Toward Exascale Computing Systems: An Energy Efficient Massive Parallel Computational Model

Muhammad Usman Ashraf, Fathy Alburaei Eassa, Aiiad Ahmad Albeshri, Abdullah Algarni
Department of Computer Science
King Abdulaziz University (KAU)
Jeddah, Saudi Arabia

***Abstract*—The emerging Exascale supercomputing system expected till 2020 will unravel many scientific mysteries. This extreme computing system will achieve a thousand-fold increase in computing power compared to the current petascale computing system. The forthcoming system will assist system designers and development communities in navigating from traditional homogeneous to the heterogeneous systems that will be incorporated into powerful accelerated GPU devices beside traditional CPUs. For achieving ExaFlops (10^{18} calculations per second) performance through the ultrascale and energy-efficient system, the current technologies are facing several challenges. Massive parallelism is one of these challenges, which requires a novel energy-efficient parallel programming (PP) model for providing the massively parallel performance. In the current study, a new parallel programming model has been proposed, which is capable of achieving massively parallel performance through coarse-grained and fine-grained parallelism over inter-node and intra-node architectural-based processing. The suggested model is a tri-level hybrid of MPI, OpenMP and CUDA that is computable over a heterogeneous system with the collaboration of traditional CPUs and energy-efficient GPU devices. Furthermore, the developed model has been demonstrated by implementing dense matrix multiplication (DMM). The proposed model is considered an initial and leading model for obtaining massively parallel performance in an Exascale computing system.**

Keywords*—*Exascale computing; high-performance computing (HPC); massive parallelism; super computing; energy efficiency; hybrid programming; CUDA; OpenMP; MPI

I. INTRODUCTION

The high-performance computing (HPC) community anticipates that a new supercomputing technology called the exascale computing system will be available at the end of the current decade. This powerful supercomputer system will provide a thousand-fold computing power increase over the current petascale computing system and will enable the unscrambling of many scientific mysteries by computing 1 ExaFlops (10^{18} calculations per second) [1], [22], [23]. This ultrascale computing system will be composed of millions of heterogeneous nodes, which will contain multiple traditional CPUs and many-core General Purpose Graphics Processing Units (GPGPU) devices. In the current petascale computing system, the power consumption is approximately 25-60 MW, by using up to 10 M cores. According to this ratio, the power consumption demand of the exascale computing system will be more than 130 Megawatts. On the way towards the exascale supercomputing system, the United States Department of Energy (US DoE) and other HPC pioneers defined some primary constraints, including power consumption (PC) ≈ 25-30 MW, system development cost (DC) ≈ 200 million USD, system time to delivery (DT) ≈ 2020 and number of cores (NC) ≈ 100 million [25]. The primary limitation for the exascale system is that it does not exist yet. However, in trying to achieve ExaFlops-level performance under these strict limitations, current technologies are facing several fundamental challenges [24]. At a broad level, these challenges can be categorized according to the themes that are listed in Table I.

TABLE I. EXASCALE COMPUTING CHALLENGES

Challenge	Description
Power consumption management	Managing power consumption through new energy-efficient algorithms and devices
Programming models	New programming models are required for programming CPU + GPU-based heterogeneous systems
Novel architectures	New architectures and frameworks that can be implemented with non-traditional processors are required
Massive Parallelism	New parallel programming approaches are required that can provide massive parallelism using new accelerated devices
Resiliency	The system should be able to provide correct computation in the face of faults in the system
Memory management mechanisms	To improve data diversity and bandwidth

One traditional way to enhance the system performance at the exascale level is to improve clock speed. However, in the future, the clock speed will be limited to 1 GHz. An alternative approach is to increase the number of cores in the system. According to the defined limitations for the exascale computing system, the number of cores should not exceed 100 million. Generally, if we increase the number of resources (cores) to enhance the performance, it ultimately will increase the power consumption for computation. Another option is to achieve 'massive parallelism' in the system to improve system performance at the exascale level. Parallelization through different PP models has already been explored and examined, with the aim of exploiting a future exascale computing system. From the start of the current decade, in consideration of the many HPC applications, which include climate and environmental modeling, computation fluid dynamics (CFD) [2], molecular nanotechnology and intelligent planetary

spacecraft [3], new versions of PP models such as High-Performance FORTRAN (HPF) [4], [7] and an explicit message passing interface (MPI) were introduced to attain petaflop-level performance in the system.

To overcome the architectural challenges of petascale systems, many new approaches were introduced, including pure parallelism, *in situ* processing [5], 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 that are required for computing thread-level parallelism in a cluster system. A new hybrid PP model was required for localizing the work from the distributed system in the spectral element method and performing efficient computations using multiple threads. 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. [6]. The hybrid model of MPI and OpenMP [21] 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. However, 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 [9]. To update the thread-level parallelism and address the overhead in thread creation/destruction and synchronization, OpenMP 4.0 was released in 2013 [8]. This new version was equipped with new features for error handling, tasking extensions, atomics and support for accelerated computation.

