Heterogeneous Systems for Energy Efficient Scientific Computing

Qiang Liu[1] and Wayne Luk[2]

[1] School of Electronic Information Engineering, Tianjin University, 300072 Tianjin, China
[2] Department of Computing, Imperial College London, SW7 2AZ London, UK

Abstract. This paper introduces a novel approach for exploring heterogeneous computing engines which include GPUs and FPGAs as accelerators. Our goal is to systematically automate finding solutions for such engines that maximize energy efficiency while meeting requirements in throughput and in resource constraints. The proposed approach, based on a linear programming model, enables optimization of system throughput and energy efficiency, and analysis of energy efficiency sensitivity and power consumption issues. It can be used in evaluating current and future computing hardware and interfaces to identify appropriate combinations. A heterogeneous system containing a CPU, a GPU and an FPGA with a PCI Express interface is studied based on the High Performance Linpack application. Results indicate that such a heterogeneous computing system is able to provide energy-efficient solutions to scientific computing with various performance demands. The improvement of system energy efficiency is more sensitive to some of the system components, for example in the studied system concurrently improving the energy efficiency of the interface and the GPU by 10 times could lead to over 10 times improvement of the system energy efficiency.

1 Introduction

Scientific computing applications, such as dense linear algebra and N-body simulation, require powerful computing engines to perform huge amounts of arithmetic operations [1]. This work explores heterogeneous computing hardware for scientific computing, aiming at relieving the increased energy demand of traditional high performance computers [2] and providing computing performance in between desktops and supercomputers. With reasonable trade-off between performance and energy and affordable price, the heterogeneous computing systems can be owned and used by organizations requiring local scientific computing.

The target heterogeneous computing systems integrate CPUs, GPUs and FPGAs, built based on a widely used host-accelerator structure. We study the heterogeneous systems' throughput, energy efficiency and energy efficiency sensitivity. These will help designers to make decisions on building heterogeneous computing systems, such as selecting devices and interconnect interfaces. The

O.C.S. Choy et al. (Eds.): ARC 2012, LNCS 7199, pp. 64–75, 2012.

aim is to design a heterogeneous platform which is able to provide a high energy-efficient solution to scientific computing with various performance demands.

Several homogeneous and heterogeneous systems have been developed for scientific applications. Ding *et al.* [3] study energy efficiency and scaling efficiency issues when many low power processors (PowerPC440) are connected. The impact of interconnect interfaces and memory accesses together with voltage and frequency scaling (DVFS) is taken into account. Wang *et al.* [4] develop an algorithm for determining workload allocation and DVFS on a system with a CPU and a GPU for scientific computation. Turkington *et al.* [5] accelerate Linpack 1000 on a platform with a CPU and an FPGA, by implementing a time-consuming subroute of Linpack 1000 on the FPGA. Fatica [6] uses a cluster, where each node has a CPU and a GPU, to speed up High Performance Linpack. A linear programming model is used in [7] to distribute workload between a CPU and a GPU, leading to accelerated FFT implementation. Tse *et al.* [8] propose a framework for accelerating financial applications on a cluster with CPUs, GPUs and FPGAs. A parallel programming approach combining OpenMP, OpenCL and C++ is proposed in [9] to facilitate the management of CPU–GPU clusters.

This paper explores adding recent GPUs and FPGAs into traditional high performance computing systems for improving system energy efficiency. A linear programming (LP) model [10] is used for computational workload allocation in such systems. This paper builds on this model and provides, for the first time, two novel results: (a) a design exploration flow for energy efficient scientific computing, and (b) detailed analysis about energy efficiency of various heterogeneous systems including FPGAs and GPUs, the sensitivity of system energy efficiency to individual system components, and energy efficiency versus power consumption.

The main contributions of this work are:

- A novel approach based on a linear programming model for exploring heterogeneous computing engines which include GPUs and FPGAs as accelerators, helping designers to find the right combinations of various computing devices and interconnections;
- Analysis of energy efficiency sensitivity and the derivative of energy efficiency with respect to power consumption, finding the system bottleneck and being aware of system power consumption when trying to scale systems; and
- Evaluation of the proposed approach on a heterogeneous system with a CPU, a GPU and an FPGA, showing that the heterogeneous computing system provides an energy-efficient solution to High Performance Linpack (HPL) and the system is more sensitive to some of the system components, *e.g.* by estimation concurrently improving the energy efficiency of the interface and the GPU by 10 times could lead to over 10 times improvement of the system energy efficiency.

