Notes from Robert*:*

I've highlighted places where things (e.g., references) need adding.

Pay attention to using the correct document styles so that sections are correctly numbered and we can generate the table of contents automatically. To display the styles go into Options from the Tools menu, select the view tab, and set a non-zero style area width (0.8 inches is good).

To update the table of contents, put the insertion point somewhere in the body of the table, and press command-option-shift-u (Mac).

I have concentrated mostly on the structure and still need to add a few more words about applications, including a digested version of Steves text. I can do this on Sunday.

The proposal is already nearly too long and

- the background/intro need to shrink a little

– the preliminary investigation stuff needs expanding

– the research, design and methods needs to have more detail for GA including application justification (Robert). The HPF component of this needs spliting into 3 subsections (initial integration of the library, PCRC based stuff, tentative look at steering etc.).

– milestones for at least the first year.

– subcontract and consoortium arrangements?

Extending The Scientific Template Library With A Shared-Memory Non-Uniform Memory Access Programming Model Coupled With HPF Language Support

Abstract: In this proposal, we build upon a set of powerful and proven paradigms for managing the complex memory hierarchy in MPPs, and fully integrate that functionality into key areas of the SciTL toolkit, greatly extending its capabilities for applications requiring non-uniform memory access and for computer systems having non-uniform memory architecture (essentially all present and future MPPs). Our approach combines the strengths of the shared-memory-programming model with the scalability and performance of the distributed-memory model. What has emerged is a highly successful Global Arrays model (GA), whose one-sided communications model yields clean code with outstanding performance and scalability, as well as compatibility with standards and common linear algebra packages. Current applications span a wide range, including molecular dynamics, computational chemistry, graphics rendering, and security value forecasting. GA is used by major centers and applications development groups nationwide and worldwide. In the spirit of a standards-based approach, this toolkit will be integrated with High Performance Fortran, in a completely standards compliant and portable fashion, so that HPF based applications can readily exploit both data and task parallelism on NUMA architectures.

1. BACKGROUND AND SIGNIFICANCE 4

1.1. GLOBAL ARRAYS 6
1.1.1. Mirrored Arrays: metacomputing extensions of GA 8
1.2. DISK RESIDENT ARRAYS: EXTENSION OF GLOBAL ARRAYS TO SECONDARY STORAGE 8

2. PRELIMINARY STUDIES 9

3. RESEARCH, DESIGN AND METHODS 10

3.1. EXTENSIONS AND IMPROVEMENTS TO GA 10
3.1.1. Higher-dimensional arrays 10
3.1.2. Additional distribution types 10
3.1.3. Support for sparse data structures 11
3.1.4. Split-phase operations 11
3.2. INTEGRATION, INTEROPERABILITY OF GA WITH STANDARD TOOLS AND LIBRARIES 11
3.3. INTERFACE TO HPF AND LANGUAGE SUPPORT 12

4. SUBCONTRACT OR CONSORTIUM ARRANGEMENTS 14

5. LITERATURE CITED 14

6. BUDGET EXPLANATION 15

7. OTHER SUPPORT OF INVESTIGATORS 15

8. BIOGRAPHICAL SKETCHES 15

9. DESCRIPTION OF FACILITIES AND RESOURCES 15

10. DEAD TEXT 16

1. Background and Significance

All current and proposed MPPs (e.g., ASCI) have or will have nonuniform memory architecture. The hardware in a particular system may support a shared-memory or message-passing programming model, but these factors effect only the relative costs of local and remote accesses not the system's fundamental Non Uniform Memory Access (NUMA) characteristics. In future systems (including ASCI systems with their SMP nodes), both the number of memory levels and the cost (in processor cycles) of accessing deeper levels can be expected to increase **[Hennessy,Petaflops]**. Essential to achieving high efficiency on such systems are non-uniform memory-access (NUMA) aware programming tools.

Yet while the efficient management of memory hierarchies is fundamental to high performance in scientific computing, existing parallel languages and tools provide only limited support for this management task **[Velde, NATO]** and focus primarily on parallelism, and more specifically on the issues of control flow, communication structures, and load balancing. Memory-related issues such as data locality and variable cost of accessing data have been largely ignored with a flat memory model (see for instance references **[Fortran90,PCN,DataParallelC,P++Manual]**). However, this simple memory model hinders the understanding and tuning of parallel program performance. The variable cost of accessing data that reside at different memory levels cannot be ignored if we are to construct efficient parallel programs.

