Self-Adaptive Heterogeneous Cluster with Wireless Network

Xinyu Niu, Kuen Hung Tsoi and Wayne Luk
Dept. of Computing
School of Engineering
Imperial College London, UK
Email: niu.xinyu10@imperial.ac.uk, {khtsoi, wl}@doc.ic.ac.uk

***Abstract*—The high performance computing (HPC) community has been exploring novel platforms to push performance forward. Field Programmable Logic Arrays (FPGAs) and Graphics Processing Units (GPUs) have been widely used as accelerators for computational intensive applications. Heterogeneous cluster is one of the promising platforms as it combines characteristics of multiple processing elements, to meet requirements of various applications. In this work, we build a self-adaptive framework for heterogeneous clusters, coupled with a customised wireless network. A runtime cluster model is implemented to predict throughput, power and thermal merits for heterogeneous clusters. Cluster configurations are scheduled to improve cluster power efficiency, as well as to reduce peak temperature of processing elements. Results show that, for monitoring operations upon heterogeneous clusters, the customised wireless network provides stable and scalable performance for negligible overhead. A high performance application is developed under the proposed framework. Experiments show that this approach can improve both power efficiency and energy efficiency of N-body simulation for more than 15 times, while reducing device peak temperature by up to 12^oC.**

I. Introduction

For highly parallel applications, GPUs and FPGAs greatly outpace CPUs in arithmetic throughput, making them strong candidates for high performance computing. The graphic-optimised architecture of GPUs shows pervasive parallelism by nature [1]. FPGAs blend customisability of application-specific integrated circuits (ASICs) and flexibility of software. Hardware architectures are customised at design-time and runtime. Computational power of systems can be increased by committing application kernels to specific accelerators. Major areas for hardware accelerators include: signal processing, physics simulation, cryptographic and finance modelling [2], [3], [4]. To further increase the computational capacity of the accelerated systems, heterogeneous processors are combined to cooperate on a single application, and application workload are distributed into multiple nodes. While cluster performance can be improved to some extent, challenges remain for better utilising the computational power of heterogeneous processing elements.

First, the heterogeneity in hardware and software complicates workload distribution and cluster synchronisation. Variations in cluster status will impact processor performance. The specific impact upon a processing element depends on its hardware architecture and application characteristics. It would be extremely difficult for static designs to handle this situation. For an application involving multiple heterogeneous nodes, the computational capacity and the power consumption of heterogeneous processing elements should be explored, to improve cluster performance. Ideally, the cluster would be dynamically reconfigured to stay at ”sweet point” of the throughput/power curve, under various cluster status.

TABLE I
Service cost of an hour of downtime.

Business Service	Cost
Brokerage Operations	\$6,450,000
Credit Card Authorization	\$2,600,000
eBay	\$225,000
Amazon.com	\$180,000
Package Shipping Services	\$150,000
Home Shopping Channel	\$113,000
Catalog Sales Center	\$90,000

Second, besides throughput and power merits, stability issues for HPC systems are getting more serious. Given a large collection of nodes and intensive computation workload put on the cluster, hardware failures become commonplace. The mean time between failures (MTBF) for the ASCI Q constellation at Los Alamos National Laboratory (LANL) is less than 6.5 hours [5]. In the computing cluster of Google, there are more than 20 reboots every day. The repercussion of hardware failures for HPC clusters could be huge. As shown in Table I, in 2003, one hour breakdown of HPC clusters can cost up to 6 million dollars [6]. As indicated by the Arrhenius equation, a 10^oC increase in device temperature will double its failure rate. Therefore, the device temperature should be carefully controlled.

Third, the cluster should be aware of its current status to be self-adaptive. A cluster model also needs to learn from cluster information to update itself. Thus status packets should be efficiently broadcast. Besides, interactions between the monitoring network and high performance applications should be minimised. Sacrificing cluster performance to integrate a monitoring network is not acceptable.

The major contributions of this work include:

- A wireless network is customised and implemented in FPGAs, as a monitoring network for the heterogeneous

cluster. Experiments show that the wireless network performance scales linearly, without impacting cluster performance.
- A runtime model is developed to predict cluster performance in power, throughput and temperature aspects. Based on this model, a scheduling algorithm is implemented to adapt cluster configurations to varying cluster status.
- A high performance application, N-body simulation, is developed to evaluate the framework efficiency. Self-adaptive support is integrated into the application. Experiment results show improvements in cluster performance.

The rest of the paper is organised as following: Section II reviews previous efforts in heterogeneous computing, monitoring network and prediction models. Section III presents the overview of the self-adaptive framework with hardware and software architectures. A monitoring network, a dynamic prediction model and a runtime scheduler cooperate to adapt cluster configurations to status variations in heterogeneous clusters. Section IV presents the customised wireless monitoring network. Section V presents the prediction model and its corresponding scheduling algorithm. Section VI presents experimental results and evaluates the proposed framework. Finally, Section VII draws the conclusion.