The rest of this paper is organized as follows. A linear programming model for workload allocation is present in Section 2. The design exploration methodology

is proposed in Section 3. The evaluation setup of the proposed approach is described in Section 4 and results are shown in Section 5. Section 6 concludes the paper with future work.

2 Workload Allocation Formulation

In this section, we will briefly present an LP model, which was used in [10] to study different workload allocation problems with regards to throughput, energy efficiency and temperature. We adopt this model to facilitate our exploration of heterogeneous systems.

Given a heterogeneous system containing H hardware computing devices, the throughput and run-time power consumption of device i are R_{di} and P_{di}, respectively, and the throughput and run-time power consumption of the communication channel between devices i and j are R_{cij} and P_{cij}, respectively. x_i is a percentage of computation workload W assigned to device i.

The execution time t_{di} of device i for performing x_i of W is

$$t_{di} = \frac{x_i \times W}{R_{di}}, 1 \leq i \leq H. \tag{1}$$

It is assumed that the host is device 1 and all data reside at the host memories. t_{d1} is the execution time of the host for computation workload x_1.

When x_i workload is allocated to device i, the host will send data to device i and receive resultant data back from it upon finishing. The amount of data involved in these transfers is $D(x_i)$, usually a linear function because the larger workload is associated with the more data transfers. As a result, the time spent on data transfers between the host and device i is

$$t_{c1i} = \frac{D(x_i)}{R_{c1i}}, 2 \leq i \leq H. \tag{2}$$

Eq.(3) shows the execution time of the heterogeneous computing system, which include computation time and data transfer time and should not be larger than the requirement execution time T.

$$t = \max(t_{d1}, \max_{i=2,H}\{t_{di} + t_{c1i}\}) \leq T \tag{3}$$

The system energy consumption e is

$$e = \sum_{i=1,H} (t_{di} \times P_{di}) + \sum_{i=2,H} (t_{c1i} \times P_{c1i}) \tag{4}$$

The first sum is the run-time energy consumption of all devices, including static and dynamic power. The second one is the energy consumed on data transfers between the host and accelerators.

The variables in this workload allocation formulation are $x_i \in [0, 1]$ $(1 \leq i \leq H)$, and are added to one, $\sum_{i=1}^{H} x_i = 1$.

The main purpose of [10] is to find the right workload allocation x_i, such that the system execution time t or the system energy consumption e is minimized, while meeting user constraints. This paper has a different focus: to exploit the insights derived from the linear programming model in exploring and optimizing system architectures, given various devices and interfaces with different throughput and power consumption parameters. In particular, based on formulae for t and e, we propose a design exploration approach for heterogeneous systems in the next section.

3 Exploration Approach

This section introduces a design exploration flow for heterogeneous systems. Let us first describe system metrics for our approach.

3.1 System Metrics

A *workload* is a set of arithmetical calculations, particularly floating point operations (FLOP).

A widely used metric for computer performance is *throughput*, which is

$$\text{throughput} = \frac{\text{number of floating point operations (FLOP)}}{\text{execution time}},$$

measured in FLOPS (Floating Point Operations Per Second).

Energy efficiency is defined below:

$$\text{energy efficiency} = \frac{\text{throughput}}{\text{power}} = \frac{\text{FLOP}}{\text{execution time} \times \text{power}} = \frac{\text{FLOP}}{\text{energy}},$$

which is measured in FLOPS/Watt or FLOP/Joule. This metric evaluates the computation rate per unit power of a computing engine. Obviously, the higher the energy efficiency is, the more promising a computing engine is. However, energy efficiency may hide one fact that a high energy efficiency computing engine has high power consumption, leading to high device temperature and thus degraded reliability.

Therefore, we study another metric in which power is paid more attention when improving system energy efficiency. The metric is the derivative of energy efficiency with respect to power (DEEP), defined as below:

$$\text{DEEP} = \frac{\Delta\text{energy efficiency}}{\Delta\text{power}}.$$

The unit of measurement is FLOPS/Watt^2. It is desirable to increase energy efficiency, but without increasing power consumption sharply.

