Self-adapting Numerical Software for Next Generation Applications

Lapack Working Note 157, ICL-UT-02-07[*]

Jack Dongarra, Victor Eijkhout[†]

August 2002

Abstract

The challenge for the development of next generation software is the successful management of the complex grid environment while delivering to the scientist the full power of flexible compositions of the available algorithmic alternatives. *Self-Adapting Numerical Software (*SANS*)* systems are intended to meet this significant challenge.

A SANS system comprises intelligent next generation numerical software that domain scientists – with disparate levels of knowledge of algorithmic and programmatic complexities of the underlying numerical software – can use to easily express and efficiently solve their problem. The components of a SANS system are:

- A SANS *agent* with:
  - An *intelligent component* that automates method selection based on data, algorithm and system attributes.
  - A *system component* that provides intelligent management of and access to the computational grid.
  - A *history database* that records relevant information generated by the intelligent component and maintains past performance data of the interaction (e.g., algorithmic, hardware specific, etc.) between SANS components.
- A simple *scripting language* that allows a structured multilayered implementation of the SANS while ensuring portability and extensibility of the user interface and underlying libraries.
- An XML/CCA-based *vocabulary of metadata* to describe behavioural properties of both data and algorithms.
- Prototype *libraries* that automate the process of architecture-dependent tuning to optimize performance on different platforms.

A SANS system can dramatically improve the ability of computational scientists to model complex, interdisciplinary phenomena with maximum efficiency and a minimum of extra-domain expertise. SANS innovations (and their generalizations) will provide to the scientific and engineering community a dynamic computational environment in which the most effective library components are automatically selected based on the problem characteristics, data attributes, and the state of the grid.

1 Introduction

As modeling, simulation, and data intensive computing become staples of scientific life across nearly every domain and discipline, the difficulties associated with scientific computing are becoming more acute for the broad rank and file of scientists and engineers. While access to necessary computing and information technology has improved dramatically over the past decade, the efficient application of scientific computing techniques still requires levels of specialized knowledge in numerical analysis, computer architectures, and programming languages that many working researchers do not have the time, the energy, or the inclination to acquire.

[*] This work was funded in part by the Los Alamos Computer Science Institute through the subcontract # R71700J-29200099 from Rice University.

[†] Innovative Computing Laboratory, University of Tennessee, Knoxville TN 37996, USA

The classic response to this situation, introduced over three decades ago, was to encode the requisite mathematical, algorithmic and programming expertise into libraries that could be easily reused by a broad spectrum of domain scientists. In recent times, however, the combination of a proliferation in libraries and the availability of a wide variety of computing platforms, including varieties of parallel platforms, have made it especially hard to choose the correct solution methodology for scientific problems. The advent of new grid-based approaches to computing only exacerbates this situation. Since the difference in performance between an optimal choice of algorithm and hardware, and a less than optimal one, can span orders of magnitude, it is unfortunate that selecting the right solution strategy requires specialized knowledge of both numerical analysis and of computing platform characteristics.

What is needed now, therefore, is a way of guiding the user through the maze of different libraries so that the best software/hardware combination is picked automatically.

We propose to deal with this problem by creating *Self-adapting Numerical Software (*SANS*)* systems that not only meet the challenges of scientific computing today, but are designed to smoothly track the state of the art in scientific computing tomorrow.

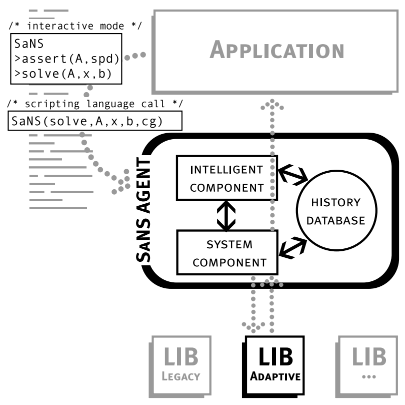
In this paper we will describe the basic ideas of SANS system, and we will sketch their realization in systems for linear equation solving, eigenvalue computations, and information retrieval, although the ideas and innovations they embody will generalize to a wide range of other operations. Like the best traditional libraries, such system can operate as ”black box” software, able to be used with complete confidence by domain scientists without requiring them to know the algorithmic and programmatic complexities it encapsulates. But in order to self-adapt to maximize their effectiveness for the user, SANS must encapsulate far more intelligence than standard libraries have aspired to. The work described below will make it possible to produce a SANS system that incorporates the following elements:

