IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 24, 2018, accepted March 28, 2018, date of publication April 9, 2018, date of current version May 16, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2823299

Performance and Power Efficient Massive Parallel Computational Model for HPC Heterogeneous Exascale Systems

M. USMAN ASHRAF, FATHY ALBURAEI EASSA, AIIAD AHMAD ALBESHRI, AND ABDULLAH ALGARNI

Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia

Corresponding author: M. Usman Ashraf (m.usmanashraf@yahoo.com)

This work was supported in part by the Deanship of Scientific Research at King Abdulaziz University, Jeddah, under Grant RG-3-611-38.

ABSTRACT The emerging high-performance computing Exascale supercomputing system, which is anticipated to be available in 2020, will unravel many scientific mysteries. This extraordinary processing framework will accomplish a thousand-folds increment in figuring power contrasted with the current Petascale framework. The prospective framework will help development communities and researchers in exploring from conventional homogeneous to the heterogeneous frameworks that will be joined into energy efficient GPU devices along with traditional CPUs. For accomplishing ExaFlops execution through the Ultrascale framework, the present innovations are confronting several challenges. Huge parallelism is one of these challenges, which requires a novel low power consuming parallel programming approach for attaining massive performance. This paper introduced a new parallel programming model that achieves massive parallelism by combining coarse-grained and fine-grained parallelism over inter-node and intra-node computation respectively. The proposed framework is tri-hybrid of MPI, OpenMP, and compute unified device architecture (MOC) that compute input data over heterogeneous framework. We implemented the proposed model in linear algebraic dense matrix multiplication application, and compared the quantified metrics with well-known basic linear algebra subroutine libraries such as CUDA basic linear algebra subroutines library and KAUST basic linear algebra subprograms. MOC outperformed to all implemented methods and achieved massive performance by consuming less power. The proposed MOC approach can be considered an initial and leading model to deal emerging Exascale computing systems.

INDEX TERMS Exascale computing, HPC, massive parallelism, super computing, energy efficiency, hybrid programming, CUDA, OpenMP, MPI.

I. INTRODUCTION

High Performance Computing (HPC) generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business [1], [2]. Supercomputing and parallel computing are the similar terms to HPC.

The fundamental idea behind the HPC is that, for instance, a single compute takes 100 hours to complete a job, and we can solve the same task by using 100 computers in 1 hour. A single node in a supercomputer may not be a more powerful but it can be when using all the resources together.

Initially, the utilization of HPC was limited to some particular applications such as simple simulations, engineering, oil and gas due to massive cost of HPC systems. However, now a day HPC is being utilized in various areas such as data mining, social media services, education sectors and industries. From recent past, Many HPC applications such as climate and environmental modelling, computation fluid dynamics (CFD) molecular nanotechnology [3], intelligent planetary spacecraft [4] and many other big data applications demand for an extreme powerful computing system to deal such applications. The HPC pioneers and researchers asserted that the emerging supercomputing systems ''Exascale systems'' will be presented till start of next decade [5], [6]. This heterogeneous architecture based HPC supercomputer framework will give a thousand-overlay increase in performance over existing Petascale systems. Such massive performing HPC system will enable the unscrambling of many

scientific mysteries by achieving ExaFlops number of calculations within secs [7]. According to development toward Exascale computing systems, it has been anticipated that it will be comprised of a huge number of heterogeneous nodes where each node will be configured with conventional multi-core CPUs and many core accelerated GPU devices [8], [9]. The arising of heterogeneity of HPC systems is leading in growingly complex platforms at a time when the demand for greater computing power continues to expand. Leading supercomputing systems, the key challenge is the massive power consumption during HPC data processing. According to current HPC supercomputing system architectures, up to 10 million number of cores are resided per node that consume about 25 to 60 MW power. Applying this power consuming ratio, Exascale computing systems will demand up to 130 MW power consumption to achieve Exaflops which is not affordable for maximum countries. However, HPC pioneers such as United States Department of Energy (US DoE), Intel, IBM and AMD defined some limitations for Exascale systems such as energy consumption $\approx$ 25-30 MW, capital cost $\approx$ 200 M USD, targeted time of delivery $\approx$ 2020-2022 and system cores should be up to 100 million [10]. It is very challenging for current HPC systems to achieve ExaFlops level performance under these hard limitations. However, a key element of the strategy as we move forward is the co-design of the applications, programming environments, frameworks and architectures. In addition, hardware breakthrough under the power consumption limitations is also required.

TABLE 1. Exascale computing challenges.

Challenge	Description
Energy efficiency	Developing more energy efficient power and cooling devices.
Interconnect technology	Providing advancement data movement schemes that provide more performance and consume less power
Memory Technology	In order to improve bandwidth and capacity, advance memory technology is required.
Scalable System Software	Scalable power and resilience aware advance systems are required.
Programming system	New programming environments are required that can provide massive parallelism and data locality.
Data management	As diversity of data is anticipated, however new data management approaches are needed that can handle 3Vs as data volume, velocity and variety.
Exascale Algorithms	New performance and energy efficient algorithms are required that can deal massive data for computation.
Algorithms for discovery, design, and decision	Facilitating mathematical optimization and uncertainty quantification for exascale discovery, design, and decision-making.
Resilience and correctness	Novel approaches are required that can ensure the correct scientific computation in term of faults and reproducibility.
Novel architectures	New architectures and frameworks that can be implemented with non-traditional processors are required
Scientific productivity	Novel software engineering tools and environments required to enhance the productivity of computational scientists.

A. EXASCALE CHALLENGES

The primary challenge for the Exascale system is that it does not exist yet. However, in trying to achieve ExaFlops level performance under the defined strict limitations, current technologies are facing several fundamental challenges. At a broad level, these challenges can be categorized in following themes in Table 1 [6].

One conventional approach to upgrade an HPC system framework is to enhance the clock speed. Due to extraordinary heat dissipation, this approach will be fixed at 1 GHz and alternative approach to increase the number of cores will be adopted [12]. According to above indicated restrictions for Exascale systems, we cannot increase the number of cores more than one hundred million. Ultimately, increase in number cores can provide targeted performance level but with massive power consumption. An alternate solution is 'massive parallelism', which required improving the programming environment. According to Jacobsen *et al.* [13], the performance of multi-level parallelism in tri-hierarchy model can be promising for Exascale computing systems. However, it should be implemented and investigated on larger clusters with more than two GPUs and different domain decomposition strategies.

This paper introduces hybrid of MPI+OMP+CUDA (MOC), a new massive parallel programming model for largescale heterogeneous cluster systems. In this article, we adopt the term tri-hybrid of ''MPI+OMP+CUDA'' abbreviated as ''MOC''. MOC provides three level of parallelism such as coarse grain, fine grain and finer granularity parallelism by computing data over inter-node, intra-node and accelerated NVIDIA GPUs devices respectively. The authors and HPC pioneers build over previous work and introduced CuBLAS (CUDA Basic Linear Algebra Subroutines library) [14] and KBAS (KAUST Basic Linear Algebra Subprograms) [15] the more performance tuning knobs that could maintain decent throughput across previous and current accelerated GPUs hardware generations, without code rewriting. Although these models provides a decent performance but cannot be considered for emerging Exascale computing systems due to massive power consumption. As discussed above the characteristics of Exascale systems that will deal with big data HPC applications. However, with respect to two fundamental HPC metrics ''performance and power consumption'', MOC outperformed to existing state-of-the-art on larger dataset computations. MOC attains asymptotically up to 30% and 40% speedup against the best implementations on heterogeneous multiprocessor CPUs and accelerated NVIDA GPUs.

Further, the paper is organized in such way that section II carried out a detailed background of parallel programming models used in MOC proposed model. In addition,

we have accomplished a comprehensive literature review of these models with different hierarchies and reported the HPC facing challenges for today and future Exascale computing systems. Section III highlights our contribution in current study. Section IV demonstrates the MOC model in detail and presents the features and functionalities of inter-node, intra-node and GPU computations in MOC. Section V describes the experimental setup including platform, HPC metrics and measurement mechanism of these metrics. Furthermore, section VI shows the experimental results of MOC and compared with existing state-of-the-art implementations. Section VII describes the critical parameters necessary to tune the MOC model for Exascale computing systems. In addition, this section presents an anticipated configuration model for Exascale system and we conclude in section VIII.

