Adaptive Computing Library for Quantum Monte Carlo Simulations

Zeki Bozkus, Ahmad Anbar, and Tarek El-Ghazawi

***Abstract*—Quantum Monte Carlo (QMC) methods are used in many scientific computer simulation as their core kernels. The implementation of QMC for distributed NUMA clusters may have load balancing issues at petascale level because of its random nature. We are studying on a simulation for inhomogeneous ultra-cold atoms on optical lattice, for which we developed a QMC algorithm with hybrid MPI+OpenMP programming model. This hybrid model uses the nested parallelism such that the outer loops are parallelized by MPI, while the inner loop relies on OpenMP parallelism. In this work, we presented an adaptive computing approach which learns the system work load dynamically by using our Adaptive Computing Library at run-time and then creates sufficient amount of OpenMP threads based on the availability of the system resources during the execution. The implementation shows that our adaptive approach can get very good load balancing without unnecessary overheads and can significantly provide performance increases up to 20% increases in comparison to MPI-only implementation on a XE6m Cray super computer.**

***Index Terms*—Hybrid parallel programming, load balancing, QMC simulation.**

I. Introduction

Many scientific simulations use quantum Monte Carlo method (QMC) at their most time consuming kernels. QMC method provides an accurate description of many-body physics which can be applied to problems relevant to chemistry, biology, physics, material science and even drug design at molecular level. In our collaborative and interdisciplinary work with physics department of George Town University, we want to build scalable and efficient an optical lattice simulator with ultra cold atoms by using inhomogeneous dynamical-mean field theory (IDMFT) in which again most time consuming portions of the simulation are QMC methods [1], [2].

These calculations are computationally intensive and require very large high performance computing systems to able to study realistic simulations. Thus, the simulator is originally written with FORTRAN+MPI programming model to run distributed computer cluster with manager-worker paradigm. However, we observed that load balancing problem occurred since each MPI process performing QMC which is random by nature. We implemented the hybrid MPI+OpenMP programming to overcome load balancing issues. However, this cause memory stalling problem on Cray XE6m, where each compute node has non-uniform memory access (NUMA) memory design. When the system is fully busy with MPI processes [3] by launching new OpenMP threads [4] which stresses the memory bandwidth, cause starvation to some threads. To overcome this difficulty, we design Adaptive Computing Library (ACL) which can be used to improve the hybrid MPI+OpenMP or MPI+thread programming model.

The cluster system, which consists of shared memory nodes with several multi-core CPUs connected to a high speed network to comprise a distributed memory system, is the most widely available hardware architecture for the high-performance computing community. On these systems, the hybrid parallel programming model with MPI internodes communication with combination of a shared memory programming model to manage intranode parallelism has become a common approach to parallel programming. ACL provides a library to help hybrid programmers to improve the performance by managing the process vs. thread balance. User of ACL will not create more threads than necessary so that their NUMA or even SMP cluster will not cause memory congestions with unwanted parallelism.

In this paper, we present our work to improve the performance of a physics simulation (QMC application) for ultra-cold atoms in optical lattice by using a hybrid programming paradigm at very large scale high performance computing cluster consists of NUMA nodes. We developed ACL library to improve the hybrid parallel programming further at this NUMA platform. We demonstrate the API of ACL and how to use it for hybrid programming environment. We compared the performance evaluation of MPI-only, MPI+OpenMP and our adaptive hybrid model of the inhomogeneous ultra-cold atoms simulation on a Cray XE6m supercomputer.

Section II of the paper presents an overview description of theory used in the simulation and briefly describing the main QMC kernel, and Section III provides hardware view of the NUMA platform. Section IV presents MPI parallelization of the simulation, while Section V explains a hybrid implementation of the simulation. Section VI presents our adaptive hybrid approach, followed with a description of the interface and implementation of our library. An evaluation of three models in terms of load balancing and performance is in Section VII. Related work and the conclusion are presented in Section VIII and Section VIIII, respectively.