- An *intelligent component* that includes an *automated data analyzer* to uncover necessary information about logical and numerical structure of the user's data, *a data model* for expressing this information as structured metadata, and a *self-adapting decision engine* that can combine this problem metadata with other information (e.g. about past performance of the system) in order to choose the best library and algorithmic strategy for solving the current problem at hand;
- A *history database* that not only records all the information that the intelligent component creates or acquires, but also all the data (e.g., algorithm, hardware, or performance related) that each interaction with a numerical routine produces;
- A *system component* that provides the interface to the available computational resources (whether on a desktop, in a cluster or on a Grid), combining the decision of the intelligent component with both historical information and its own knowledge of available resources in order to schedule the given problem for execution;
- A *scripting language* that generalizes the decision procedure that the SaNS follows and enables scientific programmers to easily make use of it; and
- One or more *prototype libraries*, for instance for sparse matrix computations, that accept information about the structure of the user's data in order to optimize for execution on different hardware platforms.

A SANS system can dramatically improve the ability of computational scientists to solve an important range of challenging problems with maximum efficiency and a minimum of extra-domain expertise. Moreover, as these SANS innovations are generalized, they will give the scientific computing community a dynamic computational environment in which the most effective library components are automatically selected and integrated on the basis of the particular problem posed, the data structures available, and the state (and behavior) of the computational environment at run-time. The SANS metadata scheme allows us to capture this self-adaptive process in databases that will provide an indispensable resource for future library developers.

The fact that current numerical libraries require detailed, specialized knowledge that most potential users are unlikely to have is a limitation on their usability that is becoming increasingly acute. With SANS it will become possible to endow legacy libraries with computational intelligence, and to develop next generation libraries that make it easier for users to realize the full potential of current day computational environments. This investigation into the potential for self-adaptation in scientific software libraries will lay the foundation necessary to meet the incredibly challenging demands of computational science over the next decade.

2 Outline of the structure of self-adaptive software



A Self-adaptive Numerical Software system has three components: an Agent (called the SANS Agent), a Scripting Language, and the underlying Adaptable Libraries. The SANS Agent is the software that accepts the data from the user application in order to pass it to a chosen underlying library. These libraries can be of a traditional type, but more interestingly they can adapt themselves to the available hardware, setting algorithm implementation parameters such that performance is optimized with respect to machine characteristics [48, 12].

The scripting language provides the interface between the user and the intelligent agent. With this scripting language we turn what used to be a mere call – or series of calls – to a library into a script that can convey contextual information to the intelligent system, which may use this information to make a more informed choice of software for solving the user's problem. Through the use of keywords and control structures in the scripting language we make it possible for the user to pass various degrees of information about the problem to be solved. In the cases where the user passes little information, the intelligent agent uses heuristics to uncover as much of this information as is possible.

The SANS agent consists of three parts. The Intelligent Component is that part of the software that uses encoded knowledge of numerical analysis to analyze the data (section 3.1). The System Component (section 3.4) knows about hardware, both in general terms and regarding the current state of the network and available resources. These two components engage in a dialogue to determine the best algorithm and platform for solving a given user problem. The third component is the History Database (section 3.5) where performance data regarding problems solved is stored. This stored knowledge is then used by the intelligent and network components to inform their decisions, and possibly tune their decision-making process.

SANS systems can various usage modes, depending for instance on the level of expertise of the application user, and on the way the system is called from the application code.

- For a non-expert user, a SANS system acts like an expert system, fully taking the burden of finding the best solver off the user's hands. In this scenario, the user knows little or nothing about the problem – or is perhaps unable to formulate and pass on such information – and leaves it up to the intelligent software to analyze structural and numerical properties of the problem data.
- Users willing and able to supply information about their problem data can benefit from a SANS system in two ways. Firstly, decisions that are made heuristically by the system in expert mode can now be put on firmer ground by the system interrogating the user or the user passing on the information in the calling script. Secondly, users themselves can search for appropriate solution methods by using the system in 'testbed' mode.

- Finally, expert users, who know by what method they want to solve their problem, can benefit from a SANS system in that it offers a simplified and unified interface to the underlying libraries. Even then, the system offer advantages over the straightforward use of existing libraries in that it can supply primitives that are optimized for the available hardware, and indeed, choose the best available hardware.