In addition, we investigate how significant the effect of improving energy efficiency of each individual device and interface is on the heterogeneous system energy efficiency, *i.e.* sensitivity analysis. This is desirable due to the fact that

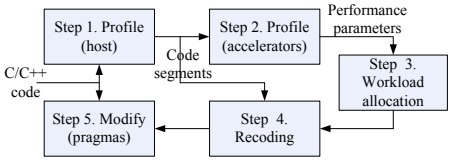

Fig. 1. Design exploration flow

```
void main()
{
    ...
    #pragma omp parallel default(shared)
    {
        #pragma omp sections
        {
            #pragma omp section
                sgemm_cpu(K, M1, N, alpha, beta, A, B, C);
            #pragma omp section
                sgemm_gpu(K, M2, N, alpha, beta, A, B, C);
            #pragma omp section
                sgemm_fpga(K, M3, N, alpha, beta, A, B, C);
        }
    }
    ...
}
```

Fig. 2. The main program performing SGEMM on the three devices

improving energy efficiency of a heterogeneous system is possibly blocked by some of the system components, such as the interface bandwidth. By identifying the bottleneck, the system energy efficiency can be further improved, or the other parts of the system could be downgraded to reduce costs.

3.2 Design Exploration Flow

The design exploration flow of the heterogeneous systems is shown in Fig. 1 and summarized as follows.

Step 1. Profile the original program of a scientific computing application in C/C++, by means of tools such as *gprof* on the host, to identify the time-consuming segments (workload W) of the program. Usually these segments contain loop nested arithmetic operations.

Step 2. Profile each of the time-consuming segments on different accelerator devices to obtain parameters, such as R_{di}, P_{di}, R_{cij} and P_{cij}. There are two methods to perform this profiling: a) using mathematical formulations of throughput and power consumption of each device and interconnection; and b) executing the code segments on each device to measure these parameters, respectively. From the designers' point of view, the former is more promising, because rewriting and executing the codes on devices which may not be used in the final system is avoided. However, it is not always easy to precisely formulate the performance metrics of different devices. Speed and power consumption of FPGA-based systems with various optimizations have been formulated in [11] and an analytical GPU performance model can be seen in [12]. In this paper, we focus on system exploration, and thus use the second method.

Step 3. Allocate workloads over different devices. This step can be performed at compile time if the workloads are *a priori* known and can be performed at run-time otherwise. We use the LP model described in the previous section.

Step 4. Rewrite those code segments, which will be executed on accelerator devices, considering various necessary optimizations based on hardware device properties. For example, kernel functions are written as regards register and

shared memory sizes for GPUs. A code transformation approach from C to a C-like hardware description for FPGA has been proposed previously in [13].

Step 5. Modify the original program, including adding codes for data transfers between the host and accelerators and inserting OpenMP *pragmas* at where the time-consuming code segments originally execute to invoke the parallel executions of them on multiple devices, following the workload allocation determined at Step 3. An example is shown in Fig. 2.

In our future work, the above flow will be gradually automated. In the scope of this paper, as regards the purpose of exploring system designs, we follow this flow to carry out experiments manually with a typical benchmark of dense linear algebra applications in the next section.

4 Experiment Setup

Benchmark. In this work, High Performance Linpack (HPL) [14] is used as a benchmark to evaluate the proposed design exploration approach. Linpack solves a dense system of linear equations and is widely used to evaluate high performance computers. In this paper, we look at the function SGEMM in HPL, which performs the following computation:

$$C = \alpha AB + \beta C,$$

where $A \in \mathbb{R}^{M \times K}$, $B \in \mathbb{R}^{K \times N}$, $C \in \mathbb{R}^{M \times N}$ and $\alpha, \beta \in \mathbb{R}$. All data are in the single-precision floating point format. The number of float point operations (FLOP) in SGEMM is $MN+2MKN+MK$ ($O(N^3)$). This function is the most time-consuming part in HPL [6].

In our experiments, we partition matrices A and C horizontally, $C_j = \alpha A_j B + \beta C_j$, and assign different sizes of workload $M_i(N + 2KN + K)$ ($M_i = x_i M$) to different computing devices i, and on each device we also parallelize the computation of rows of $C_{M_i \times N}$ based on the number of parallel processing units available.

Here the number of data transferred $O(N^2)$ between host device 1 and accelerator device i is

$$D(x_i) = x_i MK + x'_i KN + x_i MN + x_i MN, \tag{5}$$

where x'_i is a Boolean variable which is zero if $x_i = 0$; otherwise 1.