Recently, a dramatic change occurred in hardware technology development and new powerful computational devices were introduced, such as the General-Purpose Graphical Processing Unit (GPGPU) by NIVIDIA [10], AMD [48], ARM [49] and Many Integrated Cores (MIC) by Intel [11], [12]. These devices are thousands-fold more powerful than the traditional CPU devices. These Single-Instruction Multiple-Data (SIMD)-architecture-based many-core devices contain thousands of cores and are capable of performing thread-level execution. The old GPU models were only used for graphics processing, whereas the latest devices are able to perform general-purpose processing as well. To program GPUs, many PP models have been introduced, including OpenCL [20], OpenACC [50], CUDA and OpenMP [16], which are also available for GPU programming. So far, CUDA is considered the most capable model for performing thread-level optimization. Nevertheless, parallelized thread execution has been transformed from conventional CPU cores to GPU-accelerated devices. A detailed comparative study has been conducted by Ashraf et al. [19].

II. Navigation in the Hierarchy Level

Parallelism has brought about a great revolution in system performance enhancement. Parallelism was introduced in the 90s. The Terascale computing systems were based on coarse-grained parallelism, which was accomplished at the inter-node level through single-hierarchy models such as MPI [31]. To enhance the parallelism, a dual-hierarchy model was introduced for petascale supercomputing systems [32]. The objective of the petascale system was to achieve both coarse-grained and fine-grained parallelism through inter-node and intra-node processing. Many dual-hierarchy-level approaches were proposed to achieve both types of parallelism, including Hybrid MPI + OpenMP. In this dual-level hybrid model, MPI was used to achieve coarse-grained parallelism and OpenMP was used to achieve fine-grained parallelism at the thread level. The major problem with this model was massive power consumption while transferring data over CPU cores [33]. To overcome the power consumption challenge, new energy-efficient devices are introduced, such as GPGPU and MIC. From the software perspective, new programming approaches and models are required that can utilize these energy-efficient accelerated devices with traditional CPU cores through massive parallelism [23]. To achieve massive parallelism in the system, the hierarchy level in PP models is shifted from dual to tri-level, which is considered a promising level for future exascale computing systems. To add a third level of parallelism to the current homogeneous MPI + OpenMP model, a new tri-level model has been considered, which will be a hybrid MPI + OpenMP + X model [34].

Leading to a hybrid approach for massive parallelism, a new tri-level hybrid PP model was proposed for symmetric multiprocessor (SMP) cluster architectures in [12]. This model was based on message passing for inter-SMP node communication, loop directives by OpenMP for intra-SMP node parallelization and vectorization for each processing element (PE). The fundamental objective of this method was to combine coarse-grained and fine-grained parallelism. MPI was used to achieve coarse-grained parallelism and OpenMP was used to achieve fine-grained parallelism by parallelizing loops inside each SMP node. The hybrid approach is advantageous over flat MPI as it does not allow the passage of messages in all SMP nodes. This tri-level hybrid model was implemented to solve 3D linear elastic problems [35] by achieving a performance of 3.80 TFLOPS. In addition, tri-level hybrid and flat MPI programming models achieve similar performance. However, the hybrid model outperforms flat MPI in problems with large numbers of SMP nodes. Due to its monolithic power consumption, this model is not applicable to the exascale computing system. However, according to Amarasinghe et al. [36], unanimous implementation of existing models and powerful GPU devices for better performance of the system should be reinvestigated. For the future exascale system, the tri-level 'X' model will be considered as an additional model that will be responsible for the programming of accelerated GPU devices. To determine the X factor in the tri-level hybrid model, critical studies were conducted, where several models were proposed and compared with respect to performance, computation, optimization and many other metrics [26]-[29]. Evaluations showed that the current compiler of over-simplified 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. These metrics prevent full utilization of available resources for HPC heterogeneous computing systems.

Eventually, we finalized the X model as CUDA to compute accelerated GPU devices. Fig. 1 shows the fundamental navigational model for massive parallel programming.

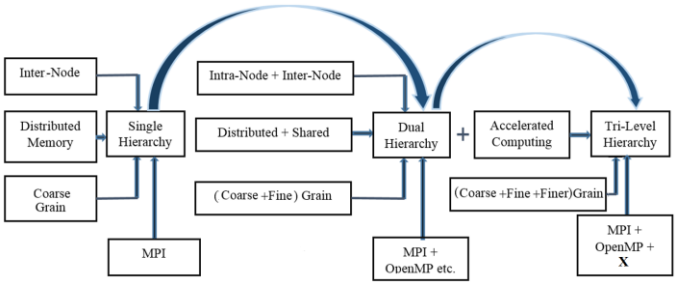

Fig. 1. Hierarchy navigation in the programming model.

This tri-level hybrid model is capable of achieving both coarse-grained and fine-grained parallelism through inter-node and intra-node processing over a heterogeneous cluster system. Leading to this architecture, we have proposed an initiative PP hybrid (MPI, OpenMP and CUDA) model with an optimized approach, which will be a promising framework for achieving the desired performance for exascale computing systems through massive parallelism.

A. MPI

MPI is a well-known traditional independent library that has been used for communication among the explicit processes in a distributed computing system. Historically, the standard version of MPI is considered the MPI-1.0 version from 1994. Many modifications, additions and clarifications have been made in different versions. Recently, in 2015, a new, mature version of MPI, namely, 'MPI 3.1', was released, to which many new features had been added, including environment management, point-to-point message passing, process creation and management, and collective communications [15]. Throughout HPC revolutionary development, MPI has been a prominent model for message passing in distributed nodes and multi-processor systems. In the future, it has been predicted that MPI will remain the best option for message passing among heterogamous devices over the cluster system, even though the original MPI designer did not focus on the exascale computing system, which requires some MPI specifications such as the maintenance of global state per process, memory management during communication within MPI processes, and process synchronization [37]. These MPI specifications must be adapted for the exascale computing system.

