Integrated Data and Task Management for Scientific Applications

Jarek Nieplocha[1], Sriram Krishamoorthy[1], Marat Valiev[1], Manoj Krishnan[1], Bruce Palmer[1], and P. Sadayappan[2]

[1] Pacific Northwest National Laboratory,
Richland, WA 99352, USA
{jarek.nieplocha,sriram,marat.valiev,
manoj,bruce.palmer}@pnl.gov
[2] The Ohio State University,
Columbus, OH 43210, USA
saday@cse.ohio-state.edu

Abstract. Several emerging application areas require intelligent management of distributed data and tasks that encapsulate execution units for collection of processors or processor groups. This paper describes an integration of data and task parallelism to address the needs of such applications in context of the Global Array (GA) programming model. GA provides programming interfaces for managing shared arrays based on non-partitioned global address space programming model concepts. Compatibility with MPI enables the scientific programmer to benefit from performance and productivity advantages of these high level programming abstractions using standard programming languages and compilers.

Keywords: Global Array programming, computational kernels, MPI, task management, data management.

1 Introduction

Since the dawn of distributed memory parallel computers, the development of application codes for such architectures has been a challenging task. The parallelism in the underlying problem needs to be identified and exposed; the data and computation then must be partitioned and mapped onto processors to achieve load balancing, and finally the interprocessor communication required to exchange the data between individual processors has to be carefully orchestrated to avoid deadlocks and unnecessary delays. When communication costs cannot be completely eliminated, alternative approaches should be taken to minimize the adverse effects of communication on scalability. To achieve good performance as well as scalability, knowledge of the network topology, memory hierarchy, and even some understanding of the underlying implementation of communication libraries has been required. Factors such as these have made parallel programming a difficult task, leaving it to a limited number of expert scientific programmers.

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 20–31, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Over the last two decades, a number of high level programming languages and libraries have been created to address the challenges of the software development for scalable architectures. Unfortunately, most of these high-level programming models have not been given enough time and/or resources to mature and advance from the proof-of-concept to the production stage. There are several reasons contributing to this problem, including: immature run-time systems used to implement advanced programming models, shortage of funding available to the research community that would be required for adequate validation, testing, debugging, and optimizing of prototype programming models, constantly evolving and advancing parallel architectures, and unrealistic expectations from the users about performance and scalability of applications implemented using prototype implementations of novel programming models.

For over a decade, we have been working on the development of the Global Arrays (GA) programming toolkit[1]. By working closely with application developers, we have been fortunate to be able define, tune, and evaluate the feature set and implementation choices for advancing GA. One of the key decisions in development of GA was interoperability with the mainstream programming model in scientific computing, MPI. GA has enabled scientific programmers to start with the low-level, low-productivity standard and use more advanced features of globally addressable data structures. The global address space abstractions can greatly improve programmer productivity by freeing the programmer from explicitly partitioning data and orchestrating the communication required to access the data. They allow focussed efforts on other key aspects of parallel computations such as parallelism or load balancing. The GA toolkit has in fact been adopted by several very large applications and used successfully by scientists without any formal training in parallel computing.

This paper provides an overview of the integration of data and task parallelism in the context of the Global Array programming model. An overview of the GA capabilities is provided. An application example is provided to show how these capabilities can be used in a real complex scientific application that requires advanced management of distributed data and tasks executing on processor groups.

2 Application Motivations

Exploiting all available forms of parallelism is becoming increasingly important for programming forthcoming high-end systems. Such systems, containing tens or hundreds of thousands of processors, present a challenge to many important scientific applications. Many applications that require these high-end systems tend to be composed of algorithms with variable computation/communication granularity. The question on how to partition computational resources and manage them to execute the overall application effectively is becoming critical to our ability to take advantage of the massively parallel hardware. One strategy to limit the negative effect of Amdahl's law on the overall efficiency and scalability of the application is execute the finer granularity algorithms on smaller subsets of processors, where their efficiency and speedup are high.