II. Related Work

FPGAs and GPUs as co-processors can offer significant performance improvements for high performance applications. FPGA-centric platforms have been widely developed, including Berkeley Emulation Engine 2 (BEE2) [7], Maxwell [8], Reconfigurable Computing cluster (RCC) [9] and Max station. The performance gain comes from dedicated communication network, customised memory access and fully pipelined data-paths. GPUs are preferred for high clock frequency and massive amount of floating point units available on the chip. Tokyo-tech Supercomputer and Ubiquitously Accessible Mass-storage Environment (TSUBAME) contains 170 NVIDIA Tesla C1070 cards, increasing throughput from 56.43 TFlops to 77.48 TFlops [10]. Heterogeneous nodes involving CPUs, GPUs and FPGAs have been reported recently. The Quadro Plex (QP) cluster [11] consists CPUs and GPUs for computing, and FPGAs for network operations. The Axel cluster [12] is the first heterogeneous cluster where CPUs, GPUs and FPGAs working concurrently for high performance computing.

Implementing an efficient and scalable monitoring network has always been a key challenge for the HPC community. In 2004, a distributed monitoring network called Ganglia was built in University of California, Berkeley [13]. Wired network resources are utilised to monitor the system status of CPU-centric clusters with small overhead. Recent developments in wireless sensor network (WSN) enable system operators to investigate the server room environment. The RACNet [14] is a wireless sensor network for CPU-based data centres. Temperature variations in server rooms are detected, and operations of cooling systems are adapted to reduce their power consumption. In 2009, a CPU status scheduler [15] was proposed for a CPU-based Beowulf cluster, to reduce the cluster peak temperature.

There have been various studies about optimizing and configuring applications under performance constraints. In 2008, an application-aware power model [16] was proposed. Based on voltage, frequency and parallelism of CPUs, the model predicts performance at runtime. The model parameters are obtained by empirical results. Energy and power saving is achieved by optimisation from the model. In 2009, a method [17] was proposed to trade-off the CPU performance and power consumption by moving threads between cores.

The research presented in this paper provides a self-adaptive platform to combine the monitoring network and the prediction model, for heterogeneous clusters. Previous works are extended to the field of heterogeneous computing. Reconfigurability of FPGAs is utilised to customise the wireless network. The characteristics of heterogeneous processing elements and high performance applications are tightly integrated into the runtime model. The aim of our works is to show, for heterogeneous clusters, runtime properties can be explored for better cooperation between the parallel devices.

III. Self-adaptive Framework

A. Framework Architecture

In a computing node of the heterogeneous cluster, there are three processing elements: an AMD Phenom Quad-Core CPU, an NVIDIA Tesla C1060 card and a Xilinx Virtex-5 LX330 FPGA hosted on an ADM-XRC-5t2 card [18]. All processing elements have their own local memory banks to accelerate local processing. Intra-node communication is supported by PCIe system bus on mother board. CPU-based Ethernet network is deployed in the cluster to execute inter-node communication. Another inter-node communication channel is the wireless network driven by FPGAs.

An appropriate software hierarchy is essential such that high performance applications and the monitoring network can cooperate properly in the heterogeneous cluster. As shown in Figure 1, high performance applications are running under message passing interface (MPI) framework. For computational intensive parts, parallel threads targeting at heterogeneous processing elements are instantiated under OpenMP extensions.

Implemented in FPGAs, network drivers are running in parallel with high performance applications. The network driver consists of a digital base-band, a Medium Access Control (MAC) layer and a network layer. Data interface and control units are developed to support data flow between CPUs, network drivers and peripheral circuits. On top of the network layer, an application layer is built in the CPU. Monitored information is processed at this layer. A scheduling algorithm is implemented to update model coefficients, and to schedule cluster configurations.

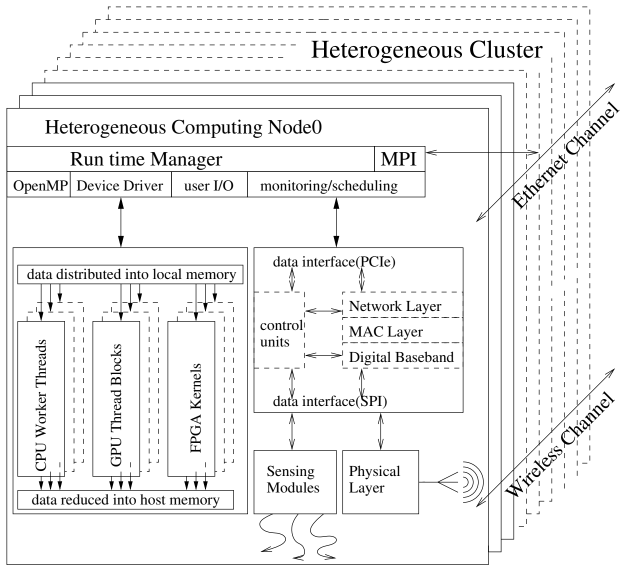

Fig. 1. Software architecture of proposed framework.

B. *Self-adaptive Computation*

As discussed in Section I, the heterogeneity in hardware architectures and varying cluster environment limits achievable cluster performance. The self-adaptive framework presented in this work is capable of adjusting cluster configurations during runtime, to cover the heterogeneity and to adapt to cluster status variations. The framework learns from impacts of current and previous configurations, and predicts cluster performance in the future. Workload distribution and device concurrency can then be dynamically scheduled to provide optimal cluster configurations under various situations.

