Proc. 5th Int. Symp. High Performance Distributed Comp. HPDC-5
(best paper award)

Shared Memory NUMA Programming on I-WAY

J. Nieplocha and R. J. Harrison

Pacific Northwest National Laboratory, P.O. Box 999, Richland WA 99352

Abstract

The performance of the Global Array shared-memory non-uniform memory-access programming model is explored on the I-WAY, wide-area-network distributed supercomputer environment. The Global Array model is extended by introducing a concept of mirrored arrays. Latencies and bandwidths for remote memory access are studied, and the performance of a large application from computational chemistry is evaluated using both fully distributed and also mirrored arrays. Excellent performance can be obtained with mirroring if even modest (0.5 MB/s) network bandwidth is available.

1 Introduction

We are interested in studying the performance of current massively parallel chemistry codes on distributed computing resources, in particular the variable latency, limited bandwidth, distributed environment of I-WAY (described below). Our applications employ the Global Array (GA) non-uniform memory-access (NUMA) programming model [1-3] and reference data in blocks to increase data locality in order to reduce sensitivity to latency. The GA toolkit provides an efficient and portable "shared-memory" programming interface for distributed-memory computers. Each process in a multiple-instruction multi-data (MIMD) parallel program can asynchronously access logical blocks of physically distributed matrices, without need for explicit cooperation by other processes. Unlike other shared-memory environments like Linda [4] or TreadMarks [5], the GA model exposes to the programmer the NUMA characteristics of the high performance computers and acknowledges that access to remote data is slower than to local data. Information about the distribution of shared data is available and very efficient access to the local portion of distributed data is provided. Global Arrays were designed to complement rather than replace the message-passing programming model. The programmer is free to use both the shared-memory and message-passing paradigms in the same program, and to take advantage of existing message-passing software libraries.

The Information Wide Area Year (I-WAY) [6] was proposed as a high-performance network linking several supercomputers and visualization environments located across the country, and was planned to use Asynchronous Transfer Mode (ATM) technology. The initial goal had been to provide OC-12c (622 Mbps) rate connection to some major supercomputer sites and OC-3c (155 Mbps) to the other sites by the Supercomputing'95 conference in December 1995. As a part of the Supercomputing'95 conference, a number of demonstrations of applications capable of using I-WAY network, supercomputer and/or visualization environments took place. For most of the time, several of the supercomputer sites participating in the I-WAY experiments were connected only at tens of kilobyte/s rates.

Some of the existing networks that were targeted for integration into the I-WAY include: CASA, vBNS, ESnet, AAINET, ATDNET, NAS NREN, MAGIC, TCG SONET and ACTS. Perhaps even more challenging than the hardware targets of the project were the system software efforts that intended to provide a view of supercomputer resources connected to the network as a single general-purpose integrated computing resource and overcome the computer-hardware heterogeneity, geographic distribution, varying network connectivity, localized and independent management of resources at supercomputer centers [7,8].

This paper is based on results of a project on NUMA programming in the distributed environment and the Supercomputing'95 demonstration prepared by Pacific Northwest National Laboratory and Argonne National Laboratory. The paper is organized as follows. In Section 2, we review the Global Arrays programming model. In Section 3, we describe implementation of GA in the I-WAY environment, provide performance results, and introduce a mirrored array approach. Section 4 outlines a large computational chemistry application, NWChem. In Section 4, we present performance results of NWChem on two Intel Paragons and two IBM SPs connected with wide-area network. The conclusions are given in Section 6.

2 Global Arrays

The Global Array (GA) programming model [1,3] was motivated by the algorithmic requirements of computational chemistry applications and by the NUMA characteristics of modern parallel architectures. NUMA memory hierarchy -- formed by registers, on-chip cache, off-chip cache, main memory, and virtual memory if available -- is present in the vast majority of sequential and parallel architectures. Parallel computers add at least one extra layer to the NUMA hierarchy of sequential computers -- remote memory.

The key concept of GA is that it provides a portable interface through which each process can independently, asynchronously, and efficiently access logical blocks of physically distributed matrices, with no need for explicit cooperation by other processes, see Figure 1. By removing unnecessary processor interactions that are required in the message-passing paradigm to access remote data, the Global Array model greatly simplifies parallel programming. In this respect, it is similar to the shared-memory programming model. However, the GA model also acknowledges that remote data is slower to access than local (i.e., remote memory as yet another layer of NUMA), and it allows data locality to be explicitly specified and used. By recognizing the communication overhead for remote data access, GA promotes optimization by data reuse and locality of reference.

Global Arrays were designed to complement rather than replace the message-passing programming model. The programmer is allowed to use both the shared-memory and message-passing paradigms in the same program, and to take advantage of existing message-passing software libraries.

