Energy Efficient Simulation Computing of Electrocardiogram with TK1 Board

Wenfeng Shen, Feng Qiu, Yaopeng Hu, Yanghua Shen, and Xin Zhu

***Abstract*—The computer simulation in cardiac models has become an increasingly powerful tool in the study of cardiac electrophysiology. To precise analysis the electrophysiological mechanism, it not only demands a significant computing capacity due to its heavy workload, but also has to be available and convenient in many medical experiments, which seems to be contradictory in a conventional computer perspective. However, with the emergence of new heterogeneous computing platforms, a surging number of scientific applications are dramatically improving upon the energy efficiency of existing solutions. Under such circumstances, this paper presents a metric method for energy efficiency and several optimizing strategies, to fully exploit the energy efficiency of applications. To verify these approaches, we describe a new experience adapting the computer simulation of electrocardiogram(ECG) based on a whole-heart model for a Jetson Tegra K1(TK1) System on Chip(SoC) board, which was previously only possible on desktop-level platforms. The experiments are conducted to evaluate the performance and power draw in different situations on different platforms. In order to investigate the characteristics of unified memory in TK1 board, an extra experiment was provided. The experimental results show that our proposed methods combining with the new hardware can lead to a encouraging consequence.**

***Index Terms*—Energy efficient, Parallelization, TK1 board, Unified memory, Computer simulation of ECG.**

I. Introduction

THE whole-heart modeling and computer simulation of electrocardiogram (ECG) has always been a key issue in the filed of cardiac electrophysiology [1]. Compared with medical clinical research, this subject enables us to better study working principles of cardiac electrophysilogy by fully reproducing the electrophysiological characteristics of heart, which can not be achieved in most traditional anatomy experiments [2].

However, due to the enormous computational burden of cardiac simulation, it was mainly operated on the supercomputers in the early stage. To break the computational bottleneck, parallelism techniques were employed into this subject to improve the performance. For example, with the aid of OpenMP and MPI protocols, Zhu et al. built a whole-heart model model on a four-node cluster of shared memory computers [3]. Tudel et al. used a Silicon Graphic Origin 2000 computer with 64 processors to realize a parallelization for the simulation of QRST integral maps with a membrane based on the virtual heart model [4]. Since the calculation capability mostly relies on multi-processors or multi-nodes of computers, the poor scalability confines the improvements on performance to a great extent. In the past decade, with the rising of various acceleration components, a lot of researchers had realized that a good performance can be realized when applications were implemented with parallelization on servers or workstations equipped with Graphics Processing Unit(GPU). For instance, Sato et al. utilized a GPU to accelerate the simulation of electrical wave propagation in myocardium and obtained a speedup of 30 times compared with the completed work of 2D tissue simulation with a single 2.0 GHz AMD Opteron processor [5]. It is important to note that the computational capacity of these platforms is increasing dramatically. But these platforms are still too expensive to become easily available everywhere. Recently, a growing number of computer modelings and simulations of ECG have been applied to ordinary personal computers(PCs) with x86 architecture. Like the previous work [6], [7], [8] from our team showed that PCs with CPU-GPU heterogeneous architecture can provide a good computing environment for the computer simulation of ECG with some parallelism strategies. However, up to now, most of researchers have only focused on improving the performance, that is, through promoting computation efficiency to reducing completion time of applications.

At the same time, with the constant extending of the parallel scale, the power consumption of high performance computers increases greatly. Moreover, the power consumption of applications of computer systems draw unprecedented attentions since the Defense Advanced Research Projects Agency (DARPA) has introduced the high productivity computing systems in [9]. Numerous studies have concentrated on how to balance the two goals of promoting computation efficiency and decreasing power draw, that is to say, how to make a trade-off to realize an energy efficient application.

To solve this problem, on the one hand, many innovative and flexible hardware platforms give developers better options of optimizing the energy consumption [10]. More and more scientific computing applications have been used in these mobile devices. For examples, a mixed-signal ECG System on Chip(SoC) is capable for portable ECG monitoring applications with low power consumption was achieved by Kim et al [11]. Raj et al. developed a reliable automatic monitor based on the condition of patient's hearts, which is processed on an ARM-based SoC with just several watts [12]. On the other hand, two energy-conscious scheduling algorithms using dynamic voltage scaling (DVS) was presented by Lee et al., and their work showed that the test

This work is partially supported by The National Key Research and Development Program of China (NO.2017YFB0701600), Shanghai Innovation Action Plan Project under the grant No. 16511101200, Shanghai University Material Genetic Engineering (No.14DZ2261200), the Japan Society for the Promotion of Science under Grants-In-Aid for Scientific Research 15H04678 and Shanghai University High Performance Computing Center.

W. Shen, F. Qiu, and Y. Shen are with the School of Computer Engineering Science, Shanghai Institute for Advanced Communication and Data Science, Shanghai University, Nanchen Road, Shanghai 200444 China (e-mail: wfshen@mail.shu.edu.cn).

Y. Hu is with Department of Physiology, School of Medicine, Fukuoka University, Fukuoka, Fukuoka 814-0180, Japan.

X. Zhu is with Biomedical Information Technology Lab, the University of Aizu, Aizu-Wakamatsu, Fukushima 965-8580, Japan.

performance is very compelling in terms of both application completion time and power consumption [13].

In terms of the modeling and simulation computing of ECG, these miniature and portable devices fitly have the advantages of significant computing capacity and lower power consumption, which enable clinical staff to use them in a hand-held device or offer medical experimental workers a quick precise calculation result during the time of making medical diagnosis. To this end, this paper makes a contribution to achieving an energy efficient computational application according to the proposed metric method. What's important, some optimizing approaches, including some parallelization strategies, an adapted scheduling algorithm and some dynamic controllable mechanisms on Jetson Tegra K1 SoC board, are presented. In the demonstrations, the scenario is the computer simulation of ECG based on the Wei-Harumi whole-heart model.

The rest of this paper is organized as follows. In section II, we will briefly introduce some related background knowledge, including some features of TK1 board and some work of ECG simulation computation. In section III, the method of how to measure energy efficiency in this work, and the details of how to conduct the optimizing strategies and adjustable mechanisms to improve energy efficiency will be described. The experimental evaluations of performance and power consumption of TK1 board and that of an ordinary PC will be exhibited in section IV. An investigation of unified memory of TK1 board will also be addressed. Then we shall make a discussion on all of the experiments in the same section. Last but not least, we draw a conclusion and propose future work in section V.