Systems that address locality management include High Performance Fortran (HPF) **[hpfbook],** the Message Passing Interface (MPI) **[MPI]**, Split-C **[SplitC],** and Compositional C++ (CC++) **[ccpp]**. HPF augments Fortran 90's flat memory model with data distribution directives, which can provide an implicit specification of locality. However, HPF can specify only data- parallel algorithms, and requires advanced compiler technology. MPI distinguishes local and remote memory explicitly, but requires that remote memory access be performed by means of explicit send calls with matching receives. Split-C and CC++ use global pointers to represent references to remote data.

Recognizing deficiencies of the existing programming models, we developed abstractions and programming tools, that can facilitate the explicit management of memory hierarchies by the programmer, and hence the efficient programming of scalable parallel computers. The abstractions comprise local arrays, global (distributed) arrays, and disk resident arrays located on secondary storage. The tools comprise the Global Arrays library **[1, 2, 3]**, which supports the transfer of data between local and global arrays, and its extension to high-performance secondary storage the Disk Resident Arrays library **[DRA]**, for transferring data between global and disk resident arrays.

The Global Array (GA) NUMA programming model and supporting toolkit has been proven as an effective and easy-to-use parallel programming tool for developing real large-scale applications **[references]**. These applications were able to obtain impressive speedups and scalability on massively-parallel computers thanks to support that GA and Disk-Resident Arrays (DRA) provide for high-performance programming in NUMA environments: control of data locality, recognition of the memory hierarchy, and explicit mechanisms for such transfers. GA is already optimized for shared-memory multiple processing SMP nodes of MPP computers and should match very well the architecture of the ASCI systems.

From the user perspective, GA does not require expert skills in parallel programming. The high-level array-oriented API is close to the matrix notation that describes mathematical formulation of physical problems. The scientists who are not parallel programming experts find GA operations on arrays much more familiar, easier to learn and use than the low-level message-passing model. Efficient and scalable parallel algorithms are also readily designed and implemented since the necessary blocking of accesses to data is fully supported by GA. These attributes of GA are the key to achieving very-high productivity levels in the development codes using GA such as NWChem **[12]**.

From the machine perspective, GA is a memory model and as such, it can deliver the highest possible performance (**Robert, this is a good point but it needs further explanation for some readers**). On a shared-memory machine, native shared-memory operations are used (e.g., CRAY T3E, SGI SMP). On a distributed-memory machine, the highest-performance message-passing tools are used. On a cluster of SMPs, shared-memory access is used wherever possible, and otherwise mechanisms to move data (e.g., message passing) are used. GA programs are fully portable between all of these architectures. Furthermore, the performance of such NUMA parallel programs are readily modeled in terms of the fundamental machine characteristics (e.g., latencies, bandwidths, cross-section bandwidths, etc.).

The GA differs from other tools in SciTL. Unlike P++ (and toolkits based on P++), that targets data-parallel execution model, GA supports both task- parallelism and data-parallelism. Unlike Tulip [**Tulip**] that uses polling for remote memory access on message-passing systems (such as IBM SP-2), GA implementation uses platform-specific low-level communication mechanisms that do not require polling. While polling is feasible in a run-time system supporting languages used to develop complete applications, it is not practical for many applications that use large standard libraries. This is because it would require modifications (inserting polling calls) to *all* of the application code including such libraries [**Bal**].

High Performance Fortran is a parallel language that is highly relevant to the DoE scientific community, where most application codes are still written in Fortran. It offers improved ease of use and generality compared to GA, as well as the advantages of being an accepted industry standard, vendor support, and a wider user-base. Unfortunately, HPF-1 is limited to data parallel operations, while many algorithms including computational chemistry and subsurface transport problems of contaminated nuclear production sites have both and task parallel components. A simple example would be the MP2 method. The first part of such a calculation is a “four-index transformation”, which is task parallel with an on-the-fly transposition of a distributed, sparse matrix. It involves a complex, nested, loop structure, which can be executed in parallel. Because the task durations vary widely, dynamic load balancing of this loop structure is necessary to obtain reasonable performance. The second major part of an MP2 calculation, the “energy evaluation”, is readily implemented in a data parallel fashion.

