International Journal of Scientific Engineering and Science
Volume 4, Issue 1, pp. 20-24, 2020. ISSN (Online): 2456-7361

# RSA Implementation on a Hetrogeneous Cluster Architecture using CPUs, GPUs, and MIC

Sanaa A. Sharaf
Department of Computer Science, King Abdulaziz University, Jeddah, Saudi Arabia
Email address: ssharaf @ kau.edu.sa

***Abstract***— *There is a great degree of difficulty in addressing problems of significant computational complexity such as the Routing and Spectrum Assignment (RSA) problem on heterogeneous clusters comprising CPUs, CUDA GPUs and Intel Many Core (MIC) architecture. A scheduling approach is employed in the present study to address the RSA issue on a heterogeneous cluster. Such an approach relies on the speed of heterogeneous architectures for task planning. More specifically, the study seeks to use appropriate parallel computing paradigms (e.g. MPI, OpenMP, CUDA) on separate architectures to address the problem and afterwards refer to the speed of task solving to determine how many tasks each architecture should be allocated. Findings show that the proposed approach is more effective than pure CPU-based application in terms of overall execution time, as made apparent by relative comparison of the number of nodes possessed by each architecture. Furthermore, as very high CPU numbers can remove the effect of employing architectures of high speed, there is an indirect correlation between the speedup and the elevated number of CPUs. On the other hand, system use and, implicitly, the number of tasks that can be introduced in the system at the same time are affected by the use of a high number of CPUs in a single task to attain a notable speedup factor. Therefore, code amendment for the purpose of task allocation to the architectures is discussed by the study as well.*

***Keywords***— *Heterogeneous Architectures, Routing and Spectrum Assignment Problem, Task Scheduling.*

## I. BACKGROUND

Standard multicore microprocessors (CPUs), graphics processor units (GPUs), and Intel many integrated core (MIC) architectures are the main components of contemporary high-performance computing (HPC) clusters. Consequently, the computational resources in one cluster vary significantly. Therefore, the developers' competencies and the employed parallel computing paradigm determine to a considerable extent the process of writing a code capable of exploiting the heterogeneous resources effectively. Furthermore, the way in which heterogeneous systems perform depends greatly on the scheduling approach used to manage the lack of uniformity.

The present study aims to address the issue of Routing and Spectrum Assignment (RSA), which is among the problems of great computational complexity in the network field. Brute force algorithm will be employed to address this issue on multicore CPUs, on GPU and on MIC, respectively. After computation of the real run time for every one of these architectures, a particular scheduling approach will be applied to improve use of hardware resources and speedup by allocating a suitable workload to the architectures. Furthermore, code amendment to ensure compatibility with the chosen scheduling approach will be discussed as well.

The structure of the remainder of the study is as follows: The RSA problem is the focus of the second part, while the application of the various architectures is discussed in the third part. A succinct presentation of the employed scheduling approach is provided in the fourth part, while the alterations made to the parallel code to satisfy the conditions of the scheduling approach are discussed in the fifth part. Finally, conclusions and recommendations for further research are outlined in the sixth part.

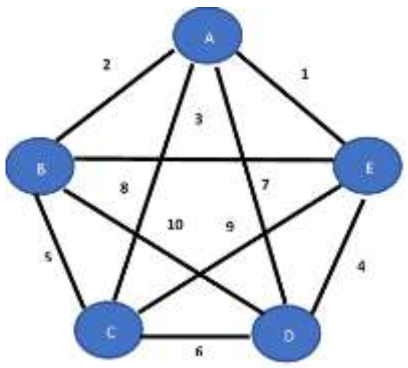
## II. ROUTING SPECTRUM ALLOCATION PROBLEM

The technology showing the greatest potential in next-generation backbone transport networks is the elastic optical network (EON) [1][2]. This type of network is characterized by the separation of the optical fiber frequency spectrum into a large number of narrow frequency slots with spectrum width that does not vary. A channel can be created by any successive slot series and in turn can generate an optical link comprising a route and a channel (i.e. light-path) by switching in the network nodes. Through assignment of the minimum needed bandwidth, EONs permit capacity acquisition [3][4]. A major problem in this regard is RSA, which is geared towards identifying the optimal path and create a light-path within an all-optical channel for existing end-to-end requirements [5]. As an NP-hard optimization issue, RSA requires taking into account certain restrictions in order to make the existing spectrum as efficient as possible. There is a great deal of difficulty in addressing the RSA issue and a number of related algorithms have been recently suggested. For instance, a range of Integer Linear Programming (ILP) models have been considered by some studies [6].