Users willing to supply metadata or algorithm choices to the system will be able to do so in two ways. First, we will develop a GUI that interrogates the user. However, if the library call appears in an inner loop of the code, this would be inappropriate and the user can resort to passing data and options through the calling script.

3 The SaNS Agent

3.1 The Intelligent Component

The Intelligent Component of a SANS system is the software that accepts the user data and performs a numerical and structural analysis on it to determine what feasible algorithms and data structures for the user problem are. We allow the users to annotate their problem data with 'metadata' (section 3.6), but in the most general case the Intelligent Component will do this by means of automated analysis (section 3.2). Moreover, any rules used in analyzing the user data and determining solution strategies are subject to tuning (section 3.3) based on performance data gained from solving the problems. Below we present each of these aspects of the SANS agent in turn, including detailed examples of how the components could engage with and be used by our driver applications.

3.2 Automated analysis of problem data

Users making a request of a SANS system pass to it both data and an operation to be performed on the data. The data can be stored in any of a number of formats, and the intended operation can be expressed in a very global sense ('solve this linear system') or with more detail ('solve this system by an iterative method, using an additive Schwarz preconditioner'). The fewer such details the user specifies, the more the SANS will have to determine the appropriate algorithm, computational kernels, and computing platform. This determination can be done with user guidance, or fully automated. Thus, a major component of a SANS is an intelligence component that performs various tests to determine the nature of the input data, and makes choices accordingly.

Some of these tests are simple and give an unambiguous answer ('is this matrix symmetric'), others are simple but have an answer that involves a tuning parameter ('is this matrix sparse'); still others are not simple at all but may involve considerable computation ('is this matrix positive definite'). For the tests with an answer on a continuous scale, the appropriateness of certain algorithms as a function of the tested value can only be preprogrammed to a limited extent. Here the self-adaptivity of the system comes into play: the intelligence component will consult the history database of previous runs in judging the match between algorithms and test values, and after the problem has been solved, data reflecting this run will be added to the database. Illustrations of the kinds of analyses the SANS agent must do include the following:

3.3 Self-Tuning Rules for Software Adaptation

The Intelligent Component can be characterized as self-tuning in the following sense: The automated analysis of problem data concerns both questions that can be settled quickly and decisively, and ones that can not be settled decisively, or only at prohibitive cost. For the latter category we will use heuristic algorithms. Such algorithms typically involve a weighing of options, that is, parameters that need to be tuned over time by the experience gained from problem runs. Since we record performance data in the history database (section 3.5) of the SANS Agent, we have a mechanism to provide feedback for the adaptation of the analysis rules used by the Intelligent Component, thus leading to a gradual increase in its intelligence.

3.4 The System Component

The System Component of the SANS agent for the managing the different available computation resources (hardware and software), which in today's environment can range from a single workstation, to a cluster, to a Computational Grid. This means that after the intelligent component has analyzed the user's data regarding its structural and numerical properties the system component will take the user data, the metadata generated by the intelligent component, and the recommendations regarding algorithms it has made, and based on its knowledge of available resources farm the problem out to a chosen computational server and a software library implemented on that server. Eventually the results are returned to the user. Empirical data is also extracted from the run and inserted into the database; see section 3.5.

However, this process is not a one-way street. The intelligent component and system component can actually engage in a dialogue as they weigh preferred algorithms against, for instance, network conditions that would make the available implementation of the preferred algorithm less computationally feasible.

One aspect of the metadata passed by the intelligent component is the analysis of the structure of the data. Based on this, especially in the case of sparse data, the system component can choose an optimized kernel to use. Such kernels, for instance for the matrix-vector product or an ILU solve, can often be passed as user-supplied operations to existing libraries. In this way, the system component can adapt the execution to the structure of the data. Conversely, the system component can decide on a variant of a certain kernel purely based on its availability on the hardware supplied, and decide to rearrange the user data from one storage format to another. This may then require re-analysis of the shuffled data by the intelligent component.

If sufficient resources are available, and certain chosen algorithms have a degree of unpredictability in their behavior, such as the number of iterations of iterative methods, the system component can decide on a poly-algorithmic approach [8], where the same problem is to be solved with very different software libraries, and on different machines. In case on solution method was marked by the intelligent component as to be preferred, the other solver will serve as a 'backup' for the main one. In some cases they may have equal standing; as the solution becomes available from one computational server, the system component will tell the other servers to abort the task.