II. Theory: Inhomogeneous Quantum Mechanical Systems

The inhomogeneous quantum mechanics is one of the most important fields used to solve difficult problems in physics.

Manuscript received September 25, 2013; revised January 3, 2014.

The authors are with the Department of Electrical and Computer Engineering and the George Washington University (e-mail: {zbozkus, anbar, tarek}@gwu.edu).

This project is to build an optical lattice simulator which models inhomogeneous mixture of Boson and Fermion atoms moving on the optical lattice at ultra-cold temperature. The model is restricted to the case where only the Fermion atoms can move from site to site and the Bosons are stationary but have an interaction with the Fermions if they exist on the same site. Furthermore, we look at only nearest neighbour hopping for the Fermions. The system, because it is at the atomic scale, is governed by quantum mechanics; more precisely it is a many body quantum mechanical problem subject to the differential equation (i.e. the equation of motion) [1], [2]:

$$(-\delta_{ik}\frac{\partial}{\partial t} - H_{ik})G_{kj} = \delta_{ij}\delta(t) \quad (1)$$

where the indices *i,j* denote discrete indices that label space points on the optical lattice. *t* is the time variable and the matrix H_{ij} is a complicated function of space time points given by the following Eq. 2:

$$\begin{aligned} H_{ij} &= -\sum_{ij} t_{ij} c_i^{\dagger} c_j + U_{bf} \sum_i c_i^{\dagger} c_i b_i^{\dagger} b_i \\ &+ \frac{1}{2} U_{bb} \sum_i b_i^{\dagger} b_i (b_i^{\dagger} b_i - 1) \\ &+ \sum_i (V_i - \mu) c_i^{\dagger} c_i + \sum_i (V_i^b + E_b) b_i^{\dagger} b_i \end{aligned} \quad (2)$$

Each term in the above equation exhibits some type of interaction between the two types of atoms. The important thing to note is that only the *first* term has terms that connect two lattice points *i* and *j*. Because we restrict ourselves to only nearest neighbor interaction we see that the matrix H_{ij} is sparse. This is the important observation of the above equation the other terms simply add to the diagonal components of *H*.

Returning back to Eq. l our ultimate goal is to compute the matrix G_{ij}. It turns out that the physical quantities of interest depend only on the diagonal elements of G_{ij}, i.e. G_{ii}. The system uses QMC solver to calculate for this quantity for each G_{ii}. The QMC part is the most expensive part of the simulation.

Algorithm 1: The pseudo code of QMC method.
Generate initial random configurations
DO *g* = 1 to *N*_generations
DO *w* = 1 to *N*_walkers
DO *p* = 1 to *N*_particles
Move particle new random position
Compute Wave and Green's function ratios
Accept/reject move with Metropolis
Calculate new energy of walker
Calculate s = base on walker energy
if *s* == 0 remove walker
if *s* == 1 do nothing
if *s* > 1 create *s*—1 additional walkers
END DO
END DO
END DO

The QMC method is summed up in the Algorithm 1. The algorithm is simply comprised of three main loops. The first loop is going on the number of Monte Carlo cycle and the second and third loops go through walkers (parallel instances of systems) and particles respectively. Finally, we also have the branching part where some walkers are removed; some are untouched, while others breed new walkers by spawning new copies. Since the randomness natural of QMC algorithm, we can only know at run-time how expensive each QMC becomes. This becomes the source of load imbalance at the simulation.

III. NUMA Cluster Architecture

Any realistic scientific simulations which employ QMC methods requires to a usage of distributed memory high performance system to enable calculation to complete in a reasonable amount of time with a good accuracy. These distributed systems consists of many shared memory multi-core nodes connected to each other with a high speed interconnect. However, the share memory multi-core nodes have two different memory architectures. The first architecture is called the Symmetric Multiprocessor (SMP) architecture which has many identical processors and all of processors have equal access times to all memory regions of the system. The second architecture is called Non-Uniform Memory Access (NUMA) architecture [5] as shown Fig. 1.