Link- and path-based formulations are the two existing types of ILP formulations, depending on the kinds of variables employed. Link-based formulations require taking into account the whole space of every potential connection among any two network nodes, while path-based formulations are geared towards identifying the ideal solution between a received series of paths that serve as input for every node pair in the network. This latter type of ILP formulations is adopted in the present study, so the inputs of the RSA problems are as follows:

## A. *Network Topology*

A connected graph G(V, A), with V and A respectively representing the set of nodes and the set of directed network links (Fig. 1) [7] .



*G(V)= {A,B,C,D,E}   is a set of nodes*

*A(V)= {1,2,3,4,5,6,7,8,9,10}   is a set of directed links*

Fig. 1. Network Topology of 5 nodes

## B. *Traffic Demands*

These include a traffic demand matrix T=[tsd], with the extent of spectrum necessary for traffic transport between nodes s and d being denoted by tsd [8]. Traffic requirement to graph G if every link is associated with ten frequency slots (FS) is exemplified in Figure 2. Free slots or fragmented frequency (FF) and utilized frequency (UF) are respectively denoted by the white and grey blocks [9][5].

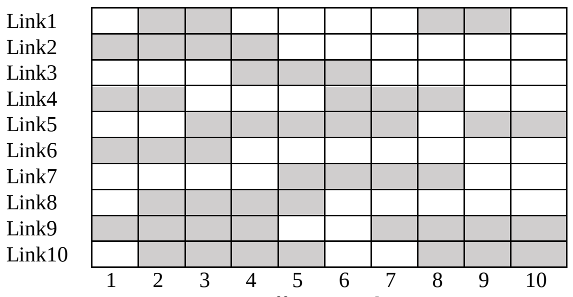


Fig. 2. Traffic Demands

## C. *Routing Paths*

These refer to a series of paths (Ksd) associated with source-destination (s, d). For instance, the route paths between source A and destination D in the network topology shown in Fig. 1 are {{7}, {1,4}, {3,6}, {1,9,6}…}.

Three major conditions are associated with RSA. The first condition is that a contiguous spectrum must be allocated to every requirement (spectrum contiguity). The second condition is that an identical spectrum must be allocated to every requirement along every path link (spectrum continuity). The third condition is that portions of the existing spectrum that do not overlap must be allocated to requirements with a link in common (non-overlapping spectrum) [5][8].

The routing algorithm, which produces every possible routing table, and the spectrum assignment, which examines the traffic demand matrix against every routing table based on spectrum assignment algorithms, are the two components of the RSA issue. The RSA seeks to reduce the number of spectra allocated on any network link as much as possible and to meet every condition.

## III. Application of RSA on Various Architectures

Brute force algorithm will be employed to address this issue on multicore CPUs, on GPU and on MIC, respectively. After computation of the real run time for every one of these architectures, a particular scheduling approach will be applied to improve use of hardware resources and speedup by allocating a suitable workload to the architectures. Furthermore, code amendment to ensure compatibility with the chosen scheduling approach will be discussed as well.

Every experiment is conducted based on Intel Compiler 2017 and Intel MPI V5. CUDA V6, GCC compiler and OpenMPI are the basis of the GPU experiments, while Intel Compiler 2015 and Intel MPI V5 with MIC native mode are the basis of the MIC experiments.