Heterogeneous System. The heterogenous system used in our experiments contains a CPU, a GPU and an FPGA, whose parameters are shown in Table 1. The GPU card is connected to the CPU host system using PCIe, while the FPGA card (ADM-XRC-5T2 [15]) uses the PCIx interface and has a converter from PCIx to PCIe to connect to the host. The parameters of the two kinds of PCI interface, measured on the platform, are also shown in Table 1.

The CPU runs 64-bit Linux OS. With 2 threads the CPU performs SGEMM using the same code as the function `HPL_dgemm()` in Intel MKL 10.3.3, but in

the single-precision data format. For GPU, we call the function `cublasSgemm()` from the CUBLAS library in the CUDA SDK, which can automatically tune the number of threads and the number of thread blocks based on input matrix sizes.

We implement the SGEMM function using Verilog for FPGA. The FPGA card has four SDRAM banks, each at the bandwidth of 256Mb/s, and three of them are used to store the input matrices and the fourth stores the results. On the target FPGA device, 32 parallel processing units are realized to perform the SGEMM computation in parallel. Resource utilization is shown in Table 1, where the number of DSP blocks embedded on-chip and slices are the constraints that limit the number of processing units running at 100 MHz to 32. In other words, using a larger FPGA could realize more parallel processing units.

The execution mode of SGEMM on the heterogeneous platform is shown in Fig. 2, where three devices with different workloads (M_1, M_2, M_3) are triggered at the same time to perform SGEMM by using openMP directives. The data transfers between CPU and GPU and between CPU and FPGA are included in the corresponding functions.

The throughput and power results for each individual device, shown in Table 1, are measured separately, when the SGEMM function is executed on the hardware platform. The results are measured several times when all devices and interfaces run in the steady state, *i.e.* running at the highest speed for a period of time, and reported as average values. The throughput of the CPU, the GPU and the FPGA is the computation rate, without considering data I/O of the devices. The power consumption of the FPGA card is obtained by monitoring the current flowing over a current sense resistor on the card [15]. The power consumption of the GPU card and the CPU system is measured by placing a AC/DC current clamp at the corresponding power supplies, respectively. The power consumption for the CPU is in fact the power consumed on the whole PC system, when removing the FPGA and GPU cards. The power consumption of the PCI interfaces are obtained by subtracting the system idle power from the power measured when data are being transferred between CPU and FPGA or between CPU and GPU, without executing computation on any of the devices.

The energy efficiency values in Table 1 are derived from the throughput and power results. All results present in the next section are calculated based on these results, using the formulations in Section 2.

5 Experimental Results

In our experiments, we vary M to change the size of input matrices of the SGEMM function, while $K = 400$ and $N = 1000$. The results from various K and N can be obtained similarly by simply partitioning the matrices.

From Table 1, it is clear that the GPU has the highest throughput in the target system, compared to the CPU and the FPGA, because the GPU has 448 processing cores for floating point operations and the SGEMM function is well matched to the GPU system structure. We could use more powerful CPUs and larger FPGAs to reduce the performance gap. For example, by estimation, a

Table 1. Hardware platform characteristics. *These are measured based on SGEMM.

Device	Intel CPU (Xeon w3505)	nVidia GPU (Tesla c2070)	Xilinx FPGA (Virtex5-vlx330t)	PCIx	PCIe
#Cores	2	448	32 (100% DSP, 78% slices, 79% RAMs)	n/a	n/a
Clock Freq	2.53 GHz	1.15 GHz	100 MHz	n/a	n/a
Lithography	45 nm	40 nm	65 nm	n/a	n/a
Throughput*	12.83 GFLOPs	558.94 GFLOPs	13.90 GFLOPs	0.21 GBs	5.79 GBs
Power*	151.0 W	164.4 W	3.87 W	4.4 W	33 W
Energy efficiency*	0.085 GFLOPs/W	3.47 GFLOPs/W	3.66 GFLOPs/W	47.7 MBs/W	175.5 MBs/W

Virtex6-vsx475t FPGA could accommodate 133 parallel processing units, and thus the throughput of the FPGA implementation could be increased by about 4 times, respectively. However, our aim here is to exploit the strengths of different devices within a system. Specifically, in the following it is shown that in the target heterogeneous system the CPU is suitable to the scenarios where significant communication is involved, and the FPGA is more promising for the scenarios with concerns about power and temperature.

