Power Profiling and Optimization for Heterogeneous Multi-Core Systems

Kuen Hung Tsoi and Wayne Luk
Department of Computing, Imperial College London
{khtsoi, wl}@doc.ic.ac.uk

ABSTRACT

Processing speed and energy efficiency are two of the most critical issues for computer systems. This paper presents a systematic approach for profiling the power and performance characteristics of application targeting heterogeneous multi-core computing platforms. Our approach enables rapid and automated design space exploration involving optimisation of workload distribution for systems with accelerators such as FPGAs and GPUs. We demonstrate that, with minor modification to the design, it is possible to estimate performance and power efficiency trade off to identify optimized workload distribution. Our approach shows that, for N-body computation, the fastest design which involves 2 CPU cores, 10 FPGA cores and 40960 GPU threads, is 2 times faster than a design with only FPGAs while achieving better overall energy efficiency.

1. INTRODUCTION

Utilizing multiple cores to perform computation in parallel is one of the most promising ways to improve performance in advanced computer systems. Besides multi-core processors, it is also possible to achieve this parallelism through customized cores in reconfigurable logic or massively parallel processors in a GPU (graphics processing unit). While computation time decreases with increasing parallelism, power consumption is now a critical consideration for computer systems.

In an ideal homogeneous architecture, processing capability and power consumption are evenly distributed among its cores. Thus the decrease of computation time and the increase of system power are linear with the number of active cores. So the total energy cost (the Power-Time product) of solving a particular problem is constant. However, these assumptions are not valid. First, performance improvement suffers from the law of diminishing return such that speedup may not scale linearly with the number of cores. Second, not all energy consumed by the system contributes to useful computation. Power consumption overhead is not linear with the number of active cores. Third, the achievable performance is limited by architectural factors such as I/O bandwidth and cache size. Conflicts and arbitration of critical resources degrade performance. The situation becomes more complicated when different types of multi-core processing units are involved. It is important to optimize the energy and power efficiency for such platforms.

In this work, we consider a single machine containing heterogeneous multi-core processing units: multi-core CPUs, custom cores in FPGAs (Field Programmable Gate Array), and massively parallel GPUs. CPUs are the most common multi-core computing device. To increase the parallelism of an application, FPGAs and GPUs are usually used as hardware accelerators. These accelerators are often connected to the CPU through the host system bus. Our aim is to use automated power profiling techniques to find the most energy efficient design (measured by the number of operations per joule) for an application, and compare that with the design with the highest performance.

There are studies in power modelling for FPGAs and GPUs and various models have been proposed [1–6]. Some of these studies provide analytical treatment of device power behaviour. In this work, we adopt a complementary approach based on profiling with actual performance and power consumption measurement, the results from which are used in calibrating simplified analytical models. Our approach reveals, for the first time, the impact of parallelism in different types of processors on speed, power consumption and energy efficiency. The novel aspects of our approach include:

- A technique for profiling performance and power characteristics of various multi-core accelerators. The approach can be applied to applications for identifying trade-off between performance, power consumption and energy efficiency of systems based on heterogeneous accelerators.
- Facilities to measure, control and optimize power efficiency for heterogeneous accelerators automatically. Current supported hardware devices include multi-core CPUs, FPGA accelerator boards and GPU platforms.
- An application-specific model for estimating the achievable energy efficiency. The model makes use of profiling results to estimate the performance of different multi-core processors. Two applications targeting CPU, FPGA and GPU illustrate the effectiveness of our approach.

The rest of the paper is organized as follows. Section 2 reviews previous work on power, performance and energy modelling of hardware accelerators. Section 3 presents a systematic approach to profile power efficiency on various hardware accelerators. Section 4 presents a power efficiency model and an optimization process for systems with heterogeneous processing units. Section 5 applies the profiling and optimization approach on an N-Body application and an Asian option pricing application for a system with CPU, FPGA and GPU. Section 6 summaries the achievements of this work.

2. BACKGROUND

In 2008, a power model [1] was proposed to predict and reduce runtime power consumption by considering the voltage, frequency and parallelism parameters as well as the performance requirement for an application. Runtime results are collected by instruments to provide data for an empirical model which adapts voltage and concurrency parameters. In 2009, a fine-grained power management scheme was proposed by moving threads between cores with heterogeneous power-performance ratios [2]. The overhead of changing thread location is significantly less than that of scaling the voltage or frequency of the core on which the thread is currently running. These methods require precision timing instrument and training for the model.