A number of 496 nodes (11,904 cores) represent the compute nodes for performance of enormous parallel tasks as well as small parallel or serial tasks. The number of standard compute nodes (9120 cores) with 96 GB (4 GB/core) is 380, while high-memory compute nodes (2688 cores) with 256 GB (10.6 GB/core) and designed for programs with high memory demands is 112. Two Nvidia Tesla K20 GPGPU ready-compute nodes (2496 CUDA cores for every card) with 96 GB for execution of applications capable of applying GPU-based accelerators are also used. Furthermore, applications capable of implementing accelerators based mainly on MIC are executed based on two Intel Phi 5110P Co-processor ready-compute nodes (120 Xeon phi cores) with 96 GB. Tables I, II and III respectively illustrate the standard architectures of regular CPU node, MIC (Xeon Phi) node and NVIDIA GPGPU (CUDA) node [10].

TABLE I. Regular Cpu-Based Compute Node [10]

| Attribute | Value |
|---|---|
| Architecture | x86_64 |
| CPU op-mode(s) | 32-bit, 64-bit |
| Byte Order | Little Endian |
| CPU(s) | 24 |
| On-line CPU(s) list | 0-23 |
| Thread(s) per core | 1 |
| Core(s) per socket | 12 |
| Socket(s) | 2 |
| NUMA node(s) | 2 |
| CPU MHz | 2399.852 |
| Memory | 96 GB |

TABLE II. XEON PHI Compute Node

| Attribute | Value |
|---|---|
| Total No of Active Cores | 60 |
| Voltage | 897000 uV |
| Frequency | 1052631 kHz |

## IV. Scheduling Strategy

The objective of this study is to minimize the execution time of RSA problem through scheduling it on multiple heterogenous architectures. Therefore, in this section we will investigate the impact of using scheduling strategy described in [11] on assigning tasks to different architectures. Subsequently, we will show how the scheduling strategy will affect the code. Since we have 1610 possible routing tables for 5 nodes complete mesh. Each task will calculate the make-span(score) of one of the routing tables. (Table V) shows the speed differences between architectures when running one task. We simply run the task separately on the architecture and calculate the execution time. This in fact can give us a ratio on which we can decide how many tasks should be assigned to a specific architecture. Specific ratio of tasks that should be assigned to different architectures from a total number of (1610 tasks) is listed in (Table VI). Achieved run times are also shown in Table VI. Comparing the results in Table IV and Table VI gives us an idea about the improvement in the total run time. For example; Using 1 Regular node, 1 CUDA node, and 1 MIC node will reduce the run time from 73911 seconds to 13657 seconds. Since the CUDA and MIC nodes are limited number we can only increase the number of regular nodes as shown on Table VI.

TABLE III. Nvidia cuda compute node [10]

| Attribute | Value |
|---|---|
| CUDA Driver Version / Runtime Version | 6.0 / 6.0 |
| CUDA Capability Major/Minor version number | 3.5 |
| Total amount of global memory | 5120 MBytes (5368512512 bytes) |
| (13) Multiprocessors, (192) CUDA Cores/MP | 2496 CUDA Cores |
| GPU Clock rate | 706 MHz (0.71 GHz) |
| Memory Clock rate | 2600 Mhz |
| Memory Bus Width | 320-bit |
| L2 Cache Size | 1310720 bytes |
| Total amount of constant memory | 65536 bytes |
| Total amount of shared memory per block | 49152 bytes |
| Total number of registers available per block | 65536 |
| Warp size | 32 |
| Maximum number of threads per multiprocessor | 2048 |
| Maximum number of threads per block | 1024 |
| Max dimension size of a thread block (x,y,z) | (1024, 1024, 64) |
| Max dimension size of a grid size (x,y,z) | (2147483647, 65535, 65535) |
| Maximum memory pitch | 2147483647 bytes |

TABLE IV. Results of implementing RSA on CPU, GPU and Xeon Phi

| Trial No. | Platform | Result (seconds) |
|---|---|---|
| 1 | 1 Regular Node (OpenMP) | 73911 |
| 2 | 5 Regular Nodes (MPI + OpenMP) | 14759 |
| 3 | 10 Regular Nodes (MPI + OpenMP) | 7380 |
| 4 | 1 GPU Node (CUDA) | 23280 |
| 5 | 1 MIC Node (Offload Mode) | 59390 |