B. OpenMP

Open Specification for Multi-Processing (OpenMP) is one of the most frequently used models for SIMD thread-level parallel execution, which determines the set of directives, environment variables and multiple library routines. These specifications are supported in FORTRAN and C/C++ for using shared memory parallelism. The most recent version, namely, OpenMP 4.5, contains various new features, including error handling, tasking extensions, atomics and accelerated computation [38]. A new synchronization strategy has been introduced, where multiple tasks are grouped and synchronized using the 'taskgroup' construct [13]. In this way, many new constructs are added into the OpenMP 4.5 version that manages the threads efficiently. Similarly, loop parallelization with unbalanced amounts of work is also optimized as 'taskloop' using new directives [14]. One shortcoming of OpenMP is that it can be applied only for shared memory platforms on a single node, and not for cluster systems, which limits the use of the MPI option for cluster computing. However, it is anticipated that OpenMP will be promising model for exascale application, to achieve massive parallelism at the thread level.

C. CUDA

Recently, NIVIDA introduced CUDA (Compute Unified Device Architecture), which is a unique thread-level parallel computing platform for programming massive parallel computing accelerated GPUs. CUDA is supported by FORTRAN and C/C++ for programming accelerated GPGPUs. The current CUDA release, namely, CUDA 8.0, which is the most feature-packed and powerful, is available with novel profiling capabilities. In addition, it supports the Pascal GPU architecture and lambda heterogeneous compilers [17], [18]. In CUDA parallel programming, an application that contains the sequential program 'CUDA Kernel' is available, which executes programs in parallel on GPU devices. The Single Program Multiple Data (SPMD)-based kernel is initialized by passing multiple parameters, including grid size and block size. Based on modern GPU architecture, the GPU Block dispatcher schedules the grid by assigning each thread to one of the computational cores, and these threads are synchronized by self-cooperation. Each block has its own shared memory, which is accessible to every core inside it. Threads process data using this shared memory within that block and return the results to the scheduler. This processed data is stored in GPU global memory, which is accessible to host CPU cores. CPU cores read data from GPU global memory and transfer data from GPU to CPU cores and memory. In this way, we can achieve massive parallelism through heterogeneous CPU + GPU computation using CUDA.

III. Tri-Level Hybrid Parallel Programming Model

In this section, we present the proposed tri-level hybrid PP model for the exascale computing system. Based on the hierarchy navigation in previous parallel programming models, the proposed approach is a hybrid of MPI, OpenMP and CUDA.

A. Inter-Node Computation

In the proposed model, initially, some fundamental specifications, such as the number of nodes, number of CPUs per node, number of CPU cores, number of accelerated GPU devices, and memory levels, are the requirements of the system on which the model is to be implemented. After obtaining these fundamental specifications of the system, the parallel computing process is initiated. The top-level inter-node parallelism was achieved through the standard-specification MPI library to parallelize the distributed nodes. Immediately after MPI initialization, some necessary statements were executed to define the MPI communication size and the ranks of the available processes in MPI communication. Usually, the process with rank '0' is considered the master process, while the rest of the processes are considered slave processes. Before

broadcasting begins, data and many other necessary parameters are distributed over connected nodes in the system. For task mapping, the master process communicates with all slave processes to distribute/gather data. To maintain synchronization while sending and receiving data, blocking methods 'MPI_Send' and 'MPI_Recv' were respectively used, instead of non-blocking methods 'MPI_Isend' and 'MPI_Irecv'. These communication methods are better synchronized and more reliable for producing pure error-free parallelism.

B. Intra-Node Computation

Once the data have been shared over all distributed nodes, the second level of multi-threaded intra-node parallel processing is initiated through OpenMP, which uses shared memory among multiple CPU cores of the system. At this stage, multiple OpenMP pragmas were used to achieve fine-grained parallelism by defining all looping and independent parallel computing statements within the OpenMP parallel region. As this is middle-level parallelism, the resources of the current and next levels of parallelism are correlated. Before entering the third step, the number of available CPU threads in the system is determined, followed by the estimation of the number of accelerated GPU devices that are installed in the system. For the optimization of resources and results, the numbers of CPU threads and GPU devices should be same. Consequently, determination of the numbers of CPU threads and GPU devices can facilitate the adjustment of their strengths by using the following pre-defined functions:

*cudaGetDeviceCount (*numGPU*)*; //get number of GPUs

*omp_set_num_threads(*numGPU*);*// Set number of Threads

cudaThreadSynchronize(); // synchronize CUDA threads

C. Accelerated GPU Computation

Within the outer scope of OpenMP, another thread level of parallelism was created through the shared memory system over accelerated GPU devices, which provide finer granularity using GPU cores. This complicated heterogeneous CPU+GPU computation is supported by different programming models using FORTRAN and C/C++. In our proposed model, we used CUDA to perform this heterogeneous computation, where the SIMD-based data segment was transferred from Host to GPU core using built-in CUDA methods. Fig. 2 presents the workflow of tri-hybrid parallel programming as follows.