In many important problem areas such as environmental remediation, drug and enzyme design, and development of new energy sources, we need to gain molecular

level understanding of macroscopic phenomena. The fate of many macroscopic processes is often dependent upon intricate details of numerous chemical transformations at the microscopic (angstrom) level. Fundamental understanding of these phenomena is highly desirable for many large-scale practical applications in biology, energy, and climate areas, providing a way for rational control. The immense dimensions of this problem make it an ideal candidate for emerging peta and exascale computing platforms, but the necessary mathematical and software tools are inadequate or virtually nonexistent. The inherent complexity of the problem in conjunction with management of vast number of computing nodes presents challenges far beyond the conventional scientific applications codes. The presence of multiple scales requires concurrent parallel engagement and information exchange between different computational kernels (e.g. quantum, classical, continuum) with potentially thousands or more coupled simulations running at the same time. These simulations will have to dynamically managed and reconfigured subject to available computing nodes and network bottlenecks including the inherent run-time failures. Given the vast numbers of potential scenarios and outcomes, the details of the simulation cannot be anticipated ahead of time requiring intelligent software for managing complex data, scheduling processor resources and task execution.

3 Global Arrays

In the traditional shared-memory programming model, data is located either in “private” memory (accessible only by a specific process) or in “global” memory (accessible to all processes). In shared-memory systems, global memory is accessed in the same manner as local memory, i.e., by load/store operations. The shared-memory paradigm eliminates the synchronization that is required when message passing is used to access non-private data. The Global Arrays toolkit combines the best features of the shared and distributed-memory programming models [1]. It implements a shared-memory programming model in which data locality is managed by the programmer through explicit calls to functions that transfer data between a global address space (a distributed array) and local storage. In this respect, the GA model has similarities to distributed shared-memory (DSM) models [2, 3] that provide an explicit acquire/release protocol. However, the GA model acknowledges that remote data is slower to access than is local data and therefore allows data locality to be explicitly specified and hence managed. Another advantage is that GA, by optimizing and moving only the data requested by the user, avoids issues such as false sharing or redundant data transfers present in some DSM solutions. The GA model exposes to the programmer the hierarchical memory of modern high-performance computer systems, and by recognizing the communication overhead for remote data transfer, it promotes data reuse and locality of reference. The GA programming model includes as a subset message passing; in particular, the programmer can use full MPI functionality on both GA and non-GA data.

The Global Array toolkit can be used as a distributed array library in an MPI-based application or as a complete programming environment based on a shared-memory approach. The core capabilities of GA are in the area of management of dense distributed arrays and a set of operations for sparse data management is available [1].

The library can be used in C, C++, Fortran 77, Fortran 90 and Python programs. The capabilities of the GA toolkit include:

1. The GA toolkit provides extensive support for controlling array distribution and accessing locality information. Global arrays can be created by (1) allowing the library to determine array distribution, (2) specifying decomposition only for one array dimension and allowing the library to determine the others, (3) specifying the distribution block size for all dimensions, or (4) specifying irregular distribution as a Cartesian product of irregular distributions for each axis. The distribution and locality information is available through library operations that (1) specify the array section held by a given process, (2) specify which process owns a particular array element, and (3) return a list of processes and the blocks of data owned by each process corresponding to a given section of an array.