TABLE V. Architectures speed differences for one task

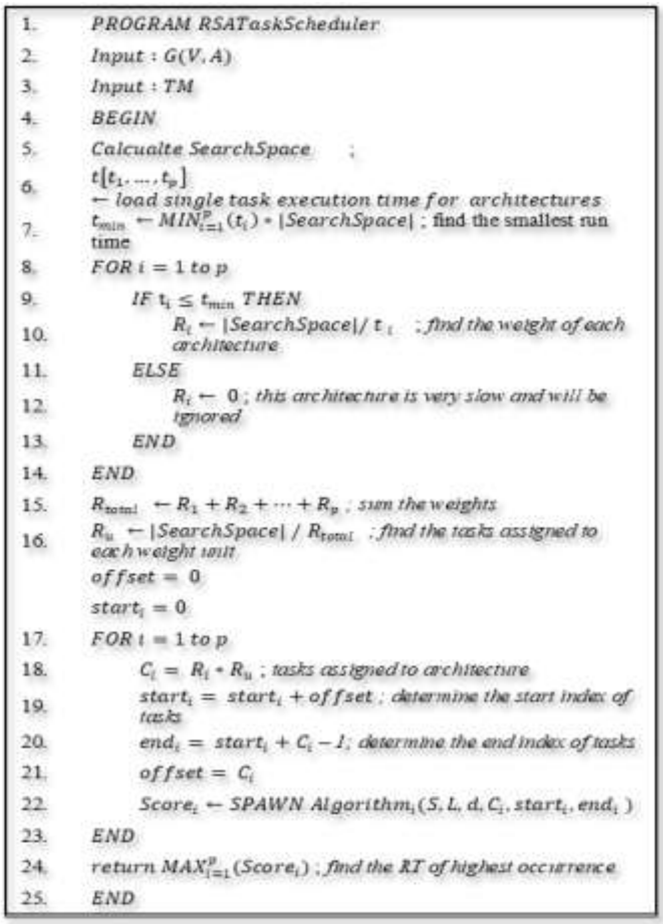
| Architecture | CPU Node | GPU Node | XEON-Phi Node |
|---|---|---|---|
| Task Execution Time (in sec) | 6.72 E-08 | 2.11 E-08 | 5.4 E-08 |

## V. Modifications to the Code

As mentioned earlier, there are two major RSA components, which are respectively concerned with producing every possible routing table and spectrum assignments in keeping with the traffic demand matrix and the routing tables produced. The breadth-first search algorithm is the graph search method employed in this study for the parallel implementation of the produced routing tables. However, any graph search method is suitable. Furthermore, to break down the problem, the RSA Task Scheduler algorithm employed in Fig. 3 is responsible for dividing the problem into chucks each contains several tasks. The main purpose of this study is to distribute the RSA implementation across different architectures to measure the performance, for this reason the chunks will execute on different architecture after ensuring that this architecture will process the chunks in reasonable time and less than processing all chunks on the fastest architecture available.

TABLE VI. Tasks assigned to architectures based on speed

| Platform | CPU Ratio % | CUDA Ratio % | MIC Ratio % | Result (Seconds) |
|---|---|---|---|---|
| 1 Regular Node + 1 CUDA + 1 MIC | 18.41865 | 58.66036 | 22.92099 | 13657 |
| 2 Regular Node + 1 CUDA + 1 MIC | 31.10769 | 49.53642 | 19.3559 | 11496 |
| 4 Regular Node + 1 CUDA + 1 MIC | 47.45365 | 37.783 | 14.76336 | 8768 |
| 8 Regular Node + 1 CUDA + 1 MIC | 64.36415 | 25.62364 | 10.0122 | 5946 |



```
PROGRAM RSATaskScheduler
Input : $G(V, A)$
Input : $TM$
BEGIN
Calcualte SearchSpace ;
$t[t_1, \ldots, t_p] \leftarrow$ load single task execution time for architectures
$t_{min} \leftarrow MIN_{i=1}^{p}(t_i) * |SearchSpace|$ ; find the smallest run time
FOR $i = 1$ to $p$
    IF $t_i \leq t_{min}$ THEN
        $R_i \leftarrow |SearchSpace| / t_i$ ; find the weight of each architecture
    ELSE
        $R_i \leftarrow 0$ ; this architecture is very slow and will be ignored
    END
END
$R_{total} \leftarrow R_1 + R_2 + \cdots + R_p$ ; sum the weights
$R_u \leftarrow |SearchSpace| / R_{total}$ ; find the tasks assigned to each weight unit
$offset = 0$
$start_i = 0$
FOR $i = 1$ to $p$
    $C_i = R_i * R_u$ ; tasks assigned to architecture
    $start_i = start_i + offset$ ; determine the start index of tasks
    $end_i = start_i + C_i - 1$; determine the end index of tasks
    $offset = C_i$
    $Score_i \leftarrow$ SPAWN $Algorithm_i(S, L, d, C_i, start_i, end_i)$
END
return $MAX_{i=1}^{p}(Score_i)$ ; find the RT of highest occurrence
END
```