II. Background

A. Jetson Tegra K1 SoC board

Tegra K1 SoC is a mobile processor with GPU which has the same advanced feature and architecture as the modern desktop GPU $--Kepler$ GPU, though it still possesses the mobile chip with low power draw designed by Nvidia [14].

The Jetson Tegra K1 SoC board (referred to as TK1 board in this paper) is an embedded development platform within the Tegra K1 SoC(CPU + GPU + ISP in one chip) and runs a Linux environment. It can provide developers with not only all Tegra's common features and interfaces by standard connectors, but also a highly flexible and extensible for special design. It has some PC-oriented features such as SATA, mini-PCIe and fan to allow continuous operation when developers need to extend disk capacity in dealing with heavy workload. There are two reasons for us to choose the TK1 board as our platform of energy efficient computing applications. Firstly, it has been designed with the particular goal of being efficient [10]; secondly, this board as a heterogeneous architecture, which contains the Quad-Core 4 ARM Cortex-A15 CPU running at up to 2.3 GHz, the 5-th low power companion Cotex core and a Nvidia Kepler streaming multiprocessor (SM) with 192 Compute Unified Device Architecture(CUDA) cores running at up to 852 MHz [14].

There are some encouraging work worth mentioning with respect to the employment of TK1 board in other applications, Serrano et al. concluded Jetson TK1 is an evident incentive that can lead to the creation of smaller and mobile medical image scanners [15]. Belloch et al. also demonstrated that Jetson TK1 is capable of executing the application in real time and it can provide a really good performance even if it is a low power device [16].

B. Memory Access of GPU

Generally speaking, the CUDA platform mainly supports the following data migration methods: traditional memory access, unified memory access, zero-copy access, pageable memory access and pinned memory access [17]. In this paper, we will focus on the traditional memory access and unified memory access, and compare these two patterns in terms of the overall application execution time.

1) Traditional Memory Access: To date, the most common way of memory access of GPU is traditional memory access(TMA). The memory of CPU(host) and GPU(device) are completely distinct since they are separated in physical structure, but they still can communicate with each other by the PCI-express bus. A typical CUDA Application Program Interface(API) in executing a program by TMA includes the following steps:

1) Invoke the Malloc() function to allocate the space of host memory for data.
2) Invoke the cudaMalloc() function to allocate the space device memory for data.
3) Invoke the cudaMemcpy() function to transfer data from the host to the device.
4) Execute CUDA kernel and store the result in the device memory.
5) Invoke the cudaMemcpy() function to transfer data from the device back to the host.
6) Invoke the Free() and cudaFree() function to free the memory space of the host and device.

From this process, it is not difficult to find that the space of host and device is undoubtedly disjoint. Hence this programming model relies on programmers to explicitly manage data between CPU and GPU [18]. There exists two specified constant movements of host-to-device and device-to-host, which may result in overheads of memory operations and extreme performance degradation [19].

2) Unified Memory Access: Unified Memory Access(UMA) is an emerging technology supported by CUDA 6.X. In this paper, the environment of experiments is TK1 board with CUDA 6.5, which offers not only a heterogeneous architecture of ARM CPU and NVIDIA GPU, but also a unified memory in CPU-GPU integrated memory.

The unified memory has implemented a pool of managed memory to unite the separate domain of CPU-GPU. It is accessible with a single pointer to both the CPU and GPU since the managed memory is shared by the CPU and GPU. The different view between the TMA and the UMA is showed in Figure 1.

The key point of unified memory is that the system automatically migrates data allocated in unified memory of host and device so that it looks like CPU memory to code running on the CPU, and like GPU memory to code running on the GPU [20]. A typical CUDA API in executing a program by UMA includes the following steps:

1) Invoke the cudaMallocManaged() function to allocate the memory space for data, and to create the buffer

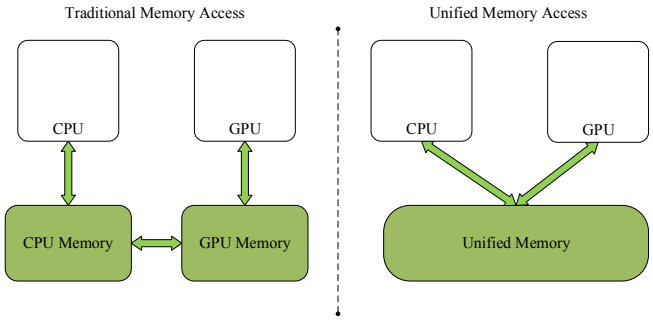

Fig. 1. Different view between the traditional memory access and the unified memory access

on GPU side and notifies the memory driver about the location and page tables.

2) Use a global pointer to access the data that stored in the unified address space when need it in CPU runtime or CUDA kernel runtime.
3) Invoke the cudaDeviceSynchronize() function to synchronize the data between the host space and device space.
4) Invoke the cudaFree() function to free the common address memory space for data.

It is necessary to note that, the host space and device space mentioned here can be understood as two logical spaces but they are treated as a coherent memory space instead of a disjoint one.

The show up of unified memory enables the memory become more optimizable, the programming efficiency become higher, and the way to make use of GPU become more simple and direct, even the complex data structures can be more widely applied to the device and the code writing. However, according to [18], [21], the performance of UMA is unsatisfactory in reality. [18] stated that although the programming productivity is high due to the on-demand fetching of data, the performance of managed memory is poor which severely restricts its flexibility and adding future optimizations. Even [21] implied that Nvidia Unified Memory removes GPU memory allocation and memory copy by automatically performing them to improve programmability, but can result in performance degradation for many workloads.

In this paper, we will make an investigation to evaluate the performance of utilizing UMA from an application's rather than benchmark's perspective, which is the execution time of the computer simulation of ECG based on the whole-heart model. In view of the simulation time, that is, the timestep determines the scale of computational burden, we choose two cases with different scale: one is, set a ECG simulation of 50-ms as a evaluating of small migrate data; and the other, set a ECG simulation of 600-ms as a evaluating of big migrate data. More details of this investigation will be described in section IV.

C. The computer simulation of ECG based on the whole-heart model

In order to diagnose and quantitatively analyze all the abnormalities of heart and eventually exhibit a noninvasive diagnostic method for heart disease, the so-called forward research of electrocardiogram and backward research of electrocardiogram emerge. The computer simulation of ECG belongs to the field of forward research of ECG. Based on a heart model, it can calculate and conclude the body surface potential from the cardiac electrical activity. Note that, compared with previous anatomical experiments or physical researches, it has significant advantage in terms of economy and security, and it is undoubtedly the foundation of the backward research of ECG.

