A Secured Large Heterogeneous HPC Cluster System using Massive Parallel Programming Model with Accelerated GPUs

Khalid Alsubhi
Faculty of Computing and Information Technology
King Abdulaziz University, Saudi Arabia

***Abstract*—High Performace Computing (HPC) architectures are expected to develop first ExaFlops computer. This Exascale processing framework will be proficient to register ExaFlops estimation every subsequent that is thousands-overlay increment in current Petascale framework. Current advancements are confronting a few difficulties to move toward such outrageous registering framework. It has been anticipated that billion-way of parallelism will be exploited to discover Exascale level secured system that provide massive performance under predefined limitations such as processing cores and power consumption. However, the key elements of the strategies are required to develop a secured ExaFlops level energy efficient system. This study proposes a non-blocking, overlapping and GPU computation based tri-hybird model (OpenMP, CUDA and MPI) model that provide a massive parallelism through different granularity levels. We implemented the three different message passing strategies including and performed the experiments on Aziz-Fujitsu PRIMERGY CX400 supercomputer. It was observed that a comprehensive experimental study has been conducted to validate the performance and energy efficiency of our model. Experimental investigation shows that the EPC could be considered as an initiative and leading model to achieve massive performance through efficient scheme for Exascale computing systems.**

Keywords*—High Performance Computing HPC; MPI; OpenMP; CUDA; Supercomputing Systems

I. Introduction

Since last three decades, High performance computing (HPC), played a fundamental role in scientific endeavour where vendors emphasized to improve system performance by dramatic increasing through on-chip parallelism. According to Top-500 supercomputers list, an improvement of 10x in system performance is discovered after every 3.6 years [1]. A supercomputer in 2012, Titan Cray XK7 was capable to achieve 18 PFLOPs under the 8.3 MW power consumption [2]. Moving on the vision to enhance system performance to solve the complex problems, Tianhe-II the current supercomputer manufactured by NUDT is capable to deliver 55.2 PFLOPs with 17MW power consumption [3]. The demand of computation for solving complex problem envisioned to develop new supercomputer [4]. This extraordinary scale processing framework will be proficient to compute 1018 FLOPS activities for each subsequent that is thousand-crease increment in current Petascale framework. As per expectations, Exascale figuring framework will be involved countless heterogeneous process hubs connected by complex systems [5]. The essential issue for HPC frameworks is that such Extreme (Exascale) processing framework doesn't exist yet, anyway everything toward Exascale is simply expectations and contemplations. To improve the system throughput, the trend has been changed from traditional way of doubling clock speeds by doubling number of cores, threads or other parallelizing mechanisms [4]. However, it is predicted that millions of cores of heterogeneous devices including CPUs and GPUs will be comprised by the Exascale computing system.

A. Exascale Computing Limitations and Challenges

As indicated by the innovation and programming approaches that are being utilized in existing Petascale registering framework, the power consumption is about 25 to 60 MW by utilizing 30 M number of centres. The interest of intensity utilization for Exascale registering framework will be more than 130 Megawatts [6]. United State Department of Energy characterized some essential limitations such as Power Consumption roughly 20-30 MW, Development Cost (D.C) up to 200 M US dollars, Delivery Time (DT) till 2020, and Cores about 100 Million [7]. However, development of targeted Exascale Supercomputer under the delimitation of these constraints is the tremendous challenge for vendors and development communities.

Leading to the massive powerful computing system, there are several challenges which are still the blockage for development toward emerging HPC systems. In [7], some primary Exascale computing challenges discussed are presented in Table I. For 21st century, these imperative difficulties are the basic way to create innovatory answers for Exascale figuring framework. Nonetheless, an emotional reformulation at both equipment and programming levels, programming models, vitality proficient strategies, investigating apparatuses and overhaul calculations are requested to accomplish the calculation in ExaFlops [8]. Since last few years the development process for Exascale computing system is being rapidly fast. Under these listed challenges, many new approaches have been proposed.

B. Software Technology Navigation

In current study, our contribution is related to challenges 1, 2, and 5 from Table I to improve the system performance through efficient and massive parallelism under minimum power consumption. From software perspectives, still it has not

been determined that at what level [9], the software framework is adoptable to achieve massive parallelism for Exascale computing systems. The recent energy efficient GPU technology introduced by NVIDIA outperforms the traditional processing on CPU cores [34, 35, 36]. Therefore, involving GPU accelerated computation in system, the hierarchy level of programming model navigation is shown in Fig. 1.