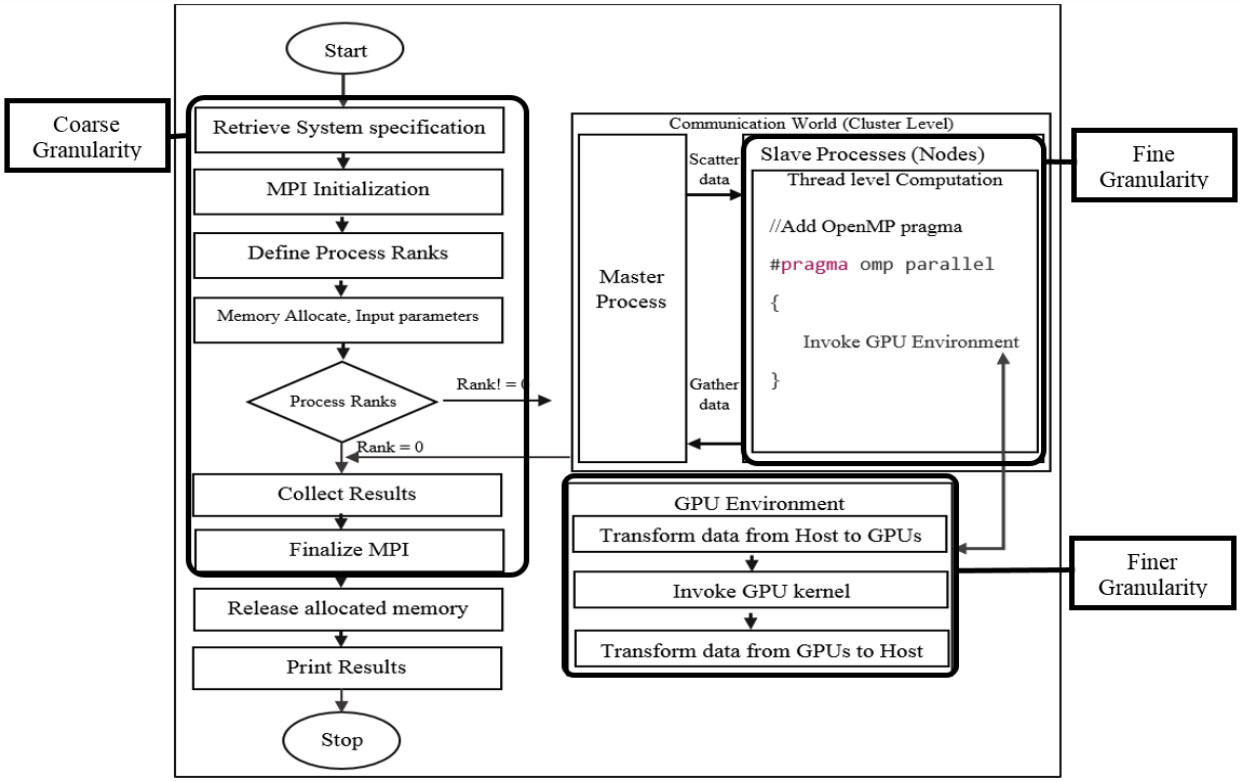

Fig. 2. Workflow of the hybrid parallel programming model.

At the same time, some fundamental information, including grid size and block size, were broadcasted with the CUDA kernel to restrict computation according to given specifications. To create a generic kernel, we defined template datatypes, which were provided by C++, that accept any datatypes as parameters and perform computations accordingly. Once parallel data computation was completed through GPU cores, it used a similar datatype from GPU to Host cores that entered again in OpenMP region. After finishing this complicated heterogeneous computation among CPUs and GPUs, the MPI master process collected all processed data and exited the parallel zone. The detailed sequence of these three levels of parallelism is illustrated in Fig. 2.

IV. EXPERIMENTAL SETUP

This section describes the experimental setup that was used to implement the proposed model. Moreover, we quantified different HPC-related metrics, including performance (number of GFlops/s) and energy efficiency (GFlops/Watt) in the system. A detailed description of these metrics is presented in this section.

A. Experimental Platform

The proposed tri-level hybrid model was implemented on an Aziz-Fujistu Primergy CX400 Intel Xeon Truescale QDR supercomputer, which was manufactured by Fujistu, at the HPC center of King Abdul-Aziz University, Jeddah, Saudi Arabia [39]. In 2015, Aziz was ranked 360th in the list of the top-500 HPC supercomputers [40]. The Aziz supercomputer is comprised of 380 regular (thin) and 112 large (fat) compute nodes. Recently, Aziz was upgraded with two SIMD-architecture-based accelerated GPU compute nodes (NVIDIA Tesla K20 GPU, 2496 CUDA Cores). Moreover, 2 MIC nodes, each with an Intel Xeon Phi Coprocessor with 60 cores, were installed. Aziz consists of a total of 11904 cores. The memories that are offered by regular and large nodes are 96 GB and 256 GB, respectively. Each node that contains an Intel E5-2695v2 processor with 2.4 GHz and 12 Cores is run by the Cent 6.4 operating system. Aziz is linked using three different networks: the InfiniBand network, the User network and the Management network. Moreover, all nodes are interconnected with one another. The file system is parallelized through the InfiniBand network. In addition, the login system and job submission are handled through the user network, while the management network is used for management purposes only. Aziz is capable of achieving 211.3 TFlops/s Linpack performance and 228.5 TFlops/s theoretical peak performance [41].

B. Performance Measurement

Performance is the first and fundamental metric of HPC systems, which is measured in Flops (number of floating-point operations per second) in the current experiments. Usually, in a parallel programing system, Flops are calculated at the peak performance of the system and for implementing algorithms. Let F_p denote the Flops at peak performance and F_m denote the Flops for implementing algorithms. F_c can be calculated as:

$$F_c = \frac{F_p}{F_m} \quad (1)$$

Using the peak performance of 211.3 TFlops/s of the Aziz supercomputer, we measured the performance range by executing target-dense MM with different datasets.