A revision of the HPF standard is now under consideration, which will include some support for task parallel operations, reduction variables, and more general data distributions. Unfortunately the designers of HPF-2 resisted introducing any non-deterministic features (needed to support dynamic load balancing) into the language, so the HPF-2 implementation of task parallel algorithms such as the MP2 four-index transformation would have to rely on the INDEPENDENT directive and on the compiler and runtime to provide high performance. Given the current philosophy of the language designers, and the complexity of compiler- or runtime-level support for dynamic load it is unlikely that acceptable performance could be obtained from mass-market implementations. The most feasible approach to supporting task and data-parallelism and taking advantage from the language support is by integrating GA with HPF in order to take advantage of the best features of both.

We propose two major activities:

- The integration of the Global Array NUMA programming toolkit into SciTL, including full interoperability with key existing SciTL components, and the extension of the Global Array toolkit to provide enhanced support for applications outside of the domains in which it is already extensively used. This work will be conducted primarily at PNNL with additional requirements analysis for the extensions to be performed with input and feedback from the SciTL user community.
- The standards based integration of GA and HPF. It will include development of GA library interfaces to HPF arrays to support task-parallelism in the framework of HPF programs. In the next phase, language support for such interfaces will be developed to replace explicit GA library calls with language constructs supported with extended HPF run-time system and GA library. This work to be primarily performed at NPAC in collaboration with PNNL.

In summary, GA would complement and extend capabilities of tools in the current SciTL by providing a NUMA-aware, portable, efficient and easy to use shared memory programming model fully compatible with MPI and interoperable with HPF.

In the following subsections, we provide some more background material about GA and DRA.

1.1. Global Arrays

The Global Arrays (GA) **[1, 2, 3]** combine the better features of message-passing and shared-memory, leading to both simple coding and efficient execution for a class of applications that operate on large distributed dense arrays, require dynamic load balancing or exhibit attendant unpredictable data reference patterns. The message-passing programming model is widely used because of its portability, yet some applications are too complex to code in it while also trying to maintain a balanced computation load and avoid redundant computations. The shared-memory programming model simplifies coding, but it is not portable and often provides little control **[Linda,Treadmarks,Midway,Orca]** over interprocessor data transfer costs. The Global Array toolkit provides an efficient and portable "shared-memory" programming interface for distributed and shared-memory systems. Unlike other shared-memory environments, the GA exposes to the programmer the nonuniform memory access (NUMA) characteristics of the high performance computers, acknowledges that access to remote data is slower than to local data, and it allows data locality to be explicitly specified and used. From the user perspective, a global array can be used as if it were stored in shared memory, except that explicit library calls are required to access it.

Development of Global Arrays has been motivated by the applications. The applications of GA include graphics rendering **[Heiland]**, financial calculations (security value forecasting), molecular dynamics **[Tjerk]** and most of all computational chemistry algorithms which calculate the electronic structure of molecules and other small or crystalline chemical systems. These calculations are used to predict many chemical properties that are not directly accessible by experiments, and play a dominant role in the number of supercomputer cycles currently used for computational chemistry. Some examples of applications that attain high parallel scalability and demonstrate the ease of use of GAs are included in the following references **[4,5,6,7,8]**.

GA were developed as part U.S. Federal High Performance Computing and Communications Initiative (HPCCI) Grand Challenge Applications project in computational chemistry, and turned out to be useful for research and production code development at PNNL's Environmental Molecular Science Laboratory (EMSL) and elsewhere in the U.S. (including Argonne National Laboratory, NCSA, Pittsburgh Supercomputing Center, Cornell Theory Center, San Diego Supercomputing Center, University of Illinois Urbana Champaign), and other countries including United Kingdom (Daresbury Laboratory), Austria (University of Vienna), Germany (Stuttgart University), Australia (Australian National University), Sweden (University of Lund) and Italy (University of Perugia). In last three years, many applications have adopted GAs, totaling nearly one-and-half million lines of code, including some 400,000 lines of new code. Nevertheless, a library-based parallel programming environment can at times be somewhat cumbersome relative to languages which directly support parallelism. Parallel languages are often able to push details away from the user's view, down to the compiler or runtime level, and offer more extensive error checking that is not possible with library approach.