The self-adaptive parts of the proposed framework include a monitoring network, a dynamic prediction model and a runtime scheduler. Hardware properties and application characteristics are extracted into the model. Based on information collected from the monitoring network, our runtime model adapts its coefficients to the varying environment. Model predictions enable the scheduler to dynamically reconfigure the heterogeneous cluster. In the meanwhile, model constraints will stabilise device temperature in a certain range. Further details of the monitoring network and the prediction model would be given in the sections that follow.

IV. Wireless Monitoring Network

To be self-adaptive, the framework should be aware of its application performance and hardware status. In our framework, device temperature, throughput and power consumption are gathered by a wireless monitoring network. The monitoring network should be light-weight and efficient to minimise the overhead for being self-aware. We refer overhead as resources consumed for collecting status data and impacts upon application performance. In terms of network performance, the implemented network needs to provide fine-grained monitoring upon cluster status variations. Monitoring resolution depends on the time step of scheduling operations. Since the cluster is scheduled every application iteration, gathering cluster status at millisecond level can satisfy the monitoring resolution, and the network overhead is acceptable. Length of the payload data is 16 bytes, leaving the required bandwidth at around 1MHz. Additionally, the network must be scalable such that the proposed framework can be deployed to large-scale clusters.

In this work, we build a FPGA-driven wireless network for the proposed framework. The novel aspects of the network include: 1) Due to the broadcasting nature of wireless technology, the wireless network can be scalable for appropriate topology. For complex scenarios such as multiple applications running in cluster, various carrier frequencies can be instantiated to isolate the communication channels and to increase available bandwidth. 2) Driven by FPGAs, the implemented networking operations will run in parallel with high performance applications. Unlike CPUs and GPUs, there are always unoccupied resources in FPGAs. The wireless network can be customised to use these resources, eliminating resource conflicts between monitoring network and high performance applications. 3) Monitoring operations in heterogeneous clusters have specific requirements and preferences. The network layers can be customised for these to improve bandwidth efficiency and to reduce consumed resources.

The physical layer of wireless network is mainly implemented in an analogue front-end. Data packets are modulated and demodulated at the carrier frequency, which varies from 2400MHz to 2498MHz. Upper layers input parameters, control signals and packets into the front-end through a Serial Peripheral Interface Bus (SPI). Transmitter and receiver chains share an external antenna. The filter parameters are configured such that transmitter synthesisers and receiver synthesisers settle at the same frequency. This allows fast turnaround between transmitters and receivers. Temperature sensors attached to processing elements are thermistors. Power consumption of processing elements is measured with the current flowing into them. Magnetic current transformers are used as current sensors. Sensed information is sampled into the FPGA through an Analogue-to-Digital Converter (ADC). A SPI controller are implemented in the FPGA to communicate with the ADC. With the dedicated communication channel between FPGAs and peripheral circuits, the latency for sampling cluster status is reduced.

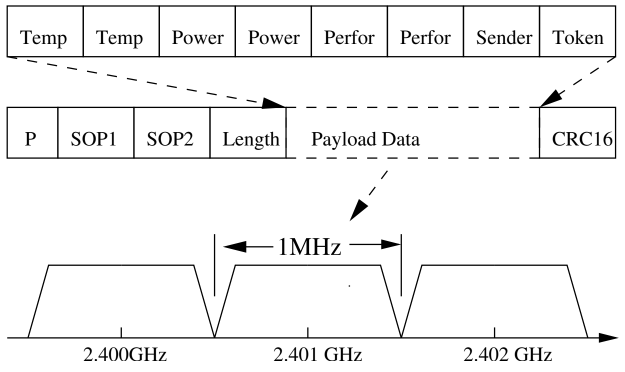

Fig. 2. Physical layer and digital base-band of proposed network.

At the digital base-band, application throughput is measured with digital counters. The measured throughput is combined with sampled device temperature and power consumption, to form payload data. The packet format is designed to provide high communication stability and high data efficiency. As shown in Figure 2, preambles, start of packet (SOP) and payload length are inserted in front of the payload data. The preamble is set as ”10101010”, a packet with such beginning will be listened. SOP is used to further verify the listened packet. The SOP is a 64-bit pseudo-noise code (PN code). Received SOP fields are correlated with data stored in the base-band. If the correlation is beyond a certain threshold, the received packet is confirmed as a valid packet. At the end of a packet, Cyclic Redundancy Check (CRC) is attached to validate contents of the received payload. Direct Sequence Spread Spectrum (DSSS) technique is used to improve the immunity to noise in the server room. As a consequence, bandwidth of the wireless network is reduced from 1Mbps to 250Kbps. Sampled packets are put into a First In, First Out (FIFO). When its MAC layer possesses control for the communication channel, the packets are transmitted by the front-end. Therefore, data sampling is isolated from network operations. Received date are decoded and fed into PCIe channels. A handshaking protocol is implemented to support data transfers between FPGAs and CPUs.