C. Power Measurement

Limiting power consumption is one of the vital challenges for current and future supercomputing technologies. The primary objective of future research for the exascale computing system is the optimal selection of hardware and software for achieving high performance under the power consumption limitations [42]. Many HPC pioneers have initiated and developed energy-efficient devices, such as NVIDIA GPGPU [43], AMD GPU [44], and Intel MIC [45]. Similarly, software development communities are trying to develop new programming models that can provide outstanding performance under energy constraints.

Generally, a system is evaluated according to its energy consumption, which indicates the power rate at which processing was executed, as described in (2).

$$E\ (kWh) = \int_0^t V.I\ \ (dt) \quad (2)$$

From the 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 (3).

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

On the basis of the dictated factors and the fundamental energy evaluation (2), we quantified these factors in the current study with respect to system performance and power consumption. The power consumption is the sum of the products of the power of each component and the corresponding duration [28]. The measurement of power consumption is divided into two categories:

1. System Specification.
2. Application Specification.

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

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

From (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 (5):

$$\boldsymbol{P_{app}(w)} = \sum_{i=1}^{Napp} \boldsymbol{P^i_{GPU}}\ (\boldsymbol{w^i}) + \ \boldsymbol{P}_{CPU}\ \sum_j^M\ (\boldsymbol{w^j}) + \ \boldsymbol{P_{mainboard}}\left(\boldsymbol{w_{app}}\right) \quad (5)$$

According to (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.

V. EXPERIMENTAL RESULTS

In this section, we investigate the proposed tri-level hybrid parallel programming model via implementation of linear algebraic Dense Matrix Multiplication (DMM) [46]. The purpose of this study was to execute the DMM in the proposed model on a heterogeneous-architecture-based Aziz supercomputer and to determine the performance and power consumption, which are vital metrics for emerging exascale computing systems. We recorded different datasets of DMM through multiple CUDA kernels, which demonstrated that multiple kernels could produce energy-efficient results simultaneously. Moreover, during execution, the parallel performance of multiple kernels and the power consumption were evaluated, which indicated that the best performance was attained using a small and optimized number of kernels in an

energy-efficient way. This is due to the optimized computation over heterogeneous CPU and GPU cores using the CUDA platform. In contrast, using many kernels provided lower performance due to unnecessary communication among non-optimized CUDA kernels. A simple implementation of DMM, along with the defined parameters, is presented in Table II.

TABLE II. A NAÏVE CODE AND PARAMETERS OF IMPLEMENTED DMM

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]

However, we were unable to find a detailed optimization strategy for DMM due to space limitations, as explained by Tiwari et al. [30]. To explore the implementation strategy for DMM, we reused the z array in the buffer registers and the x and y arrays in the caches. These kernel configurations were obtained by varying parameters. In our implementations, the achieved performance ranged from 200 to 1100 GFlops for all implemented kernels for datasets of sizes 1000 to 10000, and the average was 716 GFlops, as shown in Fig. 3.

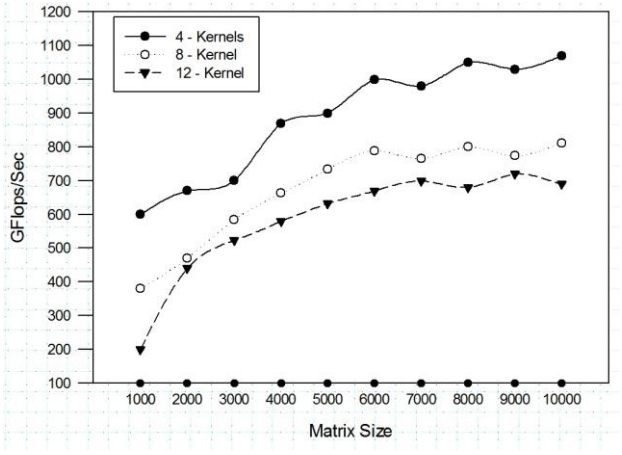

Fig. 3. Performance in DMM through multiple Kernel configurations.

During DMM computation, 4 CPU threads per node with 4 kernels achieved the best performance compared to all other configured kernels and achieved 68% of the peak performance with 1086 Gflops. Using 12 kernels produced efficient performance, but increased the energy efficiency due to unneeded communication in data processing.

Along with performance, we quantified another primary metric, namely, energy consumption, which was 28 Joules. At maximum DMM for a dataset of size 10000 through an optimized 4-kernel configuration, the quantified energy efficiency was 8.3 Gflops/W. The increment of resources affected energy efficiency dramatically and reduced it to 5.6 Gflops/W, as shown in Fig. 4.

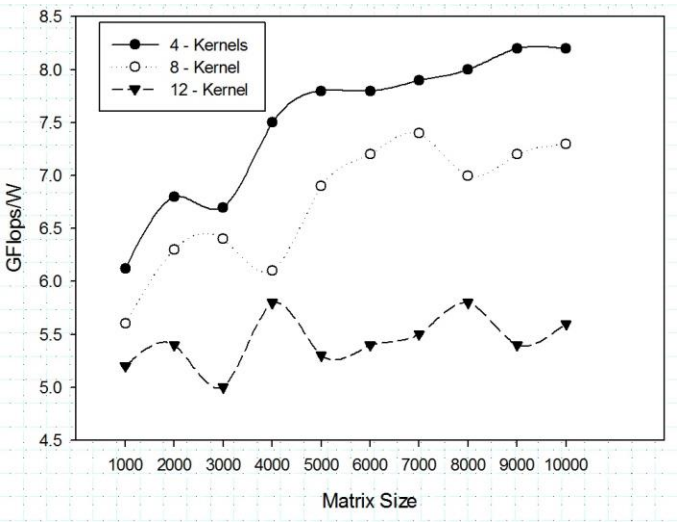

Fig. 4. Energy efficiency in DMM for different multiple-kernel configurations.

