Energy Efficient FPGA based Hardware Accelerators for Financial Applications

Jakob Kenn Toft and Alberto Nannarelli
Dept. of Applied Mathematics and Computer Science
Technical University of Denmark
Kongens Lyngby, Denmark

***Abstract*—Field Programmable Gate Arrays (FPGAs) based accelerators are very suitable to implement application-specific processors using uncommon operations or number systems.**

In this work, we design FPGA-based accelerators for two financial computations with different characteristics and we compare the accelerator performance and energy consumption to a software execution of the application.

The experimental results show that significant speed-up and energy savings, can be obtained for large data sets by using the accelerator at expenses of a longer development time.

I. Introduction

Hardware acceleration provides both speed-up and energy efficiency for computer systems. In recent years, two main classes of accelerators have emerged as the most relevant: Graphics Processing Units, or GPUs, and FPGA (Field Programmable Gate Array) based accelerators.

Modern GPUs are composed of a large array of identical computation cores to exploit parallelism. As GPUs are now fundamental components in supercomputers [1], their cores are designed to support standards to enable general-purpose computing (e.g., IEEE 756 standard for floating-point).

In contrast, FPGA based accelerators are more suitable to implement Application-Specific Processors, or ASPs, which optimize the hardware for the specific operations necessary for the application. For example, applications requiring a specific number system, such as the decimal system, or applications requiring modular operations necessary for cryptography.

The major drawback of FPGA based acceleration is the long development time compared to a software implementation of the application: the application is solely run on the CPU. There are a few solutions and suites of tools to reduce this development time, but the gap against software development is still huge [2]. In addition, the vast variety of FPGA board vendors makes hardware accelerators generally not portable across different platforms.

To summarize, FPGA based accelerators provide more hardware flexibility than GPU based accelerators because the processing cores can be tailored to execute specific operations in a specific (non-standard) number system. This flexibility is exploited to obtain better performance (lower latency, higher throughput, and lower power dissipation). However, the development costs of FPGA accelerators are the highest and the ASP is generally not portable across different platforms.

In this work, we present the results of the implementation of FPGA based accelerators for a financial computing application. The accelerator is realized by an ASP communicating to the CPU of the host computer through a standard bus. We measured the actual energy consumption by monitoring the current in both the CPU and the FPGA board.

The results demonstrate that significant speed-ups and power savings are obtained. For future developments, we propose a solution to shorten the design time and enhance the portability of ASPs developed for other platforms.

II. Applications

We chose two applications with very different characteristics (number system and communication requirements) for the hardware acceleration.

The first one is a telephone billing application based on the decimal number system. Many financial and accounting applications resort to decimal arithmetic because some decimal fractions (e.g., 0.1) cannot be represented with a finite number of bits in binary, and, in some cases, rounding errors arise [3]. These errors are not acceptable in financial and accounting applications. Only a few processors have hardware support for decimal operations, and the rest of processors implement decimal arithmetic in time consuming software routines. Our accelerator should provide the necessary hardware to execute decimal operations.

The second application is a Monte Carlo simulation for pricing of European options. An increasing number of stock market transactions are made through computer systems exploiting small variations in the price of the asset and buying/selling in a few milliseconds (high frequency trading, or HFT). The choice of buying/selling an asset is made based on the results of batch simulations which statistically determine the price to make a profit. These simulations require a lot of computer power and they can be easily parallelizable.

We give some detail on the algorithms implementing the selected applications in the following.

A. Telephone Billing

The *"TELCO"* benchmark was developed by IBM to investigate the balance between I/O time and calculation time in a telephone company billing application [4]. The benchmark

978-1-4799-5442-1/14/$31.00 ©2014 IEEE

Algorithm 1 Pseudo-code for the computations in TELCO benchmark.

```
if (calltype = L) then
  P = duration × Lrate
else
  P = duration × Drate
end if
Pr = RoundtoNearestEven(P)
B = Pr × Btax
C = Pr + Trunc(B)
if (calltype = D) then
  D = Pr × Dtax
  C = C + Trunc(D)
end if
```

Operation	Maximum	Average
addition	133	71
division	266	171
multiplication	132	69

decimal64 (16 decimal digits) operations.