II. BACKGROUND MATERIAL AND LITERATURE REVIEW

We now give a brief background of parallel programming models including MPI, OpenMP and CUDA that has been used in MOC model followed by a comprehensive literature review in section 2.2.

A. TECHNOLOGY BACKGROUND

1) MPI

MPI is a notable autonomous library that has been utilized for correspondence among the explicit procedures in distributed framework. Basically, the standard version MPI was introduced in 1994 [16]. Later on, number of modifications were in made to provide new features in different versions. In recent past, some challenges including environmental layouts, message passing in heterogeneous cluster systems, blocking/non-blocking data distribution and receiving were addressed in MPI 3.0 version [17]. Although the original development in MPI was not for Exascale consideration but a progressive improvement made it the promising consideration for emerging HPC systems. In the light of Exascale computing systems, it still requires several considerations such as low power consuming strategies in message passing among the heterogeneous cores, synchronization handling in non-blocking strategy and memory management mechanisms [18], [19].

2) OPENMP

Open Specification for Multi-Processing (OpenMP) single instruction multiple data (SIMD) based a new model was introduced for CPU thread level parallelism in 1997 [20]. OpenMP parallelize the code over CPU threads using different directives, library routines and clauses. Throughout the OpenMP development, these shared memory standards are available in C++ and FORTRAN languages. In OpenMP 4.0 version, a number of new features were introduced. These features includes ''new atomic operations'' that are used for fine grain synchronization [21]. It also contained many tasks extensions and error handling clauses that maintain the program execution. According to latest build OpenMP 4.5, OMP also facilitate the programmer to run application code on accelerated GPU devices. Along with GPU computation, synchronization between host CPUs and GPU cores was improved that is capable to run multiple task in a group format using '*taskgroup*' construct. In addition, load balancing in during loop parallelization was also improved using '*taskloop*' directive [22]. Although OpenMP has become a very famous model to deal parallel programming merely it is implementable only for single node architectures but cannot be used for cluster systems having multiple nodes. By future perspectives, it has been observed that OpenMP can be used in hybrid with MPI for future comping systems where OpenMP will perform intra-node execution.

3) CUDA

In the light of accelerated NVIDIA GPU programming, NVIDIA presented Compute Unified Device Architecture (CUDA) a remarkable model that achieve massive parallelism by running user input data over accelerated GPUs cores. CUDA architecture is also available in C++ and FORTRAN programming languages [23]. The recent stable CUDA release 8.0 introduced a new optimization schemes that improve the performance in the system. Accordingly, we can make grid and block level optimization by creating multiple stream-processors for each SM on GPU. Further new CUDA release 9.1 was presented that contained the new profiling mechanisms. This new build was supportive for multiple GPU architectures including pascal and lambda compilers [24]. In CUDA model, sequential code is parallelized by executing through CUDA kernel.

According to basic structure of CUDA programming, before calling CUDA kernel, some pre-processing are performed where firstly memory allocation is performed for GPU devices with equal number of variables used at host side. Further, data is transferred from host to GPU devices using particular methods provided by CUDA. Once data transformation is confirmed, CUDA kernel is called for GPU computation. At this stage, we can use multiple CUDA kernels to run over multiple accelerated NVIDIA GPU devices. A detailed overview has been presented in figure 1 [25].

B. LITERATURE REVIEW

Parallelism has brought about a great revolution to enhance the performance in the computer. Parallelism was introduced firstly in the 90s and still being explored to deal targeted Exascale computing systems. The primary objective of Exascale systems is to deal big data HPC applications such as climate and environmental modelling, computation fluid dynamics (CFD), molecular nanotechnology, intelligent planetary spacecraft and many other applications that are required to run on HPC systems. In order to deal these HPC applications, a variety of PPMs such as High-Performance FORTRAN (HPF) [26] and an explicit message-passing interface (MPI) were introduced to attain TFlops. Terascale computing systems were based on coarse-grained parallelism that

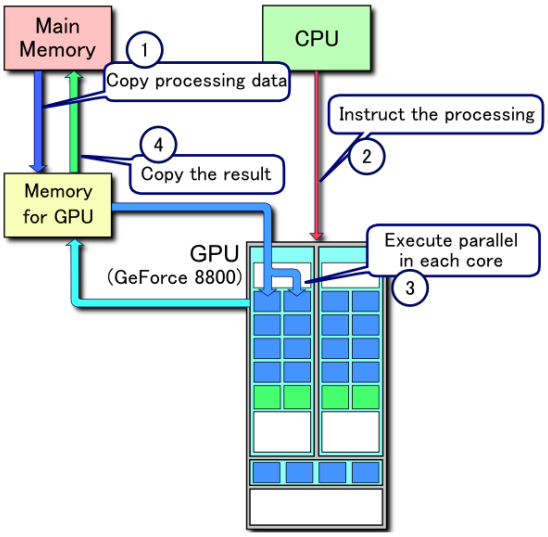

FIGURE 1. Processing flow on CUDA.

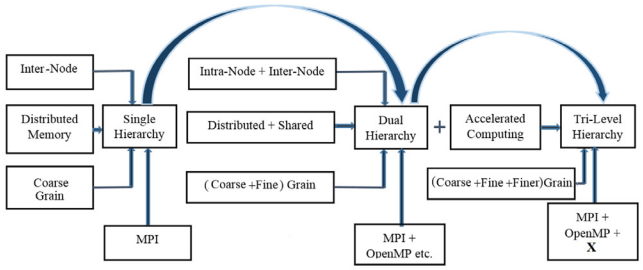

FIGURE 2. Hierarchy Navigation in the Programming Model.

was accomplished at the inter-node level through single-hierarchy models such as MPI. To increase the performance in Terascale systems, many new approaches were introduced, such as pure parallelism, *in situ* processing [27], and out-of-core and multi-resolution techniques. However, pure parallelism was conceived as a suitable paradigm. These suggested models were not able to address the challenges of the higher-order CFD applications and required thread-level parallel computing in a cluster system. Later on, dual-hierarchy model (MPI + X) was introduced for Petascale supercomputing systems [28]. The objective of the Petascale system was to achieve both coarse-grained and fine-grained parallelism through inter-node and intra-node processing. Therefore, a hybrid model of MPI (to parallelize data at the inter-node level) and OpenMP (to parallelize at the intra-node level) was proposed by Dong *et al.* [29]. This hybrid technique enhanced the performance in solving HPC applications. The hybrid model of MPI and OpenMP [30] for coarse-grained parallelism shows good scalability compared to single-hierarchy-level parallelism (pure MPI and pure OpenMP 3.0) with respect to both the problem size and the number of processors for a fixed problem size. Nevertheless, the use of multiple threading in a hybrid paradigm increases the thread management overhead in thread creation/destruction and synchronization considerably with the increase in the number of threads [31]. To update the thread-level parallelism and address the overhead in thread creation/destruction and synchronization, OpenMP 4.0 was released in 2013 as discussed in last section. This new version was equipped with new features for error handling, tasking extensions, atomics and support for accelerated computation. The primary challenge with this hybrid model was massive power consumption while data transferring and communication among CPU processors [32]. However, new hybrid PPM approaches and hardware devices were required for localizing the work from the distributed system in the spectral element method and performing efficient computations using multiple threads.

Since a decade ago, a sensational change happened in hardware advancements in IT. Many-cores architecture based new energy-efficient and powerful devices has been introduced such as Graphical Processing Unit (GPU) by NVIDIA [33], Many Integrated Cores (MIC) by Intel [36]. These GPUs devices were also introduced by different pioneers such as ADM-GPU [34], ARM [35]. These Single-Instruction Multiple-Data (SIMD)-architecture-based many-cores devices contained thousands of cores in single chip that are much powerful than conventional CPUs. The legacy GPU devices were applicable only for graphical processing but new GPU models called General-Purpose Graphical Processing Unit (GPGPU) are available to compute general purpose data processing in HPC applications along with graphical computation. These GPGPUs can be programmed using several accelerated programming models such as OpenCL [37], [38], OpenACC [39], CUDA and OpenMP that can program GPUs. According to different experiences and consequences, CUDA is most promising model for accelerated programming which is optimizable at thread level.

