Loadbalancing on Parallel Heterogeneous Architectures: Spin-image Algorithm on CPU and MIC

Ahmed Eleliemy[1] Mahmoud Fayze[2] Rashid Mehmood[3] Iyad Katib[3] Naif Aljohani[3]

[1]HPC Group, University of Basel, Basel, Switzerland, ahmed.eleliemy@unibas.ch
[2]Fujitsu & Computer Science, Ain-Shams University, Cairo, Egypt, Mahmoud.Fayez@ts.fujitsu.com
[3]High Performance Computing Center, King AbdulAziz University, Jeddah, Saudi Arabia, {rmehmood, iakatib, nraljohani}@kau.edu.sa

Abstract

Loadbalancing of computational tasks over heterogeneous architectures is an area of paramount importance due to the growing heterogeneity of HPC platforms and the higher performance and energy efficiency they could offer. This paper aims to address this challenge for a heterogeneous platform comprising Intel Xeon multi-core processors and Intel Xeon Phi accelerators (MIC) using an empirical approach. The proposed approach is investigated through a case study of the spin-image algorithm, selected due to its computationally intensive nature and a wide range of applications including 3D database retrieval systems and object recognition. The contributions of this paper are threefold. Firstly, we introduce a parallel spin-image algorithm (PSIA) that achieves a speedup of 19.8 on 24 CPU cores. Secondly, we provide results for a hybrid implementation of PSIA for a heterogeneous platform comprising CPU and MIC: to the best of our knowledge, this is the first such heterogeneous implementation of the spin-image algorithm. Thirdly, we use a range of 3D objects to empirically find a strategy to loadbalance computations between the MIC and CPU cores, achieving speedups of up to 32.4 over the sequential version. The LIRIS 3D mesh watermarking dataset is used to investigate performance analysis and optimization.

Keywords: heterogeneous architectures, MIC, spin-image algorithm, loadbalancing, performance analysis

1 Introduction

High performance computing (HPC) systems rely on concurrent, parallel and distributed computing technologies and resources to provide much larger memories and computational power than is possible with a general-purpose computer. HPC systems have traditionally been used to solve large problems arising from engineering and science. They are now being increasingly utilized in many other areas including business, economy and social sciences. Early HPC systems have mainly been homogeneous. However, the heterogeneity of modern computing systems is on the rise due to the increasing demands for higher performance and energy efficiency. Loadbalancing of computational tasks on homogeneous platforms is generally considered a difficult problem; it is an even bigger challenge when it comes to high performance heterogeneous systems.

Two most common options for accelerator units in heterogeneous platforms are GPGPUs (General-Purpose Computation on Graphics Processing Unit) and MICs (Intel Many Integrated Core Architecture). GPUs comprise thousands of cores and could offer very high memory bandwidth and computation throughput. In the last decade, a lot of research work has been done to speed up different algorithms using such architectures (Bautista Gomez et al., 2014). Therefore, GPUs have become one of the main accelerators in many HPC facilities. Although they are low-power, low-cost and massive parallel execution units but unfortunately, they have their own limitations (Shukla and Bhuyan, 2013). They are not compatible with existing x86 C, C++, and FORTRAN source codes. To use these architectures, we have to rebuild different codes from scratch and this could inhibit HPC users.

The MIC architecture was introduced by Intel in 2012 (Rahman, 2013). It is low-power, low-cost and massive parallel execution unit Like GPGPUs. However, it is a massively parallel architecture that is compatible with x86 applications. Programming models like Pthreads, MPI and OpenMP can be used without any code modifications (Utrera et al., 2015). Due to the programming simplicity and compatibility of the MIC architecture, it is the main accelerator unit in many modern HPC facilities (IntelPR). According to the Top500 list (http://www.top500.org/lists/) of most powerful supercomputers in the world, many fastest supercomputers are powered by Intel Xeon Phi coprocessors and Intel Xeon processors. There is a high potential of using MIC coprocessors beside Intel Xeon processors (Faheem and Konig-Ries, 2014). Therefore, we have chosen MICs alongside CPUs in this study of a heterogeneous platform.