TABLE I
CLOCK CYCLE COUNT FOR A SUBSET OF DECIMAL OPERATIONS [5].

provides an example of IEEE 754-2008 compliant set of decimal floating-point operations (multiplication and addition). The benchmark is available in C and Java, and it can be executed in any computer.

The benchmark reads an input file containing a list of telephone call duration (in seconds). The calls are of two types (*"Local"* and *"Distance"*) and to each type of call a rate (price) is applied. Once the call price has been computed, one or two taxes (depending on the type of call) are applied. The price of the call must be rounded to the nearest cent (round-to-even in case of a tie), while the tax is computed by truncating to the cent.

The pseudo-code of the TELCO algorithm is listed in Algorithm 1. The algorithm is completed by adding the sum of the costs of the calls.

Because the call cost computation is executed in the decimal system, each operation (addition, multiplication, and rounding) is executed by a software routine requiring several cycles. For example, Table I reports (source [5]) the average number of clock cycles necessary to implement a subset of decimal operations in binary floating-point units.

For this reason, we chose the TELCO application as an excellent case for acceleration in an ASP where arithmetic operations are implemented by decimal hardware units.

The TELCO benchmark is characterized by having high traffic from the memory (where the duration of the calls are stored) to the processor and back to the memory (costs of calls are stored).

B. Option Pricing

The value of a security at time T, $S(T)$, for a European option can be computed by using a Monte Carlo simulation

Algorithm 2 Monte Carlo European option pricing.

```
VsqrtT = σ√T
drift = (r − σ²/2)T
expRT = e^{rT}
sum = 0
for i = 1 to n do
  St = S_0 · e^(drift + VsqrtT · Vrnd)
  if (St − K > 0) then
    sum = sum + (St − K) · expRT
  end if
end for
S = sum/n
```

to evaluate the Black-Scholes-Merton equation

$$S(T) = S(0) \cdot e^{\left[\left(\mu - \frac{\sigma^2}{2}\right)T + \sigma\epsilon\sqrt{T}\right]} \quad (1)$$

for several samples and then by computing the mean value [6]. The Monte Carlo simulation of (1) can be implemented by Algorithm 2 with the following inputs: S_0 initial security price, K strike price, r risk-free interest rate, σ security volatility, T time to expiration (in years), n number of simulations. The parameters r and σ are constant, and, therefore variables `VsqrtT`, `drift` and `expRT` are constant for the simulation.

The algorithm requires to generate random numbers with normal distribution in $(-1.0, 1.0)$.

Differently from the TELCO application, for typical simulation runs ($n > 10^5$), the Monte Carlo Option Pricing (MCOP in the following) does not require much communication with the memory and most of the execution time is spent in the computations.

III. IMPLEMENTATION PLATFORM

The hardware accelerator is implemented in the Xilinx Virtex-5 LX330T FPGA. This FPGA is embedded in the Alpha Data ADM-XRC-5T2 board and is connected to the host PC via the PCI bus. The board is also equipped with DRAM that can be accessed by the FPGA and by the host PC via Direct Memory Access (DMA). The CPU of the host PC is the Intel Core2 Duo processor clocked at 3 GHz.

The implementation of the DMA functions, along with others, is included in a Software Development Kit (SDK) provided with the Alpha Data board. The SDK includes an application-programming interface (API), VHDL functions and examples.

Fig. 1 sketches the architecture of the accelerator: CPU, FPGA, and communication. We implemented in the FPGA: the ASP for the application, a front-end processor (FEP) which handles the communication ASP-DRAM-CPU, and the DMA functions (not depicted in Fig. 1).

A. Power Measurements

To be able to evaluate the energy efficiency of the accelerator, we perform a measurement of the current consumption during the execution of the application in both the CPU and the FPGA board.