The previous hybrid models could deal only homogenous systems but not heterogeneous cluster systems. However by software perspectives, new hybrid programming models were required that could utilize these energy-efficient accelerated devices along with traditional CPUs [40], [41]. Pennycook *et al.* [42] proposed a new hybrid (MPI + CUDA) approach to implement in NAS LU benchmark. Rakić *et al.* [43] introduced the similar MPI + CUDA parallelization of a finite strip program for geometric nonlinear analysis. The hybrid of MPI+CUDA is applicable on heterogeneous cluster system where multiple CPU processors are configured along with accelerated NVIDIA GPU devices. Likely, another hybrid of OpenMP + CUDA approach was introduced to achieve massive performance by computing data over single node having heterogeneous processors [44]–[46]. The similar functionally was achieved by Howison *et al.* [47] through hybrid of MPI+ PThread. These hybrid approaches could achieve coarse grain and fine-grain parallelism through MPI and GPU computations respectively. To achieve massive parallelism in the system, the hierarchy level in PPMs was shifted from dual- to tri-level by adding another layer of parallelism. Figure 2 demonstrates the increasing hierarchy in parallel

programming models. Toward massive parallel computing using tri-hierarchy level, symmetric multiprocessor (SMP) based new model was introduced to run on larger cluster systems in [48]. In this model, communication among SMP nodes was made using message passing interface. Further second level of parallel computing starts over CPU threads within the node that was accomplished by OpenMP. Continuing parallel computing it starts third level of parallelism by vectorization for each processing element. The primary aim of this approach was to achieve massive parallelism by combining coarse and fine grains within each SMP. Hybrid approach doesn't allow the message passing in SMP nodes which was the main advantage over flat MPI. Further this tri-hybrid model was used to solve several three dimensional linear elastic problems and achieved 3.80 TFlops. From smaller executions, both tri-hybrid and flat MPI achieve better performance but hybrid model outperformed to flat MPI for larger systems with multiple SMP nodes. Although the performance in tri-hybrid model was good enough but huge power consumption was the biggest challenge for HPC technologies. According to Amarasinghe *et al.* [50], unanimous implementation of existing models and powerful GPU devices for better performance of the system should be reinvestigated. On the road toward Exascale computing systems, it has been anticipated that in tri-hybrid third level of parallel computing model X will be replaced by accelerated processing through energy efficient GPU computations. In order to decide this X model, several models were recommended in different studies [51]–[54]. These studies recommended OpenACC, CUDA at top consideration where OpenACC exceeded the performance of the Compute Unified Device Architecture (CUDA) by approximately 50%. Moreover, it exceeded CUDA's performance by up to 98%. Conversely, metrics such as optimization and program flexibility, thread synchronization and other advanced features are attainable in CUDA but not in OpenACC. Under these metrics, HPC heterogeneous systems prevents unnecessary usage of resources. Eventually, we finalized the X model as CUDA to compute accelerated GPU devices in current studies, which is expected the promising model for Exascale computing system.

III. CONTRIBUTIONS

Our contribution in this paper can be summarized as follows:

- Proposed a new tri-hybrid MPI + OpenMP + CUDA (MOC) massive parallel computing model for Exascale computing system that combine coarse-grain, fine grain and finer granularity through inter-node, intra-node and accelerated GPU computations.
- We proposed a tri-hybrid algorithm and theoretical model to evaluate MOC model complexity.
- We implemented MOC in linear algebra dense matrix multiplication using different kernel sizes and evaluated HPC metrics including performance and power consumption.
- We implemented the same problem in CuBLAS and KAUST basic linear algebra subprograms (KBLAS) the most famous Linear Algebra Subroutines libraries. Furthermore, we compare the results with MOC suggested model.
- Based on MOC consequences, anticipated configurations and predictive performance and power consumption have been presented for future Exascale computing system.

IV. TRI-HYBRID MPI+OPENMP+CUDA (MOC) PARALLEL PROGRAMMING MODEL

In this section, we have presented the proposed tri-hybrid parallel programming model for Exascale computing system. Based on the hierarchy navigation in previous parallel programming models, the proposed approach is a hybrid of MPI, OpenMP and CUDA and abbreviated as MOC. MOC contains three major level of computations such as inter-node, intra-node and accelerated GPU devices. The detailed workflow of these three parallel computing level has been illustrated in Figure 3. Each computation level has been discussed in detail as follows:

A. INTER-NODE COMPUTATION

Before interacting with MOC model, some prerequisites are necessary to determine about targeted system that includes host CPU cores and its architecture, number of racks if targeted system is larger cluster, total number of nodes in the system, the GPU devices for accelerated computing and type of GPUs, memory type and levels. Once these specifications are determined, parallel computing zones started. MOC provides basically three levels of parallel zones where first and top level is obtained through inter-node computation. Inter-node computation was achieved by MPI that communicate among host CPUS processors in all connected nodes. MPI defines two types of processes such as master process and slave process where master process is indicated with rank '0' and slave processes are represented with non-zero ranks. Before data distribution over processes, there are some fundamental MPI statements that are necessary to define these ranks and communication size over MPI world. Continuing the parallel computing, MPI master processes distribute the data over all connected nodes through slave processes. In order to distribute and receive the data, several methods are available to use. For MOC model, we implemented blocking methods MPI_Send() and MPI_Recv() for sending and receiving data. Although these methods are not as much efficient as non-blocking Isend() and Irec() but we blocking methods maintain the synchronization. In our implementation, we didn't use any optimization during data distribution however, this level of parallelism provides only coarse grain parallelism. After distrusting data over CPU processes, the next parallel computing zone started as described below.

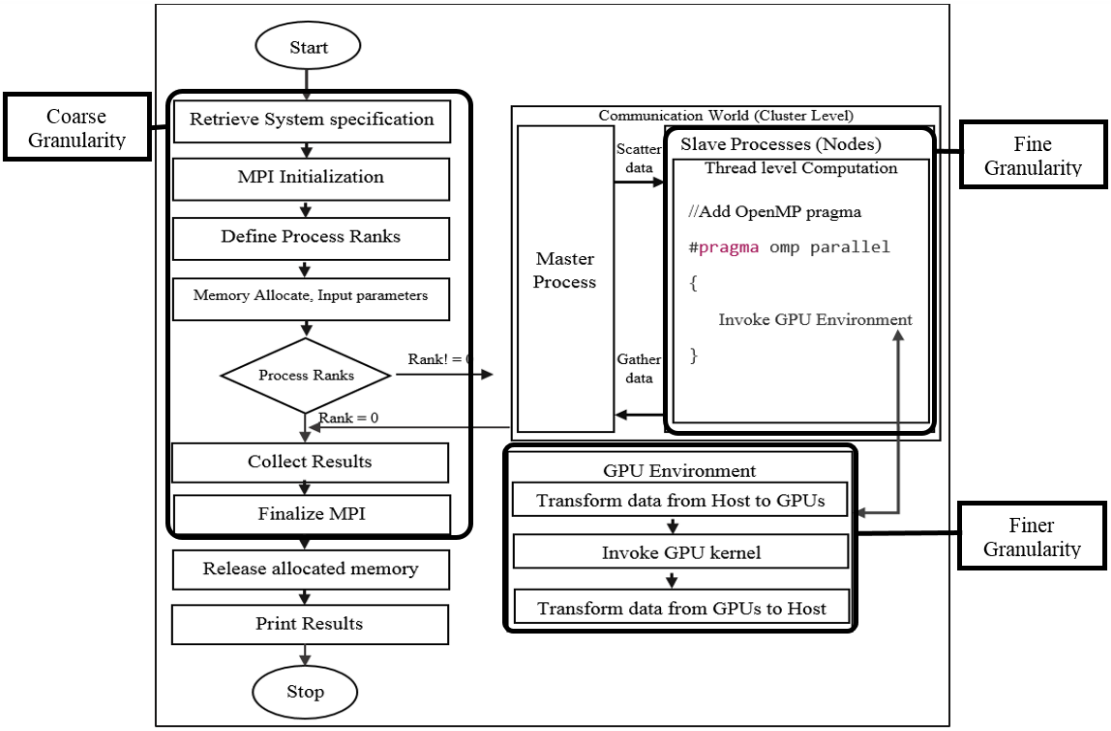

FIGURE 3. Workflow of the Hybrid Parallel Programming Model.