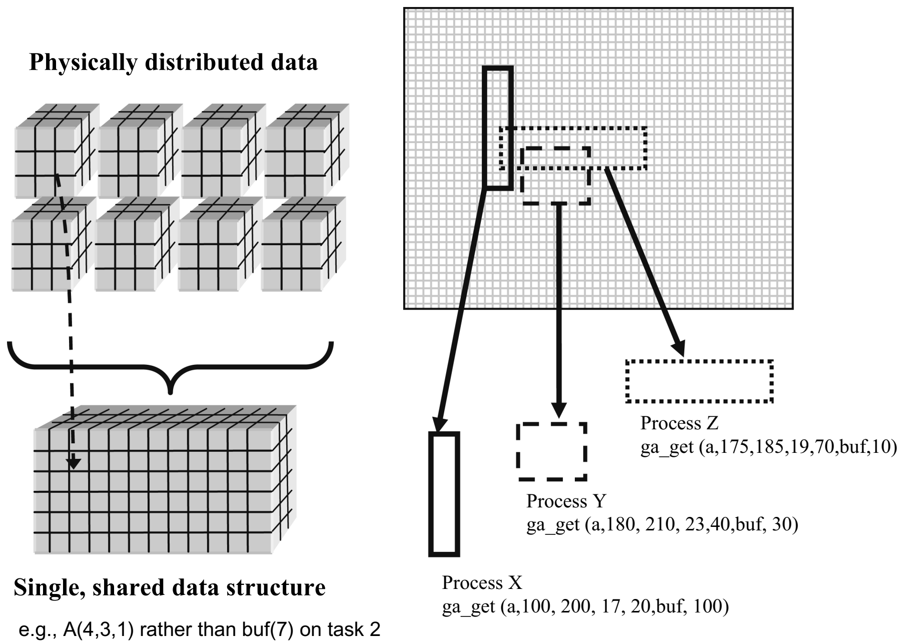

Fig. 1. Left: GA manages a distributed array as a single shared data object. As shown, any process/task can access the distributed data using global indexing (e.g., using global index A(4,3,1)). Right: Any part of the array can be accessed noncollectively as if it is located in shared memory (e.g., Process X gets a block of the global array with global indices starting at (100,17) and block size=100x3).

2. The GA toolkit offers communication calls to support for both task and data parallelism. Task parallelism is supported through the one-sided (noncollective) copy operations that transfer data between global memory (distributed/shared array) and local memory. In addition, each process is able to access directly data held in a section of a global array that is logically assigned to that process.

Atomic operations are provided that can be used to implement synchronization and ensure correctness of an accumulate operation (floating-point sum reduction that combines local and remote data) executed concurrently by multiple processes and targeting overlapping array sections.

3. The data parallel computing model is supported through the set of collectively-called functions that operate on either entire arrays or sections of global arrays. The set includes BLAS-like operations (copy, additions, transpose, dot products, matrix multiplication). Some of them are defined and supported for all array dimensions (e.g., addition). Some other operations such as matrix multiplication are limited to one- or two-dimensional arrays (however, multiplication is also offered on two-dimensional subsections of higher dimensional arrays). The set of data parallel operations has been enlarged to support the requirements of TAO. The extensions included element-wise operations on arrays (e.g., elementwise addition of two arrays, or shifting diagonal).
4. GA extends its capabilities by offering interfaces to third-party libraries in the area of linear algebra through ScaLAPACK) [4] and optimization through TAO [5, 6].

For performance reasons shared memory is used for storing global arrays within SMP nodes. Therefore, any process/task can directly access the memory allocated for a global array on any other process in the same SMP node. Although every process is guaranteed to have fast access to the portion of array it owns, all the other processes in the same SMP node are able to access this memory directly, thereby avoiding unnecessary copies. In the case of a shared memory system, such as the SGI Altix, a process can access data in the entire global array directly. An appropriate interface for task mapping to individual SMP nodes of a cluster in the parallel job was introduced to enable exploiting the performance advantages of shared memory.

Exploiting processor groups is becoming increasingly important for programming next-generation high-end systems composed of tens or hundreds of thousands of processors. To preserve compatibility of GA with MPI, GA follows the MPI approach to the processor group management as closely as possible. However, with GA the management of shared data rather than the explicit interprocessor communication is the main focus area. GA includes calls to create, access, share, and destroy shared data in the framework of the processor group management of MPI. For example, Fig. 2 illustrates the concept of using shared arrays by processor groups. The three processor groups (Group1, Group2, and Group3 in Fig. 2) execute tasks that operate on three arrays: A, B, and C. Array A is in the scope of all three processor groups. Array B is distributed on processor Group 1. Array C is distributed on processor group 3. All arrays can be accessed using collective (individual and multiple arrays) and one-sided (non-collective) operations by any processor in the group that owns the array.

The concept of the *default processor group* is a powerful capability that enables rapid development of new group-based codes and simplifies conversion of existing, non-group aware codes. Under normal circumstances, the default group for a parallel calculation in GA is the MPI world group (contains the complete set of processors), but a call is available that can be used to change the default group to a processor subgroup. This call must be executed by all processors in the subgroup. Furthermore,