As known, in the host-accelerator systems, when communication is taken into account, accelerators' nominal performance usually suffers a discount. Two main factors need consideration. The first one, from the algorithm perspective, is computation per data transfer. In SGEMM, this factor is $O(N)$, while the factor of another function SGER from Linpack, involving $O(N^2)$ floating point operations and $O(N^2)$ data transfers, is $O(1)$. We use the LP model in Section 2 to determine the workload allocation of these two functions on the three devices to maximize throughput. Results are shown in Figs. 3 and 4, where x_c, x_g and x_f are the percentages of workloads assigned to CPU, GPU and FPGA, respectively. We can observe that most of the SGER's workload is assigned to the CPU, while the SGEMM's workload is distributed and the faster device receives more.

The second one, from the hardware perspective, is communication channel bandwidth. For SGEMM shown in Fig. 4, with the fast interface PCIe, a large portion of the workload is assigned to the fast devices, regardless of the matrix size; with the slow interface PCIx, the allocation depends on the matrix size M.

It is clearly shown that the workload allocation scheme varies, according to different scenarios. The heterogeneous system is able to provide such variety. In the rest of this section, results are calculated based on the PCIe interface which connects the host and accelerators.

In addition, as shown in Table 1, although in the target heterogenous system the GPU has the highest throughput, the FPGA has the highest energy efficiency, which is 44.9 times over CPU and 1.2 times over GPU. This illustrates the strength of the FPGA in the heterogeneous system, and the impact will be more significant if a latest large FPGA is used. For example, by estimation, a Virtex6-vsx475t FPGA could provide energy efficiency in about 14 GFLOPS/W.

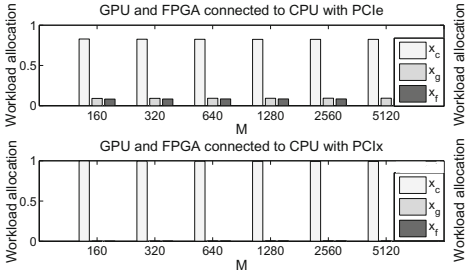

Fig. 3. SGER workload allocation

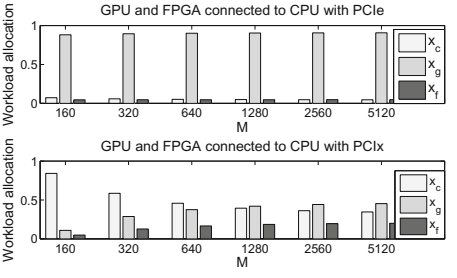

Fig. 4. SGEMM workload allocation

Table 2. Energy efficiency of heterogeneous systems for SGEMM with $M = 20480$

	CPU	CPU+FPGA	CPU+GPU	CPU+GPU+FPGA
Energy efficiency	0.06	0.12	1.08	1.11
GFLOPS/W	(1x)	(2x)	(18x)	(18.6x)

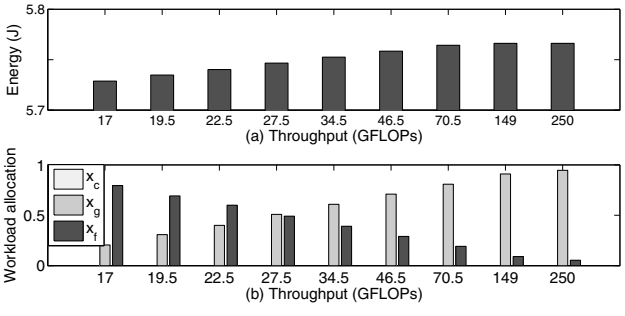

Fig. 5. SGEMM (M=20480) (a) energy consumption of the CPU+GPU+FPGA system and (b) corresponding workload allocation ($x_c = 0$), given throughput requirements

Given the different combinations of the three devices, the energy efficiency of various heterogeneous systems is shown in Table 2. The throughput and power consumption of the whole systems are measured. Note that the energy efficiency is the whole system's energy efficiency, including all kinds of overheads, such as data transfers and software protocol. Compared to the single CPU system, the heterogeneous systems improve system energy efficiency. As a point of reference, the number one in the world Green500 energy-efficient supercomputer list is 1.68 GFLOPS/W as reported in 2010 [16]. Note that the peak performance 1.11 GFLOPS/W shown in this paper is just for the function SGEMM.