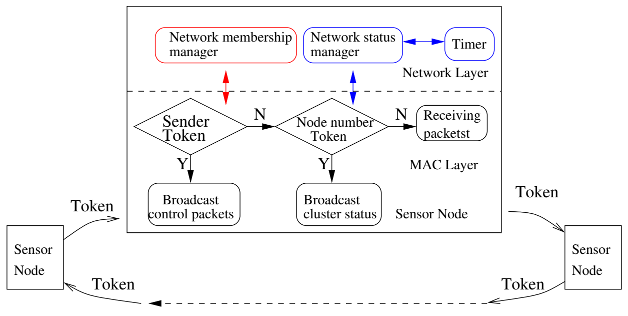

Fig. 3. MAC layer and network layer of proposed network.

At the MAC layer, an anti-collision algorithm is developed for medium access control. For a heterogeneous cluster, the communication order can be predefined for a given application. The purpose of our MAC layer is to share the wireless channel with minimal overhead. In other words, the latency introduced by the anti-collision algorithm should be minimised. A token ring topology is implemented for our monitoring network. As shown in Figure 2 and Figure 3, sender identification and token are attached at the end of the payload. If a head node is broadcasting control signals, it occupies the wireless channel until all information has been transmitted. In other scenarios, the sensor node possessing the token will transmit status of the sensed node. A new token is then generated for the following node indicated in the network membership list. By consuming and generating tokens one by one, cluster status can be gathered. This simple algorithm ensures that the medium access can be decided within several comparisons. The token ring topology enables the node information for all involved nodes to be circulated in the cluster, with a deterministic order.

On top of the MAC layer, a network layer is implemented to manage the network topology, network membership and network status during runtime. The token ring topology depends on a predefined membership to loop through all involved nodes. The initial network membership is synchronised with MPI configurations. During runtime, the membership list can be reconfigured to remove inactive nodes and to combine new nodes. Therefore, the network topology can be dynamically reconfigured. To ensure network stability, network status is monitored at the network layer. Network status for a sensor node includes initialising, receiving, transmitting and finalising. In certain cases, the token will be lost during transmission, and network status can be used to build a time-out mechanism. If a node takes too much time to receive a new token, the previous packet will be re-transmitted to restart communication. The network layer is running in parallel with the other three layers. Therefore, no additional latency is introduced.

V. Scheduling Mechanism

A runtime model is essential for our self-adaptive framework. With information collected from the wireless monitoring network, time-varying coefficients in the dynamic model are automatically updated. Application characteristics, hardware properties and runtime variations are combined into the model to generate proper predictions under current environment. Based on the predictions, a scheduling algorithm is developed to adapt cluster configurations to cluster status variations.

Previous works mainly focus on CPU-centric clusters, and system optimisation is achieved by tuning hardware parameters for a multi-stage application [16], [17]. In our work, application kernels are expanded onto heterogeneous computing. The optimisation issue is how to better utilise the heterogeneous resources in a single stage. Scheduling variables include parallelism $p_{(i,j)}$ and workload ratio $w_{(i,j)}$ for every processing element. We refer to each possible combination of $p_{(i,j)}$ and $w_{(i,j)}$ as a configuration. The label (i,j) indicates that it is the configuration for processing elements j of node i. It is assumed that there are N nodes involved in the application, and there are M processing elements in one node.

A. Basic Prediction Model

Throughput of a processing element (TH) depends on application characteristics, hardware architectures and current cluster status. Basic throughput prediction only considers static situations. Throughput is expressed as a linearised function. The heterogeneity in hardware architectures is covered by different model coefficients.

$$TH_{(i,j)} = TH_{dy(i,j)} \cdot p_{(i,j)} + \epsilon_{(i,j,k)} \quad (1)$$

TH_{dy} is the throughput when only one kernel is running in the specific processing element. It is the simplest scenario for a processing element, as memory hierarchy is dedicated to the single kernel, and no synchronisation is required for parallel threads. When parallelism $p_{(i,j)}$ is increased, it will bring overhead to protect data consistency and to share memory bandwidth. The overhead is modelled as $\epsilon_{(i,j,k)}$. The $\epsilon_{(i,j,k)}$ is modified for different parallelism, with k indicating current parallelism. Power consumption $P_{(i,j)}$ is modelled with similar strategy. $P_{dy_(i,j)}$ stands for dynamic power consumption of a single kernel, and $\eta_{(i,j,k)}$ covers non-linear factors for power consumption.

$$P_{(i,j)} = P_{dy(i,j)} \cdot p_{ij} + \eta_{(i,j,k)} \tag{2}$$

Temperature is previously modelled with thermal flow at circuit level [19], [20]. In our model, the temperature problem is investigated at system level, with temperature affected by power dissipation and cooling systems. The temperature is presented as sum of initial temperature, temperature increase due to power dissipation and temperature decrease due to cooling systems. $P_{ov(i,j)}$ presents overall power consumption of a node. Static power consumption $P_{st(i,j)}$ is measured at initialisation phase of applications. The coefficients are assumed to be static in the basic model.

$$T_{(i,j)} = T_{ini} + T_{di(i,j)} \cdot P_{ov} + T_{co(i,j)} \tag{3}$$
$$P_{ov(i,j)} = P_{st(i,j)} + P_{ij(i,j)} \tag{4}$$

The static model is trained by running applications on an isolated prototype machine. Single kernel design is executed first to measure linear coefficients. All other possible configurations are mapped onto corresponding processing elements to calculate the non-linear factors.