1) The whole-heart model: The whole-heart model mentioned by this paper is the Wei-Harumi model [1], which introduced comprehensive electrophysiological characteristics and the anisotropic fiber cells. This model is capable of a number of the simulation of pathological characteristics, such as myocardial infarction and arrhythmia. It is based on an inclined three-dimensional coordinate system, and the angle between each coordinate axis is 60 degrees. All models consist of approximately 50,000 cell elements and coordinate axes are discretized into evenly spaced layers with 1.5 mm units and the three dimensions each has 56, 56, 90 layers [22].

2) The computer simulation of ECG: According to [6], the process of a computer simulation of ECG can be described as the following three stages:

1) Simulate the propagation of cardiac electric excitation based upon the Huygens principle;
2) Calculate the ECG potentials on the volume conductor, which is simulated in the whole heart model;
3) Keep the calculation results of ECG potentials.

Wei D has established a geometric model based on the actual heart in the previous study [23], which can be abstracted as a volume conductor. The torso surface, the epicardial surface, the left endocardial surface and the right endocardial surface were divided into 344, 278, 307 and 1002 node, and 684, 552, 610 and 2002 triangles in this volume conductor [22]. We will employ the boundary element method to calculate all values of ECG potentials in all of these nodes and triangles in this reasonable artificial model.

In the first stage, every cell in the volume conductor of the Wei-Harumi model can be stimulated according to the Huygens principle which is called heart electric excitation propagation. After that, certain cells become current dipole sources, and then the potentials of body surfaces will be effected by dipole sources in the entire volume conductor. Therefore, the most important issue in ECG simulation computing is the second stage. In this stage, you need to locate every current dipole source stimulated after the propagation of cardiac electric excitation, and calculate the potential values produced by every current dipole source in four blocks' surfaces on human body [7]. The total number of triangles is about 3800. At last, all the changing of potentials of the four surfaces will be recorded to draw a curve, which can be observed and studied on the influence of clinical disease or drugs compared to clinical electrocardiogram data.

III. Methodology

A. A Metric Method of Energy Efficiency

Taking the words too literally, the raw formula of energy efficiency, η_E, can be termed as below according to [24], [25]:

$$\eta_E = \frac{Pe}{C}, \tag{1}$$

where

$$Pe = \frac{W}{T_E}, \tag{2}$$

where

$$T_E = T_C + T_{IO} + T_N. \tag{3}$$

On the one side, the performance, Pe of Equation 1, is a complete concept for measuring the quality of applications. The theoretical formula of Pe is shown in Equation 2, in which the performance will be counted by: divide overall workload of the application by overall application execution time, that is, the wall time. In general, the wall time of application execution consists of the time of computing, the time of IO during the data migration between disk and memory, and the time of network communication if application runs in a multi-devices environment. Equation 3 expresses the formal representation of T_E.

As Table I shows, it is essential to say that the unit of performance can be expressed as $GFLOPS$, because the defined W just means workloads, that is, the number all float-point operations during the time of application execution. Thus, the Pe can be represented as the number of float-point operations per second.

On the other side, the C in Equation 1 on behalf the superficial meaning of overall energy consumption of applications. Considering the applicable meaning of C, it is a controversial concept which can be interpreted as P with $Watt$ units or Q with $Kalvin$ units, due to the fact that the consumption of applications may consist of power consumption, temperature consumption, land occupation, even and costing consumption. In general, the P is used as the measurement factor of overall energy consumption of the application. Note that, one $Watt$ equals to one $Joule$ divides by one second, also equals to one $Volt$ times one $Ampere$, according to Joule's law in a pure resistance situation, as Equation 4 expresses below.

$$\begin{aligned} C &= P \\ &= U \times I. \end{aligned} \tag{4}$$

Based on all above formulas, we can easily infer that the energy efficiency, η_E, can be expressed as shown in Equation 5.

$$\eta_E = \frac{W}{T_E \times P}. \tag{5}$$

Where the unit of η_E is $FLOPS/Watt$ or $Pe/Watt$. The result is coherent with a previous study [26], where the standard performance evaluation corporation (SPEC) released the $SPECpower\_ssj^{©}$ 2008 benchmark suite to measure the power and performance characteristics of server-class computer equipment, which called the $Performance - per - Watt$.

In this thesis, in particular, in order to make a qualitative evaluation on the energy efficiency easily and directly, a simplified dependency metric of energy efficiency is shown in Equation 6.

$$\eta_E \propto \frac{W}{T_E \times P}. \tag{6}$$

Where the η_E is negatively correlated with two factors: T_E and P. While, the W in this paper can be regarded as a constant count in our measurement, because of the energy efficiency only cares qualitative evaluation rather than quantitative evaluation. In another word, the metric of Pe of the application in this study only be related to one factor, the T_E, based on the above hypothesis. Note that, the word "energy efficient" mentioned in the rest of this thesis's experiments is saying the same meaning as high energy efficiency.

B. Optimization Strategy of Execution Time-Oriented

Based on Equation 6 of section III-A, assuming that the P remains the same, when the T_E decreases, the quotient which is Energy Efficiency will accordingly increase. Thus, we decided to employ some parallelization strategies into programming, in order to make the T_C becomes smaller. Since the T_N of Equation 3 is not in view in stand-alone mode, the T_E will become smaller, that is to say, the application will become more energy efficient.

1) Parallelization in the Level of CPU: The CPU of TK1 board possesses 4-Plus-1 physical cores, which can reach a highest frequency of 2.3GHz. In order to fully exploit the computing capacity of TK1's CPU, we determine to realize a multi-thread optimization by invoking OpenMP API in a multi-thread way. OpenMP enable us to parallel accelerate the computing of applications, so as to reduce the computational time, that is to say, improve the performance of applications.

The usage of OpenMP is described in detail in our previous work [6], there is not too much explanation here except the following contents. The reason why we turn on 4 main cores of TK1's CPU is that the 5th core is just a companion core and there is no need to use it in a multi-thread case. In addition, the *#pragma* are added to indicate the parallel area with multi-thread in the code. Taking the simulation computation of ECG as example, *#pragma omp parallel* opened up the multi-thread mode in *ECG_inf()* function and *ECG_bd()* function during the application execution time.