Power consumption issues also attract much attention in the FPGA community. In 2005, a multi-level power modelling framework was proposed for the VPR tool [3]. The framework models both dynamic and static power characteristics of the connections, lookup table clusters and transitional glitches. In 2010, a method was proposed to optimize the degree of parallelism based on effects of execution energy and reconfiguration energy in an FPGA [4]. This work achieved significant energy reduction in filter designs by runtime reconfiguration of coefficient-optimized modules.

As GPUs are becoming mature platforms for general purpose computing, interest in their power and energy efficiency is rising. In 2008, an early study [5] measures the real-time energy behaviour of GPU and locates the break-even point of the power-performance trade off of an example in the CUDA SDK. In 2009, power measurement of the major components inside a GPU platform including the processing cores and the memory hierarchy was reported [7]. In the same year, more comprehensive experiments are performed [6]. Using CUBLAS and ATLAS as the workload in the experiment, real-time measured values are identified by inserting special workload as time markers.

Our approach does not require precision timing instruments or special instrumentation. It is applicable to a wide range of applications.

3. PROFILING APPROACH

3.1 Parameters and Measurement

Our experiments measure computation time and power consumption. These measurements provide the basis for power-performance trade off and optimizations. A program for a specific algorithm often needs to perform tasks which are not made explicit in the algorithm. These tasks, such as data formatting and memory management, are considered as overhead. The total runtime of a program is the sum of the time overhead (T_o) and the computation time (T_c), $T_{total} = T_o + T_c$. In this work, we focus on T_c which is defined as the time spent on the algorithm kernel. In our profiling method, the accelerators in the system are considered as power consuming atomic modules. The characteristics of the specific algorithm and implementation are captured in the measurements. As we consider the total power consumption of the system, its idle power consumption is also covered in this work.

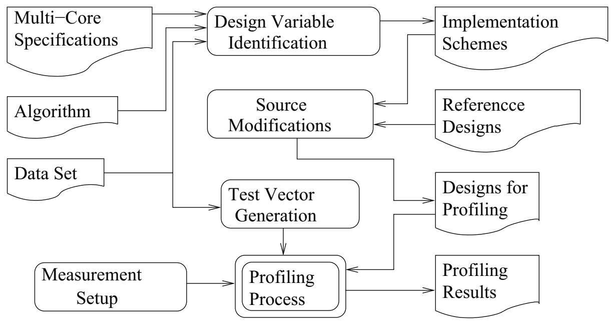

Figure 1: A profiling flow for assessing power-performance trade off in multi-core platforms.

Table 1: Common Variables

Name	**Platforms**	**Meaning**
application-specific		
loop count	N/A	kernel loop iterations
bytes/core	N/A	data for one core
data size	N/A	total data size
bytes/flop	N/A	data:operations ratio
platform-specific		
active cores (nc)	all	active cores assigned
threads (nt)	CPU/GPU	threads created
thread blocks (ntb)	GPU	# of thread blocks
threads/block (tpb)	GPU	# of threads per block
frequency (cf)	FPGA	core frequency

3.2 Profiling Flow

After identifying the properties to be measured, the design techniques and the profiling procedures are presented here. Each multi-core platform has specific features which should be captured in different ways. Figure 1 shows the steps in this profiling flow.

In the profiling flow, the first step is to extract representative design variables which can be varied to explore different implementation schemes. There are two types of variables: ones depending on the application and the data set, and ones depending on the type of multi-core computing platform. Table 1 presents some common variables used in the profiling flow. This is not a complete list and not all of them are applicable every time. Application developers need to verify these application-specific and platform-specific variables.

It is important to profile the implementations using proper data set for accurate results. This is achieved by using sample data sets which target full application execution. The test vector generation process can be embedded in the modified source code such that it can provide data segments in suitable size according to the profiling variables. Since our measurements are in the scale of seconds, the experiments may need to repeat the kernels for a prolonged execution time. The power consumption of the complete system is measured and recorded by monitoring software connected to external equipment, which is detailed in Section 4.