B. Runtime Model Extensions

For a static model, model coefficients are assumed to be constant. Networking and computing resources distributed to the application are considered as stable. In practice, the heterogeneous cluster is shared by multiple applications. It is impossible for cluster environment to satisfy such assumptions. The static model can provide an appropriate cluster configuration initially. During runtime, the simplified functions will introduce errors in predictions, limiting cluster performance.

For throughput and power consumption, a runtime coefficients α is introduced to cover impacts of resource sharing. It is updated according to errors between predicted values and measured values. In the heterogeneous cluster, an application is divided into Map and Reduce phase [21]. Reduce time means the time consumed by reducing intermediate results in parallel PEs into final results. Variations in communication channel status are reflected by updating the reduce time.

$$TH_{(i,j)} = (TH_{dy_(i,j)} \cdot p_{(i,j)} + \epsilon_{(i,j,k)}) \cdot \alpha_{(i,j)} \tag{5}$$

Temperature coefficients are non-linear in the time dimension. The time-varying factors are combined into our model by updating the model coefficients dynamically. T_{di} and T_{co} adapt themselves to approximate to the actual temperature curve. Within a time step, measured temperature and power consumption are processed to update T_{di}. T_{co} is calculated during the reduce phase. The coefficients are averaged for predictions in the following time step.

$$T_{(i,j,t)} = T_{ini} + T_{di(i,j,t)} \cdot P_{ov} + T_{co(i,j,t)} \tag{6}$$
$$T_{di(i,j,t)} = \beta_{(i,j,t)} \cdot t_{map} \tag{7}$$
$$T_{co(i,j,t)} = \gamma_{(i,j,t)} \cdot t \tag{8}$$

C. Model Constraints

With the dynamic model, cluster status can be predicted under complex cluster environment. Constraints for model parameters are necessary for making reasonable predictions. Maximum parallelism for a processing element is limited by available hardware resources. The scheduled parallelism and workload ratio should be in certain range to ensure correct configuration. Sum of the workload ratio w must be 1 for appropriate data distribution.

$$\begin{cases} p_{max} \geq p_{(i,j)} \geq 0 & :\forall(i,j) \in \{(1,1)...(M,N)\} \\ 1 \geq w_{(i,j)} \geq 0 & :\forall(i,j) \in \{(1,1)...(M,N)\} \end{cases} \tag{9}$$

$$1 = \sum_{i=1}^{M} \sum_{j=1}^{N} w_{(i,j)} \tag{10}$$

For parallel computing, execution time of distributed workload should be synchronised to eliminate cluster bottlenecks. Computation time of a specific processing element can be easily calculated. Memory access time t_m refers to time for data transfers between host memory banks and local memory banks. Execution time of cluster can be divided into execution time for Map phase t_{map} and execution time for reduce phase t_{red}. At cluster level, `MPI_Barrier` is used to protect data consistency, thus ensuring equality of reduce time t_{red}. Since t_m and t_{red} are measured during runtime, for a given parallelism $p_{(i,j)}$, the computation time t_c and the corresponding workload ratio $w_{(i,j)}$ can be calculated as:

$$f(w_{(i,j)}, t_c) = \begin{cases} \frac{w_i j}{TH_{ij}} + t_{m_{(i,j)}} & = t_i \\ \sum_{i=1}^{M} \sum_{j=1}^{N} w_{(i,j)} & = 1 \end{cases} \tag{11}$$

By synchronising the execution time, the number of scheduling variables is reduced to one.

With the calculated t_c and $w_{(i,j)}$, for a given parallelism $p_{(i,j)}$, throughput, power consumption and temperature of a processing element can be properly estimated. Peak temperature of every processing element should be limited to an acceptable level.

$$T_{(i,j)} \leq T_{max\_ij} \quad : \forall(i,j) \in \{(1,1)...(M,N)\} \tag{12}$$

D. Scheduling Algorithm

To utilise the predicted information, a scheduling algorithm is required to pick the optimal cluster configuration for the next time step. At cluster level, cluster power consumption P can be accumulated from power consumption for involved processing elements. Overall throughput is expressed as the reciprocal of predicted cluster execution time t, as overall workload is normalised to 1. Cluster power efficiency E is expressed with the ratio between TH and P.

$$TH = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N} w_{(i,j)}}{t} = \frac{1}{t} \tag{13}$$

$$P = \sum_{i=1}^{M}\sum_{j=1}^{N} P_{ij} \quad : i \in \{1...M\} j \in \{1...N\} \tag{14}$$

The scheduler is running in main thread under the MPI platform, its scheduling algorithm is shown in Algorithm 1. The time step in current model is set to be one application iteration. When heterogeneous processing elements are working on distributed workload, sensed data are collected into the application layer. Monitored cluster status is processed in parallel with high performance applications. Events such as device overheating, resource sharing and network saturation are expressed as varying runtime coefficients. The scheduler learns cluster status from the updated model. Achievable computational capacity for each processing element is limited by temperature constraints. By scheduling with runtime predictions and model constraints, the proposed framework adapt itself to stay at the optimal level. The objective of our scheduling algorithm is maximum power efficiency.