Part of the System Component is scheduling operations and querying network resources. In building self-adaptive software we will not actually engage in these aspects of systems programming; for that we will leverage our earlier – and ongoing – work in Netsolve [16, 18, 17].

3.5 History database

Self-adaptivity of our agent-based numerical library to meet the needs of diverse users on any computational environment requires a knowledge base of performance data to make intelligent choices for algorithms, data structures, architectures, and programming languages. Each interaction with a numerical routine produces valuable data ranging from iteration counts (algorithm level) to cache hits (hardware level). The middleware designed to interface between the user application and the computation grid must be able to exploit all 'known' data for each user request. Based on the problem posed by the user, the available data structures, and the state of the computational environment, the system would select the 'best' software library component(s) for solving the current problem. Categorization of performance and problem 'metadata' into relational databases should be based on the application domain as well as the state of all networks and processors defining the Grid.

Maintaining a dynamic (constantly updated) database of problems solved along with the state of the computational grid and library components used to obtain the solution can facilitate dynamic problem solving for numerous applications and also provide insights into future library component designs. In many cases, not one algorithm or approach may be viable as grid conditions change (e.g., network traffic increases during the workday or processor failures) so that the library may dynamically create a 'polyalgorithm' approach. Detecting slow convergence or a stall of any current module would be stored in both contexts: the problem being solved and the computational environment. In the course of solving the user's problem, several solution strategies (e.g., more than one preconditioner for an iterative solver for sparse linear systems of equations) may be used and recorded into the database. Utilizing past and present performance metadata facilitates dynamic (customized) solutions to large-scale problems, which cannot be generated from current numerical software libraries.

The system needs to handle various data aspects. Original user data can be annotated with metadata, or metadata can be uncovered, and after solving the problem, there is performance data for both the specific problem and the characteristics of the hardware and of software kernels used. We will store all this information in our history database to inform future decision making processes. Mostly metadata and performance data regarding the user problems are stored, but space permitting it would even be possible to store the user data itself. For information encoding we will use XML, and to ensure standardization and interoperability between SANS agents at different sites we will rely on *Repository in a Box (RIB)* [13, 14], which is software technology we have developed for creating interoperable metadata repositories that is now widely deployed in the high performance computing community.

3.6 Metadata for Self-Adapting Numerical Libraries

The operations typically performed by traditional libraries is on data that has been abstracted from the original problem. For instance, one may solve a linear system in order to solve a system of ODEs. However, the fact that the linear system comes from ODEs is lost once the library receives the matrix and right hand side. Our intelligent library will have the facility of annotating problem data with information that is typically lost, but which may inform a decision making process. First, we will implement the facility for the user to pass such metadata explicitly in order to guide the intelligent library. More importantly, however, we will design heuristics for uncovering such lost data, taking the burden completely off the user.

4 The Scripting Language

The primary interaction between the application developer and the SANS agent is by means of a 'scripting' language. Additionally, a graphical user interface will be constructed to query the user to either interactively expand script language statements or to exchange 'metadata' regarding matrix/system attributes.

The scripting language is also the layer that provides the coupling between the library interface and the SANS agent. The role of this language limits its features to provide essential functionality that cannot be duplicated with ease in other components. This language provides simple control mechanisms and data types; a script in this language would be interpreted (or alternatively compiled) to produce a program of calls to the underlying libraries.

In domain-specific computational science applications the notion of a scripting language has been typically implicit. In these disparate development environments the tendency is to express applications as scripts to invoke application-specific libraries. Our effort differs from these approaches in that our goal is the development of an explicit, prototype scripting language. This language must be general enough to cater to applications that require a broad class of matrix based computations. However, at the same time, we propose to develop a simple scripting language so that interpreting scripts into underlying library function calls adds little overhead. The goal is to provide a lightweight flexible mechanism for translating application needs and attributes into suitable smart library calls within the context of agent-based grid-computing. Our effort concerns:

- Specifying syntax and semantics so that the scripting language is suitable for a wide class of applications requiring matrix computations.
- Providing ways to express attributes of the application and the manner of use; for example, the application should be able to specify whether or not the linear solver is being used within an implicit time-stepping process (such attributes can be used for method selection to improve performance).
- Providing a basic set of control structures so that either the application developer or the intelligent agent can specify a 'poly-algorithmic' approach [8].
- Providing access to library components with a sufficient degree of detail so that the scripting language can be used for library extension. For example, to prototype a new poly-algorithmic, smart library method composed of existing library functions.