According to this navigational model, the anticipation is going to be that Tri-level model outperforms much better than traditional single or dual models [10]. It provides massive parallelism where energy efficient accelerated devices (GPGPU) collaborate with other models that deal with fine-grain and coarse-grain parallelism.

The rest of paper is organized as follows. Section II related work describes the existing state-of-art-approaches at Single, Dual, and Tri levels. Further Section III depicts a comprehensive overview of proposed EPC model, its features and components. Section IV, presents the experimental platform and applications used to evaluate EPC model. Last Section V concludes and explains the results in term of summary.

TABLE I. EXASCALE COMPUTING CHALLENGES

Challenges	***Description***
Power Consumption Management	Power consumed by the system and its management
Novel Architectures	New non-conventional architectures to support Exascale frameworks
Memory Technology	Memory management and storing systems to support massive storage.
Scalable System Software	Scalable and resilience system are needed to support sudden power fluctuation
Programming Systems	Novel programming techniques to support parallel programming libraries and frameworks.
Data Management	Efficient data management approaches are demanded.
Exascale Algorithms	New algorithms should be proposed to manage massive parallelism and advance programming.
Discovery and Design Algorithms	Discovery should be facilitated by mathematical models.
Resilience & Correctness	Faults and verification challenges should be addressed.
Scientific Productivity	Scientific productivity is necessary to through novel software tools.
Power Consumption Management	Power consumed by the system and its management

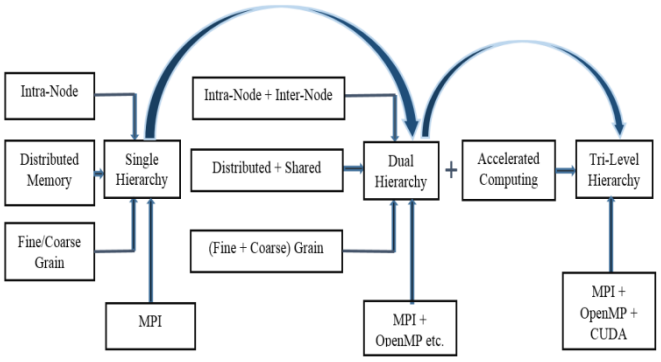

Fig. 1. Hierarchy Level of Programming Model Navigation.

II. RELATED WORK

Pushing toward HPC (High Performance Computing), equipment and programming rising advances have been examined toward Petascale registering framework in [11]. Prompting Petascale figuring framework, numerous equipment point of view methods where studied such as Conventional innovation, Preparing In-Memory structures (PIM), Digital superconductor advances, Computation Fluid Dynamics (CFD), Special-reason equipment, Web-based Petascale Computing, atomic nanotechnology and insightful planetary rocket and so on [12]. An information parallel programming language with respect to procedures for Petascale framework were proposed [13]. These models where capable to gain parallelism for both course grain and fine grain level using traditional homogenous system on multicore CPU devices [14].

In the end of recent decade, to bring scalability in system, technology trend was changed from traditional homogenous to heterogeneous cluster system where many-core devices were introduced such as General Purpose Graphics Processing Unit (GPGPU), Graphics Processing Unit (GPU) by NIVIDIA [15] and MIC (Many Integrated cores) by Intel [16]. These accelerated devices are based on Single Instruction Multiple Data (SIMD) from Flynn's classification. Beyond these accelerated devices, many parallel programming models have been proposed such as CUDA, OpenACC, and OpenCL. It has been anticipated these parallel programming models could be promising to achieve massive parallelism required for future Exascale computing system [17]. In any case, to use such incredible gadgets and models, a key component of the methodology is the co-structure of uses, designs and programming conditions at both equipment and programming level.

According to development to HPC Exascale computing system, China has a fast development towards HPC systems and consequently they introduced Tianhe II HPC system recently in 2014 [18]. Further they introduced the upgraded version named as Tianhe III [19]. Similarly, DEEP (Dynamical Exascale Entry Platform) by European Union in 2011 [20] started effort toward a new HPC Exascale computing system. SERT project funded by NAG took initiative to introduce first Exascale computing system in 2020 [21, 22]. In Japan, RIKEN [23] claimed to present first Exascale computing in start of 2020. Further, Indian Government also started Exascale computing development since 2018 and claimed to introduce in 2022 [24].

III. PRELIMINARIES

MPI has many different schemes that can be used to program a cluster system. Traditionally, two prevalent methods MPI blocking (synchronous) and non-blocking (asynchronous) are being used to distribute data over a cluster system [25, 26, 27]. In legacy systems, the whole processing was performed by CPU cores using MPI blocking method. Consequently, the processing over CPU cores was very costly with respect to energy consumption and processing efficiency. Therefore, new SIMD (single instruction multiple data) based energy efficient devices (GPUs, MIC) were introduced that contains thousands of cores on it. These cores compute data in parallel and consequently, reduce processing time and power consumption.