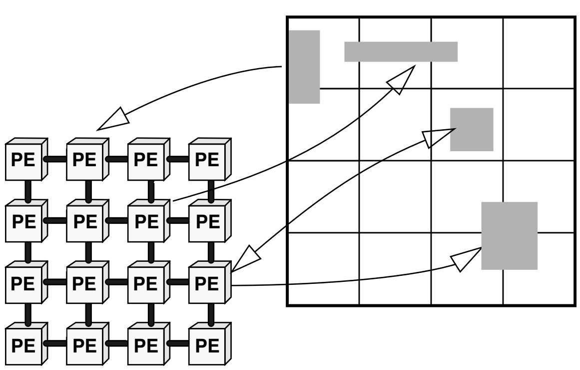

Figure 1: Asynchronous one-sided access to sections of a distributed matrix

The Global Arrays programming model can be characterized as follows:

- MIMD parallelism is provided using a multi-process approach, in which all non-GA data, file descriptors, and so on are replicated or unique to each process.
- Processes can communicate with each other by creating and accessing GA distributed matrices, and also (if desired) by conventional message-passing.
- Matrices are physically distributed blockwise, either regularly or as the Cartesian product of irregular distributions on each axis.
- Each process can independently and asynchronously access any patch of a GA distributed matrix, without requiring cooperation by the application code in any other process.
- Several types of access are provided, including *get*, *put*, *scatter, gather*, *accumulate* (floating point sum-reduction), and atomic *read-and-increment* (integer).
- GA memory is *weakly consistent*. Most of the operations that modify GA memory like *put, scatter,* and *accumulate* might return to the caller before the actual operation completes. A *barrier* operation guarantees completion of all the data transfers.
- Each process is assumed to have fast access to some portion of each distributed matrix, and slower access to the remainder. These speed differences define the data as being 'local' or 'remote', respectively. However, the numeric difference between 'local' and 'remote' access times is unspecified.
- Each process can determine which portion of each distributed matrix is stored 'locally'. Every element of a distributed matrix is guaranteed to be 'local' to exactly one process.

The GA programming model has been implemented in the Global Array toolkit [9] that currently supports two-dimensional arrays. The interface has been designed in light of emerging standards. In particular, HPF [10] certainly provides the basis for future standards definition of distributed arrays in FORTRAN. The basic functionality in the GA toolkit (create, fetch, store, accumulate, gather, scatter, and data-parallel operations) all may be expressed as single statements using FORTRAN-90 array notation and the data-distribution directives of HPF‡.

‡. What HPF does not currently provide is random access to regions of distributed arrays from within a MIMD parallel subroutine call-tree, and reduction into overlapping regions of shared arrays

The GA toolkit supports four distinct environments:

1. Distributed-memory, message-passing parallel computers with interrupt-driven communications or Active Messages (Intel iPSC/860, Delta and Paragon, IBM SP-1/2).
2. Networked clusters of multiple- or single-processors with simple message passing (using the MPI [11] or TCGMSG [12] portable message-passing libraries).
3. Shared-memory parallel computers (KSR-1/2, SGI, Convex SPP, most brands of UNIX workstations).
4. Globally-addressable distributed-memory computers (Cray T3D).

3 Global Arrays in the I-WAY environment

From the NUMA programming perspective, the I-WAY metacomputer environment provides an additional layer of remote memory with much higher latencies and lower bandwidth than are available in a supercomputer. Conceptually, this environment is similar to the clustered network of multiprocessor workstations. The main differences arise from the ratios of floating-point performance of nodes and network performance. There are three major performance factors to consider when analyzing the effectiveness of network high-performance computing:

- Floating point performance of a network compute node: tens to hundreds of GFLOPS (aggregate for all processors sharing the same network connection) for a supercomputer on the I-WAY WAN vs. tens to hundreds of MFLOPS for a typical workstation.
- Network bandwidth: the LAN supports rates from a fraction of a MB/s to tens of MB/s, see for example Myrinet network [13], vs. tens of KB/s to a few MB/s for the WAN. In addition, multiple (hundreds and even thousands) of processors have to share the bandwidth of a single link to the WAN making the effective network bandwidth per processor orders of magnitude lower than for tightly coupled clusters of workstations.
- Network latency: for the LAN, latency is in order of tens of microseconds for Fast Messages on a Myrinet network [13] to milliseconds for a LAN using standard ETHERNET vs. tens to hundreds of milliseconds for the I-WAY WAN. Unlike the WAN bandwidth that can be improved with advanced network technologies, the latency has physical limits that are currently being approached. For example, 12 milliseconds is required for light to cover the distance between the East and West coast of North America. For a 200 MFLOPS processor, this time is equivalent to 2,400,000 floating point instructions.

We observe that the relative cost of access to remote memory (weighted with floating-point performance) is much higher for I-WAY metacomputer than for clusters of workstations. Despite progress in network technology, the high latencies are and will continue to discourage frequent transfers of small amounts of data over the wide area networks.

In spite of the communication limitations of WAN computing, there are some classes of applications that can take advantage of the potentially enormous computational power of multiple supercomputers. In a two-phase scheme, the application can asynchronously prefetch data while computations take place. This approach however requires significant restructuring of the code and is applicable only to some algorithms. The replicated shared memory that has been proposed for wide area computing [14] might also be effective in hiding the network latency.

We implemented the Global Arrays NUMA programming model for the I-WAY environment and studied performance of the basic communication primitives. We also extended the model with new functionality in the spirit of replicated shared memory that can be used by the applications to reduce the effects of the high cost of communication over the WAN.

