and Lifelong Learning and is co-funded by the European Social Fund and Greek national resources.

References

[1] F. Ametrano and L. Ballabio. Quantlib-a free/open-source library for quantitative finance. *Availabl e: http://quantlib. org/(visited on 04/29/2014)*, 2003.

[2] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '10, pages 198–209, New York, NY, USA, 2010. ACM.

[3] M. Bohr. A 30 year retrospective on dennard's mosfet scaling paper. *Solid-State Circuits Society Newsletter, IEEE*, 12(1):11–13, Winter 2007.

[4] W.-c. Feng and K. W. Cameron. The green500 list: Encouraging sustainable supercomputing. *Computer*, 40(12):50–55, 2007.

[5] X. Feng, K. W. Cameron, and D. A. Buell. Pbpi: A high performance implementation of bayesian phylogenetic inference. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[6] S. Grauer-Gray, W. Killian, R. Searles, and J. Cavazos. Accelerating financial applications on the gpu. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, pages 127–136. ACM, 2013.

[7] Intel. Intel 64 and ia-32 architectures software developer manual, 2010. Chapter 14.9.1.

[8] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–, Washington, DC, USA, 2004. IEEE Computer Society.

[9] X. Li. Power Management for GPU-CPU Heterogeneous Systems. Master's thesis, University of Tennessee, 12 2011.

[10] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. Top 500 list. *Electronically published at http://www. top500. org*, 2010.

[11] S. Misailovic, D. Kim, and M. Rinard. Parallelizing sequential programs with statistical accuracy tests. *ACM Trans. Embed. Comput. Syst.*, 12(2s):88:1–88:26, May 2013.

[12] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. Treejuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. *ACM Trans. Graph.*, 22(3):453–462, July 2003.

[13] NVIDIA. NVML API Reference. http://docs.nvidia.com/deploy/nvml-api/index.html.

[14] OpenACC standard committee. The OpenACC Application Programming Interface, v2.0, June 2013.

[15] V. Prisacariu and I. Reid. fastHOG-a real-time GPU implementation of HOG. Technical Report 2310/9, Department of Engineering Science, Cambridge University, 2009.

[16] A. Rahimi, A. Marongiu, R. K. Gupta, and L. Benini. A variability-aware openmp environment for efficient execution of accuracy-configurable computation on shared-fpu processor clusters. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '13, pages 35:1–35:10, Piscataway, NJ, USA, 2013. IEEE Press.

[17] M. Ringenburg, A. Sampson, I. Ackerman, and L. C. D. Grossman. Monitoring and debugging the quality of results in approximate programs. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2015)*, Istanbul, Turkey, March 2015.

[18] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 13–24, New York, NY, USA, 2013. ACM.

[19] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 164–174, New York, NY, USA, 2011. ACM.

[20] J. E. Stone, D. Gohara, and G. Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test*, 12(3):66–73, May 2010.

[21] K. H. Tsoi and W. Luk. Power profiling and optimization for heterogeneous multi-core systems. *SIGARCH Comput. Archit. News*, 39(4):8–13, Dec. 2011.

[22] K. Yamaguchi, D. McAllester, and R. Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. In *ECCV*, 2014.

[23] Q. Zhang, F. Yuan, R. Ye, and Q. Xu. Approxit: An approximate computing framework for iterative methods. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, DAC '14, pages 97:1–97:6, New York, NY, USA, 2014. ACM.