The profiling process is performed by running the modified designs in different implementation schemes to capture the impact of energy efficiency due to the changes in design variables. This can be automated by exploiting the embedded features of the modified implementation, and by appropriate scripting programs.

First, the original application is modified using techniques discussed in Section 3.3 for each type of multi-core devices. The modified CPU and GPU programs have parametrised workload distribution and synchronization code segments. So an implementation can be generated by altering the parameters and recompiling the source code. Due to the long synthesis time, the FPGA program adopts a unified interface supporting parameter alteration at run time without recompiling. All of these programs will also monitor and output the execution time for further analysis.

Second, developers can select a set of discrete variable values to reduce the effort in exploring the design space. These values and their corresponding positions in the program are captured in a configuration file. A profiler written in Perl is used to help automating the evaluation tasks. The profiler reads in the configuration file and performs the necessary changes to generate the required executables. These executables are then launched and monitored. At the same time, the external power monitoring program records the power consumption behaviour during each execution.

Finally, the profiling results are stored in a table used to estimate the power-performance trade-off by methods such as piecewise linear approximation. An power estimator with interpolation function is created for processing profiled data and user input constraints.

3.3 Device Specific Profiling Techniques

We use OpenMP to parallelise a design for a multi-core CPU platform. The main loop is partitioned so that each thread has an identical kernel. The workload is evenly distributed among threads. To enable design space exploration, the reference design is modified to support creating different number of threads by varying the per-thread workload distribution in each iteration.

Assigning a single thread to each core may not fully utilize the silicon resource due to memory access latency of the cores and data dependency of the kernel. On the other hand, performance does not scale linearly with the number of threads due to factors such as cache efficiency, internal pipelining of the cores and thread management overhead. The profiling experiment starts with a single thread and increases the number of threads until saturation.

Figure 2 shows a common architecture for profiling multi-core FPGA design. Unlike multi-threaded CPU implementation, more complicated mechanisms are used to facilitate the profiling functions for FPGA. First, communication to external devices such as host CPU and memory modules is usually fixed with a predefined speed. If the operating frequency of the cores is changed in each experiment, synchronizing logic is required to enable correct communication across different clock domains. For this reason, asynchronous FIFOs between customized cores and external interface are inserted. The clock source to the cores under test is generated by

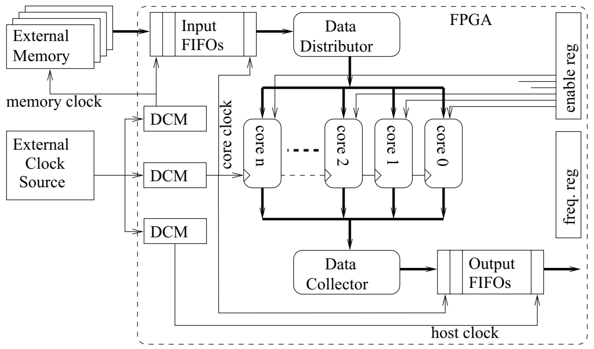

Figure 2: Common architecture of profiling a multi-core FPGA design.

a digital clock manager (DCM). Host software is able to configure the frequency of this DCM at runtime before each experiment.

Second, it is difficult to remove a core from a configured FPGA device without resynthesizing the design. Thus the profiling version of FPGA design controls the number of active cores using independent enable signals. The host program sets the enable vector at the beginning of each experiment. The major modification is to generalize the workload distribution circuit.

We represent and explore the GPU implementation schemes in a 2D space. The first dimension is the number of threads per block (*tpb*) and the second dimension is the number of thread blocks (*ntb*). GPU architecture usually limits the maximum value of *tpb* while the *ntb* is usually defined by the application developer. These two variables determine the utilization of the GPU hardware (mostly FPU cores) and thus the power consumption and performance of the specific application.

To enable the profiling functions in a GPU implementation, The *tpb* and *ntb* values are parametrised by command line operation. The kernel is modified to include a loop in which the iteration count depends on the *tpb* and *ntb* values. The amount of data processed by the kernel adapts to the implementation schemes.