Important to the successful employment of the SciTL in the DoE scientific community and beyond, is its compatibility with languages, libraries and tools that scientists use. Most scientific applications have been and continue to be developed in Fortran. It is important for the tools to support this language and follow its evolution. In particular, the High Performance Fortran (HPF) is a new industry standard for developing data parallel programs. While several present SciTL object-oriented programming toolkits implemented in C++ offer some interfaces to Fortran, their main strength can only be exploited within the C++ object-oriented framework. Although GA has been designed using object oriented principles it does not suffer from the performance problems associated with C++ implementations which could lead to the reported 50% performance loss as comparing to the Fortran implementation [**P++Manual**]. Some of these problems can be addressed by expression templates however the required sophisticated C++ compiler technology is not yet widely available. GA offers its full capabilities and equally good performance to both Fortran and C/C++ applications.

The GA was developed in light of emerging standards: the API follows Fortran 90 array notation, the library has been designed based on object oriented principles, GA is compatible with MPI, and interfaces to third-party parallel linear algebra packages such as ScaLAPACK or PLAPACK exist. 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. Extending the interoperability of GA with other tools and facilities is a subject of this proposal and should further enhance value of GA to DoE projects.

The GA are available variety of computer systems: 1) distributed-memory, message-passing parallel computers with interrupt-driven communications or Active Messages (IBM SP and Intel Paragon); 2) networked clusters of single- and multi-processor UNIX (Sun, IBM, SGI, DEC, HP) and Windows NT workstations; 3) shared-memory computers (Convex/HP SPP, SGI Origin-2000 and PowerChallenge); and 4) globally-addressable distributed-memory computers (Cray T3D/E). Several computer vendors are collaborating with PNNL researchers in developing optimized ports of GA. For instance, IBM have provided a new low-level API (LAPI) with the sole-purpose of supporting GA-like operations. This API was designed in close collaboration with the developers of GA in direct response to our requirements. Cray Research and PNNL have been collaborating on optimizations of GA and DRA libraries for the current and forthcoming Cray/SGI systems.

1.1.1. Mirrored Arrays: metacomputing extensions of GA

In recent years there has been increasing interest in metacomputing. Metacomputing environments comprise multiple supercomputers and other devices (mass storage systems, display devices) connected via local area networks (LANs) or wide area networks (WANs). They can potentially provide significant increases in the computational power accessible to a single application. From the memory hierarchy perspective, the metacomputing environment introduces one or more additional layers, with typically much higher latencies and lower bandwidth than in a supercomputer. Mirrored Arrays [**IWAY-HPDC**,**IWAY-TJS**] have been developed to address the performance characteristics of metacomputer in a manner similar to replicated shared memory [**RSM**]. Arrays are fully distributed within each machine, so the amount of data held by each processor increases by a factor roughly proportional to the number of networked supercomputers. Each supercomputer operates on its own mirrored array independently, and a special operation is provided for enforcing consistency of the different mirrored arrays. This primitive is a collective operation across all supercomputers and merges entire or user-specified sections of the mirrored arrays. Upon completion of this operation, all machines have identical copies of the specified array or array section. The result is that most GA communication occurs locally within supercomputers. Total network traffic is lower and average message size sent across the network is larger than in the fully distributed approach. This approach was very successful for a large application (Self-Consistent Field) that was able to effectively exploit multiple supercomputers (Intel Paragons and IBM SPs) connected with WANs.

1.2. Disk Resident Arrays: Extension of Global Arrays to Secondary Storage

Disk Resident Arrays (DRA) extend the GA model to another level in the storage hierarchy, namely, secondary storage. DRA introduces the concept of a disk resident array—a disk-based representation of an array—and provides functions for transferring blocks of data between global arrays and disk resident arrays. Hence, it allows programmers to access data located on disk via a simple interface expressed in terms of arrays rather than files. It extends the benefits of global arrays (in particular, the absence of complex index calculations and high-level array interface) to programs that operate on arrays that are too large to fit into memory.