Due to parallel computation, data processing over GPU cores is very fast which required a rapid data input. In this way, MPI non-blocking is appropriate approach to fully utilize these powerful devices and achieve maximum performance. In current study, we discussed three fundamental MPI non-blocking schemes as follows:

A. (S1)- MPI Non-Blocking, no Overlapping Computation

In first strategy 1 (S1) MPI non-blocking and no overlapping implemented scheme, computation does not overlap during data processing [28]. This scheme performs just like a blocking mechanism where all resources are reserved until the whole processing is completed. One disadvantage of this scheme is that many resources are reserved event though they finished their assigned tasks. Although MPI communication is capable to overlap with CUDA, but we avoided from overlapping in this implementation. During exchanging data from multiple arrays, MPI scatter and gather data for one edge while memory copying operation is proceeding for other components.

B. (S2)- MPI Non-Blocking, Overlapping Computation

The second implemented strategy for data distribution was (S2) MPI non-blocking but overlapping computation where CUDA copying operation was overlapped with MPI communication. In this strategy, CUDA kernel was decomposed into three potions where top and bottom edges were done from the middle. In such way, kernel was started with the edges which are going to be computed, rather than start exchanging on entire domain. Following non-blocking MPI mechanism, first portion started copying operation from device to host. Immediately after completing copy operation to host, middle portion of the domain started computation. Similarly, last part of exchanging operation started as soon middle potion complete its computation. This implementation strategy can be more significant by improving the overlapping computation of middle portion.

C. (S3)-MPI Non-blocking,Overlapping & GPU Computation

The final implementation was MPI (S3) non-blocking with highest amount of overlapping which is anticipated the best performing strategy for large scale cluster system [29]. Using asynchronous method, CUDA streams were enabled and started computation from middle portion that cause to for massive overlapping, MPI communication and memory operations. The important thing in this strategy is that, a very small level of changes is needed inside the CUDA kernels to perform the computations. In order to optimize the GPU threads, a flag along with grid size and number of blocks is broadcasted over the kernels to indicate a specific portion for computation.

IV. Efficient Parallel Computing Model

We presented the proposed EPC model implemented in C++. Based on the predicted Exascale computing system, EPC model was categorized into three different computing environments including cluster system, compute node, and GPU computing. Each environment contained a separate layer of parallelism as presented in Fig. 2.

Programmer interacts with EPC model through the application written in C++. Before entering in parallelism zone, data is analyzed by the programmer himself statically to know that, which statement can be parallelized. Once data is analyzed and ready for parallel computation, it entered in parallel computing zones as described in following sections.

A. Inter-Node Computation Layer

The primary degree of parallelism of the model was accomplished between hub correspondences. In view of these parameters, developer break down and appropriate over associated framework hubs utilizing an institutionalized SIMD based Message Passing Interface (MPI) library [30]. MPI blocking (synchronous) and non-blocking (no concurrent) two pervasive components are being utilized to move and assemble information over the processors. Blocking systems is utilized when a solid synchronization is required because of reliance in information. For this situation, the assets are held utilizing some pre-characterized MPI holding up explanations until the handling is finished. In our parallel registering system, information is required just to convey over the processors that subsequently gives coarse-grain parallelism at this level, along these lines we chose "non-blocking, covering with GPU calculation" the third MPI non-blocking technique as talked about in past segment. In this procedure, when information is moved no concurrently over associated hubs, it entered in second degree of parallelism portrayed in following area.

B. Intra-Node Computation Layer

The proposed model provides the second level of parallelism at Intra-node computation. At this level, the distributed data through MPI processors is further communicated with CPU threads for parallel processing. At this stage, OpenMP pragmas are used that parallelize the blocks of code either fine grain or course grain computation. OpenMP threads use the system specified threads over CPU cores and complete the executions. According to new OpenMP version, we can use multiple OpenMP pragmas for multiple blocks within single block that is the reason to achieve fine grain parallelism in the code.

C. GPU Computation Layer Acceleration

The last level of parallelism in our proposed model is Intra-node computation. In this layer of computation, the whole processing is performed on accelerated GPU devices. In this strategy, firstly the data is transferred form CPU cores to GPU that further distributed over GPU Warps. According to the structure of GPU each warp contains 32 number of cores where the number of warps can be different from GPU structure to structure. Once the data is transferred over GPU cores, GPU kernel divide the tasks into multiple GPU warps and perform all the operations in parallel. To perform the GPU computation, we can utilize the different accelerated devices such as NVIDIA GPU, AMD GPU etc. for current study, to maintain the maximum support for C++, we selected NVIDIA GPU and implemented accordingly.