As mentioned earlier, our scripting language will not be a classical programming language; rather, it is one designed to inexpensively perform a weighing of possible options to compose an adaptive solution as a 'poly-algorithm.' A script in this language will be interpreted by the SANS agent to which the application connects. Additionally, the agent may use pre-defined scripts for method composition. Finally, the agent may use scripts to dynamically compose a 'poly-algorithm' solution based on past solution history, changes in the run-time environment, etc.

We anticipate providing simple primitives for if-then-else conditionals, case-based testing, and standard loop-constructs. In addition, the scripting language will contain keywords to specify solution methods, problem attributes, data formats, etc. In short we will provide a keyword-rich language with simple control structures that can be used by both the application developer and the SANS agent to craft a 'custom' solution from the optimized implementations available in the underlying library layer.

We will start with a class of 'operation' keywords to indicate a basic set of major matrix operations, for example, linear-solution, LU/QR/Cholesky factorizations, and sparse matrix reorderings. Additionally, we expect the meaning of these high-level operations to be specified in greater detail by using a class of keywords we call 'modifiers.' For example, for the linear solution operation, a modifier might the type of solver, say 'direct' or 'iterative.' The modifiers are in a sense hierarchical, if an 'iterative' method is chosen, the application could further specify 'conjugate gradients' with 'pc ic0,' i.e., a level-zero incomplete Cholesky preconditioner. These modifiers are optional; if the application does not select a type of solver (direct or iterative), it is the responsibility of the agent to select one. In such cases, the agent may simply expand predefined scripts for each high-level method.

To intelligently perform the kind of expansion discussed in the last paragraph the agent would need to know some problem/matrix attributes. We anticipate providing a class of 'assertion' keywords that can be used within the application to specify properties such as whether the matrix is well-conditioned, or if it has full-rank or not, etc. In the event of automatic method selection, the agent would use these attributes, in conjunction with the desired high-level operation, to select predefined scripts specifying suitable 'poly-algorithm' solution methods. The class of 'assertion' keywords will also contain specific terms to describe the type of data format for the input matrix and other assorted information.

We expect a last class of keywords to refer to the computational environment—these keywords will be used to specify system specific details, e.g., the number of processors. We also anticipate allowing 'directives' which can control the action of the agent—these directives could be used to enable or disable (within script segments) automatic 'poly-algorithm' expansions by the agent.

This scripting language approach allows the user to request a service from the agent at various levels of detail. It may well be that the experienced user will have additional knowledge about the attributes of the problem which could be used to provide a better solution. In these instances, the SANS agent would use a GUI to interact with the user. Such interaction would be in the form of a sequence of query/response pairs aimed at better defining application needs and problem attributes.

We would like to reiterate that we are developing a simple scripting language primarily because it allows a structured multilayered implementation of the SANS while ensuring portability and extensibility of the user interface and underlying libraries. Existing scripting languages either lack adequate abstraction or are difficult to implement and support due to their generality; for example, Perl is not object-oriented while the Python thread module can have interactions with the module for signal handling.

5 Libraries

5.1 Exemplar libraries

We propose to exploit the automation of the process of architecture-dependent tuning of numerical kernels, replacing the current hand-tuning process with a semiautomated search procedure that our SANS agent can use. Current limited prototypes for dense matrix-multiplication (ATLAS [48] and PHIPAC [12], both involving researchers of the current proposal), sparse matrix-vector-multiplication (Sparsity [33, 32], also involving some of the current authors) and FFTs (FFTW [26, 25]) show that we can frequently do as well as or even better than hand-tuned vendor code on the kernels attempted. The importance of *automatically* doing about as well as hand-tuned code is critical: It is not just a matter of saving some time for a few programmers writing the BLAS (Basic Linear Algebra Subroutines [40, 22, 21]) at a few companies, it is a matter of making such code easily accessible. For example, The Mathworks did not include LAPACK [5] into Matlab for many years because of the difficulty of assuring the availability of fast BLAS. Only when ATLAS made fast BLAS generally available did the Mathworks put LAPACK into Matlab. Making licensing, installation and tuning issues transparent for average users is even more important.

Current projects use a hand-written *search-directed code generator* (SDCG) to produce many different C implementations of, say, matrix-multiplication, which are all run on each architecture, and the fastest one selected. Simple performance models are used to limit the search space of implementations to generate and time. Since C is generated very machine specific optimizations like instruction selection can be left to the compiler. We propose to extend this approach to a much wider range of computational kernels by using compiler technology to automate the production of these SDCGs.