2) Parallelization in the Level of GPU: TK1 board, with a Nvidia Kepler GPU, which contains 192 CUDA cores. Objectively, GPU is more suitable than CPU for performing a task, since it's data are high intensive and no disjoint to my best knowledge. Hence we choose the most suitable section in ECG simulation computation's code to implement the parallelization with the aid of CUDA API.

According to [7], in the whole ECG simulation computation procedure, $ECG\_inf()$ function took about 99% computational time, that is to say, the computation tasks of $ECG\_inf()$ function took a great majority of the whole computation tasks. The pseudo code of $ECG\_inf()$ function is displayed in Figure 2, where the I, J, K respectively represents the maximum value of three coordinate dimensions in the whole-heart model, and $ECG\_inf\_single()$ function is responsible for the ECG calculation of a single dipole. In addition, [6] has proved that the calculating of current dipole sources in ECG simulation computing is independent in three layers' loop between adjacent timesteps. As a result, we decide to employ the CUDA optimizing strategies into $ECG\_inf()$ section.

In other words, we invoke the CUDA API to sufficiently exploit the computing capacity of CUDA core in order to parallel accelerating the $ECG\_inf()$ process. The pseudo

TABLE I
SYMBOLS AND DEFINITIONS

Symbol	Definition	Unit
η_E	The energy efficiency of the application	$GFLOP/Watt$
Pe	The performance of the application	$GFLOPS$
C	The overall energy consumption during the time of application execution	$Watt$
W	The overall workload, that is, all float-point operations during the time of application execution	$Flop$
T_E	The overall wall time of application execution	$Second$
T_C	The computing time of application execution	$Second$
T_{IO}	The input/output time of application execution	$Second$
T_N	The network communication time of application execution	$Second$
P	The power consumption during the time of application execution	$Watt$
Q	The thermal consumption during the time of application execution	$Joule$
U	Voltage	$Volt$
I	Current	$Ampere$

$ECG\_inf()$
▷ Calculating ECG potentials in an infinite medium.

```
1  BEGIN
2  arrayInit();
3  ▷ Initialize the array.
4  for k ← 0 to K
5       do for j ← 0 to J
6            do for i ← 0 to I
7                 do if dipole(i,j,k) != 0
8                    ▷ Verify the excited cell is or isn't a dipole source.
9                    ECG_inf_single();
10                   ▷ Calculate the value of potential produced by dipole(i,j,k).
11                   end if
12              end for
13         end for
14 end for
15 END
```

Fig. 2. Pseudo code of ECG_inf() in the serial program

code of $gpuECG\_inf()$ function is given in Figure 3. It enunciates a whole process of the computing of potentials produced by dipole source after GPU level's parallelization. It is clear to see that $gpuECG\_inf\_single()$ function can be executed in a CUDA kernel to complete the computation of ECG potentials, on condition that the cell pass the verification of $dipole(i, j, k)$ function in every loop. Generally, the thread 0 is responsible for GPU and the remaining threads will execute the remaining tasks in a hybrid parallel programming which combines OpenMP and CUDA. Hence, we must detect the id number of current thread to decide in which the next task should be loaded.

As shown in Figure 3, the flow is: firstly, copy all needed data from host memory to device memory; secondly, call $gpuECG\_inf\_single()$ function to complete the ECG potentials' computation of each dipole; finally, copy all results of potentials from device memory back to host memory. Note that, we used the TMA API in this GPU section. As we can see in, there are two explicit memory copies exist so that the overhead of memory copy can not be ignored.

As a measurement, we try a new memory access API which derives from unified memory in TK1 board, means to reduce the overhead of memory copy. In the last section, we have described that how to use the new CUDA API to take advantage of unified memory. Figure 4 illustrates the pseudo code of $gpuECG\_inf()$ but with the UMA in a new memory architecture. Although this novel way reduces the overhead of memory copy in a level of abstraction, it is still a transparent procedure in developers' view, which means the challenge is that make clear what the unified memory exactly do during the data migration. The effect on our experiments will be analyzed in section IV.

3) Optimization in the Level of Scheduling Algorithm: Crucially, this paper makes use of the load prediction dynamic static integrated scheduling algorithm(LPDS) [7], in which proposed by Shen et al., for the better parallelism computation of the computer simulation of ECG. It is necessary to know the major advantage of this scheduling algorithm, which lies in three aspects: firstly, it achieves load balance of all tasks; secondly, it successfully reduces the overhead of normal dynamic scheduling; thirdly, it adequately exploits the computing capacity of hybrid environment. The previous

```
gpuECG_inf()
▷ Calculating ECG potentials in an infinite medium with parallelization.
 1  BEGIN
 2  malloc();
 3  ▷ Allocate memory in the host for storing data.
 4  cudaMalloc();
 5  ▷ Allocate memory in the device for storing data.
 6  memcpyHosttoDevice();
 7  ▷ Transfer the data from host to device.
 8  for k ← 0 to K
 9        do for j ← 0 to J
10              do for i ← 0 to I
11                    do if dipole(i,j,k) != 0
12                       ▷ Verify the excited cell is or isn't a dipole source.
13                       gpuECG_inf_single();
14                       ▷ Call the kernel to calculate the potentials produced by dipole(i,j,k).
15                       end if
16                 end for
17           end for
18 end for
19 memcpyDevicetoHost();
20 ▷ Transfer the data from device to host.
21 cudaFree();
22 free();
23 END
```

Fig. 3. Pseudo code of ECG_inf() in the parallel program with traditional memory access

```
gpuECG_inf()
▷ Calculating ECG potentials in an infinite medium with parallelization.
 1  BEGIN
 2  cudaMallocManaged();
 3  ▷ Allocate memory in the unified space for storing data.
 4  for k ← 0 to K
 5        do for j ← 0 to J
 6              do for i ← 0 to I
 7                    do if dipole(i,j,k) != 0
 8                       ▷ Verify the excited cell is or not a dipole source.
 9                       gpuECG_inf_single();
10                       ▷ Call the kernel and use the parameter in the unified space to calculate.
11                       end if
12                 end for
13           end for
14 end for
15 cudaDeviceSynchronize();
16 ▷ Synchronize the values stored in unified space whatever belong to host or device.
17 free();
18 END
```

Fig. 4. Pseudo code of ECG_inf() in the parallel program with unified memory access

work [7], [8] has concluded that the LPDS is more efficient than traditional dynamic scheduling in most of load predictable problems on a desktop-level hybrid architecture.