In the past, low overlapping between CPU and GPU caused the wastage of resources where GPU threads remain in idle state until the processing from other kernels is not accomplished. Usually, this inefficiency factor was found in

MPI non-blocking non-overlapping and non-blocking low-overlapping strategies that consequently waste resources utilization and decrease system efficiency.

Although MPI non-blocking was implemented in existing design as shown in Fig. 3(a) but waiting state for kernel and separate progress effected in decreasing efficiency. In each broadcasting, Isend() function/method has performed in three states including kernel initialization, kernel waiting and start sending data. During Isend() from these states, kernel stream was reserved. Once first kernel stream is complete, next one start processing. In such way, each stream waste resource utilization during waiting state. Conversely, in our proposed design, we organized these three states for every broadcasting Isend() in such way that kernels were overlapped and initialized immediately after one. Therefore, all kernel streams are now overlapping and can start processing as soon it receives data. A minor waiting state is ignorable because data sending process can be started as soon it complete its previous stage. Fig. 3(b) shows a clear benefit of proposed design that minimize delay in processing and increase efficiency through higher overlapping.

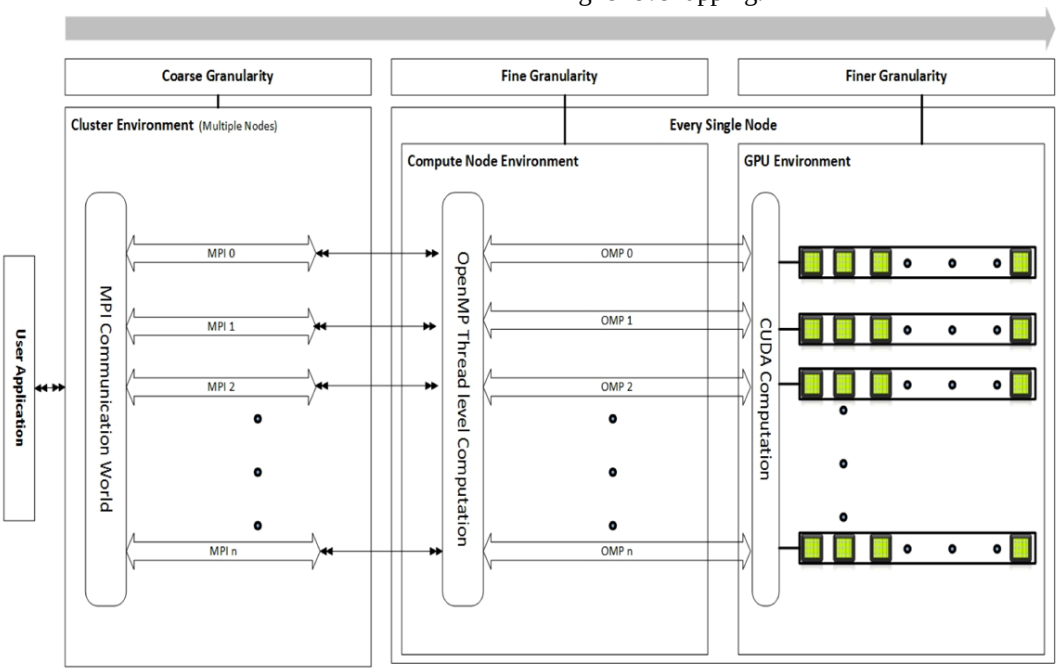

Fig. 2. EPC: A Hybrid Parallel Computational Model.