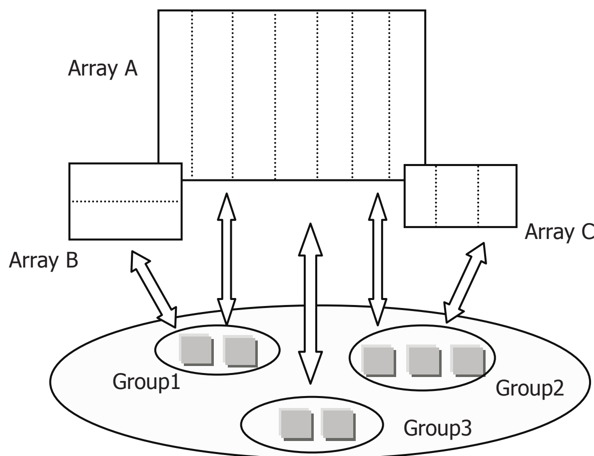

Fig. 2. An example of multilevel parallelism in Global Arrays

although it is not required, it is probably a *very* good idea to make sure that the default groups for all processors in the system (i.e. all processors contained in the original world group) represent a complete non-overlapping covering of the original world group. Once the default group has been set, all operations are implicitly assumed to occur on the default processor group unless explicitly stated otherwise. Shared arrays are created on the default processor group and global operations by default are restricted to the default group. Inquiry functions, such as the number of tasks and the task rank, return values relative to the default processor group.

4 Task Pools

Task parallelism is a popular technique for expressing parallelism in programs that exhibit irregular, sparse, or nested parallelism. Mixed task and data parallelism exists naturally in many application programs. Integrating task and data parallelism improves application performance in practice, however utilizing it may require sophisticated scheduling algorithms and software support [8]. In such programs, static partitioning can often lead to load imbalance due to irregularity in the problem spatial domain, sparsity in the data, or algorithmic characteristics of the problem where computational effort is not directly correlated with the data sizes. By decomposing the problem into tasks and dynamically mapping the computation onto available resources, these classes of applications are able to achieve scalable performance. In addition to overcoming irregularity in the computation, dynamic task scheduling can also be used to mitigate irregularity in the hardware that may be introduced through heterogeneity among processors or in the memory hierarchy.

In the early stage of development, GA model was motivated by the needs of electronic structure computational chemistry applications that deploy task level parallelism. These applications relied on atomic increment of a shared task counter to assign dynamically tasks to individual processors. However, the development of more

complex task management schemes has not been pursued by the application developers.

Recently, we have been designing different task pool management systems complementary to the GA model. Some early efforts were pursued in context of the Tensor Contraction Engine (TCE) to manage sparse tensor contraction operations performed by coupled cluster models for *ab initio* electronic structure modeling. This includes locality-aware load-balancing for in-memory computations [9], transparent memory hierarchy management for out-of-core programs [10], and work-stealing techniques [11]. All these approaches focused on tasks that are sequentially executed on a single processor. We have been working with application developers of quantum chemistry codes to define a task management pool suitable for more dynamic schemes and applicable to tasks executed on processor groups rather than individual processor. Moreover, depending on the problem and molecular properties, tasks can vary significantly in their requirements for memory and processor resources.

The abstraction provided to the user, shown in Fig. 3, enables the specification of a set of independent tasks to be executed in parallel. For each such set, all processes collectively create a task pool object using the create task pool method. Each task in the task pool is identified by the routine to be invoked to process that task, identified by a function handle, and the set of locality elements it operates upon. In addition, any private data specific to that task can also be specified. Each locality element corresponds to a global data region, identified by its global address, size, and its access mode. Three access modes are supported. Read, write, and access modes allow for put, get, and accumulate of global data. For dense and block-sparse arrays, the global address is replaced by the array handle and the specification of the data region being accessed. Tasks are added to a task pool using the add task method. The creation and addition of tasks to the task pool is done by all processors. Once all the tasks have been added to the task pool, the seal task pool method is used to seal