This paper presents an empirical approach for achieving loadbalancing and optimum performance from a heterogeneous system comprising Intel Xeon multi-core processor and MIC. The approach is investigated using a case study of spin-image algorithm (Johnson, 1997). This algorithm is selected because it is being used in a wide range of important applications including 3D Database Retrieval Systems (Assfalg et al., 2004), Face Detection (Choi and Kim, 2013), Object Recognition (Johnson and Hebert, 1999), Object Categorization (Eleliemy et al., 2013), 3D map registration (Mei and He, 2013), and registration algorithm for LiDAR 3D point cloud models (He and Mei, 2015). The spin-image algorithm is well-known for its high computational complexity and is considered an essential bottleneck in many fields.

The paper makes three contributions. (1) A parallel spin-image algorithm (PSIA) has been introduced that achieves the speedup of 19.8 on 24 CPU cores. The process of parallelizing spin-image algorithm into a set of independent tasks which can be optimally scheduled between Intel Xeon processor and Intel MIC coprocessor is depicted and described. (2) results from a hybrid version of PSIA for a heterogeneous platform comprising CPU and MIC have been provided. (3) A strategy to empirically find an optimal loadbalancing of computations between the MIC and CPU cores has been introduced using a range of 3D objects. Speedups of up to 32.4 over the sequential version have been reported. We have used a range of objects from LIRIS 3D mesh watermarking dataset for performance analysis and optimization in all our experiments. To the best of our knowledge, no hybrid implementation of the spin-image algorithm on CPUs and MICs has been reported in the literature. The study of loadbalancing as a methodology for spin- over CPU and MIC is also novel.

This paper is organized as follows. Section 2 describes the sequential spin-image algorithm. Section 3 provides the dependency analysis of the spin-image algorithm and the proposed parallel spin-images algorithm. Section 4 provides details of the heterogeneous platform and dataset we have used for experiments in this paper. Results from the experiments and their analysis are presented along with the details of the loadbalancing strategy. Finally, in Section 5, further research challenges for loadbalancing on heterogeneous platforms and a number of directions for future work are given.

2 Spin-Image Algorithm

Spin-image is an algorithm that converts a 3D shape into a set of 2D images as shape descriptors. It was introduced in (Johnson, 1997). The main concern regarding the use of the spin-image algorithm is its computational complexity, especially with the increase of depth cameras' resolution. According to the spin-image algorithm, a spin-image can only be generated at any point with known normal (oriented point). Generated spin-image can be viewed as a paper that rotates around point normal while other points touch and stick to it.

$$i = \frac{(W/2) - n \cdot (x-p)}{B} \tag{1}$$

$$j = \frac{\sqrt{||x-p||^2 - (n.(x-p))^2}}{B} \tag{2}$$

Equations 1 and 2 show how to calculate spin-image at an oriented point (p). This equation calculates two indices i and j, where the generated spin-image should be incremented by one. In fact, instead of incrementing the point at index i and j, four indices $[i,j]$, $[i,j+1]$, $[i+1,j]$ and $[i+1,j+1]$ are incremented with values $(1-a)(1-b)$, $(1-a)b$, $a(1-b)$ and ab, respectively. Equations 3 and 4 can be used to calculate the values of a and b. This process is called smoothness of spin-images.

$$a = a - i * binsize \tag{3}$$

$$b = \beta - j * binsize \tag{4}$$

The equations show that i and j are affected by two parameters; Image-width (W) and Bin-size (B). However, these parameters affect only the quality of generated spin-image and do not have impact on spin-image generation time. Moreover, smoothness process is related to the quality of the generated spin-image. Therefore, these parameters will be considered as constants and smoothness process will be ignored in this work. The below pseudo code shows the generation process spin-images .

```
procedure CalcSpinImages
W = ImageWidth
B = BinSize
Define SpinImages As List
for each p in Mesh
  Define SpinImage[W * W]
  n = normal of P
  for each x in Mesh
     i = $((W/2) - n \times (x-p))/B$
     j= $\sqrt{||x-p||^2 - (n.(x-p))^2}/B$
     SpinImage[i,j]+=1
  end for
  SpinImages.add(SpinImage)
end for
return SpinImages
end procedure
```