In this thesis, we transplant the LPDS into a embedded Linux system to realize a parallelism acceleration, which enable us to make good use of the hybrid computational

capacity of ARM-based CPU and Nvidia GPU on TK1 board. The simplified pseudo code of main program based on an adapted LPDS is shown in Figure 5, where also shows the combined hybrid programming mode with OpenMP and CUDA. Note that, some functions are listed as below:

$propagation()$: The propagation of cardiac electric excitation.

$malloc()$: The allocation of private space for CPU.

$cudaMalloc()$: The allocation of private space for GPU.

$loadPredictandSort()$ and $setDeque()$: The summation of the computational amount of all dipoles of every timestep, and then the calculated amount will be sorted into a deque in a descend order, which form a pair with their timestep [8]. The variable $head$ and $tail$ in $setDeque()$ represents the head and the tail of this deque, (e.g., $head$ means the timestep owns the most heavy computational task).

The variable $gpustaic$ and $cpustaic$ is the number of allocated tasks by means of static scheduling of LPDS in the initial stage.The variable $gpudynamic$ and $cpudynamic$ is the number of allocated tasks by means of dynamic scheduling of LPDS in the second stage.

$\#pragma\ omp\ parallel$: The declaration of OpenMP parallel program section.

$cudaFree()$ and $free()$: The release of data space in their memory.

To explain the scheduling algorithm clearly, it can be divided into two stages. In the first stage, static scheduling stage, thread 0 catches a chunk tasks from the $head$ to the $gpustatic$ of the deque, and calls $gpuECG\_inf()$ to execute a CUDA kernel in GPU. Accordingly, the other threads catch a chunk tasks from the $tail$ to the $cpustatic$ of the deque, and call $ECG\_inf()$ to execute CPU computation.

In the second stage, dynamic scheduling stage, thread 0 catches a task from the $gpudynamic$ which equals to $gpustatic$ in the previous stage, and calls $gpuECG\_inf()$ to execute a CUDA kernel in GPU, with the value of $gpudynamic$ pluses one automatically. In the meantime, a ready thread of the remaining threads catches a task from the $cpudynamic$ which equal to $cpustatic$ in the previous stage, and calls $ECG\_inf()$ to complete the computation task in CPU, with the $cpudynamic$ minuses one automatically. Until the value of $gpudynamic$ equal to the value of $cpudynamic$, the whole computation tasks will be executed completely.

main program of ECG Simulation Computation

```
1  BEGIN
2  propagation();
3  loadPredictandSort(workload);
4  setDeque(head, gpustaic, cpustaic, tail);
5  malloc();
6  cudaMalloc();
7  #pragma omp parallel
8       tid = omp_get_thread_num();
9            if tid == 0
10           then
11             memcpyHosttoDevice();
12             gpuECG_inf (head to gpustaic);
13             memcpyDevicetoHost();
14           else
15             ECG_inf(tail to cpustaic);
16           end if
17      gpudynamic = gpustaic;
18      cpudynamic = cpustatic;
19      while (gpudynamic ! = cpudynamic)
20           do if tid == 0
21           then
22             gpuECG_inf(gpudynamic);
23             gpudynamic = gpudynamic + 1;
24           else
25             ECG_inf(cpudynamic);
26             cpudynamic = cpudynamic − 1;
27           end if
28      end while
29 end #pragma omp parallel
30 cudaFree();
31 free();
32 saveResults();
33 END
```

Fig. 5. Pseudo code of the ECG simulation computation based on an adapted LPDS and parallelizations

C. Optimization Strategy of Power Consumption-Oriented

According to [15], the power consumption of computer systems with modern desktop-scale GPUs is comparatively larger. It can reach to a few tens even one hundred watts. Since the power consumption of TK1 board only takes several watts in an idle mode, it becomes the best optimizable object in our experiments theoretically.

The TK1 board possesses 4-Plus-1 physical cores and one on-chip integrated GPU, and offers many mechanisms to control the power to support applications, which includes two CPU clusters, frequency scaling of CPU and GPU, GPU offloading, and others. The developers are free to adjust the CPU and GPU core frequency, select the active CPU clusters and control the number of active cores to adjust the power consumption of the board [27]. According to Equation 6 of section III-A, assuming the P decreases when the T_E remains the same, the quotient, in other words, the η_E will undoubtedly increase.

Therefore, in order to realize an energy efficient computing on the computer simulation of ECG, we intend to fully take advantage of the steerable features of on-board. As we all know, the board has two clusters of the CPU cores: turning on 4 main cores to get higher performance with higher power consumption or turning on 5th companion Cortex core to get lower power consumption and gear generation with lower performance [27]. In general, TK1 board's CPU is setting in cluster of 4 main cores. In this mode, when the system reaches a low utilization state, the hardware and operating system will cooperate to migrate processes and threads off of the 4 main CPU cores, and move them to the 5th CPU core [28]. This mechanism is called sleep-mode. However, if developers constrainedly control the number of active cores and core frequencies when CPU is in busy runtime, the performance will be affected obviously, with power consumption's variation. Hence, we better manually

set the number of active cores and core frequencies in the idle time of CPU, while we just leave it alone to automatic adjustment in most cases.

The GPU has a same character in TK1 board but only in adjusting of frequency. In this paper, we adopt the strategy of sleep mode on CPU and GPU, which enable us to decrease the power consumption of the board. Whilst, in terms of the synthetic factors of energy efficiency, we make a cross combination for adjustable mechanism of board, which means the best situation will be launched in after the following contrast: 4 main CPU cores cluster or the 5th companion CPU core cluster; the regular 4 main CPU cores cluster or the manual 4 main CPU cores cluster; the regular GPU frequency or the constrained GPU top frequency. The result of power consumption and performance after employing cross combination situation on TK1 board will be presented in section IV.

IV. Experimental Result and Discussion

In order to evaluate the typical performance and power consumption of TK1 board, we setup an ordinary PC with desktop-level GPU as a contrast. The PC is equipped with Intel Core 2 Quad CPU Q8200, 2.33GHz, 6GB host memory and a GeForce GT 705 card, regarded as platform(a). The TK1 board with a Quad-Core 2.3GHz ARM Cortex-A15 CPU(ARMv7-A architecture), a 5-th low power companion Cortex core, and one Kepler architecture GPU running at 852 MHz(128 CUDA cores), paired with 2 GB of LP-DDR3 RAM, is regarded as platform(b). Each Cortex-A15 core has 32 KB L1 instruction and 32 KB L1 data caches. 4-core cluster has 2 MB of shared L2 cache.