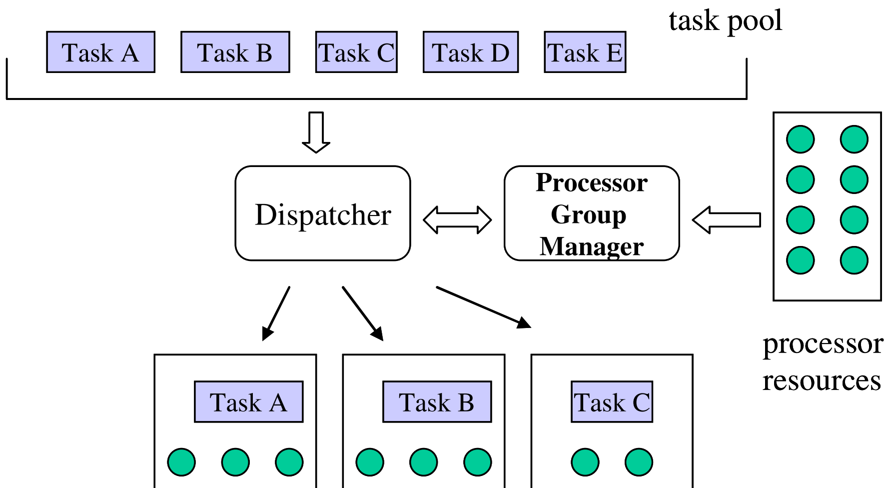

Fig. 3. Task Pool Management

the work pool. This method is invoked once for a task pool and is used to perform start-time optimizations. Subsequently, Dispatcher processes the task pool and calls Processor Group Manager to create and appropriately sized processor groups. Dispatcher then assigns tasks to the processor groups to process the tasks in the task pool. Once a task is finished a processor group can be assigned another task or destroyed. A task pool, once created, can be processed multiple times. The cost of start-time optimizations, performed once, can thus be amortized over multiple cycles.

The task pool approach for processor groups appears quite attractive for building applications with multi-level parallelism. Decomposition of the problem into tasks to be assigned to different processor groups is in particular attractive for massively parallel systems and problems that include some parts that lack sufficient parallelism to be executed on all the processors the application is using.

Our approach for integrating task and data parallelism has several performance benefits in practice. Earlier efforts [12, 13] have pursued similar goals. However they are either compile-based approach or not suited for irregular computations. Moreover, these approaches do not offer much control to the user. For example, they are more biased to task parallelism and offer less support for data parallelism, or vice versa. Some of the earlier approaches are based on MPI processor group model, which imposes additional constraints on the task parallelism as outlined in Section 4.1. In addition, our approach uses global address space model which offers significant productivity advantages in the data management area.

4.1 Processor Group Management

One of the design obstacles involved overcoming the limitations of the MPI processor group management. Under MPI all the processors in the parent group have to participate in the subgroup creation process. This constraint makes it impossible to create and destroy on-demand processor groups to match requirements of tasks on top of the queue. We designed a more flexible group management system and integrated it with ARMCI runtime-system that GA uses [7]. ARMCI in particular implements a basic set of collective communication calls on processor groups. The subgroup creation process is coordinated between processors that form a subgroup. However, this coordination can be implicit and is implemented without use of collective operations. The only assumption is that all the processors in the subgroup must be aware of the group membership.

5 Application Example

An important representative problem where the issues outlined in Section 2 start to play an important role can be found in dynamical simulation of soft matter system consisting of collection of loosely bound molecules (e.g. liquid water). Examples of such systems can be found in wide range of applications related to environmental remediation, catalysis, biology, and other areas. To provide a realistic description of these systems requires consideration of relatively large fragments (more than 103 atoms) that, combined with periodic boundary conditions, give an insight into bulk behavior which is most relevant to practical applications. Until recently problems of

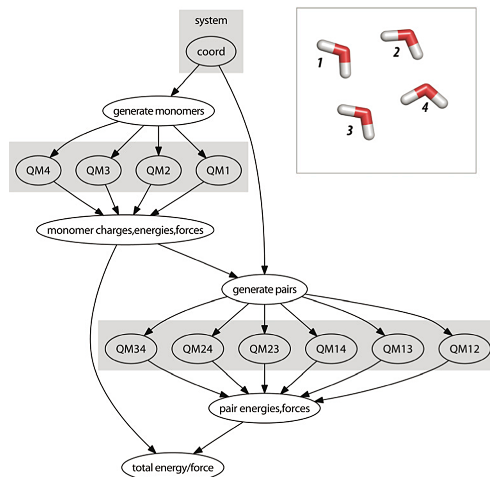

Fig. 5. Task graph for fragment based QM approach illustrated for a system containing 4 molecules