B. INTRA-NODE COMPUTATION (START FROM HERE TO NEXT)

Intra-node computation is second level of parallelism where distributed data over host CPU cores is computed within the node. This computation is performed over CPU threads. These threads can be parallelized through different parallel programming models. Once of the most famous parallel programming model to parallelize CPU threads is OpenMP. As discussed above, OpenMP can be used to program both CPU cores and GPU devices as well. In MOC implementation, we used OpenMP to program parallelize CPU threads and achieved fine grain parallelism. OpenMP programming model contains one main outer pragma that initiate the parallel zone. Each statement written within this pragma is computed in parallel. However, to achieve fine grain parallelism, we implemented multiple looping directives and sections directives, and refine the parallelism. Within these pragmas, we defined the third level of parallelism called GPU computation. In order to optimize the resource, we reserved the similar number OpenMP threads as number of available GPU devices.

C. ACCELERATED GPU COMPUTATION

The third level of parallelism in MOC model was performed through data processing over accelerated GPU devices. Each CPU process was reserved for every GPU device. Therefore, a looping statement reserve a specific GPU device every time and transfer data from host to GPU device. This data is further computed in CUDA kernel that run the code on specific GPU device. At this stage, data is computed over thousands of cores in parallel and obtained finer granularity. For a cluster system having larger number of GPU devices, it's difficult to write the kernels each time. However, MOC model contained a generic form of CUDA kernel that receive/return data in template format and execute accordingly. Once data computation over GPU devices is completed, it transferred back over host cores and controlled by OpenMP threads from where it was initiated. Similarly, OpenMP complete its execution within the pragma and return data to MPI slave processes. After receiving data from all these levels, MPI master thread collect data from slave processes and return the results back to user call. In such way, we achieve three level of parallelism from MOC model.

Algorithm in Listing 1, describes the individual role of MPI, OpenMP and CUDA in MOC model that provide three levels of parallelism as discussed above. Analysing computation and communication cost of an algorithm depicts that whether the algorithm is useful or not. Generally, running time of any algorithm depends on multiple factors such as single/multi-processor system, read/write speed to memory, bit system (32/64 bits) and the input data. Theoretically, an algorithm is evaluated by calculating the time and space complexity in it [63], [64]. Space complexity is related to memory types used in the system. Now a day, we have advanced memory devices that overcome the space issues and consequently ignore the space complexity considerations. Generally, a parallel algorithm is analysed by following factors.

Listing 1. The Tri-Hybrid MOC Algorithm

Input: *mA ← Input matrix*
mB ← Input matrix
Output: *mC ← Resulting matrix*
Declerations *i, numDev, size, rank, numProc, A_D, B_D, C_D.*

Start MPI parallel region
1. *MPI_Init()// initialize MPI parallel region*
2. *Size ← MPI_Comm_size()* // get communication size
3. *Rank ← MPI_Comm_rank () // Get MPI ranks*
4. ***If*** *Rank == master // if master rank (0)*
5. ***Then***
6. Make local initialization before Entering MPI Communication World
7. ***Else***
8. *MPI_Send()// Send data to all processes rank > 0*
9. *MPI_Wait()// check the processing periodically until data transferred to all slave processes successfully.*
10. ***If*** *Rank > 0 // if rank is slave but not master process*
11. ***Then***
12. *MPI_Recv() // Receive data from all slave processes having rank greater than zero*

Start OpenMP parallel region
13. *#pragma omp parallel // Define omp pragma using parallel clause and write code within this pragma*
14. *{*

Start CUDA parallel region
15. ***While*** *(i<numDev)**Do** //until number of GPU devices*
16. *cudaSetDevice(devID) // assign a device id to specific GPU device*
17. *cudaMemcpy() mA → A_D // copy data from host to NVIDIA GPU devices.*
18. *InvokeCUDAKernel<<<grdSize,blkSiz,>>>(A_D, A_B, A_C)*
19. *cudaMemcpy() mC ←C_D // copy data from GPU devices to host*
20. *free all device variables*

End CUDA parallel region
End OpenMP parallel region
21. ***if*** *rank == 0 // there is MPI master process*
22. ***Then***
23. *MPI_Recv() // Receive processed data from all rank*
24. *MPI_finalize()// Finalize MPI processing*

End MPI Processing
25. ***Return*** *mC // return the Results*

- Total cost (Communication cost)
- Time complexity

List. 1. The Tri-Hybrid MOC Algorithm

In order to evaluate these parameters in MOC model, let's assume that N is the problem size in bytes. In MOC model, each process p_i will process these bytes equally as $w = N/p$ where w is the workload and f is considered the percentage of workload w that cannot be processed within the parallel region. According to Amdahl's law, we consider the T_t the time factor for processing by adding t threads on each p. Using asymptotic analysis $O(N)$ approach, the required work amount for N bytes can be expressed as $O(\frac{N}{pT_t})$.

During processing of any parallel algorithm, it incurs some amount of communication overhead [65]. In MOC implantation, we tried to reduce the communication rounds from three phases including sending, computing and receiving; and assume the overhead cost as T_o. During sending round, let us assume that s bytes data will be send by a process p_i from working region. Therefore, the communication complexity for sending s bytes will be as $O(\frac{N}{S_p})$. Similarly, the multithreaded processes can compute C bytes of data using share memory. During data computation over processes, many overhead possibilities exist there such as wait time during accessing shared data, processes synchronization etc. However, these overheads during data computation can be expressed as $o = N/M$ where M is the output of data computation. Thus the complexity of data computation round can be determined as $O(\frac{N}{oCp})$. Once the program execution is done by all the processes, the third round of data receiving takes start that can be described as $O(log\ (p + Rec))$ where Rec is timestamps for receiving processed data.

Overall the total time complexity of MOC algorithm can be summarized as $(T_m = T_c + T_o)$ where T_c represents the computation cost of input data and T_o is the communication overhead cost.

$$T_c = O(\frac{N}{pT_t}) \quad \text{(a)}$$

$$T_o = O(\frac{N}{S_p}) + O(\frac{N}{oCp}) + O(log(p + Rec)) \quad \text{(b)}$$

$$T_m \approx O\left(\frac{N}{p}\left(\frac{1}{T_t} + \frac{1}{S} + \frac{1}{oC}\right)\right) + O(log(p + Rec)) \quad \text{(c)}$$

Equation (c) elaborates that the complexity in MOC algorithm depend on multiple parameters under data computation and communication among the processes.

V. EXPERIMENTAL SETUP

This section explains the selected experimental platform for proposed MOC model. In experiments, we measured different HPC metrics that includes performance factors such as execution time, number of achieved flops, power consumption and the energy efficiency in the system. A detailed description of these metrics is explained in following section.

A. EXPERIMENTAL PLATFORM

In order to evaluate the suggested MOC model, all experiments were performed on Aziz supercomputer available in

HPC centre, King Abdulaziz University, Jeddah, Saudi Arabia. Aziz-Fujistu Primergy CX400 Intel Xeon Truescale QDR supercomputer is manufactured by Fujistu [55]. According to top-500 supercomputing list, Aziz was ranked at 360th position [56]. Originally Aziz was develop to deal HPC applications in Saudi Arabia and collaborated projects. It contained 492 number of nodes which are interlinked within the racks through InfiniBand where 380 are regular and 112 are large nodes with additional specifications. Previously Aziz was capable to run the applications only on homogeneous node but due to requirements of massive parallel computing, it was upgraded by adding two SIMD-architecture-based accelerated NVIDA Tesla k-20 GPU devices where each device has 2496 CUDA cores. Moreover, two MIC devices with an Intel Xeon Phi Coprocessor with 60 cores were also installed to upgrade homogeneous computational architecture. In such way, Aziz contains total 11904 number of core in it. Regarding memory, each regular node contained by 96 GB and larger (FAT) nodes with 256 GB. Each node has Intel E5-2695v2 processor that contains twelve physical cores with 2.4 GHz processing power. Overall Aziz all nodes and accelerated devices are interlinked through three different networks including user, management and InfiniBand network. For Aziz, user network and management networks are specifically used for the login and job submission handling whereas InfiniBand to parallelize the file system. According to LINKPACK benchmarks, Aziz's peak performance was measured with 211.3 Tflops/s and 228.5 Tflops/s as theoretical performance [57]. Regarding software speficications, it run using Cent OS with release 6.4. For accelerated programming CUDA recent toolkit 9.1 is installed. It also contained many other compilers required for HPC libraries.