3 The Proposed Parallel Spin-Images Algorithm

Figure 1 shows the spin-image generation process for a given 3D Mesh. This process contains different levels of calculations. Each level contains some of the dependent and/or independent tasks. For example, Let M be a certain 3D Mesh, S is the set of all M oriented points, P_1, P_2 and $P_n \in$ S. Spin-image calculation at P_1 is totally independent of P_2, P_3 and P_n. However, at P_1, we have to scan P_2, P_3 till P_n to find different indices i, j to increment spin-image at these indices. Such calculation cannot be parallelized directly because it may be required to increment the same spin-image cell $[i,j]$ simultaneously.

Figure 2 shows the proposed PSIA. In PSIA, we have two levels of parallelism. At the first level, we calculate spin-images at different points P_1, P_2 and P_n simultaneously. While at the second level, for each point pair (P_x,

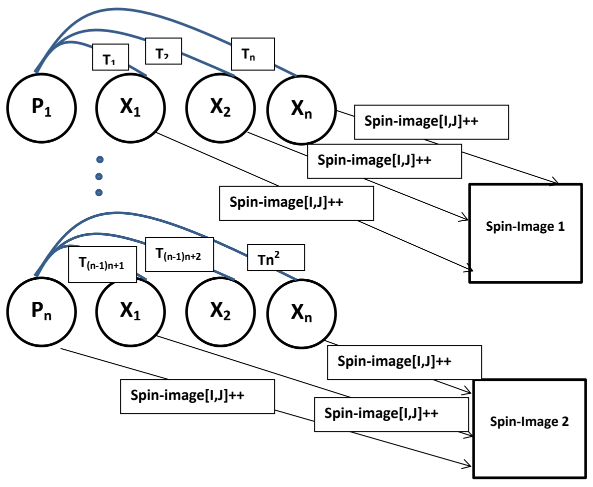

Figure 1. Generating Spin-images for a given 3D Mesh

P_1) , (P_x, P_2) and (P_x, P_n), we have a temporary spin-image that we call *Partial Spin-image*. Finally, we add all partial images to get the spin-image at point P_x. Figure 2 shows the process for computing all spin-images concurrently using partial spin-images.

4 Results and Analysis

4.1 Platform Specification

The experiments have been carried out on the *Aziz* supercomputer. Aziz supercomputer is Fujitsu made and is able to deliver peak performance of 230 teraflops. It has a total of 11,904 cores in 496 nodes, where each node comprises dual socket Intel Xeon E5-2695v2 12-core processor running at 2.4GHz. 380 of these nodes contain 96 GB memory each, while the rest of the 112 nodes contain 256 GB each, making up a total of 66 TB memory in the system. The system also contains 2 NVidia Tesla K20 GPU equipped compute nodes with 48 cores and 2 Intel Phi 5110P co-processor equipped compute nodes with 48 cores. Aziz was ranked number 360 in the June 2015 Top500 competition, currently it is at number 491 (November 2015).

The platform (part of the Aziz supercomputer) we have used for the experiments consists of 2 Intel processors E5-2695v2 each has 12 Cores (2.4GHz), 96GB RAM, Intel Xeon Phi Coprocessor 5110P (1053GHz, 60Cores). The code is written in C and compiled for Linux (CentOS 6.4) using Intel parallel studio XE 2015 version 15.0.

4.2 Experimental Data

Objects from LIRIS 3D mesh watermarking dataset (Wang et al., 2010) have been used for the performance analysis and optimization. This dataset has been selected due to its dense objects. Table 1 shows the number of vertices for each object.