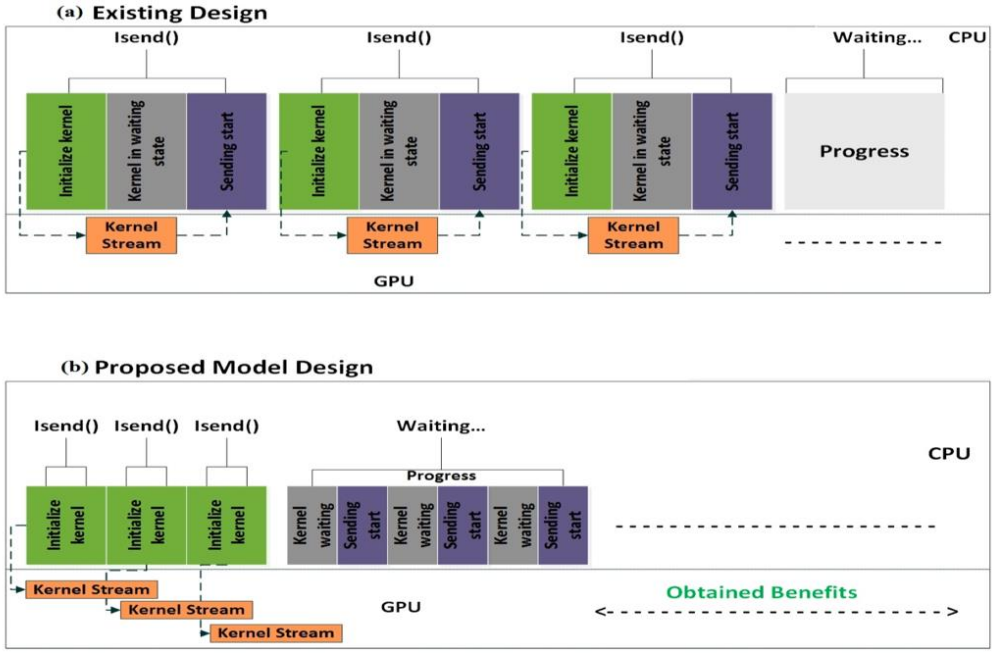

Fig. 3. Overlapping: Existing vs Proposed Design.

V. Experimental Setup

A. Platform

To perform the experiments, we have used the Aziz supercomputer the 360th positioned in 2015 top supercomputers placed in High Performance Computing Centre (HPCC), King Abdulaziz University. The Aziz supercomputer contains Xeon CPU processors along with GPU devices [31]. Aziz comprises of complete 11904 number of cores on it including both CPU and GPU cores. Regarding the Aziz memory, 96GB hubs and 256 GB individually configured in it where each hub consists of individual processor -2.4 GHz and 12 Cores- controlled CentOS 6.4 working framework. All the nodes are connected with infini-band medium to make the communication more efficient. With respect to overall efficiency, Aziz supercomputer is very powerful that is able to accomplish about 211 TFlops/s Linpack execution and about 228 TFlops/s overall [32].

B. Performance Measurment

The primary factor in High performance computing systems is Performance [33]. Conventionally, the performance of a computer system is calculated in number of Flops by attaining the peak performance and the number of flops against the targeted application execution as described in equation 1. If we consider that F_p are the flops at peak floating point and F_m are the number of flops against targeted application, therefore Fc can be determined as:

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

Using the Aziz peak performance Aziz, we have quantified the performance by executing targeted HPC applications at different datasets described in following sections.

C. Power Measurment

The second most important metric is the power consumption which is the primary challenge for current and emerging HPC systems. In current we have discussed the power consumption different perspectives. Conventionally the power consumption can be categorized in two ways including the power consumed at system level without running specific application and secondly the power consumption with some specific application computation [30]. Both categories have been specified the given equations 2,3 as follows.

$$\mathrm{P_{system}(w)} = \sum_{i=1}^{N} P_{GPU}^{i}(w^i) + P_{CPU}(\sum_{j}^{M}(w^i)) + P_{mb}(w) \quad (2)$$

In above equation, the power consumed by system is the sum of power consumed by number of configured GPUs, CPUs and mainboard.

$$\mathrm{P_{app}} = \sum_{i=1}^{N_{app}} P_{GPU}^{i}\left(w_{app}^{i}\right) + P_{CPU}\left(\sum_{j}^{M} w^i\right) + P_{mb}\left(w_{app}\right) \quad (3)$$

Similarly, the equation 3 describe the power consumed by system while running a specific application which is the sum of power consumed by number of configured GPUs, CPUs and mainboard.

VI. Experimental Results and Discussion

In this section we have presented all the determined results from the experiments where we implemented various numerical algorithms and discussed experimental results in this section. In first implementation, we run DMM application with multiple datasets through EPC model. A fundamental matrix multiplication method used in our implementation has been presented in below equation (6).

$$\begin{bmatrix} c_{11}c_{12}\ldots.c_{1p} \\ c_{21}c_{22}\ldots.c_{2p} \\ \vdots\ \ \vdots\ \ \ddots\ \ \vdots \\ c_{n1}c_{n2}\ldots.c_{np} \end{bmatrix} = \begin{bmatrix} a_{11}a_{12}\ldots.a_{1m} \\ a_{21}a_{22}\ldots.a_{2m} \\ \vdots\ \ \vdots\ \ \ddots\ \ \vdots \\ a_{n1}a_{n2}\ldots.a_{nm} \end{bmatrix} \begin{bmatrix} b_{11}b_{12}\ldots.b_{1p} \\ b_{21}b_{22}\ldots.b_{2p} \\ \vdots\ \ \vdots\ \ \ddots\ \ \vdots \\ b_{m1}b_{.2}\ldots.b_{mp} \end{bmatrix}$$