this size could only be described by classical molecular mechanics based on empirical pair potentials. In many cases however a quantum-mechanical based description would be quite beneficial providing validation for classical models as well as a description of electronic structure properties. Given that a direct quantum-mechanical based description would be unreasonable for these large systems, alternative approaches have been developed that combine both quantum mechanical and classical methods. Among them is a fragment based quantum mechanical description which utilizes a cluster-like many-body expansions where the short range quantum mechanical effects are evaluated on the quantum mechanical level while long-range electrostatic effects are handled classically [14]. For example, in the second order (binary) fragment expansion the total energy of the system is represented by

$$E^{\text{binary}} = \sum_{i_1} E_{i_1} + \sum_{i_1 < i_2} \left(E_{i_1 i_2} - E_{i_1} - E_{i_2} \right)$$

where E_i is the energy of the ith monomer and E_{ij} the energy of the dimer made from i and j molecules of the system. These individual energy terms are obtained as the eigenvalues of the Schrödinger equation

$$H_{i_1 \cdots i_k} \psi_{i_1 \cdots i_k} = E_{i_1 \cdots i_k} \psi_{i_1 \cdots i_k}$$

Here the Hamiltonian,

$$H_{i_1 \cdots i_k} = H^0_{i_1 \cdots i_k} + \sum_{i_m \notin \{i_1, \ldots, i_k\}} V_{i_m}$$

is given by the sum of the gas phase Hamiltonian $H^0_{i_1 \cdots i_k}$ of the isolated cluster $i_1 \cdots i_k$ and ,V_{i_m} ,the electrostatic interaction between the subcluster $i_1 \cdots i_k$ and other subunits i_m (not belonging to the subcluster).

$$V_{i_m}^{\text{Coulomb}}(\mathbf{r}) = \sum_{a \in i_m} \frac{Z_a}{|\mathbf{r} - \mathbf{r}_a|} - \int \frac{\rho_{i_m}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}'$$

To facilitate the calculations the electrostatic potential can be approximated by the effective electrostatic charges fit to reproduce the "true" electric field on a grid point surrounding the system.

$$\sum_{a \in i_m} \frac{Z_a}{|\mathbf{r} - \mathbf{r}_a|} - \int \frac{\rho_{i_m}(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' = \sum_{a \in i_m} \frac{Q_a}{|\mathbf{r} - \mathbf{r}_a|}$$

The calculation in the fragment-based approach proceeds according to the following scheme, see Fig. 4.

1. For each monomer in the system the Schrodinger equation for the monomer and new charges Q_a are calculated.
2. Step 1 is repeated until self-consistency is reached where charges Q_a and correspondingly energies E_i are no longer changing.
3. For each dimer in the system the Schrodinger equation is solved with other molecules represented by point charges calculated in steps 1-2.

Steps 1 and 2 are ideally suited for exploiting the parallel task pools approach. On one hand the monomer calculations appear fully parallel, but there is nontrivial nonlinear coupling through the electrostatic charges that they generate.

6 Experimental Results

The experiment evaluation was performed on a Quadrics cluster with dual 1.5GHz Itanium nodes running Linux kernel 2.6.11.

The example system considered in this work consists of 256 water molecules enclosed in a ~20A cubic box. Only the monomer part of the calculation was considered at this point. After the initialization step where the coordinates of monomer units and surrounding charges are assembled, the quantum calculations of each monomer unit are executed independently subject to available processor pool. We performed the entire self-consistent loop until charges on all the monomers have converged. This involves one-sided communication of the charges from the processor group to a global charge array which is monitored for convergence. The performance of our scheme is shown in Fig. 4, labeled *task-pool*. In this case the processor groups were uniformly sized and included two processors. For comparison, we evaluated the performance of the application, with each monomer calculation being performed by all the processors collectively (MPI_COMM_WORLD), denoted *do-all* in the figure. As our data illustrates this approach does not scale due to inherent limitation in the