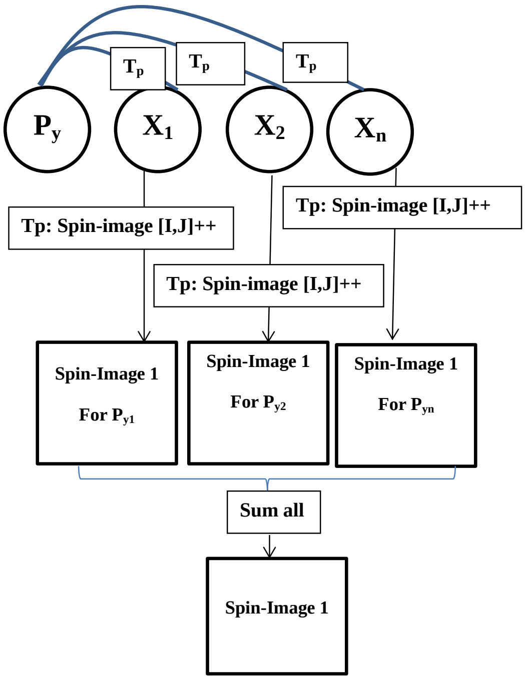

Figure 2. Proposed Parallel Spin-Image Algorithm

Table 1. 3D Objects in 3D Mesh Watermarking Dataset

Object Label	*Number of vertices*
Ramesses	826266
Horse	112642
Venus	100759
Rabbit	70658
Crank	50012
Dragon	50000
Hand	36619
Bunny	34835
Casting	5096
Cow	2904

4.3 Results

It can be seen from our discussions in Sections 2 and 3 that the execution time of spin-images generation process for any object is the key measure of the computational complexity. As mentioned in section 2, the complexity of such process is $O(n^2)$. Therefore, any increase in the number of object points leads to significant increase in the total time of spin-image generation process. In order to illustrate that fact, only percentage of 1% spin-images for each object has been generated. In figure3, the time to spin-images generation for *Bunny* and *Rabbit* is 0.73 and 2.881 seconds respectively, while their sizes are 34835 and 70658 points respectively. Simply, *Rabbit* is almost 2 times in size comparing with *Bunny*, but its spin-images generation takes almost 4 times as what it takes in the *Bunny* object. Also, it is the same for both *Horse* and *Bunny*, time to generate spin-images for *Horse* is 9 times as *Bunny*, while its

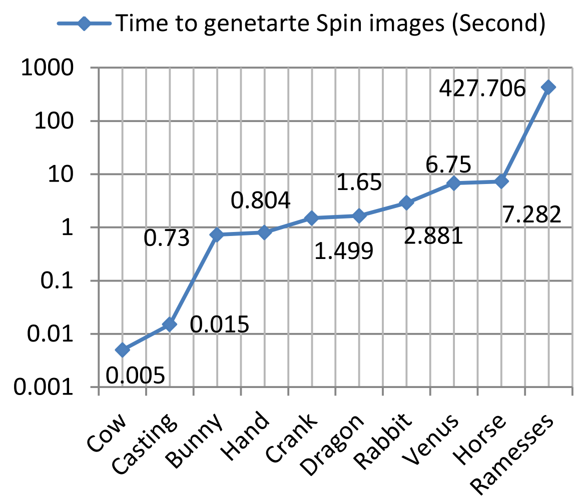

Figure 3. Sequential implementation of spin-image algorithm which number of generated spin-images equals to 1% of object size