3.1 Implementation

The implementation of Global Arrays in the I-WAY environment is based on a ‘data-server’ (or ‘hidden-agent’) concept. This idea is also exploited in a somewhat different manner in the implementation of the GA toolkit on networks of workstations. In the I-WAY environment, availability of a general-purpose message-passing system that allows communication between any pair of processes across WAN is not assumed. Instead, we implemented a simple message-passing system for point-to-point communication based on the TCP/IP protocol. The message-passing library is employed to communicate between ‘data-servers’ while local/native message-passing is used for communication between processes executing on the same machine. A ‘data-server’ node is a distinct process, which on distributed-memory architectures executes on a dedicated processor, that is devoted exclusively to the I-WAY network communication, see Figure 2. Processes executing the application code are referred to as ‘compute nodes’.

The ‘data-server’ process is excluded from the set of processes created by the user at initialization time and does not execute the application code. Instead, it polls for request messages from ‘local’ processes as well as the other ‘remote’ ‘data-servers’. Request messages from local processes are forwarded to the appropriate ‘data-servers’ using the socket message-passing library. Incoming remote requests are

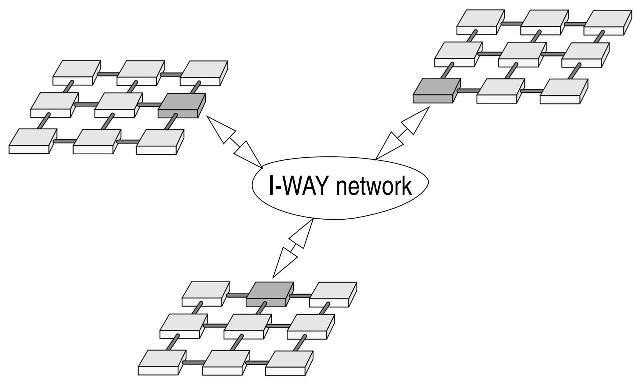

Figure 2: Only one processor (server node) on each machine can communicate directly across the network

handled differently according to the local machine architecture, see Figure 3 and 4 and the following description.

As an example, let us consider implementation of the remote store operation, *ga_put*, which copies data from the local/private memory of the calling process to the global memory accessible to any process through the GA library calls. For the class of shared- and globally-addressable - memory machines, implementation of the GA toolkit is based on the copy mechanism between local/private process memory and shared/global memory. On another class of MPPs -- message-passing distributed-memory architectures, like the IBM SP or Intel Paragon, GA is implemented with interrupt-receive functionality. On both classes of architectures, a compute node determines the logical ownership and physical location of the array section where the local data should be copied to. In case of multiple ownership, the request is decomposed and executed on a per-process basis. If a process that 'owns' a target (sub)section of the global array is executing on a remote machine, a request message containing a header record and the data is sent to the local 'data-server' process. The 'data-server' forwards the message to the remote machine through the WAN. The next stage varies depending on the machine architecture.

On shared/global-memory systems, the 'data-server' copies the data to the destination location in memory, see Figure 3. The node that is logically assigned to the corresponding section of the global array is not involved.

On distributed-memory systems, the 'data-server' node forwards the message that arrived from the network to the local node that actually holds the corresponding section of the matrix. The message invokes a handler function at the remote node. From the interrupt handler, a copy routine is

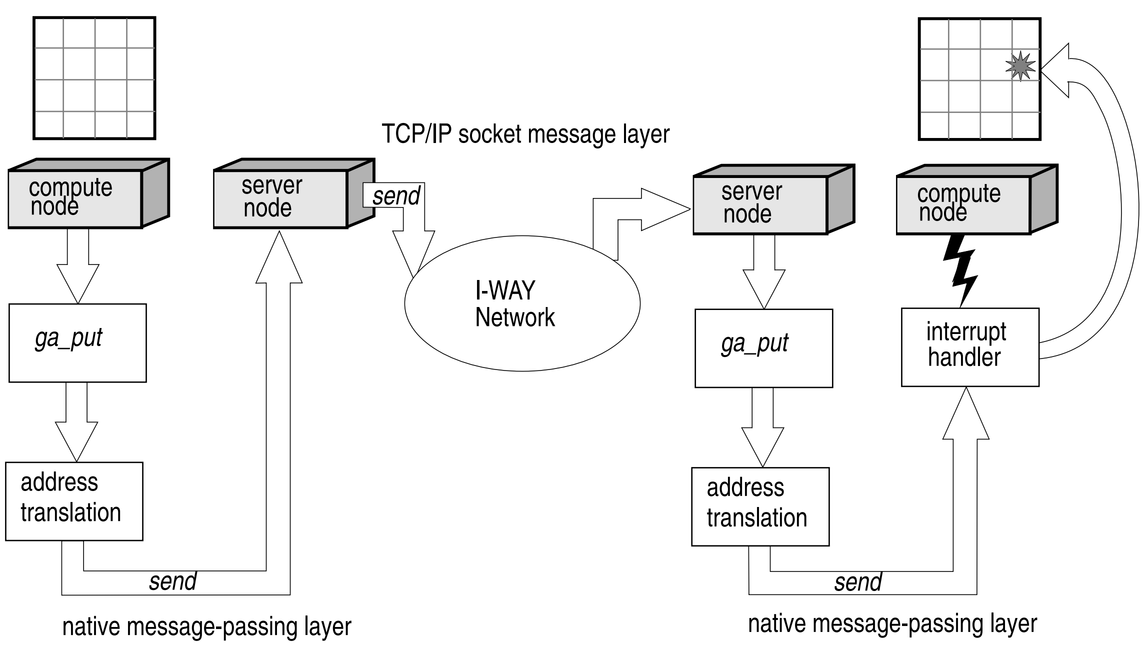

Figure 3: Implementation of a remote put operation on the message-passing systems in the I-WAY environment