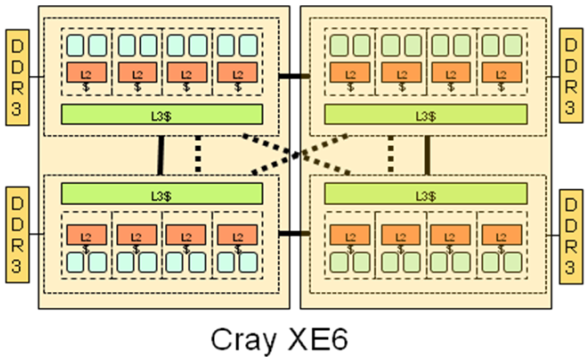

Fig. 1. NUMA architecture at Cray XE6 nodes.

The cores of NUMA architectures can access local memory faster than at non-local remote memory but still can access all memory region supported by the hardware. The NUMA architecture can scale better than SMP architecture after a number of processors. Thus, modern high performance hardware such as Cray XE6 which is used in this study has 4 NUMA nodes where each has 6 cores. That makes the total number of cores in the node 24. We believe that our ACL library will provide a support for NUMA aware programming to reduce memory contentions.

IV. Implementation of MPI Parallelization

The original parallel version of the simulation code was developed using FORTRAN+MPI programming with master-slave model where the master processor dispatches work to the slave processors, who perform the work and return the results back to the master. The master-slave model fairly balances workloads among the slaves. However, in this

simulation, every slave performs many QMC algorithms which create load imbalances during the later stages of the simulation. To overcome this issue, we considered blending coarse grain MPI parallelism with fine grain OpenMP parallelism at the shared memory nodes of the cluster.

The simulation pays importance to eliminate many-to-one communications which is very common at master-slave model that can lead to a bottleneck affecting scaling to large number of nodes. The code chooses many-to-many communications that slave nodes communicate directly amongst themselves to accumulate a final answer.

V. Implementation of Hybrid MPI+OpenMP

Hybrid MPI+OpenMP model mainly used at nested parallelism applications [6], [7]. MPI portion of the hybrid can parallelize the outside loop by distributing the iteration space among the nodes of the cluster. On the other hand, OpenMP portion of the hybrid model enable shared memory parallelism among cores inside a compute node. The requirement of two different programming models increases the programming complexity at this parallel programming. However, this model fits nicely with today high performance computing architecture. Programmer uses MPI to perform domain decomposition of data explicitly. However, OpenMP can handle shared tables and data structures among threads by locking the critical sections.

The hybrid programming can be designed with different structures. The most common one is to share the number of processors among the MPI process and OpenMP threads. We did not choose this approach because our simulation requires P^3 configuration for the number of processors. This limits the experiment configurations. In this implementation, we choose that every MPI process will have fixed 4 threads each as shown Algorithm 2 where show QMC methods of each MPI slaves parallelized by OpenMP.

Algorithm 2: The Hybridized pseudo QMC method.

```
Generate initial random configurations

DO g = 1 to N_generations
!$omp parallel do private(....) default(shared)
 DO w = 1 to N_walkers
 DO p = 1 to N_particles
Move particle new random
position
Compute Wave and Green's function ratios
Accept/reject move with Metropolis
Calculate new energy of walker
Calculate s = base on walker energy
!$omp critical
if s == 0 remove walker
if s == 1 do nothing
if s > 1 create s—1 additional walkers
!$omp critical end
END DO
END DO
!$omp parallel do end
END DO
...
Call MPI_Barrier()
```

In addition, MPI supports two different thread safety levels for hybrid programming. MPI_THREAD_MULTIPLE allows multiple threads can call MPI library. This feature can enable to overlapping communication with computation optimization. The other alternative is MPI_THREAD_FUNNELED support in which an MPI application may be multithreaded but only one thread at a time can call MPI library. We choose this support level since our MPI calls are outside of OpenMP parallel region.