4. MODELLING AND OPTIMISATION

Continuous power monitoring facilities are installed for the experiments. A FLUKE i30 current clamp is used to measure the (host) input AC current, I_h. This current clamp has $S = 100mV/A$ output sensitivity in $\pm 1mA$ resolution. The output of the clamp, V_h, is measured, in mV scale, by a Maplin N56FU digital multimeter (DMM). The readings on the DMM is collected and recorded by the host computer through USB interface. The open source QtDMM software is used as the retrieving and logging tool. The logger is set to collect readings in a sample period of $s_p = 0.1$ seconds and write them to a file with time stamp. Since $V_h = S \times I_h$, the actual power, P_h, is automatically calculated by the framework using the following equation where $V_p = 220$ Volts is the main power voltage:

$$P_h = I_h \times V_p = V_h/S \times V_p \tag{1}$$

The computation time of the application under profiling is measured by recording time stamps before and after the kernel execution. The `gettimeofday()` function is used to output the time stamps and compute the difference between them. These time stamps are also used in computing the energy consumption of the kernel. After aligning them with those inside the QtDMM log file, a user can extract the encapsulated readings and integrate them to find the total energy consumed during the kernel computation period as:

$$E_h = s_p \times \sum P_h[i] = ((s_p \times V_p)/S) \times \sum V_h[i]. \quad (2)$$

Here $P_h[0], P_h[1], ...$ correspond to the system power consumption in each sample. This method measures the total energy consumption of the host system including all accelerators. To find the active energy consumption of a specific kernel, the idle system power is subtracted from the measured reading. In our testbed, the idle system power is around 160 Watts.

We use a profiling based application-specific model to capture system behaviour. For example, many applications contain kernels of loop structure with a fixed number of iterations and a fixed number of operations inside the loop body. Even in this simple form, complex behaviour may be experienced by the system due to the execution environment and the platform architecture. Such complexity is difficult to be captured in pure mathematical model while our profiling process can provide useful information in practice.

In our current implementation, the profiling results are stored in tabular format with the parameter values and the measured results in a row. The performance values are recorded as kernel computation time in seconds. The power consumption results are recorded as the average power reading in Watts. The energy result is computed as in Equation 2.

Linear interpolation is used in characterising performance and power consumption of implementation schemes based on the profiling functions for each multi-core architecture. The modeling functions for the CPU, FPGA and GPU are in the form of: $*_c(nt)$, $*_f(nc, cf)$ and $*_g(tpb, ntb)$. The $*$ symbol can either be P for power, E for energy or T for kernel computing time. Table 1 shows a summary of the parameters. Additional checks are required in the interpolation to avoid infeasible implementation schemes.

The profiling and modeling techniques are for applications running on a single type of multi-core architecture. For systems with multiple types of multi-core processors, it is useful to allow heterogeneous processors to work collaboratively on a single application. We extend our approach below to cover a system with multiple accelerators.

To capture energy efficiency, we define *MFOpJ* as *M*illions *F*loating-point *O*perations *p*er *J*oule. For a specific implementation $\mathcal{I}$, it is calculated as:

$$MFOpJ(\mathcal{I}) = (O(\mathcal{I}) \times 10^{-6})/E(\mathcal{I}). \quad (3)$$

Here $O(\mathcal{I})$ denotes the number of floating-point (*fp*) operations required by $\mathcal{I}$, and $E(\mathcal{I})$ is the energy consumed during the execution of $\mathcal{I}$. We focus on floating-point operations since they are usually the performance bottleneck and the power hungry component in the kernel. The total number of *fp* operations of a kernel, $\mathcal{K}$, is computed as

$$O(\mathcal{I}) = l_{count}(\mathcal{K}) \times l_{size}(\mathcal{K}), \quad (4)$$

where l_{count} is the total number of iterations of the main-loop (i.e. the number of times the computation kernel is called) and l_{size} is the total number of *fp* operations inside the kernel. By substituting Equation 2 and 4, we can expand Equation 3 to:

$$MFOpJ(\mathcal{I}) = \frac{l_{count}(\mathcal{K}) \times l_{size}(\mathcal{K}) \times 10^{-6}}{((s_p \times V_p)/S) \times \sum V_h[i]}. \quad (5)$$

The unified *MFOpJ* unit models the behaviour of a specific application running on multiple types of multi-core platforms. It allows evaluating trade-off between different implementation schemes on the same hardware, and between different hardware architectures.

Utilizing the *MFOpJ* model, several steps are required to achieve optimised energy-performance trade off.