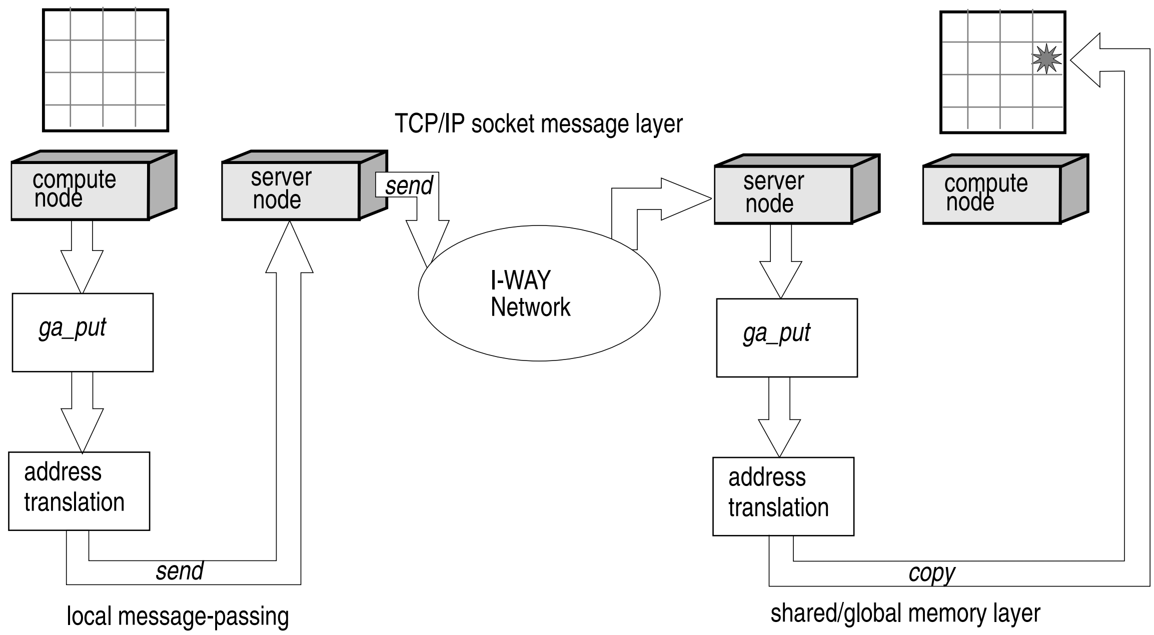

Figure 4: Implementation of a remote put operation on shared/global memory systems in the I-WAY environment

called to move the data to the destination location, see Figure 4.

The message-passing library used for the WAN communication is implemented with nonblocking socket operations and it uses a lean and primitive flow-control algorithm. Only point-to-point communication is provided. The message-passing library is currently not supporting the XDR format thus limiting its applicability to homogenous platforms. This limitation could be easily removed in the subsequent versions of this software. As a convenience to the user, the GA toolkit supplies wrappers to broadcast, integer and floating-point reduce operations that have a uniform user-interface regardless of the message-passing library (MPI or TCGMSG) that is running with GA. In the I-WAY environment, these operations are implemented on top of our socket message-passing library with additional message-packetization in GA required to avoid overflowing limited network buffers.

For current MPP architectures, a single 'data-server' per single machine suffices since the internal network can sustain a higher bandwidth that the external one. However, additional 'data-servers' might be added as WAN bandwidth increases.

3.2 Performance Experiments

The experiments were carried out using a 400-node Intel Paragon at the San Diego Supercomputer Center, a 512-node Paragon at CalTech, a 128-node IBM SP-1 at the Argonne National Laboratory and a 512-node SP-2 at the Cornell Theory Center. The simultaneous access to parallel machines was arranged with help of the system administrators at the involved sites.

The latency and bandwidth of GA primitive operations, *get, put,* and atomic *accumulate* were measured, see Table 1, for the following cases:

- Access to a local section of a global array.
- Access to a section of global array that resides on another node of the same machine.
- Two partitions were created on the same machine, and the test program was executed in the same fashion (with 'data-servers' as it would on two machines connected through the wide area network. The measurements were made for access to a section of the matrix that resides in the memory of a remote process in a different partition than that of the requesting process. The purpose of this experiment was to test the performance of the communication software excluding the WAN performance factor.

Table 1: Performance of GA primitive operations

operation		Paragon		SP-1	
		latency [ms]	bandwidth [MB/s]	latency [ms]	bandwidth [MB/s]
get	local	0.035	90	0.013	86
	remote	0.5	17	0.5	14
	part. remote	300	0.120	6.3	0.42
	WAN remote	110	0.07	168/24	0.045/0.35
put	local	0.030	93	0.013	86
	remote	0.065	35	0.075	17
	part. remote	0.080	35	0.077	1.1
	WAN remote	0.080	35	0.07/0.07	0.054/0.38
accu-mulate	local	0.160	46	0.018	73
	remote	0.065	36	0.077	17
	part. remote	0.080	35	0.091	1.1
	WAN remote	0.085	35	0.09/0.09	0.052/0.36

- Access to a section of a global array that resides in memory of a processor on a remote machine connected with WAN.