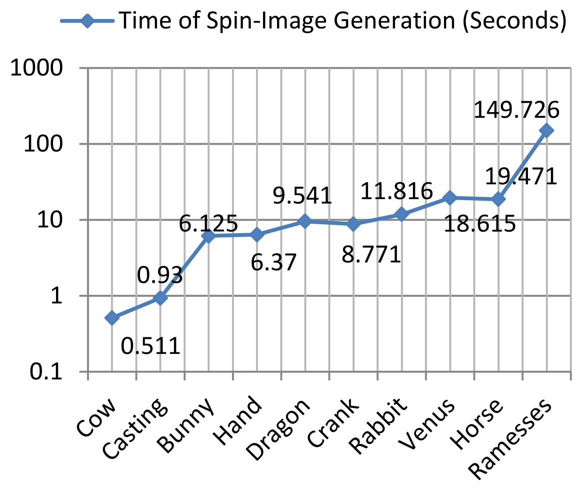

Figure 4. Sequential implementation of spin-image algorithm and the number of generated spin-images equals to 2904

is 3 times as *Bunny*. Some approaches from Object Recognition field as well as Object Categorization like (Hegazy, 2016) try to avoid such problem by avoiding spin-image generation at each point of a 3D object, it only generate spin-images at certain points which will not affect correct recognition or categorization rate. However, the problem remains specially for dense objects like LIRIS objects.

In Figure 4, the total number of generated spin-images is 2409 images per each object, this number represents the common maximum number of spin-images between all objects, because *Cow* Object is the smallest object and it contains 2409 vertices. Figure 4 shows that dense objects like *Ramesses*, *Horse*, and *Venus* consume at huge time such it can not be used in real-time systems. For example, *Horse*, *Venus*, and *Rabbit* objects take 19.471, 18.615, and 11.816 seconds respectively. Moreover *Ramesses* object takes 149.726 seconds.

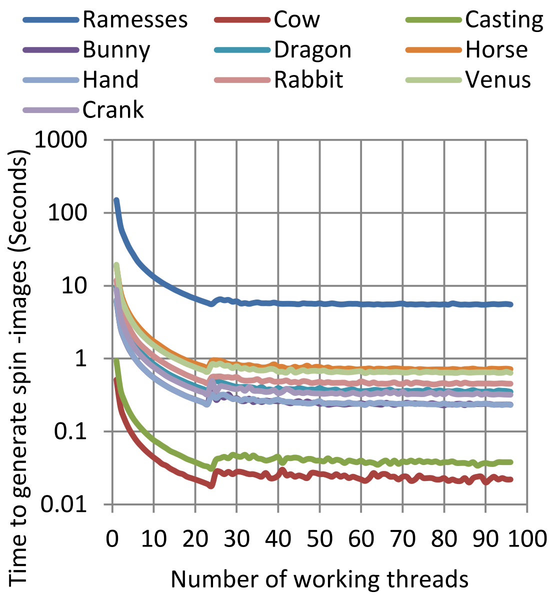

Figure 5. Performance of PSIA over 24-cores CPU and the number of generated spin-images is 2904

In fact, the *Ramesses* object is 284.5 times in size compared to the smallest object in the dataset. Therefore, it will always be a challenge to generate all of its spin-images, taking into consideration that all points of any LIRIS object are oriented points, which means that for *Ramesses* we need to generate 826266 spin-images.

Figure 5 shows the performance of our enhanced PSIA algorithm for multi-core CPUs. Note that As discussed in section 3, PSIA is a parallel version of the original spin-image algorithm where openMP threads are used to perform the parallel tasks. The figure shows that there is a reverse relation between the number of OpenMP threads and the total run-time. Simply, increasing the number of working OpenMP threads will decrease the total run-time. However, there is a turnover point on OpenMP threads axis at 24 thread, where this relation is not valid. Such relation violation is due to the physical characteristics of the used hardware, as mentioned before this experiment runs over 24-Cores CPU.

Another experiment has been conducted to investigate the scalability of the proposed parallel SIA algorithm using another shared memory architecture like Xeon phi (MIC technology). As mentioned before, we have an Intel Xeon Phi card that consists of 60 cores. There is two important feature in such hardware; First each core can run 4 concurrent threads. Second, there is no need to change any code, it is compatible with x86 processors. In other words, same code can be recompiled. Figure 6 shows same performance which means scalability of the algorithm, but it also shows the turnover point changed to be around 230 core which is logically due to the physical characteristics of the hardware platform.