1 The software reference design is analyzed to extract the l_{count} and l_{size} parameters of the kernel.
2 Representative configurations (with the nt, nc and cf parameters) are implemented.
3 The kernel power and execution time are measured and used to obtain the *MFOpJ* merit as shown in Equation 5.
4 The *MFOpJ* values of other design configurations are obtained by an interpolation program.
5 The heterogeneous multi-accelerator efficiency is modeled by eliminating duplicate system energy.
6 The *MFOpJ* is estimated for design spece exploration and energy-performance optimisation.

5. CASE STUDIES AND RESULTS

5.1 Implementation Platform

The host platform is an HP z400 workstation with an Intel Xeon W3505 CPU at 2.53GHz. The Alpha-Data ADM-XRC-5T2 acceleration card has one Xilinx Virtex-5 XC5LX-330T FPGA device and is attached to the host system through a 8x PCIe interface. The nVidia Tesla C1060 accelerator has a GTX280 GPU chip at 1.3GHz and is interfaced to the host through a 16x PCIe slot. Since it is difficult to isolate the power consumption of these devices from other host peripherals, we instead measure the total power consumption at the input of the power supply unit.

The host system runs a Linux OS with 2.6.33.4 kernel. Intel C Compiler (ICC) 11.1 is used to compile the CPU based implementations which are optimized with the `-fast` flag. Thread parallelization is implemented using the OpenMP extension. The FPGA design is synthesized and implemented using Xilinx ISE Design Suit version 12.1 with highest optimization effort level enabled in all steps for maximum speed. The GPU implementation is compiled and executed using the nVidia CUDA 3.0 environment.

5.2 N-Body Simulation

Applying the steps in Section 4, the *MFOpJ* merit figures of this example are obtained. First, we identify the $l_{count}(\mathcal{K})$ and $l_{size}(\mathcal{K})$ values as shown in the next paragraph. Second, we implement multiple designs based on the accelerator specific parameters and measure the performance as shown in Figure 3. A full design space

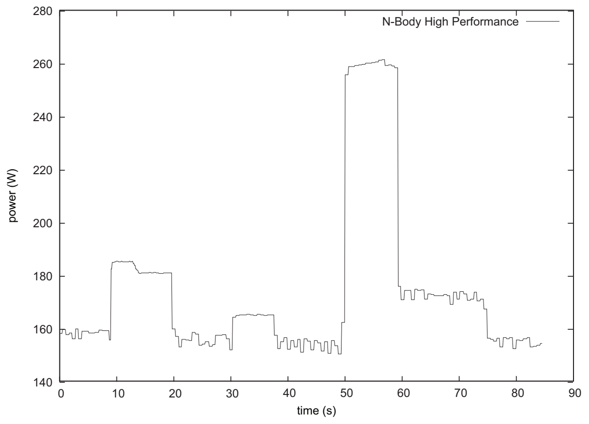

Figure 3: Power consumption of CPU, FPGA and GPU for N-Body simulation.

exploration is then performed by interpolation based on the measured results as shown in Figure 4. This merit helps us to identify the most energy efficient configuration of FPGA implementation.

The computation inside a single iteration of the kernel consists of 20 single precision floating-point operations. This kernel is iterated for N^2 times in a nested loop structure for one time step in simulation. In the experiment, we use a data set with size $N = 81920$ and partition the workload in the outer loop to minimize data dependency and communication between separate cores. So $O(nbody) = N^2 \times 20 = 1.34 \times 10^{11}$. The FPGA implementation achieves 320MHz for a single core and 200MHz for a 10-core design. The maximum number of cores in FPGA is mainly limited by the available DSP blocks used in the floating-point operators. The decrease in frequency is due to the longer connection in both control and data path.

Figure 3 shows the power behaviour of the three types of multi-core devices when running the N-Body application. In this experiment, the same data set is used to test the CPU, FPGA and GPU in three separate runs. The three pulses in Figure 3 shows these activities in the same order. The area under each pulse is the energy when the specific device is used. The GPU device continues (for 15 seconds) to consume power after the application is terminated. This is the device management overhead of the GPU.

We first measure, in *MFOpJ*, the results of the CPU based implementation when the number of threads increases. The results show that, using Intel compiler, we achieve 18 *MFOpJ* maximum efficiency and the efficiency improvement stops after the number of threads goes beyond 2, which is expected since our CPU has two physical cores.