Especially of interest are sparse kernels [34, 32]. Sparse matrix algorithms tend to run much more slowly than their dense matrix counterparts. For example, on a 250 MHz Ultrasparc II, a typical sparse matrix vector multiply implementation applied to a document retrieval matrix runs at less than 10 MFlops/s, compared to 100 MFlops/s for a vendor-supplied dense matrix-vector multiplication routine, and 400 MFlops/s for matrix-matrix. Major reasons for this performance difference include indirect access to the matrix and poor data locality in access to the source vector x in the sparse case.

However, our current optimizations can speed sparse-matrix-vector multiplication up over five fold [34]. It is remarkable how the type of optimization depends on the matrix structure: For a document retrieval matrix, only by combining both cache blocking and multiplying multiple vectors simultaneously do we get a five fold speed; cache blocking alone yields a speedup of 2.5, and multiple vectors alone yield no speedup. In contrast, for a more structured matrix arising from face recognition, the unoptimized performance is about 40 Mflops, cache blocking yields no speedup, and using multiple vectors yields a five fold speedup to 200 Mflops. In other words, the correct choice of optimization, and the performance, depends intimately on the matrix structure.

We will use these techniques as well as others and incorporate them into SANS: *register blocking*, where a sparse matrix is reorganized into fixed-size blocks that may contain some zeros, and *matrix reordering*, where the order of rows and columns are changed to reduce cache misses and memory coherence traffic on an SMP [44].

5.2 Optimization modes

We propose to build a search-directed code generator that contains two major components: a transformation engine to generate a set of candidate versions of the algorithmic kernel, and a search engine that is used to run and evaluate the various candidates to choose the best one for a given machine.

We give several scenarios that our system will need to support, as motivated by examples discussed earlier. The standard tradeoff is that the longer one waits to attempt an optimization, the more information is available about the problem, but the less time one has to do it.

Completely off-line optimization This scenario is used in PHIPAC and ATLAS, and it works well for the the dense BLAS because the computational pattern is nearly independent of the input: matrix multiplication does the same sequence of operations independent of the values stored in the matrices. Because optimization can be done offline, one can in principle take an arbitrary amount of time searching over many possible implementations for the best one on a given micro-architecture.

Hybrid off-line/run-time optimization This is the scenario in which Sparsity can work (it can be run completely off-line as well). In both cases, some kernel building blocks are assembled off-line, such as matrix-vector or matrix-matrix multiply kernels for very small dimensions. Then at run time the actual problem instance is used to choose an algorithm. For Sparsity, the problem instance is described by the sparsity pattern of the matrix A. Any significant processing of this will also overwhelm the cost of a single matrix-vector multiplication, so only when many are to be performed is optimization worthwhile.

Completely Run-time optimization This is the scenario in which just-in-time (JIT) compilers work [15, 35, 1], as well as the inspector-executor model [20] and other dynamic compilation systems [24, 6, 23]. In these cases, one has essentially all information about a problem instance, but the least time available to optimize. A standard example of inspector-executor is to examine the sparsity pattern of a sparse matrix on a parallel machine at run, and automatically run a graph partitioner like Parmetis [36] to redistribute it to accelerate subsequent matrix-vector multiplications.

Feedback Directed Compilation This involves running the program, collecting profile and other information [29, 19, 7, 3] and recompiling with this information. We will make use of this mode through the explicit incorporation of a database of performance history information.

5.3 Applying SANS to Sparse Linear Algebra

In dense linear algebra, efficient use of the memory hierarchy is probably the single most important factor in achieving high performance. Utilizing the optimal register blockings, pipeline lengths, loop unrollings and similar floating point optimizations yield significant performance gains only after the CPU to memory bottleneck is first addressed by optimizing cache use. Dense matrices are easily stored in one or two dimensional arrays; once the storage is fixed in this way, cache optimization can be done statically during installation.

For sparse linear algebra, however, the data storage and structure vary widely. In order to achieve high performance for sparse linear algebra operations, it is necessary to exploit the sparsity structure in the original data.