By providing distinct interfaces for accessing objects located in main memory and on the disk, GA and DRA render visible the different levels of the memory hierarchy in which objects are stored. Hence, programs can take advantage of the performance characteristics associated with access to these levels. Recall that memory hierarchies consist of multiple levels, but are managed between two adjacent levels at a time [**Hennessy**]. For example, a page fault causes the transfer of a data block (page) to main memory while a cache miss transfers a cache line. Similarly, GA and DRA allow data transfer between adjacent levels of memory which combined with collective I/O strategy made this model very efficient [**DRA**,**Ying**] on the parallel I/O systems. DRA read and write operations can be applied both to entire arrays and to sections of arrays (disk and/or global arrays); in either case, they are collective and asynchronous. This collective I/O strategy has been adopted in many other projects for the efficiency reasons [**MPIO**, **PASSION**, **Panda**] as well. DRA asynchronous interface to its I/O operation permits applications to overlap time-consuming I/O with computation. The DRA has been successfully used in out-of-core matrix multiplication and two chemistry applications.

2. Preliminary Studies

The Global Array concept has been already thoroughly tested in the variety of application programs with some containing hundreds of thousands lines of code.

The functionality extensions (support for higher dimensional arrays and split-phase interface) that this proposal describes fit well within the implementation framework.

Some interfaces between GA and third-party linear algebra libraries (ScaLAPACK, SUMMA, PLAPACK) have been developed but only a limited number of them have been thoroughly tested in the application codes. Further work is required to make these interfaces complete (for example providing full BLAS-3 functionality through PLAPACK) and testing to make them robust and suitable for production code development.

The interface to HPF has been recently investigated and we determined that the language standard provides sufficient information necessary to retrieve distribution information and construct GA descriptor associated with a HPF array. The EXTRINSIC interface of HPF has been before used in other projects [**need example ref from Tom**] and was proven as a reliable mechanism for interfacing HPF applications with non-HPF software.

Our recent study at NPAC determined that the recent HPF-2 standard, which is nearing approval, would provide some support for task-parallelism in chemistry applications. Unfortunately, the new features are not sufficient to preserve and support dynamic load balancing schemes in these applications. The code restructuring involved in translation of SCF algorithm, implemented with GA, to HPF-2 required replacement of load-balancing, task-farming loop with deterministic parallel loop. It is not clear if optimizing HPF-2 compilers would be capable of performing automatic dynamic load balancing when translating such loops. Moreover, HPF compilers often concentrate on parallelizing simple loops, where communication can be reduced to cooperative or collective operations. On encountering a complex loop where communication cannot sensibly be reduced to such simple patterns, a compiler may well fall back on a default strategy of sequentializing the loop. These considerations indicate that Global-Array shared-memory operations will be needed even when HPF-2 compilers become available. In particular, coupling GA library with standard, vendor-supported HPF (2) compiler could offer a powerful tool combination capable of supporting task- and data-parallelism more effectively than any of the both facilities would be capable of alone.

(I have serious reservations about the GA-MPI-HPF performance study to talk about it here -- Jarek)

We could talk about experiments with Programmable Array Compiler here but I am not sure if it is appropriate given that Rosing is not involved explicitly in this proposal.

3. Research, Design and Methods

Although GA NUMA programming model and toolkit has been very successful for many applications, the value of GA to DOE the scientific community can be improved. The improvements can target the following areas:

- integration and interoperability with other tools and libraries including those already in the SciTL
- extensions of the GA functionality and programming interfaces
- interoperability with High Performance Fortran and language support for NUMA model

3.1. Extensions and improvements to GA

3.1.1. Higher-dimensional arrays

The current implementation of GA provides support only for 1- and 2-dimensional dense arrays distributed (uniformly or nonuniformly) in the block fashion. This is a significant limitation for computational fluid dynamics and other applications with 3-dimensional problem domain. These applications require support for arrays with three- to five-dimensions to represent the physical quantities such as temperature, velocity, pressure, etc in the 3-dimensional spatial domains. We propose to extend the GA to support arrays up to seven dimensions (the Fortran limit). This is a natural extension of the current toolkit capabilities and will fit well in the current implementation framework.

3.1.2. Additional distribution types

To improve static load balancing, many parallel linear algebra algorithms adopted block cyclic distribution. The current interfaces between GA and parallel linear algebra packages, such as ScaLAPACK, that use such distributions require data reorganization. For some applications, such reorganizations could be avoided if GA supported block cyclic distribution.

3.1.3. Support for sparse data structures

The current GA applications handle sparse data structures by explicitly packing data into one- or two-dimensional dense arrays. Support for sparse arrays in GA would greatly simplify implementation of such algorithms. The details of the sparse storage would be encapsulated in the GA object and access to the data simplified.