VI. Adaptive Hybrid Method

In our simulation, MPI-only approach performs very well at the beginning stage of the simulation by keeping the system busy. The problem of load imbalance appears at the later stage of the simulation. We designed a new library where it will help the programmer to decide the number of OpenMP threads adaptively. The system will run MPI-only style until some of MPI processes start waiting for the synchronization barriers or blocking communications. Our library can tell the system to use more OpenMP threads for the inner node parallelism instead of MPI processes as time progress. Our first regular hybrid implementation suffered from memory congestion. However, the adaptive approach can solve this problem with the help of the programmer.

TABLE I: Adaptive Computing Library

Interface	Description
void **ACL_Init**(Boolean flag)	*ACL_NUMA_NODE/ACL_COMPUTE _NODE*
int **ACL_Acquire**(int myrank)	*Return available cores*
boolean **ACL_Check**(int myrank)	*Quick check*
void **ACL_Release**(int myrank)	*Release cores*
void **ACL_Sleep**(int myrank)	*Inform the blocking*
void **ACL_Finalize**(int myrank)	*Free all ACL resources*

Table I gives the number of routines in our Adaptive Computing Library (ACL) which is implemented with MPI one-sided communication. The ACL library supports two policies. It may give the number of cores availability at the NUMA nodes or at the larger COMPUTE nodes. Each MPI process will keep information about the avalibility of the nodes where they affine. The ACL_Init will initialize library and allocate internal data structures with ACL_NUMA_NODE or ACL_COMPUTE_NODE flag. ACL_Acquire routine will grant the number of cores the caller process where the appropriate threads will be created for the shared memory OpenMP. ACL_Check will return Boolean value whether the system should be MASTERONLY execution with quick system check without any ACL overhead such as internal locks. ACL_Release will return cores to the system which may be used other process in the same node. ACL_Final will free all internal data structures and all the recourses such as locks back to the system.

Algorithm 3 is giving the example of ACL usage for the hybrid QMC method. In this code, omp_set_num_threads() of OpenMP library [4] can specify the number of threads. If the ACL_Acquire return 1, the subsequent OpenMP parallel region will be executed master only without any parallelism. Otherwise, it will be executed with a varying number of

threads depending of the workload of the system.

Algorithm 3: The Hybridized pseudo code with ACL usage in QMC kernel.

```
call AC_Init(ACL_NUMA_NODE)
....

DO g = 1 to N_generations
ncores = ACL_Acquire(myrank)
call omp_set_num_thrs(ncores)
!$omp parallel do private(....) default(shared)
DO w = 1 to N_walkers
DO p = 1 to N_particles
Move particle new random
position
Compute Wave and Green's function ratios
Accept/reject move with Metropolis
Calculate new energy of walker
Calculate s = base on walker energy
!$omp critical
if s == 0 remove walker
if s == 1 do nothing
if s > 1 create s—1 additional walkers
!$omp critical end
END DO
END DO
!$omp parallel do end
call ACL_Release(myrank)
END DO
...
ACL_Sleep(myrank)
call MPI_Barrier()
...
call ACL_Finalize()
```

ACL library works for only intra node cores. It will not give core availability information for remote nodes. The reason is that there are many internal data structures and large tables for QMC which cannot redistributed for this simulation. If the redistribution capability is added to the simulation, that will increase MPI communication heavily. In addition, it requires more programming effort to restructure source code of the whole simulation. OpenMP can allow incremental parallelization where you can improve as you needed. However, MPI parallelization requires data domain decomposition which cannot be done incrementally.

Unfortunately, many FORTRAN programmers use too many COMMON blocks with SAVE keyword where they make many variables global even though some of these variables can be thread private variables. This is the problem of FORTRAN legacy code. When we make some of these false global variables back to local, it helps performance tremendously to the NUMA memory architecture.