Actually due to the type of spin-image calculations which all are floating point operations, results of running

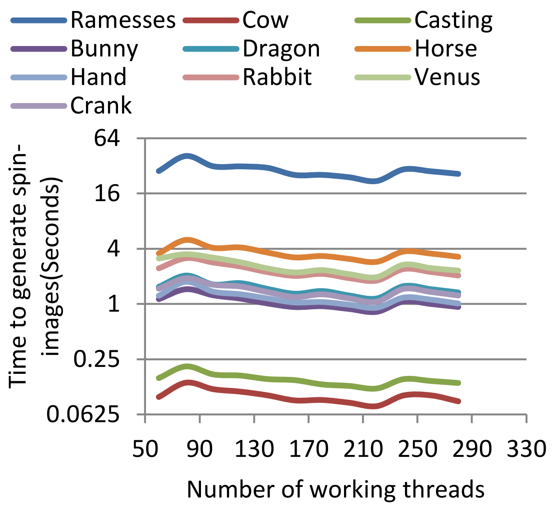

Figure 6. Performance of PSIA over 60-cores MIC accelerator and the number of generated spin-images is 2904

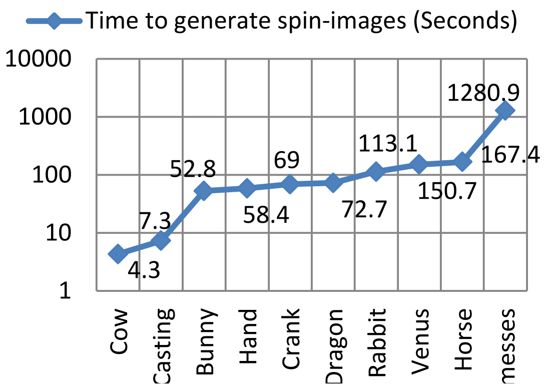

Figure 7. Sequential implementation of spin-image algorithm runs over MIC accelerator and the number of generated spin-images is 2904

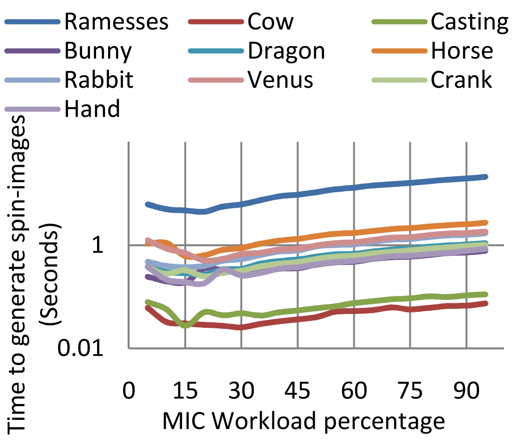

Figure 8. Performance for the hybrid implementation of PSIA at different workload distributions between CPU and MIC-cores

Table 2. The values for MIC Workload Ratios for the given objects

Object Name	R_1	R_2	*MIC Workload %*
Ramesses	0.232666667	0.095204348	29.04
Cow	0.00075	0.000334783	30.86
Casting	0.001291667	0.000521739	28.77
Bunny	0.017833333	0.003573913	16.69
Dragon	0.019291667	0.004995652	20.56
Horse	0.038833333	0.012530435	24.39
Hand	0.0135	0.003965217	22.70
Rabbit	0.02325	0.007830435	25.19
Venus	0.034875	0.00853913	19.67
Crank	0.020583333	0.004608696	18.29

PSIA over MIC is not promising as running over CPU, however, the total time to generate 2904 spin-images is reduced comparing to sequential spin-image algorithm. In order to complete the image, the original sequential spin-image has been compile to MIC architectures. Figure 7 shows that how the original sequential spin-image algorithm behave for MIC. Therefore, it expected that PSIA over MIC is not promising as running over CPU.