3.1.4. Split-phase operations

In order to improve latency tolerance in systems with multiple layers of NUMA memory hierarchy we propose to extend GA shared-memory communication operations with split-phase operations. This technique will allow overlapping computations with communications in the applications that are capable of using such mechanisms.

3.2. Integration, interoperability of GA with standard tools and libraries

Many applications that use GA require access to parallel linear algebra algorithms. We respond to this demand by developing interfaces to the state-of-the-art parallel libraries such as ScaLAPACK, SUMMA, PLAPACK, and PeIGS. The interfaces are integrated with the GA using a “black box” approach which hides from the user the complexity (and sometimes variation from one release to another) of the solver interfaces. The user only specifies global array descriptor for input and output and the appropriate GA interface handles the rest of the task. The data is rearranged to the format required by a particular third-party library, makes call to the appropriate solver, and returns the output data in the output global array. An example of such an interface is *ga_lu_solve* operation. It takes three arguments: character variable indicating if transpose operation is required, array handle for coefficient matrix, and array handle for the right-hand-side vector(s) which is overwritten by the solution upon exit. The subroutine in turn rearranges the arrays into the block cyclic distribution format, calls ScaLAPACK *pdgetrf* (factorization) and *pdgetrs* (forward/backward substitution) subroutines and returns the solution vector(s) in the global array. The subroutines *pdgetrf* and *pdgetrs* contain 8 and 13 arguments respectively as comparing to the three arguments in *ga_lu_solve*.

The GA furnishes interfaces to several parallel linear algebra algorithms available in ScaLAPACK. The following interfaces are available: LU and Cholesky factorizations, linear equation solvers, and matrix inverse. We propose providing additional interfaces to: eigensolver and Householder reduction to tridiagonal form.

Additional interfaces to PLAPACK **(we need it from George)**.

PETSc is set of numerical libraries and data structures developed for solution of partial differential equations. In addition to the distributed data structures, it contains iterative solvers, preconditioners, and unconstrained minimization algorithms. We propose to develop interface between GA and PETSc to make the numerical algorithms supported by PETSc to GA applications. By combining PETSc with Disk Resident Arrays in the GA framework, out-of-core solvers can be provided.

3.3. Interface to HPF and language support

Most scientific applications have been and continue to be developed in Fortran. It is important for the programming tools and libraries to support and interoperate with this language and follow its evolution. In particular, the High Performance Fortran (HPF) is a new industry standard for developing data parallel programs. Based on Fortran 90 syntax, it introduces a portable way to express explicitly parallelism in the code. Intuitive data mapping directives and the global name space employed in the HPF model greatly simplifies development of parallel applications. The efficiency of the resulting codes depends on the run-time support. One of the greatest strength of HPF is that it hides low-level details of the target machine architecture from the programmer. The actual parallel implementation is done by the compilation system, with the machine-specific run-time support taking responsibility of accessing remote (off-processor) data.

The HPF design and implementation reflects the current status of the compiler technology. In order to guarantee portability of the codes in a sense that the performance of the resulting codes is similar on machines with a similar computational power regardless their architecture, HPF-1 does not support directly some features which are necessary for some class of applications. In particular, there is no support for features required in dynamic, task-parallel programs, such as random access to regions of distributed arrays from within a MIMD parallel subroutine call-tree, and reduction into overlapping regions of distributed arrays. The HPF Forum recognized such features as too specific to be included in the Fortran standard, or found it premature to guarantee performance at this time.

By no means, it limits the scope of the language. HPF provides a portable way to incorporate libraries of parallel codes that are implemented in languages different from HPF, and following parallel programming paradigms different than that employed by HPF. The difference between the directly supported HPF features and the extrinsic procedures is that the latter are to be developed by the user (or a third party), and it is the user's responsibility to provide cross-platform portability. Development of parallel extrinsic procedures necessarily requires expertise in parallel computing. Therefore, we propose to develop HPF extrinsic libraries that become a part of the SciTL.

As the first step, we propose complementing HPF with the Global Arrays library. In particular, the GA non-collective one-sided communication on array sections would allow random access to regions of distributed arrays and reduction into overlapping regions of distributed arrays from within a MIMD parallel subroutine call-tree. Since GA has been designed to be as compatible as possible with HPF and GA API follows Fortran 90 and HPF array notation, most GA collective operations can be translated directly into single-line statements in HPF.