The latency was calculated for a single eight-byte word transfer by timing several hundreds of operations and taking the average. The bandwidth was measured for transfer of a 353×353 section (approximately 1MB of data) of a 710×710 square double-precision matrix that is uniformly distributed on four processors.

Unfortunately, the HiPPI CASA network between SDSC and CalTech was not operational during the time when the experiments took place (at Supercomputing'95 and in the following several months), therefore we could only use the Internet. The network bandwidth between these two supercomputer centers was found to be typically around 70 KB/s. The network latency, as measured using the *ping* program executing on a service (not compute) Paragon node, was around 35 ms. The Internet network was, of course, available in a nondedicated mode.

We used both the Internet and the vBNS ATM networks in the experiments on the IBM SP systems. The two sets of WAN bandwidth numbers in Table 1 for the SP-1 correspond to the Internet and ATM network runs. The Internet latency between these two sites varies significantly, and at the time when the measurements shown in the table were taken it was around 150ms. The ATM network latency was less variable and we measured 20-23 ms using the *ping* program. The ATM network was rated below the OC-3 performance. Unfortunately, due to the software configuration problem in connection between the ANL SP-1 and the vBNS network, outbound packets larger than 4470 bytes were transferred at extremely low rate, hundreds times slower than inbound data. This one-directional bottleneck made the ATM network unusable for running our application but we managed to measure performance of GA[‡]. The numbers for the ATM connection given in Table 1 were measured for the ANL SP-1 executing *get* operation while the CTC SP-2 executed *put* and *accumulate* operations.

The IBM SP-1 at Argonne contain processing nodes similar to the IBM RS/6000 model 370 workstations, with 62.5 MHz clock, 32-KB data and 32-KB instruction caches. "Thin nodes" of the IBM SP-2 at Cornell are similar to the IBM RS/6000 model 390 workstations, and they have the same processor clock and twice the amount of cache memory as the SP-1 counterparts. We did not use "wide nodes" of the SP-2. Interprocessor communication performance in GA is roughly 30% better on the Cornell SP-2 that on the ANL SP-1.

Bandwidth in the non-local *put* and *accumulate* operations on the Paragon is observed to be higher that the network bandwidth which is consistent with the semantics of these operations (might return before data reaches the remote destination) and relatively high amount of the system message buffer size in the NX system, 1 MB in our case, that can accommodate multiple request messages without blocking the sender. On the IBM SP, only 8KB per processor of internal message buffer space is available which limits the amount of data that can be sent to the server node without blocking.

Latency of the *get* operation in the test program executing on two partitions of the same Paragon (at CalTech) is unexpectedly high. We suspect that OSF/1 is responsible for this phenomenon, perhaps by executing the networking services in a single-threaded fashion.

‡. Since the ANL SP-1 connection to vBNS is planned to be fixed soon and the ATM network updated to OC-3 rates, we intend to rerun the GA performance test and NWChem experiments and present the updated results at the conference.

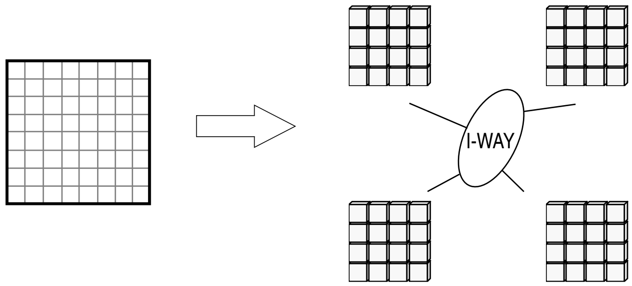

Figure 5: Matrix distributed across the I-WAY machines

3.3 Mirrored Arrays

Replicated shared memory minimizes latency in access to shared data by maintaining fully replicated copies of the data at each node [14]. Data consistency is assured by broadcasting modified pages to all the nodes. In the GA, mirrored arrays are replicated across the I-WAY supercomputers. Each machine instead of storing only a piece of a global array has an entire copy, see Figures 5 and 6. Arrays are fully distributed within each machine. However, the amount of data held by each processor is higher than when arrays are fully distributed among all the machines. Unlike the replicated shared memory, in the current approach the consistency of global memory is controlled explicitly by the user with a new function, *ga_net_merge*. The consistency is enforced only when desired. The new collective operation merges entire or user-specified sections of the mirrored arrays. Upon completion of this operation, all machines have identical copies of the entire (or specified section of the) array. In the mirrored array approach, all the other GA communication operations are executed locally on each machine. The network traffic is lower and average message size sent across the network is larger than in the fully distributed approach.