It can be observed from the results so far that the PSIA performance over CPU is better than its performance over MIC. However, using both CPU and MIC together should give better results than using any of them individually. The remaining question is how to divide the workload between CPU and MIC. According to (Faheem and Konig-Ries, 2014), because all PSIA tasks are identical with almost no dependency, workload for MIC can be calculated as follows.

Let R_1 is the ratio between the number of generated spin-images and run-time over 24 CPU cores, and R_2 is the ratio between the number of generated spin-images and run-time over 230 MIC cores. Formally, R_x = (the number of generated spin-images / total run time) / the number of working threads, where x represent the available architectures ($CPU = 1$ and $MIC = 2$) The MIC Workload Ratio W of MIC to the total workload could be calculated as $R_2/(R_1+R_2)$. The number of generated spin-images is same for both R_1 and R_2, and therefore, it could be excluded from the workload Ratios (W). The values for MIC Workload Ratios (as well as R_1 and R_2) for various objects are given in Table 2. The table shows that the lowest MIC Workload Ratio (Column 4) is for the Bunny object (16.69%), while the highest is for the Cow object (30.86%). Consequently, based on the results given in Table 2, we can conclude that the maximum performance can be obtained by allocating around 23% (average of the values in Column 4) of the total workload to MIC and the remaining part to CPU. This is not the exact optimum value, rather a value around which optimum performance can be found.

To investigate the workload distribution between MIC and CPU further, we performed another set of experiments. Figure 8 shows the results for a range of workload distributions between MIC and CPU. The first result (the leftmost) is for the case where MIC is given 5% of the workload, while CPU gets 95% of the workload. The MIC workload is increased in steps of 5% until it reaches 95%, where CPU gets a 5% of the total load. Based on the numerical values of the results depicted in the figure, the optimum MIC Workload Ratio is dependent on the object, and falls between 10% to 30%. Although further analysis of such behavior is required, our preliminary explanation is as follows. According to Equations (1), (2), (3), and (4), the values of i, j require 4 memory accesses — $[i, j], [i, j+1], [i+1, j]$ and $[i+1, j+1]$ — in order to increment the value of the spin-image. These memory accesses are avoided when the values j, j are outside the spin-image boundary. However, there are no guarantees that the MIC in the assigned workload may require, or may not require, these 4 memory accesses. This memory access behavior will be examined further in our future work.

5 Conclusions and future work

Improving loadbalancing of computational tasks over heterogeneous architectures is an area of paramount importance. This paper aimed at addressing the loadbalancing problem for a heterogeneous platform comprising CPUs and MICs. The approach proposed in this paper was investigated through a case study of the spin-image algorithm, selected due to its computationally intensive nature and a wide range of applications including 3D database retrieval systems and object recognition. The paper made three contributions. A parallel spin-image algorithm (PSIA) was introduced and its implementation achieved the speedup of 19.8 on 24 CPU cores. Results from a hybrid implementation of PSIA were presented. We empirically found a strategy to loadbalance computations between the MIC and CPU cores, achieving speedups of up to 32.4. Objects from LIRIS 3D mesh watermarking dataset were used to provide performance analysis and optimization.

The proposed PSIA on the heterogeneous platform can replace the original spin-image for many different applications as mentioned in Section 1. Also, the proposed PSIA is scalable. It can run over a different number of cores equal to N, such that N is less than or equal to the number of generated spin-images. The use of the PSIA algorithm is recommended for the case where the objects are dense. Moreover, the proposed PSIA hybrid implementation allows real-time generation of spin-images.