Algorithm 1 Scheduling Algorithm.

```
1: for $i = 1 \to N$ do
2:   for $j = 1 \to M$ do
3:     $\alpha_{(i,j)} \Leftarrow$TH(p(i,j), w(i,j), $t_{c(i,j)}$);
4:     $T_{di(i,j)} \Leftarrow$T($T_{(i,j)}$, $T_{ini(i,j)}$, p(i,j), $P_{st(i,j)}, t_{map}$);
5:     $T_{co(i,j)} \Leftarrow$T($T_{(i,j)}$, $T_{ini(i,j)}$, $t_{red}$);
6:     $p_{max} \Leftarrow$ T($T_{max}, T_{ini}$, $T_{diss}$);
7:   end for
8: end for
9: for $i = 1 \to N$ do
10:   for $j = 1 \to M$ do
11:     for $k = 1 \to p_{max}$ do
12:       ($t_c$, w) $\Leftarrow f(w,t)$;
13:       Estimate $TH$, $P$, $E$;
14:       Pick up $E_{max}$;;
15:     end for
16:   end for
17: end for
```

VI. Results

The proposed framework is evaluated in this section. A high performance application is developed under the self-adaptive framework. Additional hardware resources, power consumption and execution time are measured as framework overhead. Bandwidth efficiency, stability and scalability of the wireless monitoring network are compared with its wired counterpart. Finally, cluster performance is measured to evaluate efficiency of the self-adaptive framework.

TABLE II
N-Body FPGA implementation results.

resource	non-adaptive	self-adaptive	difference
LUT	93862(45%)	94818(45%)	956 (0.46%)
FF	115822(55%)	117773(56%)	1951 (0.94%)
DSP	180(93%)	180(93%)	0 (0%)
BRAM	101(31%)	103(32%)	2 (0.6%)

A. Benchmark and Experiment Environment

N-Body simulation is commonly used to study interactions between objects with gravitation. The simulation is an iterative process where the 3D position and velocity vectors are updated after combining the total force acting on every individual particle. N-body simulation is highly parallel. It is developed under the proposed framework to evaluate its efficiency. Maximum parallelism for CPUs, GPUs and FPGAs are 4, 10 and 10, respectively. The CPU designs are compiled using ICC with `-fast` and `-O3` flags, linked against OpenMP and OpenMPI libraries for intra-node and inter-node communication. The GPU implementation is developed under CUDA environment. The hardware designs are captured with VHDL descriptions, running at 200MHz. The dynamic power consumption for a full-speed design running in a heterogeneous node is 175.68W.

B. Framework Overhead

Integrating the self-adaptive facilities into high performance applications will inevitably introduce overhead. In our framework, the overhead includes power consumed by sensor front-ends, hardware resources used by network drivers, and additional time spent on scheduling.

Supported by power modules within heterogeneous nodes, sensor front-ends consume extra power for sensing and circulating cluster information. For a 3.3V voltage supply, the front-end runs with 0.04A current when transmitting data, and with 0.01A current when receiving data. Maximum power consumption for a sensor front-end is 0.132W. This is negligible when compared with power consumption of a node.

The number of data-paths implemented in a FPGA chip is limited by available DSP blocks. The wireless network driver is customised to avoid resource conflicts. From Table II, we can see that the network driver consumes no DSP block and less than 1% of other resources. As the network driver is implemented as isolated module, the operating frequency of the high performance application is not affected.

The scheduling algorithm is running in serial with computational intensive kernels. Scalability of proposed scheduler determines the number of nodes accommodated in the framework. The scheduler execution time is shown in Figure 4. It can be seen that the scheduler overhead scales linearly. If maximum time consumed by the scheduler is set to be 10ms, many nodes can be supported by our framework.

C. Network Performance

The performance of broadcasting cluster status either through the wireless network or through MPI over Ethernet is measured. We compare bandwidth efficiency, network stability

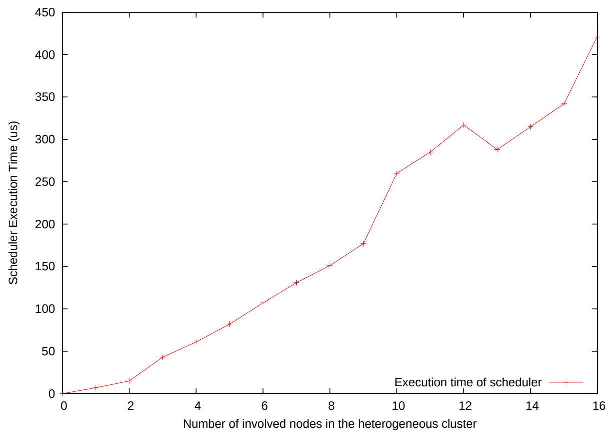

Fig. 4. Scheduler scalability.