Based on performance and energy efficiency, a tradeoff between the two metrics [47] can be determined as follows:

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

Following this tradeoff, we calculated the ratio between performance and energy efficiency, which describes the performance that is achievable for a given energy efficiency, as shown in Fig. 5. Each vertical and horizontal line represents information about performance and energy efficiency, respectively. We can fix the configuration and parameters at any intersecting point to provide maximum performance and energy efficiency. These evaluations determined that the best performance-energy efficiency that can be achieved using the proposed model on the Aziz supercomputer reached 1086 GFlops, which corresponds to an energy efficiency of 8.3 GFlops/W.

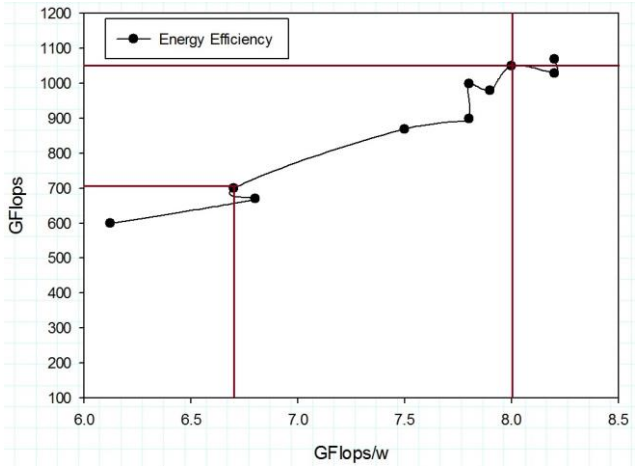

Fig. 5. Performance-energy efficiency tradeoff.

VI. EXASCALE COMPUTING SYSTEM DEMAND

The biggest challenge for the study of the emerging exascale computing system is that such a system does not exist yet. Therefore, predictive exascale-level data can be obtained using current computing systems (Terascale, petascale). In this section, a critical statistical analysis of experimental results that were obtained from the Aziz supercomputer is conducted. This statistical analysis is based on the metrics, including performance and energy efficiency that are required to satisfy the demands of the exascale computing system. The architecture of our experimental platform is heterogeneous (CPU + GPU) and is based on a cluster system that contains 11904 cores, which are integrated over 494 connected nodes. Using processing devices and memory structure, this system can provide 211.3 Tflops/s Linpack performance and 228.5 Tflops/s theoretical peak performance. In our experiments, we implemented DMM using different kernel sizes and obtained 68% of the peak performance with 1086 Gflops by consuming 28 joules of energy, which yielded 8.3 Gflops/Sec energy efficiency. This energy efficiency was determined using the fundamental formula that is given as follows:

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

However,

$$watt = \frac{Joule}{\text{Second}} \quad \text{or} \quad W = J / S \tag{6}$$

TABLE III. EXASCALE SYSTEM CONFIGURATION

Feature	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

According to (6) and the system configurations, our system consumed 130 watts at the best performance and energy efficiency. According to the exascale system constraints and predictive configurations, as listed in Table III, our system required a thousand-fold increase in current resources to perform exaFlops computations.

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

TABLE IV. METRIC ANALYSIS FOR DIFFERENT PLATFORMS

Metric	Platforms		
	Aziz Supercomputer		Exascale
	Achieved	Predictive	
Performance	1086 Gflops	≈ 230 Pflops	1 ExaFlops
P. Consumption	130 Watt	≈ 27 M.W	≈ 25 M.W
Energy Efficiency	8.3 Gflops/W	≈ 8.5 Pflops/M.W	≈ 40 Pflops/M.W

Table IV describes a statistical analysis of current and future platforms for massive computation. The currently available Aziz platforms are categorized into achieved and predictive domains. Both platforms are analyzed on the basis of the metrics that were used in DMM. In the predictive platform, the scalability of the Aziz supercomputer was considered according to the configurations of the exascale computing system, which facilitated the determination of a predictive benchmark against each metric. The predictive benchmark does not depend on the demand of the exascale system. Therefore, it can be considered an initial step in achieving the required computational level for the exascale system. In the current study, our evaluations have raised numerous challenging questions, which will open new avenues of research for scientific communities, developers and vendors in the future:

- Which programming layer is responsible for managing the dynamic behavior of resources and code irregularity, and how?
- Sometime algorithms provide better performance with less energy efficiency. What optimized method should be adopted to satisfy the trade-off between the metrics?
- Memory management plays a vital role in enhancing system performance. To increase the efficiency of memory management, what additional hooks are required?
- How can data be managed to reduce power consumption when GPU cores occupy complete the warp for small executions?

These questions suggest new challenges regarding the satisfaction of HPC metrics through massive parallelism. However, we should reconsider our implemented algorithms, frameworks, benchmarks, energy management algorithms, communication mechanisms, memory management mechanisms and load balancing mechanisms, since these factors are paramount concerns for exascale systems.