The concept of integration of stand-alone libraries, such as Global Arrays, with HPF can be pushed even further. One strategy is to provide support for user's extensions to the language. In this approach, a dedicated, programmable language processor translates user's directives into HPF statements. In particular, the language processor generates calling sequences for the user's run-time support implemented as HPF extrinsic library. We propose to develop language extensions that makes, from the user point of view, Global Arrays an integral part of the HPF run-time system accessible via HPF-like statements and/or directives. To implement the language processor to handle codes written in the extended HPF we are going to use the infrastructure developed by the Parallel Compiler Runtime Consortium [**PCRC**]. It includes HPF parser (and also C++ and Java) that transforms the source code into an intermediate representation (IR), and vice versa. The IR is implemented in an object-oriented fashion, and is modeled on Sage++ [Ganon]. A class library to perform operations on the IR is available as well. This infrastructure has been used by the PCRC to implement a HPF compiler [Li]. Here we propose to reimplement the ideas of the Programmable Array Compiler [**Rosing**] as the class library to perform operations on IR that correspond to the language extensions. The result is then reparsed to HPF, compiled using a commercial HPF compiler, and linked against the extrinsic Global Array library. The essential concept here is that the proposed language processor is programmable; that is, the functionality of the class library is described in a Language Definition File. Consequently, the same language processor can be used to integrate other run-time libraries with HPF just by modifying the language definition. Since the PCRC infrastructure is capable to process C++ and Java sources, this mechanism can be used to integrate other components of SciTL.

The following paragraphs need to shrink by about 2x. Moreover, it is not clear how to relate this work to SciTL and its existing components that target computational steering and application integration.

Another strategy that we will investigate for relevance SciTL involves dynamical software integration using interpreted protocols. The fundamental difference between this approach and that described above is that here all components of the application can be compiled independently of each other, and dynamically linked in the fly at the runtime. The base of such a system is a HPF interpreter. A prototype of it has been developed at NPAC ***[AkarsuFoxHaupt]****. The heart of the system is a HPF server implemented through the extrinsic interface. The HPF server accepts commands from a client implemented as a Java applet. The user uses the applet as a GUI. The user can interrupt, suspend and resume, execution of the precompiled HPF application at any point (including breakpoints and stepping one HPF statement at a time), and request a specific action from the HPF server. Currently, the repertoire of the user commands comprise:*

- *request for data (for debugging or visualizations),*
- *setting values of any variable in the precompiled HPF code (steering),*
- *interpretation of valid HPF statements entered from the keyboard or read from a file (HPF interpreter),*
- *request for dynamical linking and executing of precompiled shared objects (dynamical software integration).*

The control over the execution of the precompiled code, that is, dynamical switching between compiled and interpreted mode is facilitated by instrumenting the original HPF source code. This is done automatically using the PCRC infrastructure (HPF parser/reparser and the class library to transform the Intermediate Representation). All features of this integrated HPF compiler and interpreter environment are of value for the SciTL toolkit.

It provides support for interactive (and collaborative) visualizations, as well as run-time data analysis and steering; it serves as an powerful HPF debugger (particularly when combined with visualizations of distributed data objects, and HPF interpreter); it allows for rapid prototyping (capability of data manipulation in the interpreted mode and dynamical linking with precompiled *modules). Here we propose to use this system for dynamical integration of a HPF application and the Global Array library. Once the operational system is demonstrated for HPF, we will extend it to support codes implemented in Java and C++, and in particular, dynamical integration of existing libraries written in Fortran with object oriented applications.*

4. Subcontract or Consortium Arrangements

5. Literature Cited

[1] J. Nieplocha, R. Harrison, and R. Littlefield. Global Arrays: A portable ‘shared-memory” programming model for distributed memory computers. In Proc. Supercomputing 1994, pages 340-349. IEEE Computer Society Press, 1994.

[2] J. Nieplocha, R. J. Harrison, R. J. Littlefield. The Global Array programming model for high performance scientific computing. SIAM News, 28(7):12-14, 1995.

[3] J. Nieplocha, R. J. Harrison, R. J. Littlefield. The Global Array: A Non-Uniform-Memory-Access Programming Model For High-Performance Computers, The Journal of Supercomputing, 10:197-220, 1996.