B. PERFORMANCE MEASUREMENT

System performance is the primary aim of current and emerging HPC systems. The performance metric contains different factors that evaluates a system's performance such as execution time, number of achieved flops. Usually, in HPC systems, the number of flops are calculated by dividing the number of floating point operations (*FPOps*) by parallel execution time PE_t as given in equation (1).

$$Flops = \frac{FPOps}{PE_t} \tag{1}$$

Following equation (1), we measured the number of achieved flops by executing different datasets of DMM algorithm in MOC model.

C. POWER MEASUREMENT

Energy consumption is the primary challenge toward emerging HPC systems. Although this challenges is somehow addressed in current computing systems but we cannot increase the system performance under defined power consumption limitations. However, the main theme of Exascale systems is to minimize the power consumption by selection optimal hardware and software frameworks [58]. Many novel programming approaches are being introduced to program accelerated energy efficient devices that can minimize the power consumption level. Generally, a system is evaluated according to its energy consumption, which indicates the power rate at which processing was executed, as described in equation (2).

$$E(kWh) = \int_0^t V \cdot I\,(dt) \tag{2}$$

From above equation, we can calculate the total energy consumption of a system by integrating the energy consumption, which is composed of the bandwidth, memory contention, parallelism and behavior of the application in the HPC parallel system, as described in equation (3).

$$E_{system} = \int_0^t BandW\,(dt) + MemC\,(dt) + Prll\,(dt) + Bhv(dt) \tag{3}$$

On the basis of the dictated factors and the fundamental energy evaluation, we quantified these factors in the current study with respect to system performance and power consumption. For any heterogeneous cluster system, the power consumption can be calculated by summation of the products of the power of each component and the corresponding duration [59]. Generally, Power consumption in a system is categories in two types.

1) System Specification
2) Application Specification

Since the system specification has GPU devices installed in it, the power consumption is calculated by equation (4):

$$P_{system}(w) = \sum_{i=1}^{N} P^i_{GPU}(w^i) + P_{CPU}(\sum_{j}^{M}(w^j)) + P_{mainboard}(w) \tag{4}$$

From equation 4, it can be speculated that the approximate power consumption of a system is the sum of the products of the installed GPUs, CPUs and motherboard. The power consumption varies with the workload; however, on the application side, it can be quantified using equation (5):

$$P_{app} = \sum_{i=1}^{N_{app}} P^i_{GPU}(w^i) + P_{CPU}(\sum_{j}^{M}(w^j)) + P_{mainboard}(w_{app}) \tag{5}$$

According to equations (4) and (5), the power consumption in watts was measured at the idle state of the system, where only 5 watts of power were consumed by the motherboard and the remaining power was consumed by the cores of system.

VI. EXPERIMENTAL RESULTS

Tri-hybrid MOC model was implemented in linear algebraic dense matrix multiplication (DMM) algorithm [60]. We performed all the implementation on Aziz supercomputing available in King Abdulaziz University, Jeddah Saudi Arabia as the specification are described above. During experiments,

TABLE 2. Naïve code and parameters of implemented DMMA.

Kernel	Naive Code	Parameters & Domains
DMM	Do i = 1; n Do j = 1; n Do k = 1; n z(i, k)=z(i , k) + x(i, j) * y(j, k)	t_i, t_j, t_k (i,j,k tiles) ui,uj(i,j, unrolls) matrix-Size (msize) msize ∈ [1000, 2000, 3000… 10000]

we observed two fundamental HPC metrics including performance and power consumption that are biggest challenges for current and emerging HPC systems. As Aziz contained multiple GPU devices, however we computed different matrix sizes on multiple CUDA kernels (four, eight and twelve). During experiments, it was observed that multiple kernels outperformed to single kernel call with respect to performance by consuming low power. For a heterogeneous system, we noticed that larger than four kernels couldn't perform well. This happens due to unnecessary utilization of resources and communication among the devices and host cores. According to new CUDA build 9.1, the overhead of communication can be minimized by using additional CUDA statements that allow to communicate with host cores only when necessary. In contrast, four kernels outperformed throughout the executions due to an optimized resources utilization. A naïve code along with specific parameters of implemented DMM has been presented in table 2 as follows.

Above code described in table 2 is not explained completely due to space restrictions. However, readers can approach to Tiwari *et al.* [61] that has described DMM implementation and optimization strategy in details. In our implementation, z array was reused in the buffer register and x, y arrays were stored in caches to utilize in efficient way.

However, in order to quantify the performance in selected linear algebraic dense matrix multiplication application, the number of floating point (FP) operations of DMM algorithm were determined. Generally, in dense matrix multiplication algorithm (DMMA), it is apparent that the total number (TN) of FP operations to compute a product of two square matrices can be calculated using a simple formula as follows in equation (6).

$$TN\ FP\ Ops = (N^3) + ((N^2) * (N - 1)) \tag{6}$$

Once the TN of FPOps are determined, we can calculate the number of flops using equation (1). According to DMM algorithm implementation, it was observed the peak performance for datasets 1000-10000 reached up to 1 Teraflops in four kernels. For all kernels implementation the observed performance was achieved by 716 Gflops as an average as shown in Figure 4.

We noticed that the implementation of four CPU threads per node along with equal number of CUDA kernels outperformed to all other kernel configurations and consequently accomplished approximately 1 Teraflops with 68% peak

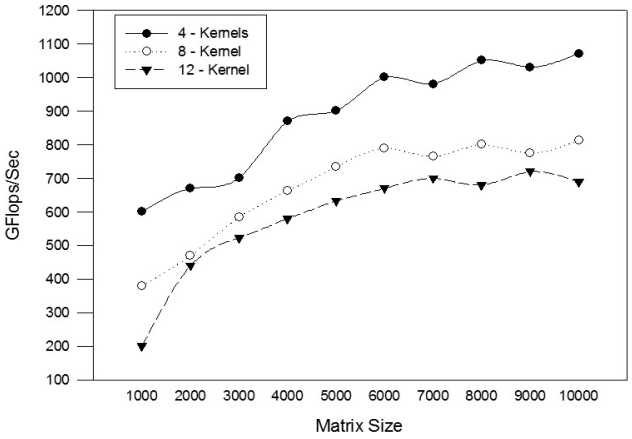

FIGURE 4. Performance in DMM through multiple kernel configurations.

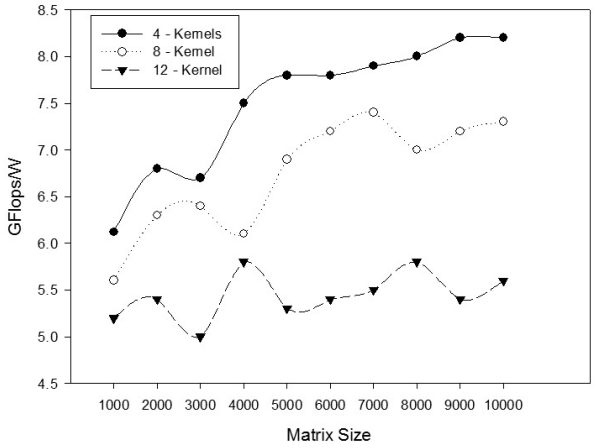

FIGURE 5. Energy efficiency in DMM for different multiple-kernel configurations.

performance. Conversely, bigger kernel sizes also achieved an adequate performance but consumed more power due to unnecessary communication among heterogeneous cores. Alongside performance, we evaluated another essential metric, to be specific, energy consuming in the system that was 28 Joules. At most extreme DMM for a dataset of size 10000 through an enhanced four kernel setup, the evaluated energy proficiency was 8.3 Gflops/W. The addition of assets influenced energy effectiveness significantly and diminished it to 5.6 Gflops/W, as appeared in Figure 5.