The operating system was installed for the x86 architecture with Linux Ubuntu 14.04.1 LTS(GNU/Linux 3.10.0) and for ARM32bit architecture with Linux Ubuntu 14.04 for Tegra(GNU/Linux 3.10.40). The program of these two platforms was compiled both with GCC v4.8 and NVCC form CUDA Toolkit6.5.

For purpose of cross-contrast, we provide two input scale: the first, the computer simulation of a 50-ms ECG is regarded as a small data scale, which is called scale(a); the second, the computer simulation of a 600-ms ECG is regarded as a big data scale, which is called scale(b).

As far as my knowledge is concerned, The platform(b) uses low voltage 12V power supply with direct current(DC), but platform(a) uses 220V power supply with alternating current(AC). Therefore, the value of power of platform(a) can be measured by using an external device −− Kill-A-Watt, which can represent the level of power imprecisely. But for platform(b), the device used in platform(a) is suitable for measuring computer's power draw with ac mode, but can not measure that of embedded board precisely. In addition, the onboard power consumption has correlation with onboard voltage, onboard current and onboard leakage current [25]. So, in next experiments for platform(b), a device named Agilent E3634A was used to provide a 12V DC power supply. This device could supply the constant DC output voltage. Meanwhile, it contains an internal automatic sampling function. Note that, the each value of power consumption or execution time or speedup of all the results are an average value by averaging 10 values after measuring 10 times.

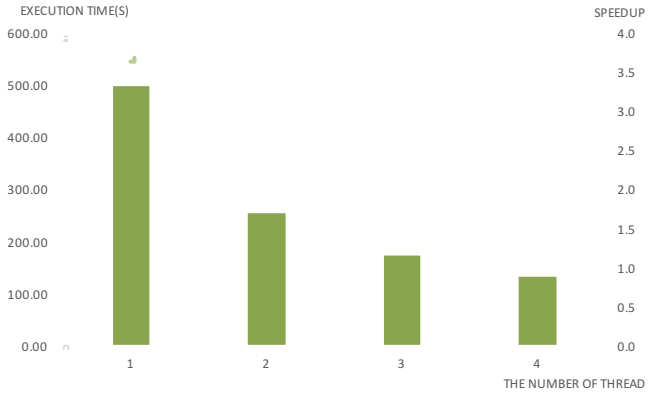

Fig. 6. The parallel results of execution time and speedup of 1 to 4 threads by OpenMP on platform(b) for scale(b)

It is necessary to say, until the computation has begun, fluctuations in power usage on all machines are relatively small [29]. And under our experimental conditions, the average power consumption of TK1 board under stabilization without applications running is 4.9 watts, and that of the PC is 56.6 watts.

A. Evaluation of Performance and Power Consumption of TK1 board

This subsection displays four experiments to study about the performance and power consumption in different situations, by changing the active number of CPU cores and the frequency of CPU or GPU of TK1 board. This set of experiments are special indication for platform(b) itself.

The first experiment is conducted with CPU loading and GPU offloading mode of scale(b) simulation, which is executed by OpenMP to realize the CPU-level's parallelization. The performance and speedup of 1 to 4 threads of CPU is given in Figure 6.

Since the CPU of TK1 board is a multi-core CPU, OpenMP enables multi-thread to handle workload in a parallel way. Hence, as you can see, the result just fit the basic rule of parallelization.

The second experiment is conducted to investigate the details of how to set CPU clusters on platform(b). As Table II(a) shows, when developers set in a multi-threads mode, taking parallelization with 4 threads as an example, the rank of overall behavior is: situation(a) $>$ situation(b) $>$ situation(c). The situation(a) is a regular mode in 4 main CPU cores mode, that is a default on-board mode with sleep-mode. The situation(b) is a manual mode in 4 main CPU cores mode, that is manually turned on 4 active CPU cores with no sleeping in applications runtime by developers. And the situation(c) is a manual mode in the 5th companion core mode for power conservation. As you know, situation(b) surely increases the computing speed and decreases the application execution time, but leads to a sharper increase of power consumption compared with situation(a). And situation(c) dramatically reduces the power consumption unless you do not care the performance anymore. What's more, the computer simulation can not be executed under situation(c), since the scale of input data is beyond the processing capacity of 5th CPU core, as the blank of Table II(a) says.

TABLE II
THE EXECUTION TIME AND POWER CONSUMPTION OF SIMULATION OF SCALE(A) AND SCALE(B) ON PLATFORM(B) IN DIFFERENT SITUATION

(a) Situation: (a) vs (b) vs (c), all work with four threads

Situation	Scale(a)	Scale(b)
(a).T_E	10.57s	132.88s
(a).P	9.09w	12.01w
(b).T_E	10.48s	132.65s
(b).P	9.54w	12.28w
(c).T_E	34.3s	none
(c).P	6.00w	6.19w

(b) Situation: (d) vs (e), both work with single thread and single GPU

Situation	Scale(a)	Scale(b)
(d).T_E	15.15s	138.43s
(d).P	7.04w	7.95w
(e).T_E	12.02s	108.24s
(e).P	7.96w	8.72w

(c) Different memory access way, both work with four threads and single GPU

Memory Access Way	Scale(a)	Scale(b)
UMA.T_E	9.97s	97.22s
UMA.P	9.35w	12.07w
TMA.T_E	8.85s	114.09s
TMA.P	9.33w	12.15w

The third investigation is in order to figure out how to set GPU frequency fitly on platform(b). As Table II(b) shows, taking parallelization with single thread and single GPU as an example, when developers set in a GPU-loading mode, the rank of overall behavior is: situation(e) $>$ situation(d). Note that, the situation(e) is a constrained GPU mode of top frequency, that is to say, developers manually adjust GPU frequency to the top frequency at the idle time in the beginning of the procedure. And the situation(d) is a regular GPU mode of alterable frequency, in which the frequency will change when the workload loaded into GPU changes. The result illustrates that, under situation(e), the power consumption just rise a little, while the performance rise much more. Hence we decide to take the constrained GPU mode of top frequency into the following experiments.

The fourth experiment is conducted to evaluate the performance and power consumption of platform(b) with a comparison between TMA and UMA. According to [18], the performance of UMA varied greatly based on program design, kernel intensity and data migration scale. Therefore, in term of different data scale of scale(a) and scale(b), this experiment adopts the situation(a) for CPU and situation(e) for GPU, to launch an investigation from an application's perspective. It turns out, on the one hand, the performance of UMA is better than TMA under big data scale input. Because the way of data migration of UMA is more suitable for big data, so long as the data scale comes within the memory capacity of CPU-GPU integrated memory. On the other hand, under the small data scale input, due to the two explicit data copies of TMA are not insignificant, the performance of UMA is worse than the TMA. Furthermore, the power consumption of taking the two different memory access way is almost the same value, as shown in Table II(c).