VII. Performance Evaluation

This section will illustrate the impact of our proposed hybrid MPI+OpenMP approaches through several experiments by running three different implementations of the QMC algorithm in the simulation on a 66-nodes Cray XE6 cluster located at George Washington University with total 1584 cores. This SMP cluster consists of 2.2 GHz AMD Opteron CPUs with a total of 6-Cores with 4 NUMA nodes for total 24 processing cores per node running Linux 2.6.2 connected with a state-of-the art Gemini 2D torus interconnect. All the codes are compiled with Cray FORTRAN compiler. We only tested our approach at Cray XE6 platform. This is a limitation in this work.

We show the performance for a baseline MPI-only version and compare with the regular hybrid MPI+OpenMP version and our adaptive version with ACL_NUMA_NODE enabled. The experiment results are obtained with an average of 5 runs. We report the timing of total simulation not only QMC timing. The simulation size is the optical lattice size where the lattice has size N^3 and we choose N to be odd, so the lattice runs from $-(N\text{-}1)/2 < x, y, z < (N\text{-}1)/2$ coordinate space.

The first experiment objective is to find the load imbalance percentage. We used CrayPat performance analyzer tool which gives minimum time and maximum time consumed to execute the code at total number of cores. Table II demonstrated that MPI-only implementation has the highest load imbalance percentage. Our adaptive solution improved the load imbalance around 7% points. The regular hybrid looks like improved imbalance percentage over MPI-only but this did not helped the runtime because of memory congestion created by more threads than available cores.

TABLE II: Adaptive Computing Library Comparing Load Imbalance Percentage between Maximum and Minimum Time Differences Measured by Craypat Profiler

Data Size	19^3	23^3	25^3	27^3
MPI Imbl.	34.00	37.00	38.00	38.00
Hybrid Imbl.	31.00	33.00	32.00	33.00
Adaptive Imbl.	28.00	30.00	31.00	30.00

The second experiment was designed to reveal how well the proposed hybrids perform compared to plain MPI version. Fig. 2 shows the total execution time on the vertical axis, and the horizontal axis denotes the problem size. The CPU times of regular hybrid are about 10% percentage worse than MPI-only implementation for every data sizes in the experiment. Adaptive solution consistently obtains better performance than MPI-only with about 20% performance gain.

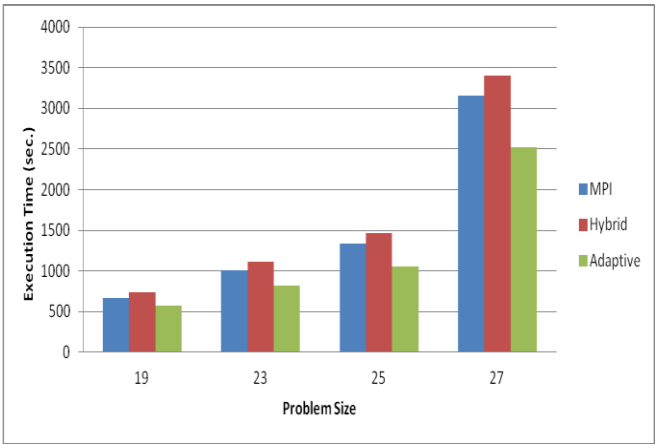

Fig. 2. Execution time of the simulation on 128+1 CPUs with varying problem sizes.

VIII. Related Work

A cluster system with multiple SMP or NUMA nodes is

the most commonly available high performance computing hardware. MPI programming can be used both within a node and among cluster's nodes. However, the other programming model for this platform is a hybrid programming model, in which a parallel program is written using a thread programming library such as Portable Operating System Interface (POSIX) threads within a node and MPI programming among nodes simultaneously. The MPI-2 Standard has clearly defined the interaction between MPI and user-created threads in an MPI program. However, Gropp and Thakur [8] have pointed out the issues involved in developing an efficient thread-safe MPI implementation without sacrificing too much performance.

The other hybrid programming with MPI as outer level parallelism and OpenMP as inner level parallelism has been extensively studied for an SMP cluster system [6], [7]. The shared address space within each SMP node is suitable for OpenMP parallelization and MPI can be employed within and across the nodes of a cluster. The MPI/OpenMP hybrid programming model is easy to apply via automatic parallelization of the compilers with some directives for loop level parallelism. Rabenseifner et al. have shown the relation between the MPI/OpenMP hybrid programming model and hardware architecture.