Performance and energy efficiency are directly proportional to each other, however the trade-off between both metrics can be determined [62] as given:

$$\frac{Performance}{Power} = \frac{Execution\ within\ the\ time\ unit}{Energy\ during\ the\ execution\ time\ unit} = \frac{work}{energy}$$

Therefore trade-off between these metrics provides the rate of accomplishable performance under given energy efficiency as presented in Figure 6 where horizontal and vertical lines

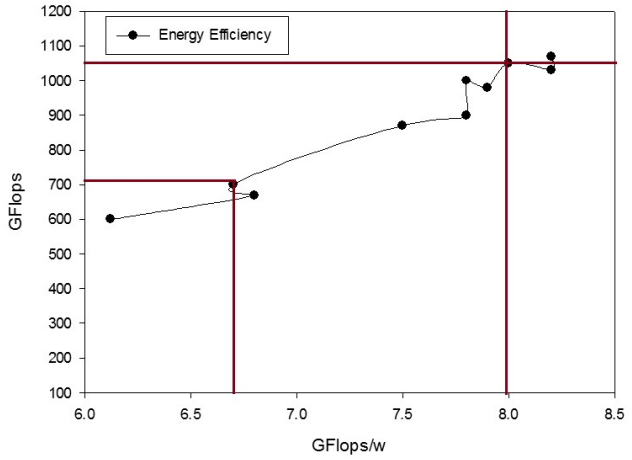

FIGURE 6. Performance-Energy Efficiency tradeoff.

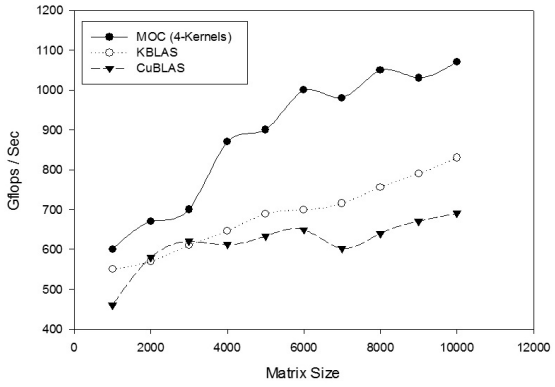

FIGURE 7. MOC performance comparison with KBLAS and CuBLAS implementations.

presents the entropy of energy efficiency and performance respectively. Any interacting point in the graph confronts the peak values of achieved performance and energy efficiency. We can settle the arrangement and parameters at any crossing point to give most extreme execution and vitality productivity. These assessments discovered that the best performance and energy efficiency which is accomplishable by utilizing the proposed demonstrate on the Aziz supercomputer was 1086 Gflops, which relates to energy efficiency of 8.3 Gflops/W.

In MOC implementations with different kernel sizes, we concluded that system performance is directly proportional to resource optimization and utilization during any dataset processing. Sometime, larger number of resources become the reason of performance decreasing in the system as (8,12 kernels) used in experiments. In order to compare the evaluated MOC performance and power consumption, we computed the similar DMM application matrix size in KBLAS and CuBLAS libraries. According to figure 7, KBLAS and CuBLAS could achieve maximum 810 and 630 Gflops/sec respectively whereas MOC achieved up to one Tflops during the same executions.

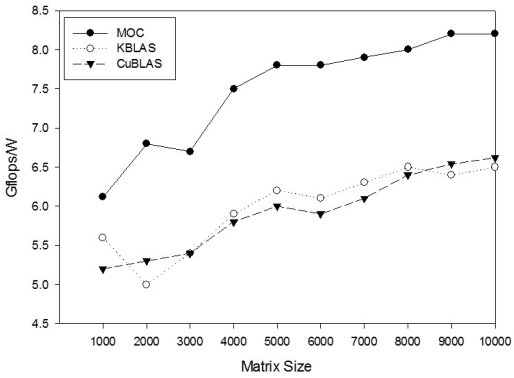

FIGURE 8. Energy efficiency in DMM for MOC vs (CuBLAS and KBLAS).

MOC outperformed throughout to KBLAS and CuBLAS in all executions matrix sizes. During power consumption quantification, there consequences were different now where MOC consumed less power as compared to all other implementations as shown in figure 8.

It was observed that MOC achieved 6.1 Gflops/w during smaller data computation whereas CuBLAS and KBLAS attained 5.2 and 5.7 number of Gflops/w respectively. By increasing matrix size, Gflops/w gradually changed in all implementations. We noticed that CuBLAS and KBLAS could achieved the maximum number of Gflops/w as 6.5-6.7 against maximum matrix size computations. In contrast, MOC attained the similar number of Gflops/w at initial matrix size executions. By increasing the matrix size, number of Gflops/w increased gradually and reached up to 8.3. Although KBLAS and CuBLAS are also optimized approaches in solving linear algebraic systems whereas MOC outperformed by depicting massive parallelism. Furthermore, the additional factor was utilization of NVIDIA GPU that processed the data in mili-seconds within small power consumption. However, we evaluated that MOC model accomplished 1086 Gflops by consuming 130 w total power consumption during larger matrix execution.

VII. EXASCALE COMPUTING SYSTEM DEMAND

The major challenge for future HPC supercomputing Exascale systems is that it doesn't exist yet. However, the development toward Exascale systems are being performed based on predictions and statistics in existing consequences. This section demonstrates a statistical analysis of performed experiments in current study. This factual investigation explore two primary HPC metrics including performance and power consumption which has been considered the challenging factors for Exascale computing systems. The current study was conducted on heterogeneous architecture based system that contained 11904 number of cores integrated in 494 number of homogenous nodes. Moreover, two k-20 NVIDA GPU devices were configured in the system for accelerated computing. The peak performance of selected platform was 211.3 Tflops/s and 228.5 Tflops/s with

TABLE 3. Exascale computing system configurations.

Features	Specification
Number of Cabinets	200
Nodes per Cabinet	384
Number of Nodes	76800
Number of Network slice	4
Total router count	19200
Peak PFlops	1258
Max Power Consumption of Processors	230 W
Max Power Consumption / Node	300 W
Max Power Consumption / System	25 MW

TABLE 4. An analysis of measuring parameters based on different platforms.

Measuring parameters	Current and future Platforms		
	Aziz		Exascale
	Attained	Predictive	
Performance	1 Teraflops	≈ 230 PFlops	1 ExaFlops
P. Consumption	130 W	≈ 27 MW	≈ 25 MW
Energy Efficiency	8.3 Gflops/W	≈8.5Pflops/MW	≈40 Pflops/MW

Linpack and theoretical performance respectively. Using this platform, MOC model achieved 1086 Gflops/s and consumed less than 28 joules energy for larger datasets. This rate of performance and energy consumption conceded the energy efficiency as 8.3 Gflops/Sec. This efficiency factor can be determined by using formula given in equation 7.

$$P\,(w) = \frac{E_{(j)}}{t_{(s)}}$$

However,

$$watt = \frac{Joule}{\text{Second}} \text{ or } \mathrm{W} = \mathrm{J} \,/\, \mathrm{S} \tag{7}$$

Following equation (7), the peak performance was achieved under the power consumption with 130 W. According to predictive Exascale systems configuration given in Table 3, if we enhance the selected platform with thousand-fold increase, it will be capable to provide Exaflops number of calculations per second.

Based on the ratio of current computation and required resources, the predictive performance and power consumption were calculated, which are presented in Table 4.

Table 4 demonstrates the HPC current and emerging platforms. Based on consequences, we categorized the selected platform Aziz supercomputer into two domains such as accomplished and prescient. However, the accomplished results against each metric demonstrates that if Aziz is scaled with Exascale configurations, MOC model can provide the predictive performance level and power consumption in it. As scalability in current architecture doesn't requires any additional frameworks that however it can achieve the predictive figures by using MOC. Therefore, MOC model can be considered as promising model for emerging Exascale computing systems if we just scale the existing HPC system with fundamental resources required for Exascale systems. The determinations from current study, elicited several challenging that open new research directions and thoughts as follows:

- As HPC systems are not confirmed about its architecture, it may homogeneous / heterogeneous, however it must be investigated that how and which layer can manage the dynamic behaviour of the system and code irregularity as well.
- In different studies, we noticed that algorithms enhance the system performance by consuming less power; it must be investigated that which optimized approach can adopt this trade-off.
- As describe in top ten challenges, memory management is one of those, however, what additional hooks can be used to increase system efficiency by reducing communication cost.
- Resource optimization should also be considered as sometime small input data occupy large number of resources.
- How we can maintain the power consumption during larger executions within a particular environment?
- How the communication overhead among the heterogeneous cores can be reduced to dilute the power consumption?