Fig. 3. *RSATaskScheduler* Pseudo Code

International Journal of Scientific Engineering and Science
Volume 4, Issue 1, pp. 20-24, 2020. ISSN (Online): 2456-7361

Both Fig. 4 and Fig. 5 illustrates the pseudo code for OpenMP and MPI implementations respectively in the first architecture CPU. In Fig. 4, RSA_OMP will take subset of routing tables from start to end and executes in parallel to calculate the score for each. At the end, the maximum score and the best routing table will be returned. RSA_MPI pseudo code presents the division of the search space across the available MPI ranks and find both mpistart and mpiend. Each rank in MPI will use procedure RSA_OMP to perform the search on its subplace.

Fig. 6 and Fig. 7 show the pseudo codes that related to the second architecture GPU which uses CUDA. In Fig. 6, RSA_GPU is the host side for CUDA implementation whereas Fig. 7 shows the actual kernel implementation in RSAKernel. The host module calculates the number of CUDA blocks and assigns the suitable number of threads per block [10].

The final architecture to study in the research is the MIC and Fig. 8 presents the pseudo code for RSA_MIC and it is similar to previous code in MPI and OpenMP.

```
1.  PROGRAM RSA_OMP
2.  Input : subspace(start, …, end)
3.  Input : TM
4.  BEGIN
5.  score ← 0
6.  $score_{max}$ ← 0
7.  $OpenMP Directive
8.  FOR ALL RT x in subspace[start, …, end]
9.      BEGIN
10.         score ← score (x, S)
11.         IF score > $score_{max}$ THEN
12.             $RT_{best}$ ← x
13.             $score_{max}$ ← score
14.         ENDIFR
15.     END
16. return ($score_{max}$, $RT_{best}$)
17. END
```

Fig. 4. *OpenMP* implementation for RSA

```
1.  PROGRAM RSA_MPI
2.  Input : subspace(start, …, end)
3.  Input : TM
4.  BEGIN
5.  RT ← 0
6.  $score_{max}$ ← 0
7.  rank ← MPI Rank
8.  $mpi_{start}$ ← rank * ((end − start)/ MPI_Size)
9.  $mpi_{end}$ ← (rank + 1) * ((end − start)/ MPI_Size)
10. Call MFP_OMP(subspace[$mpi_{start}$, …, $mpi_{end}$], S)
16. return MPI_Reduction(MAX, $score_{max}$)
17. END
```

Fig. 5. *MPI* implementation for RSA

```
1.  PROGRAM RSA_GPU
2.  Input : subspace(start, … ,end)
3.  Input : TM
4.  BEGIN
5.  threads ← 512
6.  blocks ← ceiling( (end − start) / 8 / threads)
7.  score[start, … ,end] ← 0
8.  FOR n 0 to 7
9.      BEGIN
10.         Call RSAKernel<blocks, threads>(S, n * (end-start)/8 , score)
11.     END
13. return $MAX_{i=start}^{end}$ ($score_i$)
14. END
```

Fig. 6. *GPU* implementation for RSA

```
1  Module RSAKernel
2  Input : S[1, …, T]
3  Input : offset
4  Input : TM
5  Output : score[start, …, end]
6  BEGIN
7. thread ← (blockIdx * 256 + threadIdx) + offset
8. score[thread] ← score (thread, S)
9. END
```