Sum of the given matrix can be defined as:

$$\mathrm{C_{ij}} = \sum_{k=1}^{m} \left(\mathrm{A_{ik}\ B_{kj}}\right) \quad (4)$$

Further to investigate the efficiency factor, we performed DMM implementation in suggested tri-level hybrid model with all MPI strategies (S1, S2 and S3) discussed in section (3).

By increasing matrix multiplication datasets, 'S3' increased the efficiency gradually and depicted the best performance compared to 'S1' and 'S2', and achieved 68% of peak performance in Tflops. Unlikely, 'S1' and 'S2' could attain the efficiency within range of 700-800 Gflops which was the initial throughput in 'S3' implementation. With large dataset computation, we observed that 'S1' declined the system efficiency which was eventually cause of over waiting during data distribution as shown in Fig. 4.

Along with performance, we quantified energy efficiency which is considered the primary metric for current and emerging HPC technologies. Likewise, the consequences in performance efficiency, 'S3' throughout increased energy efficiency at all datasets computation and accomplished 8.2 Gflops/w as shown in Fig. 5.

Further, we implemented 2-D Laplace application utilizing Jacobian iterative strategy where we run all models. By and large, the fractional differential conditions are ordered in a way like conic but here we have discussed only elliptic equation as $\mathrm{U_{xx}}\ (\mathrm{x,y}) + \mathrm{U_{yy}}\ (\mathrm{x,y})$ [22]. Be that as it may, the specific elliptic condition called "2-D Laplace condition" [23] utilized in current investigations is presented as follows in equation 7:

$$\frac{\partial^2 \mathrm{U}}{\partial \mathrm{x}^2}\ (\mathrm{x,y}) + \frac{\partial^2 \mathrm{U}}{\partial \mathrm{y}^2}\ (\mathrm{x,y}) = 0 \quad (5)$$

We implemented 2-D Laplace Jacobian iterative method in EPC proposed model using all strategies. The mesh size was increased dramatically in the range of 1000-8000. Fig. 6 and 7 demonstrate the consequences of 2-D Laplace method against both metrics (performance and energy efficiency). The similar efficiency ratio of 'S1' in matrix multiplication was found in 2-D Laplace solver method in range of 390-700 Gflops/sec. Although, efficiency increased gradually in 'S1' but we can rely on it due to poor throughput.

We also evaluated energy efficiency in 2D Laplace equation method (see Fig. 6). 'S3' provided the best energy

efficiency as compared to other strategies. We noticed that 'S2' was also prominent and achieved energy efficiency up to 8.3 Gflops/w but 'S1' wasted a lot of energy throughout the computation and achieved 7.4 Gflops/w at maximum mesh size.

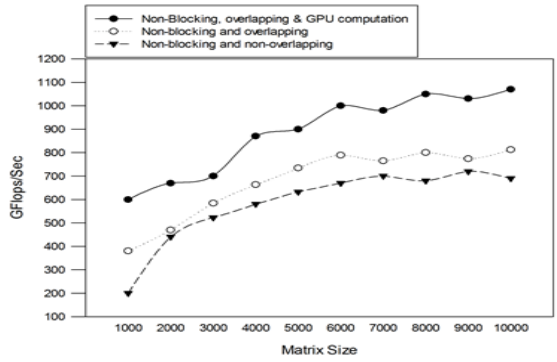

Fig. 4. Performance efficiency in all Strategies during MM Computation.

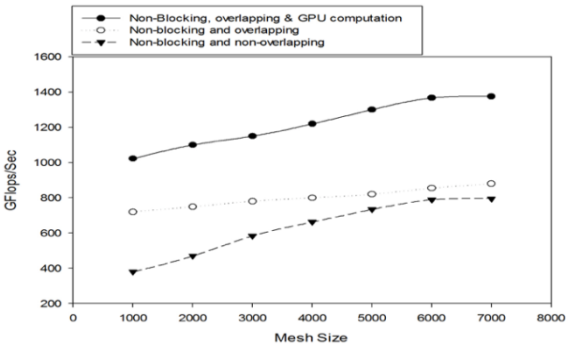

Fig. 5. Performance Efficiency in All Strategies during 2-D Laplace Solver.