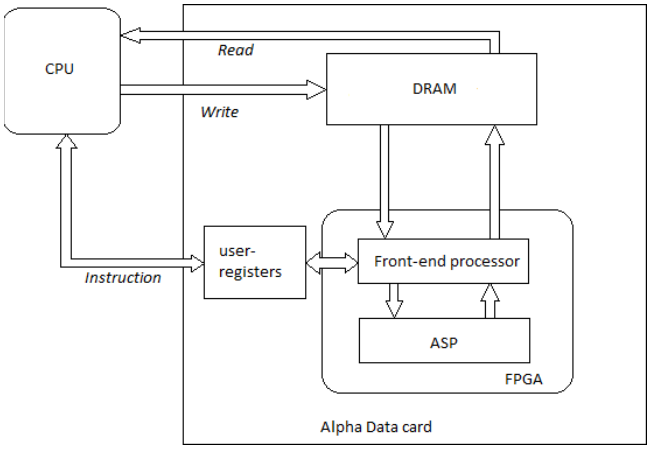

Fig. 1. Overview of the accelerator platform.

The power dissipated in the CPU is monitored by a Hall's effect current sensor which is connected between the power supply and the motherboard's CPU power socket. We connect a digital multimeter to the sensors' terminals and dump the readings (voltage drop proportional to current flowing in the CPU) in a file for batch processing. More detail on the measurement set-up is described in [7].

The FPGA board power monitoring is done by reading the voltage drop across a 7.5 $m\Omega$ shunt resistor in series with the FPGA board power supply V_{CC}. Also in this case, the readings V_{meas} are done with a digital multimeter and stored in a file. The instantaneous power dissipation is then computed by $P = \frac{V_{meas}}{7.5 \cdot 10^{-3}} V_{CC}$.

The power measured in the FPGA board is inclusive of the FPGA chip itself, the DRAM and all the peripherals on the board. Although Virtex-5 family FPGAs are equipped with pins to monitor the FPGA core power dissipation, these pins are not accessible in the Alpha Data board.

IV. TELCO Accelerator

The architecture of the accelerator for the TELCO application is derived from the one presented in [8] with some important enhancements.

First, the accelerator of [8] could only handle small sets of data (call duration to process) because data were sent from the CPU directly to the ASP (FPGA) by using a buffer and not to the DRAM on the FPGA board. In this version of the accelerator, we designed the front-end processor, implemented on the FPGA along the ASP, to handle both the data transfer (via DMA) with the host PC, and the communication CPU-ASP (via specific instructions).

Second, we apply some optimization on the ASP to make the computation more efficient when executed on FPGA platforms, and re-pipelined the unit to work at a clock frequency of 70 MHz.

Third, we perform a comprehensive performance evaluation based on actual measurements, including energy consumption.

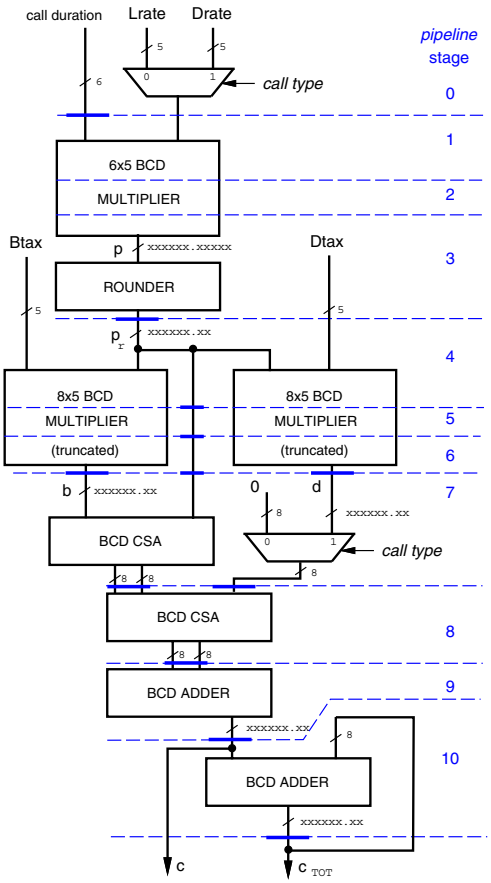

Fig. 2. ASP for TELCO accelerator.