There are essentially two approaches to recognizing and exploiting this sparsity structure. The most tractable solution involves presenting to the user many different specialized storage formats, and allowing the user to choose the format that best exploits his/her sparsity structure in order to maximize cache reuse. This approach is widely used today, as in [31, 11, 37, 45, 47, 28]. The advantage of this solution is that the input format is fixed and assumed to be appropriate to the data structure, just as with dense BLAS. Choosing one of the more optimizable data structures (such as one of the block compressed storage schemes), should allow us to directly leverage the Level 2 BLAS kernels developed for dense Linear Algebra. Less dense structures can benefit from the proposed Level 1 BLAS research in a similar manner. Intermediately sparse storages will require development of hybrid kernels, whose development should springboard directly from the dense work.

The second approach to exploiting input data structure involves having the software recognize sparsity patterns without user intervention, as in [37, 2]. Recognizing contiguous pieces of data in the original user's operands, such as subdiagonals or rectangular blocks, allows the computational kernel to maximize the data reuse at the highest levels of the memory hierarchy. In this way, recognizing patterns in a sparse matrix can lead to large savings in the memory traffic, and corresponding improvements in the performance of these kernels. To achieve this effect, the initial data structure may need to be modified in order to better utilize the specific features of a particular architecture. Since the most optimal data structure strongly depends on the user's data, it is important to dynamically perform this analysis at run-time, rather than statically during the installation process.

In order to exploit such sparsity patterns, it is necessary to explore techniques that aim at improving the poor data locality behavior of irregular sparse matrix computations. Existing compiler optimization techniques only show how to improve the data locality for regular loop structures. Striping the matrix by blocks of columns, or even partitioning it into blocks has been shown to potentially reduce the cache miss ratio (see [37]). Pattern matching techniques can also be used to recognize diagonals, or blocks of diagonals. These coarse techniques can be refined to minimize the amount of fill-in in the newly created data structure. We are interested in pursuing the development of dynamic optimization techniques based on the application of pattern matching methods to the same blocked compressed sparse storage schemes. Existing basic performance models [37, 46] evaluating the performance effects of dynamic changes in the storage scheme will also be further developed and used as building blocks for the SANS approach to sparse software. Of particular importance is the tuning of these models to specific architectures for greater accuracy. Objective functions could thus be evaluated at run-time taking into account hardware and software constants as well as the current data storage scheme.

As we have seen, there already exists a large body of research on the optimization of sparse linear algebra kernels. The literature further contains examples of using compiler- like optimizations coupled with dense code to generate sparse codes, as in [9, 10, 11, 39, 38]. This amount of research has inevitably resulted in the production of highly usable software libraries, such as [43, 4, 42]. However, most of this current software relies to a great extent on the user for sparsity analysis, and the compiler for floating point optimization.

ATLAS's previous dense work makes clear that we can expect to do quite a bit better in the floating point optimizations necessary for at least some of the storage structures. We hope that applying SANS techniques to the sparsity analysis will not only allow less sophisticated users to enjoy this increased performance as well, it will allow them to continue to enjoy it through successive hardware upgrades.

Sparsity analysis is particularly relevant to modern libraries such as Aztec or PETSc [43] because these libraries tend to hide the internal storage format from the user as much as possible. In such cases, a default storage scheme, chosen more for its generality than its performance characteristics, may be used by naive users even when a more optimal storage scheme is available. Sparsity analysis can help identify better performing storage options. Obviously, however, whatever sparsity analysis is performed must not become so burdensome that it overwhelms the actual compute time. In iterative methods, where the same operation will be performed hundreds or thousands of times with almost the same data, the cost of this analysis step can be amortized over all the calls, so that a relatively costly analysis can be done in the interest of optimization. In direct methods, a more lightweight, and therefore probably less efficient, analysis should be designed and employed.

Thus, we see two independent lines of research emerging for the application of SANS to sparse kernels for portable high performance. The first one involves extending current ATLAS features, such as the matrix vector multiply code generator, in order to support sparse optimizations concentrating primarily on blocked compressed storage schemes. The second research focus is on the characterization and evaluation of dynamic optimizations based on pattern matching techniques for those adequate storage schemes.