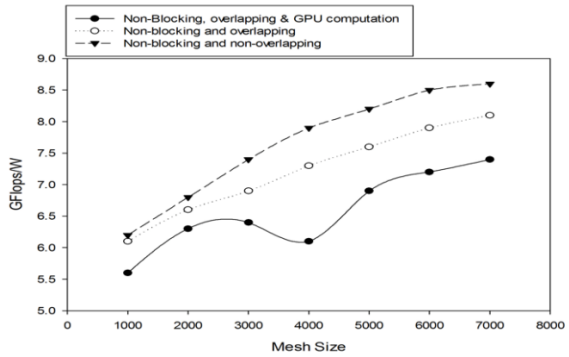

Fig. 6. Energy Efficiency 2-D Laplace.

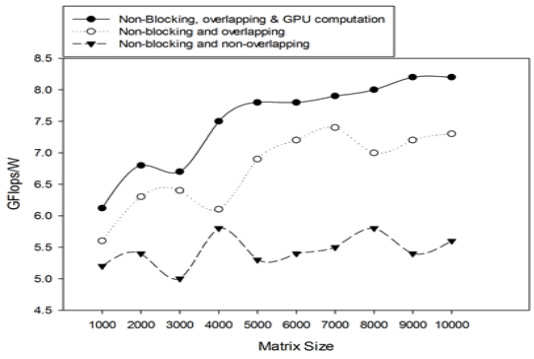

Fig. 7. Energy Efficiency in MM.

VII. Conclusion

The emerging HPC models are relied upon to grow first Exaflops PC to contain a huge number of heterogeneous process hubs connected by complex systems till next half decade. This Exascale processing framework will be skilled to figure one Exaflops estimation for each subsequent which is thousands-crease increment in current Petascale framework. In current study, we have discussed the extensive constraints for Exascale systems and perspective challenges for current technologies. In this research, the proposed model is a novel secure and efficient parallel programming approach which is tri-level hybrid of MPI, OpenMP and CUDA. In MPI, we implemented different strategies (S1, S2 and S3) under non-blocking mechanism. Further to evaluate the efficiency factors, the proposed model was implemented with all these strategies in two benchmarking HPC applications including DMM and two dimensional Laplace equation. Consequently, in both applications, we found that 'S3' strategy (non-blocking, overlapping and GPU computation) performed the best in providing performance efficiency and energy efficiency comparatively to (S1and S2). Therefore, hybrid of proposed model with 'S3' MPI strategy can be consider as promising model to achieve required performance and energy efficiency for Exascale systems. By future perspectives, this model is required to be executed a large cluster system that can meet the minimum requirement for Exascale system configurations.

References

[1] Liao, G., Guo, D., Bhuyan, L. and King, S.R., 2008, November. Software techniques to improve virtualized I/O performance on multi-core systems. In Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (pp. 161-170). ACM.

[2] Bland, B., 2012, November. Titan-early experience with the titan system at oak ridge national laboratory. In High Performance Computing, Networking, , 2012: (pp. 2189-2211). IEEE.

[3] Liao, X., Xiao, L., Yang, C. and Lu, Y., 2014. MilkyWay-2 supercomputer: system and application. Frontiers of Computer Science, 8(3), pp.345-356.

[4] Shalf, J., Dosanjh, S. and Morrison, J., 2010, June. Exascale computing technology challenges. In International Conference on High Performance Computing for Computational Science (pp. 1-25). Springer, Berlin, Heidelberg.

[5] Perarnau, S., Gupta, R. and Beckman, P., 2015. Argo: An exascale operating system and runtime. In High Performance Computing, Networking, Storage and Analysis, SC15.

[6] Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J. and Karp, S., 2008. Exascale computing study: Technology challenges in achieving exascale systems. (DARPA IPTO), Tech. Rep, 15.

[7] Reed, D., et. al., 2015. (ASCAC) Report: Exascale Computing Initiative Review. USDOE Office of Science.

[8] Ang, J.A., Barrett, R.F., Benner, R.E., Burke, D., Chan, C., Cook, J., Donofrio, D., Hammond, S.D. Hemmert, K.S. Kelly, S.M. and Le, H., 2014, November. In Hardware-Software Co-Design for High Performance Computing (Co-HPC).

[9] Ceruzzi, P.E., 2005. Moore's law and technological determinism: reflections on the history of technology. Technology and Culture, 46(3), pp.584-593.

[10] Barker, Kevin J., et al. "Entering the petaflop era: the architecture and performance of Roadrunner." Proceedings of the 2008 conference on Supercomputing. IEEE Press, 2008.