TABLE III
THE EXECUTION TIME AND POWER CONSUMPTION OF SIMULATION OF SCALE(A) AND SCALE(B) ON DIFFERENT PLATFORM

(a) Platform: (a) vs (b), both work with four threads in situation(a)

Platform	Scale(a)	Scale(b)
(a).T_E	9.97s	156.68s
(b).T_E	10.57s	132.88s
(a).P	66.28w	71.81w
(b).P	9.09w	12.01w

(b) Platform: (a) vs (b), both work with single GPU in situation(e)

Platform	Scale(a)	Scale(b)
(a).T_E	10.14s	142.32s
(b).T_E	15.15s	145.43s
(a).P	73.79w	76.13w
(b).P	7.96w	8.72w

B. Comparison of Performance and Power Consumption between TK1 board and PC

The second set of experiments is conducted on platform(a) and platform(b). In the first experiment, the computer simulation of ECG of scale(a) and scale(b) is implemented by OpenMP with four threads and situation(a) provided by the last set experiment, which means a regular mode with four main cores. The results of application execution time and power consumption are exhibited in Table III(a).

From Table III(a) we can find that, taking scale(b) as an example, the platform(b) is about 1.18X faster than platform(a) in this mode. And the power consumption of platform(b) is 5.98X lower than platform(a). As you can see, the performance of TK1 board is slightly higher than that of the PC, but the power draw of TK1 board is much less than that of the PC. To explain why, the reason is that the TK1 board is just devised for high intensive computing and low power applications, but the PC has miscellaneous parts so that is power draw is wasted too much in other pathway.

The application execution time and power consumption on platform(a) and platform(b) with single GPU are examined in the second experiment. Here the GPU is in the situation(e) which means single constrained GPU mode top frequency. The results of simulation of scale(a) and scale(b) are shown in Table III(b), it is no surprise to see that the power consumption of TK1 board is much lower than that of the PC. And it is not difficult to find that, the performance of GPU of TK1 board is slightly lower than that of the PC, because of the TK1 board surely does well in plenty of scientific applications, in term of the excellent computing power, but which is still far behind that of most desktop-level GPU.

C. Comparison of Energy Efficiency between TK1 board and PC

According to the experimental results of the aforementioned experimentations, we determine to adopt situation(a) for the parallelization of four threads, situation(e) and UMA way for the parallelization of GPU, into the last one experiment, which parallels with four threads and single GPU

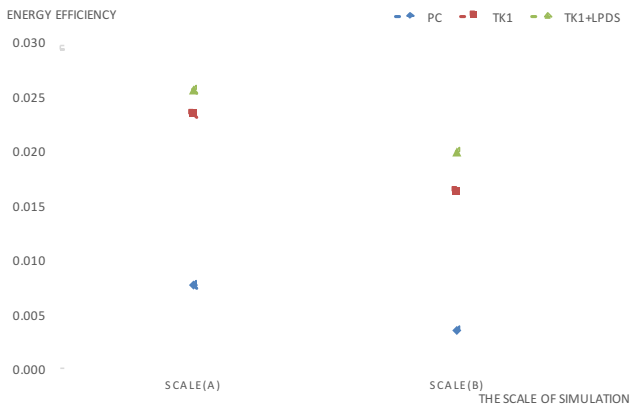

Fig. 7. The results of energy efficiency of simulation of scale(a) and scale(b) with four Threads and single GPU on platform(a) and platform(b)

for the simulation of scale(a) and scale(b). And the adapted LPDS will be added to make a contrast test.

According to Equation 6 in section III-A, we calculate results of energy efficiency for two simulation scales in following three cases: case(a): the parallel computing of ECG simulation on the PC with the four threads and single GPU; case(b): the parallel computing of ECG simulation on TK1 with the four threads and single GPU; case(c): the parallel computing of ECG simulation on TK1 with the four threads and single GPU plus the adapted LPDS. As the results illustrated in Figure 7, it is quite encouraging to find out the rank result of energy efficiency is: case(c) $>$ case(b) $>$ case(a). On the one side, the energy efficiency of small scale input is better than that of big scale input. When the workload increases multi-fold, the time of applications execution just has a small rise, owing to the strong computing ability. On the other side, the energy efficiency of case(c) under the simulation of scale(a) does not increases as fast as that of scale(b), due to the performance of ECG simulation computing can not be effected significantly by applying the LPDS, especially under small input data. The reason is, the small scale input means that there are not too much dipoles exist in the simulation timesteps. As a result of, the predication and sorting of LPDS will not work effectively.

V. Conclusions and Future Work

In this paper, we have examined application execution time, power consumption and energy efficiency with the demonstration of the computer simulation of ECG based on the whole-heart model, with a comparison between a TK1 board and an ordinary PC. According to our experiments and analyses, it is easy to see that the performance of TK1 board's CPU can reach the same level of that of the PC, and the TK1 board's GPU performs closely to that of the PC. Particularly, the energy efficiency of TK1 board is 3X-5.67X higher than that of the PC, and even can be 3.25X-6.67X higher than that of the PC with the aid of the adapted LPDS, based on the metric methods.

In a word, we can draw a conclusion that the TK1 board can provide an easy available environment with good performance, low power consumption and especially high energy efficiency under the proposed effective approaches. This paper adequately exploits the energy efficiency of the miniature embedded device for computer simulation of ECG, and offers a convenient and portable way for clinical staffs, medical experimental workers and other researchers, in their scientific applications.

Speaking of the future work, we would like to illuminate how exactly the API works and the data migrates in UMA way. Last but not least, in the future, we put our experiments into Jetson Tegra X1 or Tegra X2 SoC board, so the load prediction dynamic static integrated scheduling algorithm might be improved further, with better optimization under larger input data.

References

[1] D. Wei, "Whole-heart modeling: progress, principles and applications," *Progress in Biophysics and molecular Biology*, vol. 67, no. 1, pp. 1753–5166, 1997.