A. Accelerator Set-Up and Operations

To run the application on the FPGA accelerator, we need to set-up the device (program the FPGA). The following phases are necessary to run the application on the accelerator:

1) program the FPGA (load the bitstream);
2) activate the DRAM (self-training);
3) execute the application in the FPGA;
4) deactivate the accelerator (FPGA closing).

We discuss in Sec. VI the impact of the different phases on the overall performance of the accelerator.

During the application execution in the accelerator, phase 3, the following operations occur (refer to Fig. 1):

A) Data are loaded from virtual memory of host PC;
B) Data are written to DRAM of FPGA board;
C) ASP reads data from DRAM, processes data, and write them back in DRAM;
D) Results are read from DRAM of FPGA in host PC memory.

B. TELCO ASP

The ASP implementing the TELCO computation is depicted in Fig. 2. One of the advantages of the FPGA implementation is that we can tailor the processor to the necessary computations and deviate from standards.

For the TELCO ASP, we assume that the call duration (in seconds) is a 6-digit integer decimal number in the

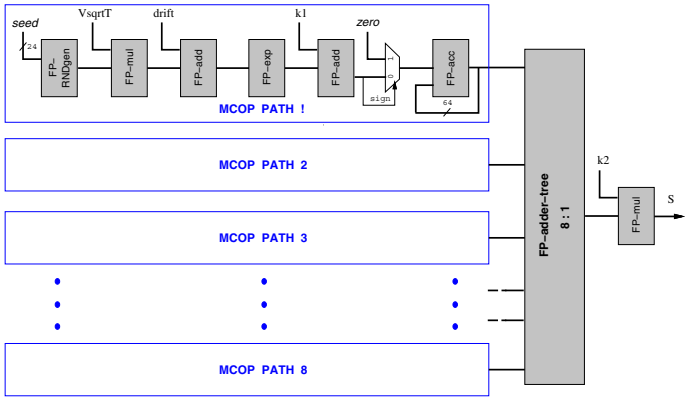

Fig. 3. ASP for 8-path MCOP accelerator.

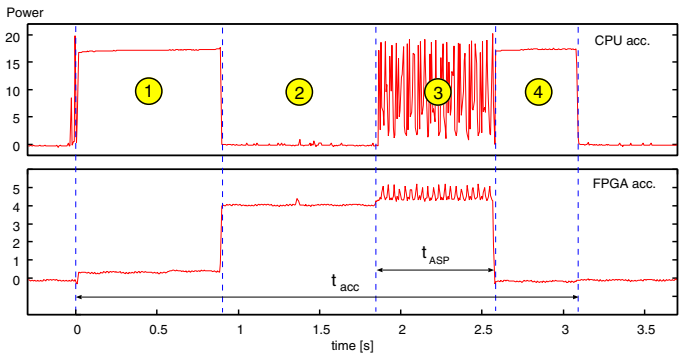

Fig. 4. Execution time in accelerator (1,000,000 calls).

range $[0, 10^6)$ ($10^6 s$ corresponds to 11.5 days), and a 5-digit decimal fractional number for the call rate and taxes. The call rate is selected by a multiplexer depending on the type of the call. The cost of each call and the total cost (C and C_{TOT}, respectively, in Fig. 2) are represented by 8-digit decimal numbers (6 integer and 2 fractional digits) for values up to $999,999.99$ (e.g., euro). Detail on the blocks in Fig. 2 can be found in [8].

The ASP is pipelined in 11 stages and clocked at a frequency of 70 MHz for a latency of $10 \times 14.3\ ns = 143\ ns$, and an ideal[1] throughput of 70 million of calls processed per second.

V. MCOP Accelerator

Algorithm 2 is mapped in the ASP sketched in Fig. 3. The algorithm is easily parallelizable by unrolling the loop in P parallel paths, labeled "MCOP PATH X" in Fig. 3. The $P = 8$ paths are then recombined by an adder tree. The algorithm is implemented in IEEE compliant *binary32* floating-point (FP) format[2].