The results and analysis of our approach for loadbalancing on heterogeneous platforms show great promise. However, there are some concerns regarding the proposed PSIA that need further investigation. For example, PSIA is based on creating partial spin-images, which means that higher memory resources are required. Further investigation in needed to find out the memory profile of the proposed PSIA algorithm compared to the original version? The loadbalancing strategy needs to be investigated further with a wider range and size of objects. Analytical or heuristic formulations need to be devised. We need to investigate various cases where, for instance, one of the computing resources does not have sufficient memory to take its workload: is it better to distribute the workload based on the characteristics of the work or the amount of work? Finding answers to these questions forms our motivation for the future work. We also plan to add the energy efficiency dimensions to this work. We plan to look at optimizing the loadbalancing strategy against energy efficiency, memory profile and computational performance.

Acknowledgment

This work is supported by the King AbdulAziz University's High Performance Computing Center (`http://hpc.kau.edu.sa`). The spin-image computations described in this paper are performed on the *Aziz* supercomputer, part of the HPC Center.

References

J. Assfalg, G. D'Amico, A. Del Bimbo, and P. Pala. 3D content-based retrieval with spin images. In *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, volume 2, pages 771–774, June 2004. doi:10.1109/ICME.2004.1394314.

L. Bautista Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, K. Pattabiraman, P. Rech, and M. Sonza Reorda. GPGPUs: How to combine high computational power with high reliability. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–9, March 2014. doi:10.7873/DATE.2014.354.

K. S. Choi and D. H. Kim. Angular-partitioned spin image descriptor for robust 3D facial landmark detection. *Electronics Letters*, 49(23):1454–1455, Nov 2013. ISSN 0013-5194. doi:10.1049/el.2013.1577.

A. Eleliemy, D. Hegazy, and W.S. Elkilani. MPI parallel implementation of 3D object categorization using spin-images. In *Computer Engineering Conference (ICENCO), 2013 9th International*, pages 25–31, Dec 2013. doi:10.1109/ICENCO.2013.6736471.

H. M. Faheem and B. Konig-Ries. A new scheduling strategy for solving the motif finding problem on heterogeneous architectures. *International Journal of Computer Applications*, 101(5), September 2014.

Y. He and Y. Mei. An efficient registration algorithm based on spin image for Lidar 3D point cloud models. *Neurocom-puting*, 151, Part 1:354 – 363, 2015. ISSN 0925-2312. doi:http://dx.doi.org/10.1016/j.neucom.2014.09.029.

D. Hegazy. Symmetric multi-processing 3d object categorization model using a spin-point curvature selection strategy. *Egyptian Computer Science (ECS) Journal*, 40(1):73–83, January 2016.

A. Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997.

A.E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5):433– 449, May 1999. ISSN 0162-8828. doi:10.1109/34.765655.

Y. Mei and Y. He. A new spin-image based 3D map registration algorithm using low-dimensional feature space. In *Information and Automation (ICIA), 2013 IEEE International Conference on*, pages 545–551, Aug 2013. doi:10.1109/ICInfA.2013.6720358.

Intel Newsroom. *Intel Delivers New Architecture for Discovery with Intel Xeon Phi Coprocessors*. Available via https://www.bayesfusion.com/ [accessed November 8, 2016].

R. Rahman. *Intel® Xeon Phi^{TM} Coprocessor Architecture and Tools. The Guide for Application Developers*. Apress Media, ISBN13: 978-1-4302-5926-8, 2013.

S.K. Shukla and L.N. Bhuyan. A hybrid shared memory heterogeneous execution platform for PCIe-based GPDGUs. In *High Performance Computing (HiPC), 2013 20th International Conference on*, pages 343–352, Dec 2013. doi:10.1109/HiPC.2013.6799140.

G. Utrera, M. Gil, and X. Martorell. In search of the best MPI-OpenMP distribution for optimum Intel-mic cluster performance. In *High Performance Computing Simulation (HPCS), 2015 International Conference on*, pages 429–435, July 2015. doi:10.1109/HPCSim.2015.7237072.

K. Wang, G. Lavoué, F. Denis, A. Baskurt, and X. He. A benchmark for 3D mesh watermarking. In *Proc. of the IEEE International Conference on Shape Modeling and Applications*, pages 231–235, 2010.