Figure 4 shows an interpolated surface for the *MFOpJ* profiling results of the FPGA implementation. Number of cores and core frequency are variables in the experiments. From the results, we can see that the design with 10 cores at 200MHz is the most energy efficient scheme for FPGA. The performance is relatively static with the frequency changing, while it is more linear with respect to the number of cores. This is due to the fact that the possible range of frequency variation is only $(320 - 200)/200 = 60\%$ while there are 10 times difference in the core number. Although the surface in Figure 4 extends to all implementation schemes in the full range of both variables, the practical interpolation function will identify the infeasible schemes and returns invalid *MFOpJ* values for them.

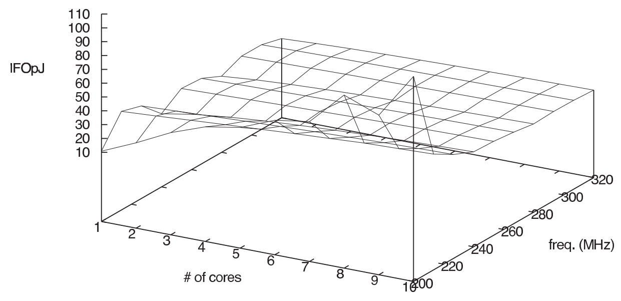

Figure 4: FPGA based N-Body simulation results (interpolated).

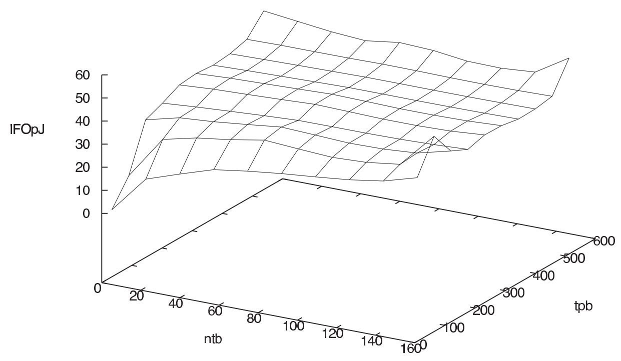

Figure 5: GPU based N-Body simulation results (interpolated).

Figure 5 shows the *MFOpJ* results of GPU designs obtained in a similar way as in the FPGA experiments. The variables are the number of thread blocks (ntb) and number of threads per block (tpb). From the results, we observed that power consumption and performance are relatively static when $ntb \geq 4$ and $tpb \geq 128$. The *MFOpJ* performance drop significantly when the variable values go below these threshold.

5.3 Asian Option Pricing

The kernel of the Monte Carlo (MC) Asian Option simulation is to evaluate the pricing values along a given path. The MC simulation size is set to $N = 10^6$ and each path contains $step = 3650$ intermediate points. 14 floating-point operations are required to calculate one data point and thus $O(asianopt) = N \times steps \times 14 = 5.11 \times 10^{10}$. The paths are then partitioned and distributed to different cores to achieve parallelism. In this application, random numbers are generated on the same core where they are processed and the outputs are accumulated from the cores. So the effects of memory access and data communication are negligible. Currently, operations in fixed-point arithmetic in the random number generation function are not covered in this experiment since we focus on floating-point operations.

Figure 6 illustrates the power and performance characteristics of three performance optimized implementations running on the CPU, FPGA and GPU hardware. To profile this application, the same set of variables

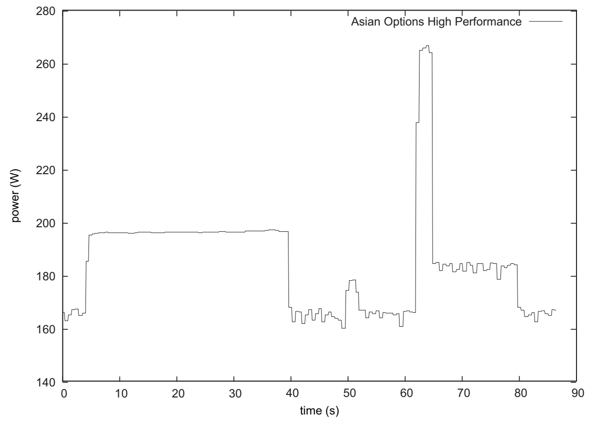

Figure 6: Power consumption results for different types of multi-core processors for Asian option pricing.