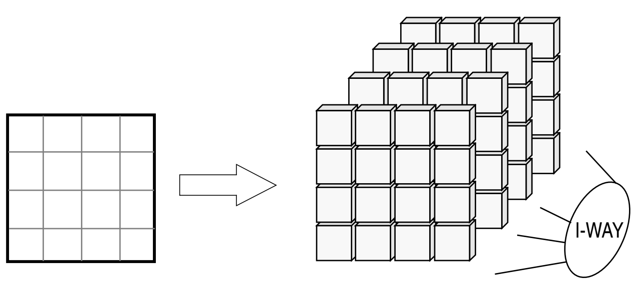

Figure 6: Mirrored matrix on the I-WAY machines

There are several choices in the implementation of the merge operation. Since different sections of the global arrays might be modified on the individual machines, the semantics of *ga_net_merge* allows non-identical sections of the array to be specified on each machine. To handle the possible overlap of section in contributions to the global merge operation, merging is based on the element-wise addition operation, which addresses the needs of the application described in this paper, NWChem:

$$y^{merged} = \sum_{i=1}^{M} y_i \quad , \qquad (1)$$

where M denotes the number of machines contributing to the collective update of the of element y. The current merging operation could be extended to allow specification of other possible choices for the merging operator by the application at run-time.

Mirrored arrays are implemented using the same software infrastructure as in the fully distributed approach, see Section 4.1. All the data traffic across the network is directed through the single selected compute nodes located on each machine, see Figure 2. Since the merge operation is collective, there is no need to apply the 'data-server' scheme and all the processes can execute the application code.

4 NWChem and Distributed-data SCF

NWChem is a computational chemistry package for massively parallel computers developed at Pacific Northwest National Laboratory [15]. The majority of methods in NWChem are used to determine the electronic structure of molecules by forming approximate solutions to the non-relativistic Schrödinger equation within the fixed nuclei approximation. The simplest of these methods is the Hartree-Fock or self-consistent field (SCF) method, in which the molecular wavefunction, a non-separable function of the coordinates of all N electrons, is approximated as an antisymmetric product of N one-electron functions. Each of these one-electron functions, or molecular orbitals (MOs) is expanded in an underlying basis set (typically atom-centered Gaussian-like functions) and the MOs are then determined by minimization of the total energy by variation of the expansion, or MO, coefficients, C.

The non-linear SCF optimization is implemented in NWChem using a pre-conditioned conjugate gradient scheme. The most expensive computational step is computation of the derivative of the energy w.r.t. the MO coefficients which is closely related to the Fock matrix, F:

$$F_{ij} = h_{ij} + \frac{1}{2} \sum_{kl} [2(ij|kl) - (ik|jl)] D_{kl} \tag{2}$$

where $D = CC'$ is the density matrix, and h_{ij} and $(ij|kl)$ are one- and two-electron integrals over the underlying basis functions. The basis is typically of dimension $N = O(10^{2-3})$ and it is easy to see that even allowing for sparsity the two-electron integrals, which carry four labels, are very numerous and must be re-computed as needed, rather than computed once and stored.

Symmetry and sparsity in the matrix of integrals $(ij|kl)$ must be exploited to avoid redundant computation. The major computation is evaluation of the non-zero integrals and the largest data requirements are due to the Fock and density matrices, both $O(N^2)$. There are two major problems:

1. distributing and accessing the matrices so as to minimize communication costs
2. maintaining load balance in the presence of sparsity and large variation in the cost of integral evaluation.

Our distributed-data algorithm is modeled after that of a blocked matrix-multiply algorithm for a high-performance workstation with a small fast memory cache. This is because non-uniform memory access (NUMA) is the fundamental performance issue for both workstations and MPPs. The cost of accessing an element of the distributed density or Fock matrices must be offset by using this element multiple times. To achieve this re-use of data we simply strip-mined the fourfold loop over the basis function indices on the two-electron integrals, Eq. 2. Since the computation is a quartic function of the block size while the communication is only a quadratic function, the block size may be adjusted to ensure the computation time dominates the communication time.

We previously strip-mined by grouping basis functions according to their (usually atomic) centers [16]. This had the advantage that the geometrical sparsity may be exploited in the outer strip-mining loops. However, the granularity was fixed and was inadequate for calculations on large molecules in small basis sets, or when using machines with very high latencies. Our most recent algorithm groups atoms into blocks, looping over quartets of blocks of atoms, and then evaluating all integrals within a quartet of blocks of atoms. Sparsity may still be exploited in the outer loops and the performance gains include:

- many fewer, larger messages,
- elimination or reduction of communication costs to less than one percent, and
- reducing sensitivity to message-passing latency,
- the ability to vectorize integral computation over many thousands of atomic quartets.

5 NWChem on the I-WAY

5.1 Performance of fully distributed approach

Our initial objective, given the projected performance numbers for the I-WAY, was to run NWChem unmodified on the I-WAY metacomputer and to tolerate the WAN latency and bandwidth by:

- increasing the block size within the SCF Fock matrix construction, and
- mapping communication intensive linear algebra operations to execute locally within individual MPPs.

The unavailability of CASA and the poor performance of ATM between Cornell and Argonne, however, produced unacceptable performance in this mode. It was not possible to increase the block size so as to hide latency without introducing intolerable load imbalance.

To illustrate this we ran a single iteration of an SCF (for C_4H_{10} molecule) calculation fully distributed between sixteen Intel Paragon nodes, eight at SDSC and eight at Caltech. This distributed calculation took 317 seconds, to be compared with execution times within the same Paragon of 6.6 seconds and 13.2 seconds on sixteen and eight nodes respectively.

