A Context-Aware Library for Mathematical Approximations

Roberto Alejandro Hidalgo Castro and Lucas Wanner

[1]Institute of Computing – UNICAMP
Av. Albert Einstein, 1251 – 13083-852 – Campinas, Brazil.

{r164787,lucas}@ic.unicamp.br

Abstract. *We have built a mathematical library that includes a series of functions with different implementations with varying precision. We couple this library with a system service that monitors the computer context, including energy consumption, and according to this context, using specified rules, dynamically changes the implementations used by the target applications. Our case studies show that our library can trade-off at most of 4% degradation in application quality up to 40% savings in energy consumption.*

1. Introduction

Recently, approximate computing has emerged as a promising approach to the energy-efficient implementation of digital systems. Approximate computing relies on the ability of many systems to tolerate some loss of quality in the computed result. By relaxing the need for fully precise operations or where an approximate result is sufficient, approximate computing techniques allow considerably improved energy efficiency.

Approximate computing has been explored at the compiler level, with works like EnerJ [Sampson et al. 2011] and PetaBricks [Ansel et al. 2009]; at runtime with Green [Baek and Chilimbi 2010], and ViRUS [Wanner and Srivastava 2014], and through hardware architectures like Doppelanger [Miguel et al. 2015] and DRUM [Hashemi et al. 2015].

These works have obtained progress in energy saving by cost of quality as well; Virus [Wanner and Srivastava 2014] managed to get saving in energy by 40% for the application Blackscholes and 50% for the application Swaptions just for a cost of at most 4% degradation in quality; Green [Baek and Chilimbi 2010] obtained an energy improvement of 28% by the lost of quality of les than 1% in the application Blackscholes.

We develop a functional adaptation technique through multiple code paths, with a focus on dynamic control monitoring for variable cost and quality objectives. Our results show us that with the use of our mathematical library we can achieve considerable savings in energy consumption at a low cost in quality degradation. This work also shows that if we build policies around the implementations of these library, we can lead our target application to reach an energy consumption goal over time, and our results show how the energy at the end oscillates around this goal.

2. System design and implementation

Our application support system allows applications to dynamically trade function implementations inside the application and with this trading increasing or decreasing the quality of the results but with the advantage of decreasing or increasing the energy consumption of the application when is being run at the operating system. To allow this change in the function implementations in the target application, our system has a library that has been built containing many implementations of the same function. When possible, this library is linked automatically to the application and the system is able to check the performance of the application at runtime, measure the current energy consumption and depending on this, to trigger instructions to change the function implementations.

The goal of this change is to increase or decrease the energy consumption of the application but at the cost of the least lost of quality as possible. In the following parts, we explain how this library is built and linked to the application and how the system controls the performance of the application and alters the quality at runtime.

2.1. Variable quality library design

This library was built with in C language, using C tools like macros to allow a multiple declaration of functions and to change implementations of the functions dynamically. As we said, certain function has multiple implementations and to distinguish between each of them, we are using levels. With this library, we should be able to change these levels inside the applications (each level represents a certain grade of quality reduction with purpose of decreasing the energy consumption). To easily change the level of a function, we are using helpers to get/set the quality level. Source code and examples for the library are available at https://github.com/RobertoHC/MathContextAwareLibrary.

This library will support C math functions, each of which has four basic implementations. These implementations are divided in four main versions for each function: double, float, approximate and faster approximate. We use levels to easily refer to each of these implementations.

2.2. Dynamic Adaptation and Monitoring System

In order to dynamically switch implementations depending on the context, the library includes a UNIX signal handler that receives asynchronous commands from a context monitoring service and uses the $\_set\_qlevel$ and $\_get\_qlevel$ helpers that are available for each of the library functions to change implementations using function pointers.

A system daemon monitors the resource usage of the applications and sends signals to reduce or increase the quality when appropriate. The monitoring system has an energy goal set as a percentage of the energy consumption at the highest quality (the default quality of the application). We work with two signals, SIGUSR1 and SIGUSR2 to reduce and increase quality, respectively. This step is done periodically by the monitoring daemon, trying to reach the energy goal at each iteration. This system uses energy counters to estimate the energy consumption.

3. Experiments

We develop a methodology for the evaluation of the developed library in the target applications and the results obtained in this work.

3.1. Experimental Setup

For our experiments, we use VarEMU [Wanner et al. 2013], a simulation framework that allows us to emulate variations in power consumption and to adapt to these variations in software. VarEMU supports configurable models for static and dynamic power. For this work we used the default model which is fitted to a Cortex M class ARM chip [Wanner et al. 2013]. In our experiments, we report energy consumption and execution time savings relative to the standard app with no modifications. Energy is given in Joules and execution time in cycles. These numbers are normalized in our experiments. The virtual machine created trough VareEMU interacts with it through memory mapped registers, and VarEMU measure the energy and execution time by creating a checkpoint for all VarEMU registers.