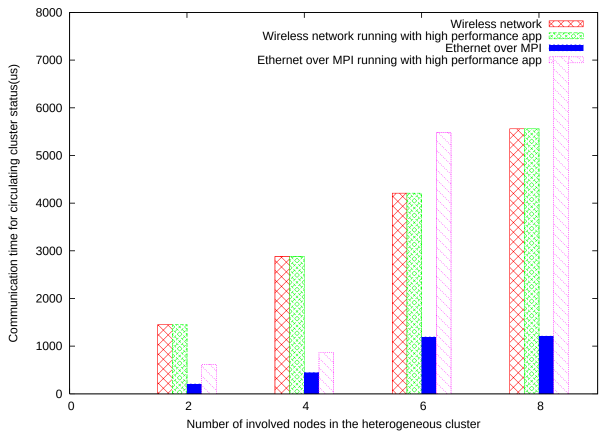

Fig. 5. Network performance.

and network scalability, when up to 8 nodes are involved. In the Ethernet-based MPI implementation, the synchronous `MPI_Allgather` model is used. The experiment results are presented in Figure 5.

While Ethernet is running at Gbps level, its network performance is limited by the overhead for packet formats and protocol stacks. The wired network outperforms the wireless network by 5 times, when there are 8 nodes in the cluster. Given the huge difference in available bandwidth, the wireless network shows better efficiency for broadcasting packets with small payload.

In practice, the monitoring network and high performance applications are running in the heterogeneous cluster simultaneously. Therefore, in the second scenario, the network operations are integrated into high performance applications. Driven by hardware, the wireless network runs in parallel with applications, its network performance is not affected. The MPI network operations generate computational workload in CPUs. The conflicts in computational resources dramatically decrease the performance of Ethernet. With only 250Kbps bandwidth, the wireless implementation outperforms the wired network, when more than 4 nodes are involved in the application. It is reasonable to expect the speed-up will keep increasing when cluster size further scales, as the computational workload generated by network operations is proportional to cluster size.

In both scenarios, the communication time for broadcasting data through the wireless channel scales linearly, while the scalability of Ethernet over MPI depends on processor status. Therefore, the efficient, light-weight and scalable wireless monitoring network is suitable for our approach.

D. Cluster Performance

Finally, we investigate the cluster performance under the proposed framework. To evaluate the self-adaptive design, cluster performance for a static design is also measured. In the static implementation, parallel devices run with full speed, and workload is distributed according to their throughput measured off-line. Figure 6 shows the execution time of both static and self-adaptive designs, under varying cluster status. Cluster execution time of the static design only reaches optimal level at several application iterations, when cluster status satisfies assumptions of the static design. The self-adaptive design, on the other hand, automatically adapt itself for cluster status variations. Predications from the runtime model ensure that the high performance application can stay at the optimal level for current cluster status. Therefore, the proposed framework can better utilise the available computational capacity of heterogeneous processing elements.

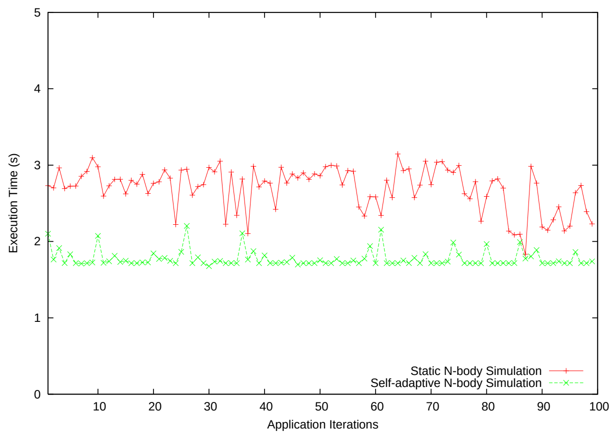

Fig. 6. Cluster execution time variations.

Figure 7 summarises cluster execution time, power efficiency and energy efficiency. The static implementation is used as a reference design. During runtime, the self-adaptive applications are dynamically reconfigured to stay at the optimal level. It can be seen that the cluster reaches maximum throughput and maximum power efficiency at different configurations. Compared with the self-adaptive implementation aiming at maximum throughput, the design staying at optimal power efficiency level reduces cluster throughput by 28%. After scheduling under the self-adaptive framework, the cluster power efficiency and the cluster energy efficiency are increased by 15.7 times and 16.3 times, respectively.

While cluster configurations are scheduled to improve cluster performance, cluster stability is also increased. Under the self-adaptive framework, computational capacity of processing

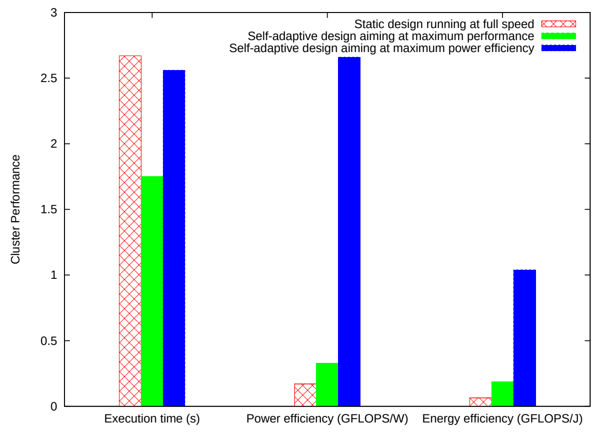

Fig. 7. Cluster performance.

elements will be dynamically reduced, as device temperature is approaching its upper limit. GPUs are overheated when static N-body simulation is running in the cluster. The device temperature before and after scheduling are presented in Figure 8. It can be seen that the peak temperature of overheated processing element is reduced by 12^oC. As indicated by the Arrhenius equation, this halves the hardware failure rate.