5.2 Performance of mirrored array approach

Within the Fock-matrix construction computational kernel, only the Fock matrix is being updated using floating-point accumulation. Elsewhere in the program, only N^2 matrices are being manipulated using dense linear algebra operations (matrix multiplication, diagonalization) and these will not dominate the computation as long as they are executed in parallel within each MPP connected to the WAN. This special structure of the SCF algorithm, which is not found in higher accuracy computational chemistry methods, made it easy to modify the code to use mirrored arrays with all data and computation replicated between the networked MPPs except in the expensive Fock build. The required modifications were:

- adopting static rather than dynamic load balancing in order to eliminate the associated inter-MPP communication,

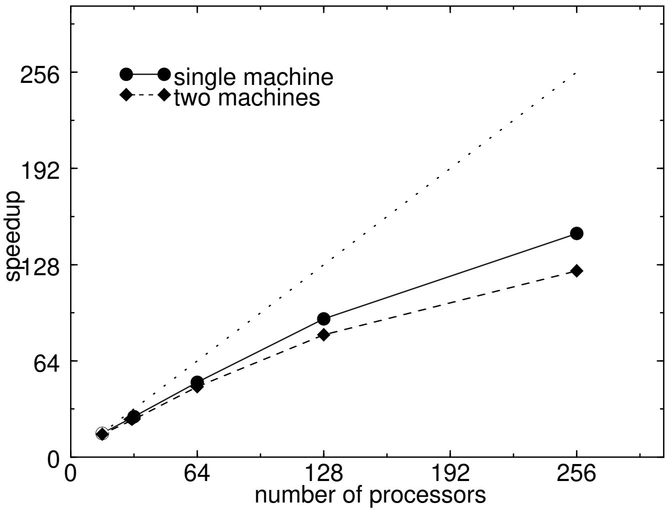

Figure 7: Scaling on single vs. two Paragons connected with WAN

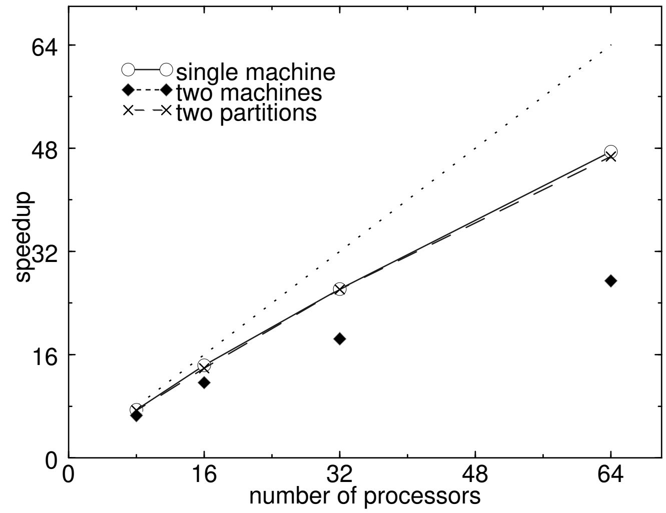

Figure 8: Scaling on single vs two IBM SPs connected with WAN and two partitions of the same machine connected with ethernet

- make the Fock-build algorithm refer to the total number of compute nodes (on all MPPs connected with WAN) rather than just the 'local' number, and
- force a merge of the partial Fock matrices across the WAN.

These changes affected only 50 lines of code in the NWChem package. Computation of the derivative of the SCF energy w.r.t. the coordinates of the nuclei is necessary to determine molecular structure, and proceeds in much the same manner as computation of the Fock matrix, except that now the more expensive derivative two-electron integrals must be computed. This computation was also distributed across the WAN.

Using the mirrored approach, the small calculation for C_4H_{10} discussed above, took just 7.6s using 16 Paragon nodes, eight at SDSC and eight at Caltech, again to be compared with the time of 6.6s for 16 nodes within the same Paragon.

In order to provide a more realistic evaluation we performed an SCF energy and gradient calculation on an 18-crown-6 ether with lithium ion in a modest basis set with a total of 219 basis functions. This is not a very large calculation but is sufficient to explore aspects of performance important to larger calculations. Figures 7 and 8 display the speedup obtained running the calculation distributed between two Paragons (Figure 7) and two IBM SP systems (Figure 8). In Figure 7, the speedup is computed assuming a speedup of four on four nodes. In Figure 8, the single node execution time was available. The speedup was calculated relative to the SP-1. In the distributed case, better performance of SP-2 nodes did not affect the completion time since the same number of nodes was used on both the SP-1 and the SP-2, and the workload was divided evenly. For comparison the entire computation took 4160.9 seconds and 12405.04 seconds on eight nodes of a single IBM SP-1 or Intel Paragon, respectively. The performance of the mirrored approach is satisfactory, and the degradation in performance arises primarily from the overhead of the WAN merge operation, which is sensitive only to network bandwidth ($O(N^2)$ data must be communicated). The better scaling of the distributed Paragon code versus the distributed SP code is readily understood since the IBM SP execution time is lower and there was also less bandwidth available over the WAN connecting the SPs compared with that between the Paragons.

In order to explore the behavior of the application given a higher-performance network, we ran the same calculations distributed between two partitions of the IBM SP at Argonne, using TCP/IP over local ethernet to communicate between the two partitions. A bandwidth of 0.5 Mbytes/second was obtained, and as can be seen from the third set of data in Figure 8 this was sufficient to obtain essentially identical speedup to the equivalent single partition calculation.