A manager-worker-based parallelization algorithm for Quantum Monte Carlo (QMC-MW) is presented at [9] on heterogeneous parallel computers and they compared with the pure iterative parallelization algorithm (QMC-PI).

A new hybrid model is developed by using MPI+MPI by [10]. In this model, they used the new MPI extension at MPI 3.0 for shared memory programming at the node level parallelism [11]. They demonstrated an average performance improvement %40 at a QMC implementation with MPI+MPI.

The developers of popular QMCpack [12] and QWalk [13] have invested time and effort into hybridizing MPI-only code with shared memory libraries, such as OpenMP to get node-level parallelism for SMP nodes. However, our QMC implementation is more NUMA-aware implementation by using ACL library.

IX. Conclusion

Through this study we developed a hybrid parallel programming model that combined the strength of MPI's coarse grain parallelism with the strength of OpenMP's fine grain approach to overcome load imbalance problem occurred MPI-only implementation at a Physics simulation for inhomogeneous ultra-cold atoms on an optical lattice problem. Since the simulation spends 90% of time on QMC algorithm, we used three different implementation codes of QMC's algorithm employing three different parallelization paradigms: MPI-only, a hybrid MPI+OpenMP and the adaptive hybrid which is optimized with our Adaptive Computing Library (ACL) on a cluster of NUMA nodes. Each implementation employed the advanced features of the underlining programming model to achieve the best possible performance gains.

We evaluated the scalability of the proposed hybrid MPI+OpenMP model and the Adaptive hybrid model on the QMC code in the simulation by comparing the baseline MPI-only implementations on up to 128 cores. We found that hybrid MPI+OpenMP implementation cannot provide improvement over MPI-only implementation. Furthermore, it even degraded the performance 10% ranges because of memory congestions created by more threads than available cores. On the other hand, our adaptive hybrid solution with our ACL library demonstrated a 20% performance gain on some configurations in comparison to MPI-only implementation by improving load balancing problem.

Acknowledgment

We would like to thank Prof. Jim Freericks at George Town University Physics department for developing the theory of the simulation as well as providing us MPI-only implementation of the simulation. We thank Michael Lujan at George Washington University Physics department to understand the simulation mathematics. This work was supported in part by the National Science Foundation under grant number OCI-0904887.

References

[1] J. K. Freericks, "Transport in multilayered nanostructures: The dynamical mean-field theory-approach," Imperial College Press, London, 2006.

[2] J. K. Freericks, "Modeling mixtures of different mass ultracold atoms in optical lattices: An illustration of high efficiency and linear scaling on the cray xt4 via a capability applications project at erdc," in *Proc. the HPCMP Users Group Conference*, Seattle, WA, IEEE Computer Society, Los Alamitos, CA, July 14–17, 2008, pp. 424-430.

[3] M. Forum, "MPI: A message-passing interface standard," University of Tennessee Knoxville, TN, USA UT-CS-94-230, 1994.

[4] OpenMP Architecture Review Board. (May 2008). OpenMP Application Program Interface Version 3.0. [Online]. Available: http://www.openmp.org/mp-documents/spec30.pdf

[5] T. H. Dunigan, J. S. Vetter, J. B. W. Iii, and P. H. M. Worley, "Performance evaluation of the Cray X1 distributed shared-memory architecture," *IEE Micro*., pp. 30-40, 2005.

[6] L. Smith and M. Bull, "Development of mixed mode MPI / OpenMP applications," *Sci. Program.*, vol. 9, pp. 83-98, 2001.

[7] R. Rabenseifner *et al.*, "Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes," presented at the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, 2009.

[8] W. Gropp and R. Thakur, "Issues in developing a thread-safe MPI implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. vol. 4192, B. Mohr *et al.*, Eds. pp. 12-21, 2006.