Table 2: Results Summary

Designs	**Schemes**	**Power** (W)	**Time** (s)	***MFOpJ***
N-Body simulation				
CPU	$nt \geq 2$	196.7	36.6	18.6
FPGA	$cf = 200, nc = 10$	167.6	7.4	108.2
GPU	$ntb \geq 60, tpb \geq 128$	264	9.26	54.9
Hybrid 1	-	301	3.9	114.3
MC Asian option pricing				
CPU	$nt \geq 2$	196	35.3	7.4
FPGA	$cf = 200, nc = 10$	177	1.83	157.8
GPU	$ntb \geq 60, tpb \geq 128$	266	2.57	74.7
Hybrid 2	-	277	1.1	167.7

are used as in our N-Body simulation. The number of FPGA cores ranges from 1 to 10, operating from 200MHz to 320MHz. As in the N-Body example, results are obtained for different implementation schemes by changing the design variables. The results show that this design can be scaled more linearly than the N-Body simulation due to the absence of memory and communication overhead.

5.4 Summary of Results

The experimental results of the two applications are summarized in Table 2. The first three designs of each application are optimized for performance. In the hybrid designs, the workload is partitioned to different processors according to their performance-optimized results. The objective is to allow the processors to finish the assigned workload such that the total computing time is minimized. In the N-Body application, the hybrid design includes both CPU, FPGA and GPU implementations in the same configuration as shown in the table. The workload distribution is 10% to CPU, 50% to FPGA and 40% to GPU. In the Asian option pricing application, since the CPU performance is so slow that only FPGA and GPU are used: 60% to the FPGA and 40% to the GPU.

Although these hybrid designs are created for performance, they also achieve the best *MFOpJ* scores. This is because of the large idle power consumption (around 160W) of the host system. When this idle power becomes the dominant factor, shorter runtime will consume less idle energy and thus less overall energy.

The combined energy efficiency cannot be obtained by summing the *MFOpJ* values of all separate multi-core processors, since the idle components will be repeated. We can instead use the active power component *MFOpJ'*, which is different from those shown in Table 2. For example, the FPGA and GPU N-Body designs score 2387 and 139 in terms of *MFOpJ'*. In this case, the most efficient design in terms of dynamic energy would be using the FPGA alone.

6. CONCLUSION

This paper presents an approach of optimizing designs for energy efficiency and computing performance based on three different multi-core architectures: CPU, FPGA and GPU. Our approach includes the measurement facilities, the implementation profiling flow, and the combined energy-performance model and optimisation process. This approach has been applied to several applications, two of which are included in this paper. Future work includes supporting a wider range of applications and FPGA run-time reconfiguration.

Acknowledgement The support of Imperial College London Research Excellence Award, UK Engineering and Physical Sciences Research Council, Alpha Data, nVidia and Xilinx is gratefully acknowledged. The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement number 248976 and 257906.

7. REFERENCES

[1] M. Curtis-Maury *et al.*, "Prediction models for multi-dimensional power-performance optimization on many cores," in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, 2008, pp. 250–259.

[2] K. K. Rangan *et al.*, "Thread motion: fine-grained power management for multi-core systems," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 302–313, 2009.

[3] F. Li *et al.*, "Power modeling and characteristics of field programmable gate arrays," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 11, pp. 1712–1724, Nov. 2005.

[4] T. Becker, W. Luk, and P. Y. K. Cheung, "Energy-aware optimisation for run-time reconfiguration," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 55–62.

[5] M. Rofouei *et al.*, "Energy-aware high performance computing with graphic processing units," in *Proc. Workshop on Power Aware Computing and Systems*, 2008.

[6] R. Suda and D. Q. Ren, "Accurate measurements and precise modeling of power dissipation of CUDA kernels toward power optimized high performance CPU-GPU computing," in *Proc. Int. Conf. on Parallel and Distributed Comput. App. and Tech.*, 2009, pp. 432–438.

[7] S. Collange *et al.*, "Power consumption of GPUs from a software perspective," in *Proc. Int. Conf. on Computational Science*, 2009, pp. 914–923.