6 Conclusions

We have demonstrated some limited success in exploiting the I-WAY using our NUMA programming model to perform molecular electronic structure calculations. The complete GA NUMA model was readily ported, fully distributed, to I-WAY. The bandwidths and latencies on I-WAY, as available to us, however, were inadequate to enable its effective use. An order of magnitude reduction in latency (to about 10 milli-seconds) and an increase of bandwidth by two magnitudes (to 10 MBytes/second) are required before the GA NUMA model is useful in electronic structure calculations without incorporating additional latency hiding mechanisms. Such mechanisms (e.g., asynchronous or split operations) greatly increase the complexity of the application code and increases the gulf between basic physics/ chemistry and the application. Since the goal of GA was to make the development of scalable parallel applications easier, increases in the complexity of the programming model are resisted unless absolutely necessary for performance. There are now over one million lines of code that make use of GAs, and NWChem alone exceeds 250,000 lines.

The mirrored array approach was very successful for the SCF algorithm, and given a modest increase in WAN bandwidth (to about 1 MB/s) it would be possible to make effective use of distributed supercomputing resources for this application. This does not, however, apply to other computational chemistry algorithms, such as multi-reference single- and double-excitation configuration interactions or coupled cluster methods, which have such great demands for high bisection bandwidth that it seems unlikely they could be satisfied by any WAN in the near future.

7 Acknowledgments

This work was performed under the auspices of the High Performance Computing and Communications Program of the Office of Computational and Technology Research, U.S. Department of Energy under contract DE-AC066-76RLO1830 with Battelle Memorial Institute which operates the Pacific Northwest National Laboratory. This work has made extensive use of software developed by the Molecular Science Software Group of the Molecular Science Research Center, PNNL, which is part of the Environmental and Molecular Science Laboratory. This research was performed in part using the Intel Paragon system operated by CalTech on behalf of the Concurrent Supercomputing Consortium, IBM SP-1 at Argonne National Laboratory, IBM SP-2 at Cornell Theory Center and Intel Paragon at San Diego Supercomputer Center. Access to these facilities was provided by the Pacific Northwest National Laboratory. We thank Heidi Lorenz-Wirzba and Jan Lindheim at CalTech, George Kremenek at SDSC, Gene Rackow at ANL and Daniel Ira Sverdlik from CTC for their help in arranging simultaneous access to supercomputer resources at their sites. We also thank Ian Foster for valuable discussions.

8 References

1. J. Nieplocha, R.J. Harrison, R.J. Littlefield, “Global Arrays: A Portable Shared-Memory Programming Model for Distributed-Memory Computers”, *Proc. Supercomputing’94.*
2. J. Nieplocha, R.J. Harrison, R. J. Littlefield, “The Global Array Programming Model for High Performance Scientific Computing”, *SIAM News*, August 1995.
3. J. Nieplocha, R.J. Harrison, R. J. Littlefield, “Global Arrays: A Non-Uniform-Memory-Access Programming Model for High-Performance Computers”, to appear in *The Journal of Supercomputing*.
4. N. Carriero and D. Gelernter, “Linda in Context”, *Communications of the ACM*, Vol 32, No. 4, 1989.
5. C. Amza et al., “TreadMarks: Shared Memory Computing on Networks of Workstations”, *IEEE Computer*, Vol. 29, No. 2, 1996.
6. I-WAY: information available at `http://www.iway.org`.
7. I. Foster, J. Geisler, B. Nickless, W. Smith, S. Tuecke, “Software Infrastructure for High-Performance Distributed Computing”, to appear in Proc. HPDC-5, 1996.
8. A. S. Grimshaw, W.A. Wulf, J.C. French, A.C. Weaver, P.F. Reynolds Jr.,”Legion: The Next Logical Step Toward a Natiowide Virtual Computer”, Technical Report CS-94-21, Dept. Computer Science, University of Virginia, 1994.
9. Anonymous Ftp: *ftp.pnl.gov:pub/global/* contains the latest public-domain release of the GA toolkit.
10. C. Koelbel et al., *The High Performance Fortran Handbook*, The MIT Press, Cambridge, MA, 1994.
11. Message Passing Interface Forum, *MPI: A Message-Passing Interface*, University of Tennessee, May 5, 1994.
12. R.J. Harrison, “Portable Tools and Applications for Parallel Computers,” *Int. J. Quant. Chem*., pp. 847-863, 40, 1991.
13. S. Pakin, M. Lauria, and A. Chien, “High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet”, *Proc. Supercomputing’95*.
14. M. Oguchi, H. Aida, and T. Saito, “A Proposal for a DSM Architecture Suitable for a Widely Distributed Environment and its Evaluation”, *Proc. 4-th IEEE Int. Symp. HPDC,* 1995.
15. D. E. Bernholdt et al., “Parallel Computational Chemistry Made Easier: The Development of NWChem,” *Int. J. Quantum Chem. Symp.*, vol. 29, 1995.
16. R.J. Harrison et al., ‘Towards High-Performance Computational Chemistry: A Scalable Self Consistent Field Calculations,’ *J. Comp. Chem.*, Vol. 17, No. 1, 1996.