The above concluded facts open new research directions and challenges for development communities and researchers. Based on these facts, the suggested MOC model must be implemented in different complex HPC applications and observe the system behaviour.

VIII. CONCLUSIONS

HPC innovation is being moved from the Petascale to the extraordinary "Exascale" processing framework. This powerful system will required massive power consumption to provide Exaflops number of calculation in secs. However, HPC pioneers, researchers and development communities defined some hard limitations that should be considered for any Exascale system. These limitations includes majorly power consumption, performance level, delivery time and number of configured cores. Based on these limitations, current technologies are facing several challenges where accomplishing massive parallelism under energy constrains is one of those. Current study proposed a new tri-hybrid MOC (MPI + OpenMP + CUDA) parallel programming model that attained massive performance through monolithic parallelism in the system. In order to evaluate MOC model, we implemented in linear algebraic dense matrix multiplication application and observed different metrics such as performance and power consumption during different dataset executions. It was observed that MOC with four kernels outperformed against eight and twelve kernels implementations. Further, MOC with peak performance was compared with other most prominent implementations including KLBAS and CuBLAS.

We observed that MOC achieved a tremendous performance and reached up to 1 Teraflops within 130 W power consumption. Based on experimental consequences, we presented a predictive performance and power consumption of selected platform if we increase it up to Exascale configurations. Although these predictive results were not meeting the Exascale figures but enhance the performance by decreasing three time power consumption as in current computation systems. However, the suggested MOC model can be conceived as a promising model for emerging Exascale computing systems.

By future perspectives, we must rethink and fix the determined challenges toward Exascale systems. The major challenge for Exascale is that it doesn't exist yet. However, we cannot assure that it will be homogeneous or heterogeneous architecture based systems. Therefore, we need an adaptive hybrid programming model that can deal both homogenous and heterogeneous architecture systems.

REFERENCES

[1] D. Eddelbuettel, "CRAN task view: High-performance and parallel computing with R," Comprehensive R Arch. Netw., Tech. Rep. 2018-03-20, 2018.
[2] Inside HPC. *What is High Performance Computing*. Accessed: Jan. 10, 2018. [Online]. Available: http://insidehpc.com/hpc-basictraining/what-is-hpc/
[3] M. Zhou, "Petascale adaptive computational fluid dynamics," Ph.D. dissertation, Rensselaer Polytech. Inst., Troy, NY, USA, 2009.
[4] J. J. Dongarra and D. W. Walker, "The quest for petascale computing," *Comput. Sci. Eng.*, vol. 3, no. 3, pp. 32–39, May 2001.
[5] R. Brower *et al.* (Oct. 2017). "Lattice QCD application development within the US DOE exascale computing project." [Online]. Available: https://arxiv.org/abs/1710.11094
[6] J.-L. Vay *et al.* (Jan. 2018). "Warp-X: A new exascale computing platform for beam-plasma simulations." [Online]. Available: https://arxiv.org/abs/1801.02568
[7] S. Perarnau, R. Gupta, and P. Beckman, "Argo: An exascale operating system and runtime," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2015.
[8] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *Proc. Int. Conf. High Perform. Comput. Comput. Sci.*, 2010, pp. 1–25.
[9] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Commun. ACM*, vol. 58, no. 7, pp. 56–68, 2015.
[10] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 374–388, 2009.
[11] *ASCAC Subcommittee for the Top Ten Exascale Research Challenges*, U.S. Dept. Energy State, Washington, DC, USA, 2014.
[12] T. N. Theis and H.-S. P. Wong, "The end of Moore's Law: A new beginning for information technology," *Comput. Sci. Eng.*, vol. 19, no. 2, pp. 41–50, 2017.
[13] D. A. Jacobsen and I. Senocak, "Multi-level parallelism for incompressible flow computations on GPU clusters," *Parallel Comput.*, vol. 39, no. 1, pp. 1–20, 2013.
[14] *CUDA Toolkit 4.0 CUBLAS Library*, Nvidia Corp., Santa Clara, CA, USA, 2011, pp. 59–60.
[15] A. Abdelfattah, D. Keyes, and H. Ltaief, "KBLAS: An optimized library for dense matrix-vector multiplication on GPU accelerators," *ACM Trans. Math. Softw.*, vol. 42, no. 3, 2016, Art. no. 18.
[16] J. Dongarra, "MPI: A message-passing interface standard version 3.0," *High Performance Computing Center Stuttgart (HLRS)*, 2013.
[17] J. Dinan, P. Balaji, D. Buntinas, D. Goodell, W. Gropp, and R. Thakur, "An implementation and evaluation of the MPI 3.0 one-sided communication interface," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 17, pp. 4385–4404, 2016.
[18] (Jun. 20, 2017.). *Message Passing Interface*. Accessed: Aug. 3, 2017. [Online]. Available: https://computing.llnl.gov/tutorials/mpi/
[19] (Jan. 18, 2018). *OpenMPI: Open Source High Performance Computing*. Accessed: Feb. 10, 2018. [Online]. Available: https://www.open-mpi.org/
[20] (Jan. 3, 2018). *OpenMP*. Accessed: Feb. 11, 2018. [Online]. Available: http://www.openmp.org/
[21] I. Karlin *et al.*, "Early experiences porting three applications to OpenMP 4.5," in *Proc. Int. Workshop OpenMP*, 2016, pp. 281–292.
[22] A. Podobas and S. Karlsson, "Towards unifying OpenMP under the task-parallel paradigm," in *Proc. Int. Workshop OpenMP*, 2016, pp. 116–129.
[23] *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corp., Santa Clara, CA, USA, 2017.
[24] (Jan. 24, 2018). *NVIDIA Accelerated Computing*. Accessed: Feb. 15, 2018. [Online]. Available: https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html
[25] (Jan. 30, 2018). *CUDA*. Accessed: Feb. 15, 2018. [Online]. Available: https://en.wikipedia.org/wiki/CUDA
[26] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems," *Parallel Comput.*, vol. 37, no. 9, pp. 562–575, 2011.
[27] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova, "*In-situ* processing and visualization for ultrascale simulations," *J. Phys., Conf. Ser.*, vol. 78, no. 1, p. 012043, 2007.
[28] P. D. Mininni, D. Rosenberg, R. Reddy, and A. Pouquet, "A hybrid MPI–OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence," *Parallel Comput.*, vol. 37, nos. 6–7, pp. 316–326, 2011.
[29] S. Dong and G. E. Karniadakis, "Dual-level parallelism for high-order CFD methods," *Parallel Comput.*, vol. 30, no. 1, pp. 1–20, 2004.
[30] M. U. Ashraf and F. E. Eassa, "Hybrid model based testing tool architecture for exascale computing system," *Int. J. Comput. Sci. Secur.*, vol. 9, no. 5, pp. 245–252, 2015.
[31] S. Jin and D. P. Chassin, "Thread group multithreading: Accelerating the computation of an agent-based power system modeling and simulation tool–C GridLAB-D," in *Proc. IEEE 47th Hawaii Int. Conf. Syst. Sci.*, Jan. 2014, pp. 2536–2545.
[32] M. Hennecke, W. Frings, W. Homberg, A. Zitz, M. Knobloch, and H. Böttiger, "Measuring power consumption on IBM Blue Gene/P," *Comput. Sci.-Res. Develop.*, vol. 27, no. 4, pp. 329–336, 2012.
[33] T. Hoegg, G. Fiedler, C. Koehler, and A. Kolb, "Flow driven GPGPU programming combining textual and graphical programming," in *Proc. ACM 7th Int. Workshop Program. Models Appl. Multicores Manycores*, 2016, pp. 88–97.
[34] N. Rajovic, L. Vilanova, C. Villavieja, N. Puzovic, and A. Ramirez, "The low power architecture approach towards Exascale computing," *J. Comput. Sci.*, vol. 4, no. 6, pp. 439–443, 2013.
[35] N. Rajovic, A. Rico, N. Puzovic, C. Adeniyi-Jones, and A. Ramirez, "Tibidabo[1]: Making the case for an ARM-based HPC system," *Future Generat. Comput. Syst.*, vol. 36, pp. 322–334, Jul. 2014.
[36] A. Duran and M. Klemm, "The Intel many integrated core architecture," in *Proc. IEEE Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2012, pp. 365–366.
[37] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Comput. Sci. Eng.*, vol. 12, no. 3, pp. 66–73, 2010.
[38] M. U. Ashraf and F. E. Eassa, "OpenGL based testing tool architecture for exascale computing," *Int. J. Comput. Sci. Secur.*, vol. 9, vol. 5, pp. 238–244, 2015.
[39] M. Wolfe *et al.*, "Implementing the OpenACC data model," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May/Jun. 2017, pp. 662–672.
[40] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, "CULA: Hybrid GPU accelerated linear algebra routines," *Proc. SPIE*, vol. 7705, p. 770502, Apr. 2010.
[41] S. Tomov, J. Dongarra, and M. Baboulin, "Towards dense linear algebra for hybrid GPU accelerated manycore systems," *Parallel Comput.*, vol. 36, nos. 5–6, pp. 232–240, 2010.
[42] S. J. Pennycook, S. D. Hammond, S. A. Jarvis, and G. R. Mudalige, "Performance analysis of a hybrid MPI/CUDA implementation of the NASLU benchmark," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 23–29, 2011.
[43] P. S. Rakić, D. D. Milašinović, Ž. Živanov, Z.Suvajdžin, M. Nikolić, and M. Hajduković, "MPI–CUDA parallelization of a finite-strip program for geometric nonlinear analysis: A hybrid approach," *Adv. Eng. Softw.*, vol. 42, no. 5, pp. 273–285, 2011.