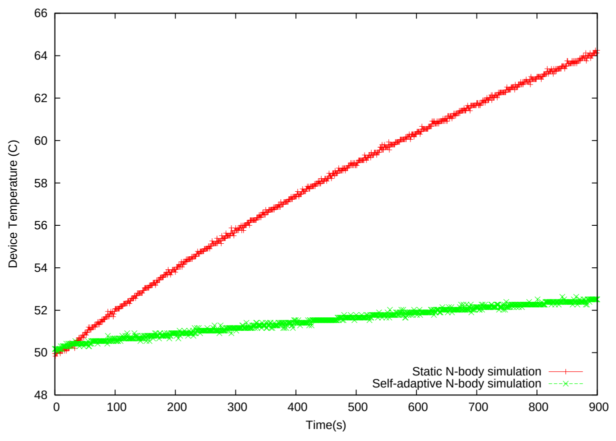

Fig. 8. Cluster performance.

VII. Conclusion

In this paper, we presented a self-adaptive framework for heterogeneous cluster. A customised wireless network is implemented as a monitoring network for heterogeneous clusters, and cluster configurations is dynamically scheduled, based on the sensed data and a runtime model. Experiment results show the wireless network provides stable and scalable performance, with negligible overhead. The dynamic framework improves power efficiency and energy efficiency of N-body simulation by 15.6 times and 16.3 times, respectively, while the device peak temperature is reduced by up to 12^oC. Current and future research includes performing extensive experiments to support partial reconfiguration in FPGA for improving energy optimization, exploring more complicated scenarios such as running multiple applications in the heterogeneous cluster, and extending the scheduling mechanism to support frequency and voltage scaling.

References

[1] NVIDIA, "Nvidia's next generation cuda compute architecture: Fermi."

[2] K. Tsoi, C. Ho, H. Yeung, and P. Leong, "An arithmetic library and its application to the n-body problem," in *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2004, pp. 68–78.

[3] A. L. Masle, W. Luk, J. Eldredge, and K. Carver, "Parametric encryption hardware design," in *ARC '10: Proceedings of the 6th International Symposium on Applied Reconfigurable Computing*, 2010, pp. 68–79.

[4] L. M.-P., C.-C. Cheung, C.-W. Cheung, W. P.P.M., L. I.K.H., Y. W.M.M., W.-S. Yuen, C. K.S.K., K.-S. Leung, and L. P.H.W., "Cpe: a parallel library for financial engineering applications," in *IEEE Transactions on Computer*, 2005, pp. 70–77.

[5] C.-H. Hsu and W.-C. Feng, "Reducing overheating-induced failures via performance-aware cpu power management," in *Proceedings from the 6th International Conference on Linux Clusters: The HPC Revolution*, 2005, pp. 54–64.

[6] W. Chunfeng, "Making a case for efficient supercomputing," in *ACM Queue*, 2003.

[7] C. Chang *et al.*, "BEE2: a high-end reconfigurable computing system," *Design & Test of Computers, IEEE*, vol. 22, no. 2, pp. 114–125, March-April 2005.

[8] R. Baxter *et al.*, "Maxwell - a 64 FPGA supercomputer," in *Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems*, 2007, pp. 287–294.

[9] R. Sass *et al.*, "Reconfigurable computing cluster (RCC) project: Investigating the feasibility of FPGA-based petascale computing," in *Proc. Symposium on Field-Programmable Custom Computing Machines*, 2007, pp. 127–140.

[10] T. Endo and S. Matsuoka, "Massive supercomputing coping with heterogeneity of modern accelerators," in *Proc. IEEE Int. Symp. on Parallel and Distributed Processing*, 2008, pp. 1–10.

[11] M. Showerman *et al.*, "QP: A heterogeneous multi-acceleator cluster," in *Proc. LCI Int. Conf. on High-Performance Clustered Computing*, 2009.

[12] W. L. Kuen Hung Tsoi, "Axel: A heterogeneous cluster with fpgas and gpus," in *FPGA '10 Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, 2010, pp. 115–124.

[13] M. L. Massie, B. N. C. b, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, 2004.

[14] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao, "Racnet: A high-fidelity data center sensing network," *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, 2009.

[15] F. Hu and J. J. Evans, "Power and environment aware control of beowulf clusters," *Cluster Computing*, vol. 12, pp. 299–308, 2009.

[16] 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.

[17] 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.

[18] Alpha-Data Parallel System Ltd, *ADM-XRC-5T2 User Manual*, 2008.

[19] T. Chantem, R. Dick, and X. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," in *Proc. International Conf.Design, Automation and Test in Europe 2008*, Mar. 2008, pp. 288–293.

[20] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Proceedings. 30th Annual International Symposium on Computer Architecture, 2003.*, Jun. 2003, pp. 288–293.

[21] L. Ralf, "Google's MapReduce programming model – revisited," *Science of Computer Programming*, vol. 68, no. 3, pp. 208–237, 2007.