[9] M. T. Feldmann, J. C. Cummings, D. R. Kent, R. P. Muller, and W. A. Goddard, "Manager–Worker-Based model for the parallelization of quantum Monte Carlo on heterogeneous and homogeneous networks," *Journal of Computational Chemistry*, vol. 29, issue 1, pp. 8–16, 15 January 2008.

[10] T. Hoefler, J. Dinan, D. Buntinas, P. Balaji, B. Barrett, R. Brightwell, W. Gropp, V. Kale, and R. Thakur, "MPI+MPI: A new, hybrid approach to parallel programming with MPI plus shared memory computing," *Computing*, 2013.

[11] V. Tipparaju *et al.*, "Investigating high performance RMA interfaces for the MPI-3 standard," presented at the 2009 International Conference on Parallel Processing, 2009.

[12] K. P. Esler, J. Kim, D. M. Ceperley *et al.*, "Quantum monte carlo algorithms for electronic structure at the petascale; the endstation project, "*Journal of Physics: Conference Series*, vol. 125, no. 1, 2008, 012057.

[13] L. K. Wagner, M. Bajdich, and L. Mitas, "Qwalk: A quantum monte carlo program for electronic structure," *Journal of Computational Physics*, vol. 228, no. 9, pp. 3390 - 3404, 2009.

Zeki Bozkus received the M.S. and the Ph.D. degrees in computer science from Syracuse University, NY, USA, in 1990 and 1995, respectively. He worked as a senior compiler engineer at the Portland Group, Inc. for six years. He worked as a senior software engineer at Mentor Graphics for the parallelization of Calibre product line for 11 years. He is now an assistant professor at the Computer Engineering Department of Kadir Has University since 2008. His primary research interests are in the parallel programming algorithms, parallel programming languages, and compilers. He is in sabbatical at George Washington University as a visiting professor.

Ahmad Anbar graduated from the Faculty of Computer and Information Sciences (FCIS), Ain Shams University, Egypt in year 2000. He worked in teaching in the university since then. He also worked as an information analyst in Electronic Data Systems (EDS), Cairo branch for three years. Ahmad got his masters degree from Ain Shams University in 2006. His master was about resources management in Grid environments. Since Fall 2008, Ahmad started his Ph.D. in The George Washington University. He joined the High Performance Computing Lab (HPCL) as a research assistant. His main research in HPCL is targeting the support of UPC on many-core architectures.

Tarek El-Ghazawi is a professor in the Department of Electrical and Computer Engineering at The George Washington University, where he leads the university-wide Strategic Program in High-Performance Computing. He is the founding director of The GW Institute for Massively Parallel Applications and Computing Technologies (IMPACT) and a founding co-director of the NSF Industry/University Center for High-Performance Reconfigurable Computing (CHREC). El-Ghazawi's research interests include high9 performance computing, computer architectures, reconfigurable, embedded computing and computer vision. He is one of the principal co-authors of the UPC parallel programming language and the first author of the UPC book from John Wiley and Sons. He has received his Ph.D. degree in Electrical and Computer Engineering from New Mexico State University in 1988. El-Ghazawi has published about 200 refereed research publications in this area. Dr. El-Ghazawi has served in many editorial roles and is currently an Associate Editor for the IEEE Transactions on Computers. He has chaired and co-chaired many international conferences and symposia including the 2009 Conference on Partitioned Global Address Space (PGAS) Programming Models and Languages (PGAS2009), The 10th IEEE International Conference on Scalable Computing and Communications (ScalCom-10), 2010, and the 9th ACS/IEEE Conference on Computer Systems and Applications, AICCSA2011. Dr. El-Ghazawi's research has been frequently supported by Federal agencies and industry. He serves or has served on many advisory boards including the Science Advisory Panel of the Arctic Region Supercomputing Center. Professor El-Ghazawi was elected to a Fellow of the IEEE with the citation "for contributions to reconfigurable computing and parallel programming".