[44] J. Guan, S. Yan, and J.-M. Jin, "An openMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems," *IEEE Trans. Antennas Propag.*, vol. 61, no. 7, pp. 3607–3616, Jul. 2013.

[45] F. Lu, J. Song, F. Yin, and X. Zhu, "Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters," *Comput. Phys. Commun.*, vol. 183, no. 6, pp. 1172–1181, 2012.

[46] R. Reyes and F. de Sande, "Optimization strategies in different CUDA architectures using llCoMP," *Microprocess. Microsyst.*, vol. 36, no. 2, pp. 78–87, 2012.

[47] M. Howison, E. W. Bethel, and H. Childs, "Hybrid parallelism for volume rendering on large-, multi-, and many-core systems," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 1, pp. 17–29, Jan. 2012.

[48] K. Nakajima, "Three-level hybrid vs. flat MPI on the Earth Simulator: Parallel iterative solvers for finite-element method," *Appl. Numer. Math.*, vol. 54, no. 2, pp. 237–255, 2005.

[49] T. Nguyen-Thoi, G. R. Liu, K. Y. Lam, and G. Y. Zhang, "A face-based smoothed finite element method (FS-FEM) for 3D linear and geometrically non-linear solid mechanics problems using 4-node tetrahedral elements," *Int. J. Numer. Methods Eng.*, vol. 78, no. 3, pp. 324–353, 2009.

[50] S. Amarasinghe *et al.*, "ASCR programming challenges for exascale computing," *Rep. Workshop Exascale Program. Challenges*, 2011.

[51] T. Hoshino, N. Maruyama, S. Matsuoka, and R. Takaki, "CUDA vs OpenACC: Performance case studies with kernel benchmarks and a memory-bound CFD application," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, May 2013, pp. 136–143.

[52] J. A. Herdman *et al.*, "Accelerating hydrocodes with OpenACC, OpenCL and CUDA," in *Proc. IEEE SC Companion High Perform. Comput., Netw., Storage Anal. (SCC)*, Nov. 2012, pp. 465–471.

[53] A. Lashgar, A. Majidi, and A. Baniasadi. (Dec. 2014) "IPMACC: Open source OpenACC to CUDA/OpenCL translator." [Online]. Available: https://arxiv.org/abs/1412.1127

[54] S. Christgau, J. Spazier, B. Schnor, M. Hammitzsch, A. Babeyko, and J. Waechter, "A comparison of CUDA and OpenACC: Accelerating the tsunami simulation EasyWave," in *Proc. Workshop Archit. Comput. Syst. (ARCS)*, Feb. 2014, pp. 1–5.

[55] (Sep. 22, 2014). *Fujitsu to Provide High-Performance Computing and Services Solution to King Abdulaziz University*. Accessed: Jul. 6, 2017. [Online]. Available: http://www.fujitsu.com/global/about/resources/news/press-releases/2014/0922-01.html

[56] (Jun. 2015). *King Abdulaziz University*. Accessed: Aug. 3, 2017. [Online]. Available: https://www.top500.org/site/50585

[57] *Aziz—Fujitsu PRIMERGY CX400, Intel Xeon E5-2695v2 12C 2.4GHz, Intel TrueScale QDR*. Accessed: Aug. 3, 2017. [Online]. Available: https://www.top500.org/system/178571

[58] L. A. Barroso, "The price of performance," *Queue*, vol. 3, no. 7, pp. 48–53, Sep. 2005.

[59] D. Ren and R. Suda, "Power efficient large matrices multiplication by load scheduling on multi-core and GPU platform with CUDA," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSE)*, vol. 1. Aug. 2009, pp. 424–429.

[60] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh, "Parallel algorithms for dense linear algebra computations," *SIAM Rev.*, vol. 32, no. 1, pp. 54–135, 1990.

[61] A. Tiwari, C. Chen, J. Chame, M. Hall, and J. K. Hollingsworth, "A scalable auto-tuning framework for compiler optimization," in *Proc. IPDPS*, Rome, Italy, May 2009, pp. 1–12.

[62] H. Anzt, B. Haugen, J. Kurzak, P. Luszczek, and J. Dongarra, "Experiences in autotuning matrix multiplication for energy minimization on GPUs," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 17, pp. 5096–5113, 2015.

[63] J. Y.-T. Leung, Ed., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, 2004.

[64] G. C. Fox, R. D. Williams, and G. C. Messina, *Parallel Computing Works!* New York, NY, USA: Elsevier, 2014.

[65] *Introduction to Parallel Computing*. Accessed: Feb. 15, 2018. [Online]. Available: https://computing.llnl.gov/ tutorials/parallel_comp/

M. USMAN ASHRAF was born in Paddali, Sialkot, Pakistan, in 1988. He received the B.Sc. degree in mathematics from the University of the Punjab, Pakistan, in 2007, and the M.S. degree in computer science from the University of Lahore, Pakistan, in 2014. He is currently pursuing the Ph.D. degree in computer science from King Abdulaziz University, Jeddah, Saudi Arabia. From 2010 to 2014, he was a Senior Software Engineer with Coeus Software Solutions, GmbH. He is currently a member of the Software Engineering Group, King Abdulaziz University Jeddah, Saudi Arabia. His research interests include high performance computing, parallel computing, exascale computing, and software engineering.

FATHY ALBURAEI EASSA received the B.Sc. degree in electronics and electrical communication engineering from Cairo University, Egypt, in 1978, and the M.Sc. and Ph.D. degrees in computers and systems engineering from Al-Azhar University, Cairo, Egypt, in 1984 and 1989, respectively, with joint supervision with the University of Colorado, USA, in 1989. He is currently a Full Professor with the Computer Science Department, Faculty of Computing and Information technology, King Abdulaziz University, Saudi Arabia. His research interests include agent based software engineering, cloud computing, software engineering, big data, distributed systems, and exascale system testing.

AIIAD AHMAD ALBESHRI received the B.S. degree in computer science from King Abdulaziz University, Jeddah, Saudi Arabia, and the Ph.D. degree in computer science from the Queensland University of Technology, Australia, in 2013. He is currently an Assistant Professor with the Department of Computer Science, King Abdulaziz University, Saudi Arabia. His research interests include cloud computing, security in cloud computing, storage in cloud computing, parallel computing, and big data.

ABDULLAH ALGARNI received the bachelor's degree from King Abdulaziz University Jeddah, Saudi Arabia, and the master's and Ph.D. degrees from the College of Natural Sciences, Colorado State University, USA, in 2016, all in computer science. He is currently an Assistant Professor and the Chairman of the Computer Science Department, King Abdulaziz University, Jeddah, Saudi Arabia. His research interest includes software vulnerabilities, software risk management and mitigation, quantitative evaluation, software risk assessment, software engineering, and software security.