Moreover, the heterogeneous system, containing CPU, GPU and FPGA, could potentially provide a high energy-efficient solution to scientific computing with various performance demands. Fig. 5 illustrates energy consumption of the heterogenous system and the corresponding workload allocation targeting at

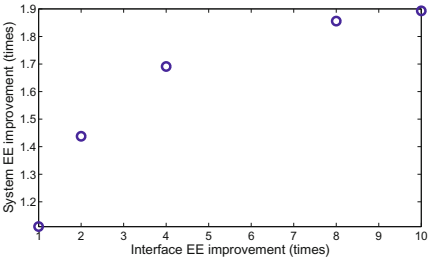

Fig. 6. System energy efficiency (EE) sensitivity to interface

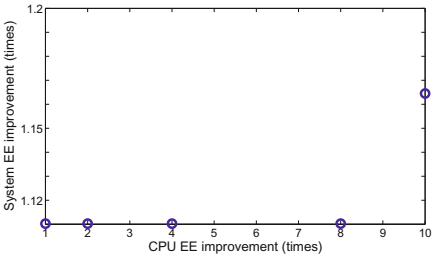

Fig. 7. System energy efficiency (EE) sensitivity to CPU

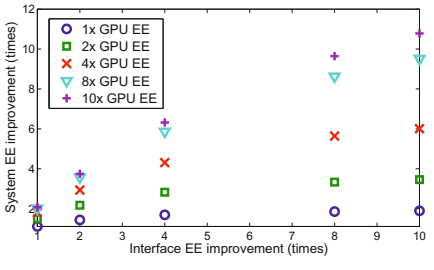

Fig. 8. System energy efficiency (EE) sensitivity to GPU

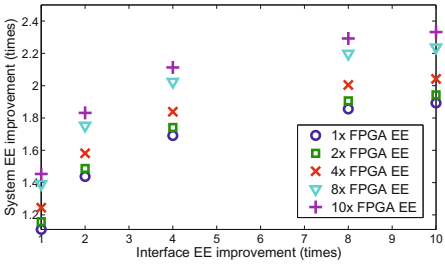

Fig. 9. System energy efficiency (EE) sensitivity to FPGA

minimizing run-time energy, given different throughput requirements for performing SGEMM. In Fig. 5 (a) we can see that the system (CPU+GPU+FPGA) has a varied energy consumption as the throughput requirement varies. The variation could be explained by the fact shown in Fig. 5 (b). The majority of the workload is assigned to the FPGA in the early stage, and then as the throughput requirement increases the workload starts to be allocated to GPU and the system energy consumption increases, and when more and more workloads are allocated to GPU the system energy consumption is dominated by the GPU.

Furthermore, we investigate how effective improving energy efficiency of each device and the interface individually is on the whole system energy efficiency. By simulation, we increase the energy efficiency of CPU, GPU, FPGA and the interface between them by 2, 4, 8, 10 times, respectively. Results are estimated and shown in Figs 6–9. For example, in Fig. 6 improving the energy efficiency of the interface alone by 10 times leads to 1.9 times improvement of the system energy efficiency. In Fig. 7, CPU's energy efficiency is increased alone and only after 8 times improvement applied the system energy efficiency starts to show very limited improvement. This is because in the target platform the CPU's energy efficiency is 40 times lower than the other two devices, as shown in Table 1. In Fig. 8, the energy efficiency of the interface and GPU is increased at the same time. It can be seen that the impact of increasing GPU's energy efficiency is more significant with the high energy efficiency interface. Similar results can be observed in Fig. 9 as well for FPGA. Overall, it is clearly shown that the energy

Table 3. Energy efficiency and power consumption of heterogeneous systems for SGEMM with $M = 20480$

	CPU+GPU+FPGA+FPGA	CPU+GPU+FPGA+GPU
Energy efficiency	1.14 GFLOPS/W	1.45 GFLOPS/W
Power	477.28 W	632.96 W

efficiency of the utilized heterogeneous computing system is more sensitive to the concurrent improvement of the interface and the GPU, 10 times improvement in the interface and the GPU leading to over 10 times improvement of the system energy efficiency. Of course, feasibility and difficulty of achieving these improvements should be taken into account. Although the results are estimated, this sensitivity analysis indicates the direction, in which system requirements could be met.