[2] X. Zhu, D. Wei, and O. Okazaki, "Computer simulation of clinical electrophysiological study," *Pacing and Clinical Electrophysiology*, vol. 35, no. 6, pp. 718–729, 2012. [Online]. Available: http://dx.doi.org/10.1111/j.1540-8159.2012.03379.x

[3] H. Zhu, Y. Sun, G. Rajagopal, A. Mondry, and P. Dhar, "Facilitating arrhythmia simulation: the method of quantitative cellular automata modeling and parallel running," *Biomedical engineering online*, vol. 3, no. 1, p. 29, 2004.

[4] M.-C. Trudel, B. Dubé, M. Potse, R. M. Gulrajani, and L. J. Leon, "Simulation of qrst integral maps with a membrane-based computer heart model employing parallel processing," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 8, pp. 1319–1329, 2004.

[5] D. Sato, Y. Xie, J. N. Weiss, Z. Qu, A. Garfinkel, and A. R. Sanderson, "Acceleration of cardiac tissue simulation with graphic processing units," *Medical & biological engineering & computing*, vol. 47, no. 9, pp. 1011–1015, 2009.

[6] W. Shen, D. Wei, W. Xu, X. Zhu, and S. Yuan, "Parallelized computation for computer simulation of electrocardiograms using personal computers with multi-core cpu and general-purpose gpu," *Computer methods and programs in biomedicine*, vol. 100, no. 1, pp. 87–96, 2010.

[7] W. Shen, Z. Luo, D. Wei, W. Xu, and X. Zhu, "Load-prediction scheduling algorithm for computer simulation of electrocardiogram in hybrid environments," *Journal of Systems and Software*, vol. 102, pp. 182–191, 2015.

[8] W. Shen, L. Sun, D. Wei, W. Xu, X. Zhu, and S. Yuan, "Load-prediction scheduling for computer simulation of electrocardiogram on a cpu-gpu pc," in *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*. IEEE, 2013, pp. 213–218.

[9] J. Kepner, "Hpc productivity: An overarching view," *International Journal of High Performance Computing Applications*, vol. 18, no. 4, pp. 393–397, 2004.

[10] K. R. Stokke, H. K. Stensland, C. Griwodz, and P. Halvorsen, "Energy efficient video encoding using the tegra k1 mobile processor," in *Proceedings of the 6th ACM Multimedia Systems Conference*. ACM, 2015, pp. 81–84.

[11] H. Kim, S. Kim, N. Van Helleputte, A. Artes, M. Konijnenburg, J. Huisken, C. Van Hoof, and R. F. Yazicioglu, "A configurable and low-power mixed signal soc for portable ecg monitoring applications," *IEEE transactions on biomedical circuits and systems*, vol. 8, no. 2, pp. 257–267, 2014.

[12] S. Raj, G. P. Chand, and K. C. Ray, "Arm-based arrhythmia beat monitoring system," *Microprocessors and Microsystems*, vol. 39, no. 7, pp. 504–511, 2015.

[13] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1374–1381, 2011.

[14] V. P. Nikolskiy, V. V. Stegailov, and V. S. Vecher, "Efficiency of the tegra k1 and x1 systems-on-chip for classical molecular dynamics," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, July 2016, pp. 682–689.

[15] E. Serrano, J. G. Blas, A. Verza, and J. Carretero, "Simulation platform for x-ray computed tomography based on low-power systems," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2015, pp. 416–426.

[16] J. A. Belloch, A. Gonzalez, R. Mayo, A. M. Vidal, and E. S. Quintana-Ortí, "Evaluating the potential of low power systems for headphone-based spatial audio applications," *Procedia Computer Science*, vol. 51, pp. 191–200, 2015.

[17] R. S. Santos, D. M. Eler, and R. E. Garcia, "Performance evaluation of data migration methods between the host and the device in cuda-based programming," in *Information Technology: New Generations*. Springer, 2016, pp. 689–700.

[18] W. Li, G. Jin, X. Cui, and S. See, "An evaluation of unified memory technology on nvidia gpus," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 1092–1098.

[19] Y. Ukidave, D. Kaeli, U. Gupta, and K. Keville., "Performance of the nvidia jetson tk1 in hpc," in *Proceedings of the 2015 IEEE International Conference on Cluster Computing*, ser. CLUSTER '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 533–534. [Online]. Available: http://dx.doi.org/10.1109/CLUSTER.2015.147

[20] Nvidia, "Unified memory in cuda 6," https://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/, accessed August 25, 2017.

[21] R. Landaverde, T. Zhang, A. K. Coskun, and M. Herbordt, "An investigation of unified memory access performance in cuda," in *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*. IEEE, 2014, pp. 1–6.

[22] X. Zhu, D. Wei, and H. Wang, "Simulation of intracardial potentials with anisotropic computer heart models," in *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*. IEEE, 2005, pp. 1071–1074.

[23] D. Wei, *Whole Heart Modeling and Computer Simulation*. Boston, MA: Springer US, 2005, pp. 81–117. [Online]. Available: https://doi.org/10.1007/978-0-387-49963-5_3

[24] H. Abdulbaset, A. Mohammad, and K. R. Raveendra, "Analysis of energy and spectral efficiency in urban shadowing environment," *IAENG International Journal of Computer Science*, vol. 43, no. 2, pp. 237–244, 2016.

[25] D. Castells-Rufas, A. Saà-Garriga, and J. Carrabina, "Energy efficiency of many-soft-core processors," *CoRR*, vol. abs/1601.07133, 2016. [Online]. Available: http://arxiv.org/abs/1601.07133

[26] Spec, "Specpower_ssj2008," https://www.spec.org/benchmarks.html#power, accessed August 28, 2017.

[27] V. Vecher, V. Nikolskii, and V. Stegailov, *GPU-Accelerated Molecular Dynamics: Energy Consumption and Performance*. Cham: Springer International Publishing, 2016, pp. 78–90.

[28] J. E. Stone, M. J. Hallock, J. C. Phillips, J. R. Peterson, Z. Luthey-Schulten, and K. Schulten, "Evaluation of emerging energy-efficient heterogeneous computing platforms for biomolecular and cellular simulation workloads," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, vol. 00, May 2016, pp. 89–100. [Online]. Available: doi.ieeecomputersociety.org/10.1109/IPDPSW.2016.130

[29] K. Keipert, G. Mitra, V. Sunriyal, S. S. Leang, M. Sosonkina, A. P. Rendell, and M. S. Gordon, "Energy-efficient computational chemistry: Comparison of x86 and arm systems," *Journal of Chemical Theory and Computation*, vol. 11, no. 11, pp. 5055–5061, 2015.