VII. CONCLUSIONS

HPC technology is being shifted from the petascale to the extreme "exascale" computing system. On the road to the exascale system, due to some strict limitations on energy consumption, system cost, number of cores and time to delivery, there are many vital challenges for vendors and development communities. One of these major challenges is to achieve massive parallelism through energy-efficient mechanisms. In this study, we have proposed a new Tri-Level hybrid (MPI + OpenMP + CUDA) parallel programming model. The proposed model is applicable for heterogeneous (CPU + GPU) distributed systems, to achieve massive parallelism with coarse, fine and finer granularity. To evaluate the proposed model, we implemented DMM with different datasets through multiple kernels. All implementations were performed on an Aziz - Fujitsu PRIMERGY CX400, Intel Xeon E5-2695v2 12C 2.4 GHz, Intel TrueScale QDR supercomputer. We evaluated our model using HPC metrics, including performance, power consumption and energy efficiency. Moreover, we provided some predictive results as

to the performance that will be achievable through exascale-level scalability in the current system. Based on the results of our implementations, the proposed model can be considered a pioneering model for HPC applications. As the exascale system does not yet exist and all the implementations and results are predictive, we must reconsider the generic challenges, including the implementation of algorithms, frameworks, benchmarks, energy management algorithms, and communication mechanisms.

By future perspectives, we have specified some additional questions that are open challenges, while achieving extreme performance with energy efficiency through massive parallelism in the HPC system. Moreover, fixed optimization in a heterogeneous computing system is not possible. Nevertheless, an adaptive framework is required for adjusting the model to the system configuration and the application requirements.

REFERENCES

[1] Perarnau, Swann, Rinku Gupta, and Pete Beckman. "Argo: An Exascale Operating System and Runtime." (2015).

[2] Zhou, Min. Petascale adaptive computational fluid dynamics. Diss. RENSSELAER POLYTECHNIC INSTITUTE, 2009.

[3] Dongarra, Jack J., and David W. Walker. "The quest for petascale computing." Computing in Science & Engineering 3.3 (2001): 32-39.

[4] Jin, Haoqiang, et al. "High performance computing using MPI and OpenMP on multi-core parallel systems." Parallel Computing 37.9 (2011): 562-575.

[5] [5] Ma, Kwan-Liu, et al. "In-situ processing and visualization for ultrascale simulations." Journal of Physics: Conference Series. Vol. 78. No. 1. IOP Publishing, 2007.

[6] Dong, Suchuan, and George Em Karniadakis. "Dual-level parallelism for high-order CFD methods." Parallel Computing 30.1 (2004): 1-20.

[7] Shafto, Mike, et al. "Modeling, simulation, information technology & processing roadmap." NASA, Washington, DC, USA, Tech. Rep 11 (2012).

[8] Martineau, Matt, Simon McIntosh-Smith, and Wayne Gaudin. "Evaluating OpenMP 4.0's Effectiveness as a Heterogeneous PP Model." Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International. IEEE, 2016.

[9] Jin, Shuangshuang, and David P. Chassin. "Thread Group Multithreading: Accelerating the Computation of an Agent-Based Power System Modeling and Simulation Tool--C GridLAB-D." 2014 47th Hawaii International Conference on System Sciences. IEEE, 2014.

[10] Hoegg, Thomas, et al. "Flow Driven GPGPU Programming combining Textual and Graphical Programming." Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores. ACM, 2016.

[11] Shan, Hongzhang, et al. "Thread-level parallelization and optimization of NWChem for the Intel MIC architecture." Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores. ACM, 2015.

[12] Nakajima, Kengo. "Three-level hybrid vs. flat mpi on the earth simulator: Parallel iterative solvers for finite-element method." Applied Numerical Mathematics 54.2 (2005): 237-255.

[13] Terboven, C., Hahnfeld, J., Teruel, X., Mateo, S., Duran, A., Klemm, M., Olivier, S.L. and de Supinski, B.R., 2016, October. Approaches for Task Affinity in OpenMP. In International Workshop on OpenMP (pp. 102-115). Springer International Publishing.

[14] Podobas, Artur, and Sven Karlsson. "Towards Unifying OpenMP Under the Task-Parallel Paradigm." International Workshop on OpenMP. Springer International Publishing, 2016.

[15] Dinan, James, et al. "An implementation and evaluation of the MPI 3.0 onesided communication interface." Concurrency and Computation: Practice and Experience (2016).

[16] Concise Comparison Adds OpenMP Versus OpenACC To CUDA Versus OpenCL Debates "techenablement.com/concise-comparision-adds-openmp-versus-openacc-to-cuda-versus-opencl-debates/", 12 Nov 2016.

[17] Fleuret, François. "Predicting the dynamics of 2d objects with a deep residual network." arXiv preprint arXiv:1610.04032 (2016).

[18] NVIDIA Accelerated Computing "developer.nvidia.com/cuda-downloads", 02 Nov 2016.

[19] Ashraf, Muhammad Usman, Fadi Fouz, and Fathy Alboraei Eassa. "Empirical Analysis of HPC Using Different Programming Models." (2016).

[20] Ashraf, Muhammad Usman, and Fathy Elbouraey Eassa. "OpenGL Based Testing Tool Architecture for Exascale Computing." International Journal of Computer Science and Security (IJCSS) 9.5: 238.