Fig. 7. *RSA Kernel* implementation for RSA

```
1.  PROGRAM RSA_MIC
2.  Input : subspace(start, …, end)
3.  Input : TM
4.  BEGIN
5.  score ← 0
6.  $score_{max}$ ← 0
7.  $pragma offload
8.  $OpenMP Directive
9.  FOR ALL RT x in subspace[start, …, end]
10.     BEGIN
11.         score ← score (x, S)
12.         IF score > $score_{max}$ THEN
13.             $RT_{best}$ ← x
14.             $score_{max}$ ← score
15.         ENDIFR
16.     END
17. return ($score_{max}$, $RT_{best}$)
18. END
```

Fig. 8. Pseudo code for MIC implementation for RSA

## VI. Conclusion

Heterogeneous architectures can be used to significantly enhance and speed up the run time of computationally intensive problems with proper scheduling strategy and suitable parallel computing paradigms. Having equivalent or at least a comparable number of different architectures can result in a tangible speedup. Future work may include studying different intelligent algorithms for scheduling and distributing highly computationally algorithm on different architectures to increase the performance. Another research direction could investigate is the power consumption and the effective algorithms that could reduce the power consumption and maintain the performance and Quality of Service (QoS) in general.

## ACKNOWLEDGMENT

The simulations in this work were performed at King Abdulaziz University's High Performance Computing Center (Aziz Supercomputer) (http://hpc.kau.edu.sa).

## REFERENCES

[1] F. Shirin Abkenar and A. Ghaffarpour Rahbar, "Study and Analysis of Routing and Spectrum Allocation (RSA) and Routing, Modulation and Spectrum Allocation (RMSA) Algorithms in Elastic Optical Networks (EONs)," *Opt. Switch. Netw.*, vol. 23, pp. 5–39, Jan. 2017.

[2] J. Simmons, *Optical network design and planning*. Springer International Publishing, 2014.

[3] Y. Hadhbi, H. Kerivin, and A. Wagler, "A novel integer linear programming model for routing and spectrum assignment in optical networks," in *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2019, pp. 127–134.

[4] Y. Wang, X. Cao, and Y. Pan, "A study of the routing and spectrum allocation in spectrum-sliced Elastic Optical Path networks," in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 1503–1511.

[5] H. Alizadeh Ghazijahani, H. Seyedarabi, J. Musevi Niya, and N.-M. Cheung, "Optimized Routing and Spectrum Assignment for Video Communication over an Elastic Optical Network," *arXiv Prepr. arXiv1909.06536*, 2019.

[6] L. F. Delvalle, E. Alfonzo, and D. Pinto-Roa, *EONS: An online RSA simulator for elastic optical networks*. 2016.

[7] I. Olszewski, "Routing and Spectrum Assignment in Spectrum Flexible Transparent Optical Networks BT - Image Processing and Communications Challenges 5," 2014, pp. 407–417.

[8] M. Fayez, I. Katib, G. N. Rouskas, T. F. Gharib, H. Khaleed, and H. M. Faheem, "Recursive algorithm for selecting optimum routing tables to solve offline routing and spectrum assignment problem," *Ain Shams Eng. J.*, Nov. 2019.

[9] F. Lezama, G. Castañón, A. M. Sarmiento, and I. B. Martins, "Differential evolution optimization applied to the routing and spectrum allocation problem in flexgrid optical networks," *Photonic Netw. Commun.*, vol. 31, no. 1, pp. 129–146, 2016.

[10] N. A. H. M. Faheem, B. Koenig-Riez, Mahmoud Fayez, Iyad Katib, "Solving the Motif Finding Problem on a Heterogeneous Cluster using CPUs GPUs and MIC Architectures," *Math. Comput. Sci. Ind.*, pp. 226–232, 2015.

[11] H. Faheem and B. König-Ries, "A New Scheduling Strategy for Solving the Motif Finding Problem on Heterogeneous Architectures," *Int. J. Comput. Appl.*, vol. 101, pp. 27–31, Sep. 2014.