Instead of performing the multiplication $S_0 \cdot e^{(\cdots)}$ in each cycle of the loop (Algorithm 2), we divide K by S_0 as a pre-computation (in the CPU) and compare $e^{(\cdots)}$ directly to `k1`$=K/S_0$. Similarly, we remove the multiplication by `expRT` out af the loop. The correct value of S is restored in the last stage by performing the multiplication by `k2`$= S_0 \cdot$`expRT`$\cdot \frac{1}{n}$.

The most critical FP-unit is the accumulator (at the end of each path). The FP-accumulator is designed to sustain a throughput of one result (per path) per clock cycle. Detail on the FP-units implementation can be found in [9].

For the MCOP accelerator the FEP provides the parameters: n, `VsqrtT`, `drift`, `k1` and `k2`. The DRAM is not used.

The ASP is pipelined in 29 stages and clocked at a frequency of 80 MHz for a latency of $29 \times 12.5\ ns = 363\ ns$, and an ideal throughput of 22 M elements processed per second.

[1]When the ASP is not slowed down by the I/O.

[2]In the 2008 revision on the IEEE standard 754 *binary32* replaces the wording *single-precision*.

VI. Performance and Power Measurements

To evaluate the performance and energy consumption of the accelerators, we compared the execution of the application in software, run in the CPU of the PC hosting the FPGA board, and the execution in the accelerator: CPU and FPGA board.

We run several batches of data to derive trends and to see how the execution in the accelerator scales with the data set.

The Intel Core2 two CPUs are set to the maximum performance, corresponding to a frequency of 3 GHz, for both the software and the accelerator execution.

We performed the measurements by reading timers in the C executable (both software and accelerator executions), and by reading the distance between edges in the waveforms logged by the multimeter.

A. TELCO Benchmark

We run several batches of call data (25,000 to 1,000,000 calls) for the TELCO application.

Fig. 4 shows the plots derived from the actual measurements performed for the 1,000,000 calls case for the accelerator (CPU+FPGA average power dissipation).

For all experiments, the FPGA set-up time, labeled (1) and (2) in Fig. 4, is 1.9 s on the average, while the FPGA closing time, labeled (4), is 0.5 s on the average. The total overhead for operating the FPGA is about 2.4 s, independently of the data-set size.

In Table II, we list the timing measurements for the software execution (CPU SW) and for the accelerator. For the accelerator, we list separately the latency of the ASP t_{ASP}, labeled (3) in Fig. 4, and the total latency for the run t_{acc}. We also list the average time to process one call in the SW execution t^C_{SW}, and in the ASP t^C_{ASP}. Moreover, we report the speed-up of the execution of the whole run t_{SW}/t_{acc}, and the speed-up of the calculation per call t^C_{SW}/t^C_{ASP}.

Table II shows that for 250,000 calls the software and accelerator execution have the same latency. For a smaller number of calls, the FPGA set-up overhead makes the accelerator execution much slower than the software solution.

By looking at the time per call in the ASP, we notice that for 50,000 calls and above, t^C_{ASP} settles between 0.72–0.76 μs. Considering that ideally the ASP can process a call per clock cycle (14.3 ns), the I/O, and not the ASP, sets the throughput

LATENCY per RUN								
	calls	10,000	25,000	50,000	100,000	250,000	500,000	1,000,000
CPU SW	latency t_{SW} $[s]$	0.120	0.280	0.520	1.040	2.600	5.240	10.280
	time per call t^C_{SW} $[\mu s]$	12.00	11.20	10.40	10.40	10.40	10.48	10.28
Accelerator	(3) t_{ASP} $[s]$	0.037	0.037	0.038	0.074	0.182	0.362	0.723
	latency$^{(*)}$ t_{acc} $[s]$	2.447	2.447	2.448	2.484	2.592	2.772	3.133
	time per call t^C_{ASP} $[\mu s]$	3.70	1.48	0.76	0.74	0.73	0.72	0.72
Speed-up t_{SW}/t_{acc}		0.05	0.11	0.21	0.42	1.00	1.89	3.28
Speed-up t^C_{SW}/t^C_{ASP}		3.24	7.57	13.68	14.05	14.29	14.48	14.22

$^{(*)}$ t_{acc} is the sum of (1), (2), (3) and (4).