Finally, based on the previous results we experiment with adding one more GPU and one more FPGA to the current heterogeneous platform. The estimated results are reported in Table 3. It can be seen that the energy efficiency by adding a GPU is 1.27 times higher than by adding an FPGA in the target system. However, the power consumption of the system after adding a GPU is very high and the system requires a powerful power supply over 800 W. The energy efficiency derivative with respect to power (DEEP) for adding one GPU is 0.0016 GFLOPs/W^2 and is 0.0037 GFLOPs/W^2 for adding one FPGA. Therefore, devices with higher DEEP factors are more promising when system energy efficiency is expected to improve within certain power constraints, such as power supply and cooling system limitation.

Overall, designers of heterogeneous systems should carefully choose computing engines and interfaces to achieve required performance with respect to applications, cost and power consumption budgets. The approach proposed in this paper provides designers with such an exploration tool.

6 Conclusion

This paper explores heterogeneous computing hardware, including CPUs, GPUs and FPGAs, for scientific computing to maximize system energy efficiency while considering system performance and power consumption. The sensitivity of the system energy efficiency to individual system components is also studied. The approach presented could help system designers to evaluate and choose the right combinations of high energy-efficient devices and interfaces, when designing energy efficient scientific computing systems.

Our design exploration approach is evaluated using Linpack on a hardware platform containing a CPU, a GPU and an FPGA, and results show that the heterogeneous computing system could provide a high energy-efficient solution to scientific computing with various performance demands.

In the future, we will extend our design exploration approach to cover applications which have a mixture of diverse computation and communication behavior.

Such applications have potential to benefit significantly from appropriate optimization of heterogeneous computing systems.

Acknowledgment. This work was supported in part by UK EPSRC, by the European Union Seventh Framework Programme under Grant agreement number 248976 and 257906, by the HiPEAC NoE, by Alpha Data, by Celoxica, by nVidia, and by Xilinx.

References

1. Eijkhout, V., et al.: Introduction to high-performance scientific computing (May 2011), http://www.tacc.utexas.edu/eijkhout/istc/istc.html
2. Feng, W.-C.: The importance of being low power in high performance computing. Cyberinfrastructure Technology Watch Quarterly 1 (2005)
3. Ding, Y., et al.: Towards energy efficient scaling of scientific codes. In: IPDPS, pp. 1–8 (April 2008)
4. Wang, G., Ren, X.: Power-efficient work distribution method for CPU-GPU heterogeneous system. In: ISPA, pp. 122–129 (September 2010)
5. Turkington, K., et al.: FPGA based acceleration of the linpack benchmark: A high level code transformation approach. In: FPL, pp. 1–6 (August 2006)
6. Fatica, M.: Accelerating linpack with CUDA on heterogenous clusters. In: GPGPU-2, pp. 46–51 (March 2009)
7. Ogata, Y., et al.: An efficient, model-based CPU-GPU heterogeneous FFT library, pp. 1–10 (April 2008)
8. Tse, A., et al.: Dynamic scheduling Monte-Carlo framework for multi-accelerator heterogeneous clusters. In: FPT, pp. 233–240 (December 2010)
9. Barak, A., et al.: A package for openCL based heterogeneous computing on clusters with many GPU devices. In: Int. Conf. on Cluster Computing Workshops and Posters, pp. 1–7 (September 2010)
10. Liu, Q., Luk, W.: Objective-driven workload allocation in heterogeneous computing systems. In: FPT (December 2011)
11. Liu, Q., et al.: Combining optimizations in automated low power design. In: DATE, pp. 1791–1796 (2010)
12. Hong, S., Kim, H.: An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In: ISCA, pp. 152–163 (2009)
13. Liu, Q., et al.: Optimising designs by combining model-based and pattern-based transformations. In: FPL, pp. 308–313 (2009)
14. Petitet, A., et al.: HPL - a portable implementation of the high-performance linpack benchmark for distributed-memory computers, version 2.0, http://www.netlib.org/benchmark/hpl/
15. Adm-xrc-5t2 data sheet, http://www.alpha-data.com/pdfs/adm-xrc-5t2.pdf
16. The green 500, http://www.green500.org/