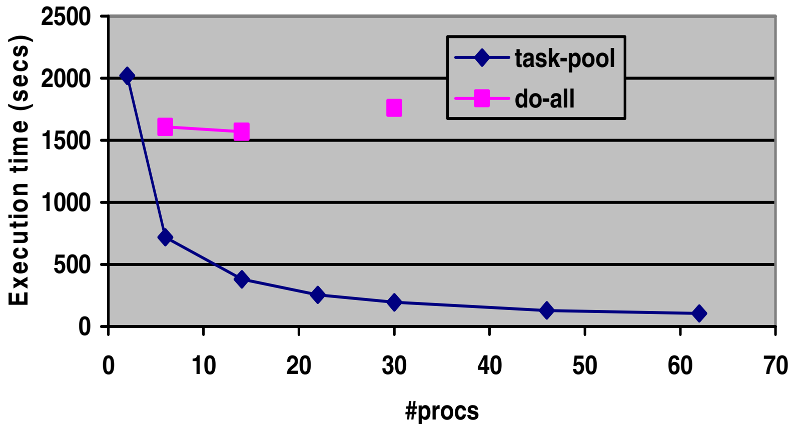

Fig. 6. Execution time for the monomer QM calculations

scalability of the quantum calculation. We observe almost linear speedup with the number of processors in our task pool-based implementation.

7 Conclusions and Future Work

We described an integrated data and task management system in context of the Global Arrays toolkit that supports a global address space programming model. Our task management system can handle variable sized processor groups. This capability was used for a quantum mechanical application with complex data and task management requirements. The experimental results demonstrate very good scaling. Our plans for future work include more experimental validation with other applications, and adding locality optimization to the task pool management, and implementing a distributed version of the task pool Dispatcher to handle very large numbers of processors.

References

1. Nieplocha, J., Palmer, B., Tipparaju, V., Krishnan, M., Trease, H., Apra, E.: Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit. International Journal of High Performance Computing and Applications 20(2) (2006)
2. Bershad, B.N., Zekauskas, M.J., Sawdon, W.A.: Midway distributed shared memory system. In: 38th Annual IEEE Computer Society International Computer Conference – COMPCON SPRING 1993, February 22-26 1993, pp. 528–537. IEEE, Piscataway (1993)
3. Cox, A.L., Dwarkadas, S., Lu, H., Zwaenepoel, W.: Evaluating the performance of software distributed shared memory as a target for parallelizing compilers. In: 1997 11th International Parallel Processing Symposium, IPPS 1997, April 1-5 1997, pp. 474–482. IEEE, Los Alamitos (1997)

4. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK: A Linear Algebra Library for Message-Passing Computers. In: Proceedings of Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN (1997)
5. Benson, S., McInnes, L., Moré, J., Sarich, J.: Toolkit for Advanced Optimization (TAO) User Manual. ANL/MCS-TM-242 (2004), http://www.mcs.anl.gov/tao
6. Benson, S., Krishnan, M., McInnes, L.C., Nieplocha, J., Sarich, J.: Using the GA and TAO toolkits for solving large-scale optimization problems on parallel computers. ACM Trans. Math. Softw. 33(2), 11 (2007)
7. Nieplocha, J., Tipparaju, V., Krishnan, M., Panda, D.: High Performance Remote Memory Access Communications: The ARMCI Approach. International Journal of High Performance Computing and Applications 20(2) (2006)
8. Chakrabarti, S., Demmel, J., Yelick, K.: Modeling the benefits of mixed data and task parallelism. In: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures, June 24-26, 1995, pp. 74–83 (1995)
9. Krishnamoorthy, S., Nieplocha, J., Sadayappan, P.: Data and computation abstractions for dynamic and irregular computations. In: Bader, D.A., Parashar, M., Sridhar, V., Prasanna, V.K. (eds.) HiPC 2005. LNCS, vol. 3769. Springer, Heidelberg (2005)
10. Krishnamoorthy, S., Catalyurek, U., Nieplocha, J.: Hypergraph partitioning for automatic memory hierarchy management. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2006) (November 2006)
11. Dinan, J., Krishnamoorthy, S., Larkins, B., Nieplocha, J., Sadayappan, P.: Scioto: A framework for global-view task parallelism (under submission)
12. Bal, H.E., Haines, M.: Approaches for Integrating Task and Data Parallelism. IEEE Concurrency 6(3), 74–84 (1998)
13. Rauber, T., Rünger, G.: Library support for hierarchical multi-processor tasks. In: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, November 16, 2002, pp. 1–10 (2002)
14. Kamiya, M., Hirata, S., Valiev, M.: Fast electron correlation methods for molecular clusters without basis set superposition errors. Journal of Chemical Physics 128(7), 74103 (2008)