[11] Sameh, A.H., 2015. Parallel Sparse Linear System and Eigenvalue Problem Solvers: From Multicore to Petascale Computing. PURDUE UNIV LAFAYETTE IN.

[12] Dongarra, J.J. and Walker, D.W., 2001. The quest for petascale computing. Computing in Science & Engineering, 3(3), pp.32-39.

[13] Zima, H.P., 2007. From FORTRAN 77 to locality-aware high productivity languages for peta-scale computing. Scientific Programming, 15(1), pp.45-65.

[14] Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., & Chapman, B. (2011). High performance computing using MPI and OpenMP on multi-core parallel systems. 37(9), 562-575.

[15] Roberge, V., Tarbouchi, M. and Okou, F.A., 2017. Distribution system optimization on graphics processing unit. IEEE Transactions on Smart Grid, 8(4), pp.1689-1699.

[16] Satish, Nadathur, et al. "Fast sort on cpus, gpus and intel architectures." Intel Labs (2010): 77-80.

[17] Bastem, B., Unat, D., Zhang, W., Almgren, A. and Shalf, J., 2017, August. Overlapping Data Transfers with Computation on GPU with Tiles. In Parallel Processing (ICPP), 2017 46th International Conference on (pp. 171-180). IEEE.

[18] Collis, S.S., 2014. Computers are changing and so must our codes.. (No. SAND2014-20299PE). Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States).

[19] Lendino, J. (2016, June 29). Meet the new world's fastest supercomputer: China's TaihuLight. Retrieved from https://www.extremetech.com/extreme/230458-meet-the-new-worlds-fastest-supercomputer-chinas-taihulight

[20] Kunkel, J. M., Balaji, P., & Dongarra, J. (2016). High Performance Computing: 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016, Proceedings. Basingstoke, England: Springer.

[21] STFC,https://www.scd.stfc.ac.uk/Pages/SCD_Science_Highlights_2016.pdf, 2016.

[22] NCF, European Exascale Software Initiative.

[23] D'Hollander, E., Dongarra, J., & Foster, I. (2013). Transition of HPC Towards Exascale Computing. Amsterdam, Netherlands: IOS Press.

[24] India to Launch $730M National Supercomputing Mission. (2014, October 7). Retrieved from https://www.hpcwire.com/2014/10/06/india-launch-730m-national-supercomputing-mission/

[25] Ashraf, M. U., Fouz, F., & Alboraei Eassa, F. (2016). Empirical Analysis of HPC Using Different Programming Models. International Journal of Modern Education and Computer Science, 8(6), 27-34.

[26] Hassani, A., Skjellum, A. and Brightwell, R., 2014, June. Design and evaluation of FA-MPI, a transactional resilience scheme for non-blocking MPI. In Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on (pp. 750-755). IEEE.

[27] Ahmed, H., Skjellumh, A., Bangalore, P. and Pirkelbauer, P., 2017, September. Transforming blocking MPI collectives to Non-blocking and persistent operations. In Proceedings of the 24th European MPI Users' Group Meeting (p. 3). ACM.

[28] Morgan, B., Holmes, D.J., Skjellum, A., Bangalore, P. and Sridharan, S., 2017, September. Planning for performance: persistent collective operations for MPI. In Proceedings of the 24th European MPI Users' Group Meeting (p. 4). ACM.

[29] Hoefler, T., Squyres, J., Bosilca, G., Fagg, G., Lumsdaine, A. and Rehm, W., 2006. Non-blocking collective operations for MPI-2. Open Systems Lab, Indiana University, Tech. Rep, 8.

[30] Hahnfeld, J., Cramer, T., Klemm, M., Terboven, C. and Müller, M.S., 2017, September. A Pattern for Overlapping Communication and Computation with OpenMP^* Target Directives. In International Workshop on OpenMP (pp. 325-337). Springer, Cham.

[31] Fujitsu to Provide High-Performance Computing and Services Solution to King Abdulaziz University. (n.d.). Retrieved from http://www.fujitsu.com/global/about/resources/news/press-releases/2014/0922-01.html.

[32] King Abdulaziz University, www.kau.edu.sa

[33] L. Ahmad, A. Majidi, and A. Baniasadi. "IPMACC: Open source OpenACC to CUDA/OpenCL translator." (2014).

[34] G. David, and N. S. Trudinger. Elliptic partial differential equations of second order. springer, 2015.

[35] B. Richard L., and J. D. Faires. "Numerical Analysis. Brooks/Cole, Thomson Learning." Inc 206 (2001): 772.

[36] Lilja, D. J. (2005). Measuring Computer Performance: A Practitioner's Guide. Cambridge, England: Cambridge University Press.