[21] Ashraf, Muhammad Usman, and Fathy Elbouraey Eassa. "Hybrid Model Based Testing Tool Architecture for Exascale Computing System."International Journal of Computer Science and Security (IJCSS) 9.5 (2015): 245.

[22] Shalf, John, Sudip Dosanjh, and John Morrison. "Exascale computing technology challenges." International Conference on High Performance Computing for Computational Science. Springer Berlin Heidelberg, 2010.

[23] Reed, Daniel A., and Jack Dongarra. "Exascale computing and big data."Communications of the ACM 58.7 (2015): 56-68. Cappello, Franck, et al. "Toward exascale resilience." International Journal of High Performance Computing Applications (2009).

[24] Marc Snir, Robert W Wisniewski, Jacob A Abraham, Sarita V Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, Andrew A Chien, Paul Coteus, Nathan A DeBardeleben, Pedro C Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. Addressing failures in exascale computing. International Journal of High Performance Computing Applications, 28(2):129{173, May 2014.

[25] Reed, Daniel, et al. DOE Advanced Scientific Computing Advisory Committee (ASCAC) Report: Exascale Computing Initiative Review. USDOE Office of Science (SC)(United States), 2015.

[26] Hoshino, Tetsuya, et al. "CUDA vs OpenACC: Performance case studies with kernel benchmarks and a memory-bound CFD application." Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. IEEE, 2013.

[27] Herdman, J. A., et al. "Accelerating hydrocodes with OpenACC, OpenCL and CUDA." High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:. IEEE, 2012.

[28] Lashgar, Ahmad, Alireza Majidi, and Amirali Baniasadi. "IPMACC: Open source OpenACC to CUDA/OpenCL translator." arXiv preprint arXiv:1412.1127 (2014).

[29] [29] Christgau, Steffen, et al. "A comparison of CUDA and OpenACC: accelerating the tsunami simulation easywave." Architecture of Computing Systems (ARCS), 2014 Workshop Proceedings. VDE, 2014.

[30] A. Tiwari, C. Chen, J. Chame, M. Hall, and J. Hollingsworth. A Scalable Auto-Tuning Framework for Compiler Optimization. In IPDPS'09, Rome, Italy, May 2009.

[31] Gabriel, Edgar, et al. "Open MPI: Goals, concept, and design of a next generation MPI implementation." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer Berlin Heidelberg, 2004.

[32] Mininni, Pablo D., et al. "A hybrid MPI–OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence." Parallel Computing 37.6 (2011): 316-326.

[33] Hennecke, Michael, et al. "Measuring power consumption on IBM Blue Gene/P." Computer Science-Research and Development 27.4 (2012): 329-336.

[34] Jacobsen, Dana A., and Inanc Senocak. "Multi-level parallelism for incompressible flow computations on GPU clusters." Parallel Computing 39.1 (2013): 1-20.

[35] Nguyen-Thoi, T., et al. "A face-based smoothed finite element method (FS-FEM) for 3D linear and geometrically non-linear solid mechanics

problems using 4-node tetrahedral elements." International journal for numerical methods in Engineering 78.3 (2009): 324-353.

[36] Amarasinghe, Saman, et al. "ASCR programming challenges for exascale computing." Report of the 2011 Workshop on Exascale Programming Challenges. 2011.

[37] Message passing Interface, https://computing.llnl.gov/tutorials/mpi/ , 20 June, 2017 [03 Aug, 2017]

[38] Royuela, Sara, et al. "OpenMP Tasking Model for Ada: Safety and Correctness." Ada-Europe International Conference on Reliable Software Technologies. Springer, Cham, 2017.

[39] Fujitsu to Provide High-Performance Computing and Services Solution to King Abdulaziz University, http://www.fujitsu.com/global/about/resources/news/press-releases/2014/0922-01.html, 22 Sep, 2014 [06 July, 2017]

[40] King Abdulaziz University, https://www.top500.org/site/50585, June 2015 [03 Aug, 2017]

[41] Aziz - Fujitsu PRIMERGY CX400, Intel Xeon E5-2695v2 12C 2.4GHz, Intel TrueScale QDR, https://www.top500.org/system/178571, June 2015 [03 Aug, 2017]

[42] L. A. Barroso. The price of performance. Queue, 3(7):48–53, September 2005.

[43] Foley, Denis, and John Danskin. "Ultra-Performance Pascal GPU and NVLink Interconnect." IEEE Micro 37.2 (2017): 7-17.

[44] Rohr, David, et al. "An energy-efficient multi-GPU supercomputer." High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on. IEEE, 2014.

[45] Chrysos, George. "Intel® Xeon Phi™ coprocessor-the architecture." Intel Whitepaper 176 (2014).

[46] Gallivan, Kyle A., Robert J. Plemmons, and Ahmed H. Sameh. "Parallel algorithms for dense linear algebra computations." SIAM review 32.1 (1990): 54-135.

[47] Anzt, Hartwig, et al. "Experiences in autotuning matrix multiplication for energy minimization on GPUs." Concurrency and Computation: Practice and Experience 27.17 (2015): 5096-5113.

[48] Rajovic, Nikola, et al. "The low power architecture approach towards exascale computing." Journal of Computational Science4.6 (2013): 439-443.

[49] Rajovic, Nikola, et al. "Tibidabo: Making the case for an ARM-based HPC system." Future Generation Computer Systems 36 (2014): 322-334.

[50] Wolfe, Michael, et al. "Implementing the OpenACC Data Model." Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International. IEEE, 2017.