ENERGY CONSUMPTION per RUN								
	calls	10,000	25,000	50,000	100,000	250,000	500,000	1,000,000
CPU SW	E_{SW} $[J]$	1.91	4.68	8.82	17.95	44.55	90.51	181.36
	P_{ave} $[W]$	15.95	16.72	16.96	17.26	17.14	17.27	17.64
Accelerator (CPU+FPGA)	E_{acc} $[J]$	30.08	29.61	29.57	29.90	31.41	34.21	38.92
	P_{ave} $[W]$	12.29	12.10	12.08	12.04	12.12	12.34	12.42
Ratio E_{SW}/E_{acc}		0.064	0.158	0.298	0.600	1.418	2.646	4.660

$P_{ave} = E_{run}/t_{run}$.

TABLE II
TELCO: LATENCY AND ENERGY CONSUMPTION FOR EXPERIMENTS WITH DIFFERENT DATA SETS (NUMBER OF CALLS).

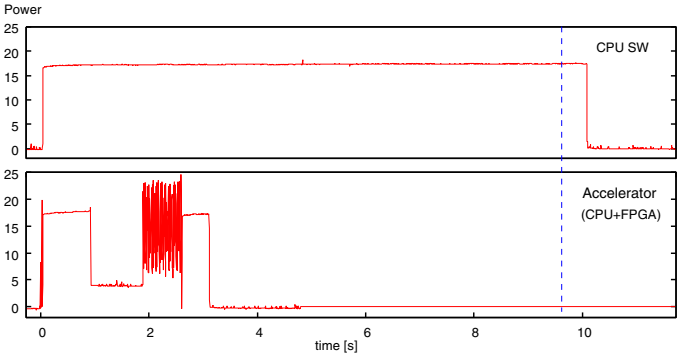

Fig. 5. TELCO: energy consumption to process 1,000,000 calls.

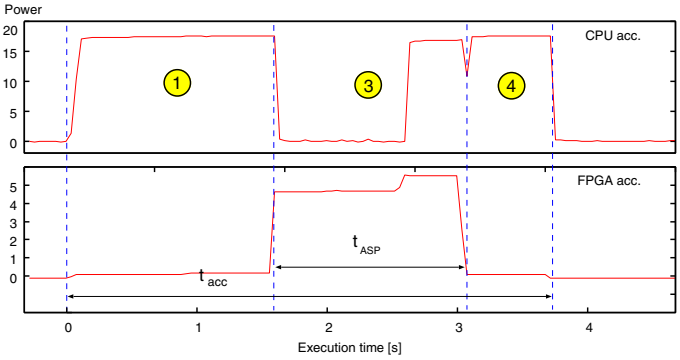

Fig. 6. MCOP: execution time and energy consumption to simulate 256M elements.

of the accelerator. As anticipated, the application execution is *"I/O bound"*: about 95% of t_{ASP} is spent in I/O.

In Table II, we also list the energy consumption measured in the experiments. We show a visual comparison in Fig. 5 for the 1,000,000 calls run. The table and the figure are scaled to have power $P = 0$ when the FPGA is idle and the CPU is not running the application. The idle power for FPGA and CPU are 9.5 W and 4.0 W, respectively.

Despite the large FPGA set-up overhead, for data sets of 250,000 calls and above, the accelerator is more energy efficient.

B. MCOP Simulation

For the Monte Carlo Option Pricing simulation, we run two batches of data: a small set of $n = 100,000$ random values (called elements in the following) and a large set of $n = 256 \times 10^6$ (256M) elements.

Fig. 6 shows the plots for the 256M elements simulation. Differently from the TELCO case, the DRAM on the FPGA board is not used and the phase labeled (2) is skipped in this case. Because the ASP is larger than the TELCO case, the programming phase (1) is longer for this processor: 1.6 s on the average. The closing time (4) is about 0.5 s on the average.

In Table III, we list the timing measurements and the energy consumption for the software execution (CPU SW) and for the accelerator. We list the average time to process one element in the SW execution t^e_{SW}, and in the ASP t^e_{ASP}. Moreover, we report the speed-ups and energy ratios.