3.2. Results

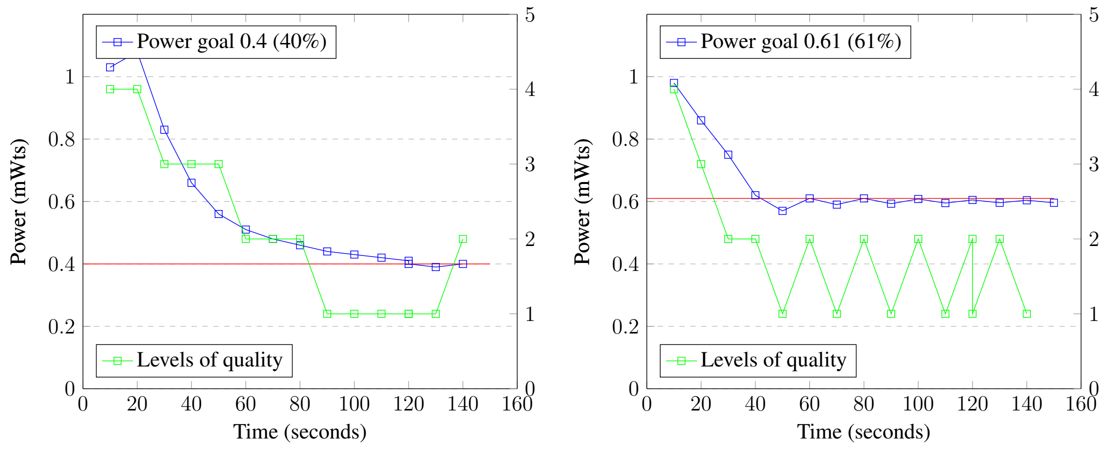
In the following table, we show the results of using library function implementations for each of the target applications, showing the trade-off between time/energy consumption and application quality.

Table 1. Approximate Computing Frameworks

Applications	Versions					
	Medium quality			Low quality		
	time (% savings)	energy (% savings)	quality loss	time (% savings)	energy (% savings)	quality loss
Blackholes	10.61	9.52	1.578	20.85	20.34	2.678
Bodytrack	17.5	17.7	1.855	16.4	16.2	2.137
Facesim	0.02	0.03	0	0.02	0.02	0
Ferret	17.62	11.15	1.457	21.22	15.21	2.726
Vips	12.04	12.1	0.835	18.02	18.2	1.957
Swaptions	7.92	7.91	1.486	19.92	20.88	3.486
Whetstone	40.37	40.34	5.9/3.3 (MIPS)	40.45	40.69	5.9 /3.3 (MIPS)
FFT	61.6	61.8	2.542	62.5	62.9	3.611
Basicmatch	38.72	38.83	0.122	41.72	42.83	0.234
Susan	7.47	6.82	0.147	9.47	9.82	0.347

We use the metric NRMSE (normalized root mean square error) to analyze the output quality of each of the applications (except for the case of Whetstone, whose output is in MIPS). For most of the applications, there is a good reduction in execution time as well as in energy consumption. We got over 15% energy reduction in six of the applications we evaluated, without losing more than 4% of their quality output (using NRMSE). In Susan where we got low time and energy savings, with similarly negligible quality loss, resulting in modest savings for a very low lost in quality. In particular in the case of Facesim, seems that the use of math functions that system supports doesn't have too much impact in the whole application performance.

In addition to the evaluation for each individual application, we also evaluate how the monitoring system works with the applications. Figure 1 shows how the monitoring system works with two of the evaluated applications: FFT and Basicmath. This monitoring system starts with a power goal and it sends signals to the application, so it decreases its quality output (which can be seen in the be plots as levels) and this causes a lower power consumption, getting closer to the power goal. In the graphics the average power consumption always gets closer to the goal overtime, and it usually oscillates at the end over this goal.



(a) FFT (b) Basicmath

Figure 1. Average power across time

4. Conclusion

We developed an application support system that includes a polymorphic math library and a dynamic context monitoring and quality adaptation system. We perform function replacement in polymorphic versions of the standard C math library function in Linux. The application case studies using this library shows that we can trade off a marginal effect on output quality, at most 4% degradation in application quality for up to 40% savings in energy consumption and time execution.

Acknowledgements

This work was supported in part by CNPq, CAPES, and FAPESP.

References

Ansel, J., Chan, C., Wong, Y. L., Olszewski, M., Zhao, Q., Edelman, A., and Amarasinghe, S. (2009). Petabricks: A language and compiler for algorithmic choice. *SIGPLAN Not.*, 44(6):38–49.

Baek, W. and Chilimbi, T. M. (2010). Green: A framework for supporting energy-conscious programming using controlled approximation. *SIGPLAN Not.*, 45(6).

Hashemi, S., Bahar, R. I., and Reda, S. (2015). Drum: A dynamic range unbiased multiplier for approximate applications. In *ICCAD'15*, pages 418–425. IEEE Press.

Miguel, J. S., Albericio, J., Moshovos, A., and Jerger, N. E. (2015). Doppelganger: A cache for approximate computing. In *MICRO-48*, pages 50–61. ACM.

Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., and Grossman, D. (2011). Enerj: Approximate data types for safe and general low-power computation. *SIGPLAN Not.*, 46(6):164–174.

Wanner, L., Elmalaki, S., Lai, L., Gupta, P., and Srivastava, M. (2013). Varemu: An emulation testbed for variability-aware software. In *CODES+ISSS '13*. IEEE Press.

Wanner, L. and Srivastava, M. (2014). Virus: Virtual function replacement under stress. In *HotPower'14*, Berkeley, CA, USA. USENIX.