[4] R. Harrison, M. Guest, R. Kendall, D. Bernholdt, A. Wong, M. Stave, J. Anchell, A. Hess, R. Littlefield, G. Fann, J. Nieplocha, G. Thomas, D. Elwood, J. Tilson, R. Shepard, A. Wagner, I. Foster, E. Lusk, and R. Stevens. Toward high-performance computational chemistry: II. A scalable self-consistent field program. Journal of Computational Chemistry, 17(1):124-132, 1996.

[5] D. Bernholdt and R. Harrison. Large-scale correlated electronic structure calculations: The RI-MP2 method on parallel computers. Chemical Physics Letters, 250:477-484, 1996.

[6] H. Dachsel, H. Lischka, R. Shepard, J. Nieplocha, and R. Harrison. A massively parallel multireference configuration interaction program - the parallel COLUMBUS program. Journal of Chemical Physics, 1997.

[7] A. Wong, R. Harrison, and A. Rendell. Parallel direct four-index transformations. Theoretica Chimica Acta, 1996.

[8] D. Bernholdt and R. Harrison. Orbital-invariant second-order many-body perturbation theory on parallel computers: An approach for large molecules. Journal of Chemical Physics, 102(24):9582-9589, 1995.

[9] J. Nieplocha and R. J. Harrison, Shared Memory NUMA Programming on I-WAY, Proc. of IEEE High Perform. Distr. Comp. HPDC-5, 1996. (HPDC-5 Best Paper award)

[10] Jarek Nieplocha and Ian Foster, Disk Resident Arrays: An Array Oriented Library for High Performance IO, Proc. Frontiers on Massively Parallel Comp. Frontiers'96, 1996.

[11] H. Fruechtl, R. A. Kendall, R. J. Harrison, and K. G. Dyall. A scalable implementation of RI-SCF on parallel computers. Intl J. Quantum Chem. Symp., 1997 (in press).

[12] D. Bernholdt et al. Parallel computational chemistry made easier: The development of NWChem. Intl J. Quantum Chem. Symp., 29:475-483, 1995.

[Tulip] P. Beckman, D. Gannon, Tulip: A portable run-time system for object-oriented systems, Proc. IPPSi'96, pp. 532:536, 1996.

[Bal] K. Langendoen, J. Romein, R. Bhoedjang, H. Bal, Integrating polling, interrupts and thread management, Proc. Frontiers'96, pp13-22, 1996.

[Rosing] Matt Rosing's Programmable Array Compiler

[P++Manual] D. Quinlan, A++/P++ Manual, vesion 0.6.8, technical report LA-CC-96-1, Los Alamos Nat. Laboratory. 1996.

6. Budget Explanation

7. Other Support of Investigators

HPCCI project at PNNL

8. Biographical Sketches

9. Description of Facilities and Resources

This work will be performed using computing facilities available at PNNL, NPAC and other DoE supercomputing sites. The computing resources at PNNL include: 472-node IBM SP with Power-2 Super nodes, SMP multiprocessor SGI systems (Origin-2000 and PowerChallenge) and numerous workstations. A smaller, next generation IBM SP system with SMP nodes will be available at PNNL in 1998. The full suite of parallel software, including HPF compiler, supported by IBM is provided. In addition, through other arrangements, access to Cray T3E is available through NERSC.

10. Dead Text

In attempts to exploit massively parallel computation, the critical effort has typically been developing the applications software needed to take full advantage of available hardware capabilities. In fact, many of the successes of the past have been the result of heroic software development efforts, that could not be cost effectively duplicated across the spectrum of problems of importance to DOE and the nation. Yet, to achieve the promise of massively parallel computing, we need to surmount this problem with better tools and more scalable and extensible libraries. Large collections of processors with their high performance communications infrastructure are inherently more complex to program and use than smaller systems, so achieving good efficiency requires considerable attention to detail, matching the characteristics of the algorithm to the capabilities of the system. A number of approaches have been developed to hide some of the complexity of the MPP system, and no single approach is optimal for every application. The most successful approaches employ language constructs or building blocks that expose enough of the system architecture to permit library and application developers to obtain good performance, while hiding mundane bookkeeping. The best approaches also enable portability across system manufacturers and system generations by encapsulating details of the underlying hardware.