Similarly to the TELCO case, for a relatively small number of elements the MCOP execution in the accelerator is totally inefficient as the software execution time is very small compared to the FPGA programming time. For a large number of elements, when millions of floating-point operations must be executed, the parallelism of the ASP and the throughput of 8 elements processed per clock cycle make the accelerator execution significantly faster.

Moreover, as the average power dissipation in the software execution is similar to that in the accelerator execution, lower latency results in lower energy.

LATENCY per RUN			
	n	100,000	256M
CPU SW	latency t_{SW} $[s]$	0.028	71.04
	time per element t^e_{SW} $[\mu s]$	0.280	0.277
Accelerator	t_{ASP} $[s]$	1.00	1.50
	latency$^{(*)}$ t_{acc} $[s]$	3.27	3.73
	time per element t^e_{ASP} $[\mu s]$	10.0	0.006
Speed-up t_{SW}/t_{acc}		0.008	19.0
Speed-up t^e_{SW}/t^e_{ASP}		0.028	46.2

$^{(*)}$ t_{acc} is the sum of (1), (3) and (4).

ENERGY CONSUMPTION per RUN			
	n	100,000	256M
CPU SW	E_{SW} $[J]$	0.54	1013
	P_{ave} $[W]$	13.58	14.27
Accelerator (CPU+FPGA)	E_{acc} $[J]$	43.08	70.31
	P_{ave} $[W]$	13.15	18.84
Ratio E_{SW}/E_{acc}		0.013	14.40

$P_{ave} = E_{run}/t_{run}$.

TABLE III
MCOP: LATENCY AND ENERGY CONSUMPTION FOR EXPERIMENTS WITH THE TWO DATA SETS.

VII. Conclusions and Future Work

The experimental results of Table II and Table III show that for a sufficiently large number of elements to process, the accelerator execution is faster and more energy efficient. For the TELCO benchmark, from data sets of 500,000 calls, both computation speed-up and energy savings increase almost linearly with the set size.

However, the FPGA set-up and closing time constitute a large overhead and unless this time is reduced by modifying some of the SDK functions, the software solution is preferable for smaller data sets.

The main drawback is the much longer development time necessary for the accelerator. However, the design of a more advanced front-end processor can drastically reduce this time. The FEP should seamlessly interface the accelerator system with any ASP to realize a design-and-plug ASP paradigm.

We are currently working on the optimization of the FPGA set-up and on the design of such front-end processor.

References

[1] Top 500 Supercomputer Sites. Top 500 List. [Online]. Available: http://www.top500.org/lists/

[2] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. A. Vissers, "A Low-Latency Library in FPGA Hardware for High-Frequency Trading (HFT)," in *Hot Interconnects'12*, 2012, pp. 9–16.

[3] M. F. Cowlishaw, "Decimal floating-point: algorism for computers," in *Proc. of 16th Symposium on Computer Arithmetic*, June 2003, pp. 104–111.

[4] IBM Corporation. "The "telco" benchmark". [Online]. Available: http://speleotrove.com/decimal/telco.html

[5] M. Cornea, C. Anderson, J. Harrison, P. T. P. Tang, E. Schneider, and C. Tsen, "A Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic Using the Binary Encoding Format," in *Proc. of the 18th Symposium on Computer Arithmetic*, July 2007.

[6] J. C. Hull, *Options, Futures and other Derivatives*, 8th ed. Prentice Hall, 2012.

[7] P. Albicocco, D. Papini, and A. Nannarelli. "Direct Measurement of Power Dissipated by Monte Carlo Simulations on CPU and FPGA Platforms". IMM Technical Report 2012-18. [Online]. Available: http://orbit.dtu.dk/

[8] N. Borup, J. Dindrop, and A. Nannarelli, "FPGA Implementation of Decimal Processors for Hardware Acceleration," in *Proc. of the 29th NORCHIP Conference*, Nov. 2011.

[9] J. S. Hegner, J. Sindholt, and A. Nannarelli, "Design of Power Efficient FPGA based Hardware Accelerators for Financial Applications," in *Proc. of the 30th NORCHIP Conference*, Nov. 2012.