As with dense optimizations, there already exists a large body of research results on optimization of sparse linear algebra kernels which feeds into our work on making high performance portable. Generic software and documentation as in [43, 4, 42] nicely identify current practice in terms of interfaces and methods of choice. Most of the software optimization there, however, relies on the compilers and appropriate design decisions. The literature contains examples of using compiler-like optimizations coupled with dense code to generate sparse codes, as in [9, 10, 11, 39, 38]. Work that will be more immediately usable involves generating optimal kernels based on known storage sparsity patterns, as in [30, 11, 37, 45]. Finally, some work has also been done on performing sparsity analysis, as in [37, 2]. Just as in our work on dense kernels, we will explore these research results within the SANS framework in order to discover how to make the performance gains they lead to portable. Historically, we have found that this involves extending many of these areas of research. In particular, research on sparsity analysis and its concrete application needs further development to characterize its range of applications as well as to quantify its benefits.

The proposed research on sparse kernel optimization is two fold. First, we plan to reuse the current ATLAS static optimization knowledge with particular emphasis on block compressed storage schemes. This concretely leverages the important optimization results obtained during the development of the dense Level 2 BLAS kernels within the ATLAS framework. The general blocking strategy can be adapted; and the optimal floating point scheduling for dense general matrix-vector multiplication can be re-used in the context of block compressed storage schemes. We expect to validate this work by analyzing experimental results on various architectures featuring various hardware resources. The second research aspect deals with developing techniques aiming at improving the poor data locality behavior of irregular sparse matrix computations. Existing compiler optimization techniques only show how to improve the data locality for regular loop structures. Striping the matrix by blocks of columns, or even partitioning it into blocks has been shown to potentially reduce the cache miss ratio (see [37]). Pattern matching techniques can also be used to recognize diagonals, or blocks of diagonals. These coarse techniques can be refined to minimize the amount of fill-in in the newly created data structure. We are interested in pursuing the development of dynamic optimization techniques based on the application of pattern matching methods to the same blocked compressed sparse storage schemes. Existing basic performance models [37, 46] evaluating the performance effects of dynamic changes in the storage scheme will also be further developed and used as building blocks for the SANS approach to sparse software. Of particular importance is the tuning of those models to specific architectures for greater accuracy. Objective functions could thus be evaluated at run-time with respect to hardware and software constants as well as the current data storage scheme.

It is our belief that the combination of these two research activities with the precise empirical knowledge of machine specific parameters — will allow for the automated generation of highly efficient basic sparse kernels.

5.4 Applying SANS to Communication

Up to this point the SANS research proposed has focuses on the optimization of serial computation, but another potentially rich vein of inquiry deals with the optimization of parallel communication operations and libraries. Perhaps the most obvious starting point for such an investigation is a widely used basic operation such as broadcast. Of course there are already many well-known algorithms for optimizing different aspects of this critical operation. For instance, on most modern interconnects, a hypercube spanning tree is the topology of choice for small messages. As message size grows, however, preservation of optimality demands a shift from a latency-reducing algorithm to a bandwidth-reducing algorithm. Determining with any precision where one type algorithm becomes more efficient than another may be a nontrivial exercise, as it depends on message and broadcast group size, as well as the usual hardware and software issues. Even within the class of bandwidth-reducing algorithms, differing architectures may affect the choice of algorithm. With shared Ethernet, for example, a ring-based topology may be more efficient than other alternatives due to its low contention. Clearly such cases are natural candidates for the application of empirical methods.

This research will require development of SANS at a very fundamental level. Even the modest goal of optimizing the broadcast will require the solution to problems involved in doing the necessary timings for parallel computations. If SANS is successful in optimizing this simple operation, a host of parallel building block operations can be similarly addressed. In particular, the same kind of techniques can be applied to reductions and all-to- all communication, for instance. Further, this work should be extensible to both higher and lower level operations.

In the higher level range, it is worth investigating algorithm-specific topologies where the flow of operations across the processors gives more esoteric topologies preference in timings. Such a case can be found in ScaLAPACK's factorization routines. In these routines, each processor column contains an NB-wide panel of matrix columns. The active processor column factorizes its column panel, and broadcasts the factored panel to all processors. The next active processor column is the one immediately to the right of the current active processor column. Because of this flow of broadcasts, pipelines are maintained, and thus a pipelining broadcast such as ring is used. However, in this algorithm, the broadcast is the last time-critical thing the active processor column does. The next active processor's time is much more critical, and thus a modified ring, where the current active column performs the sends of the next active column, can be more optimal than an unmodified ring.

As an example of a lower level optimization, even such a simple operation as point to point send/receive can be optimized for various interconnects using the SANS method. It is not uncommon for certain architectures to employ differing interconnects for internal communication. For example, cluster-based machines constructed by connecting several independent building block units to make one machine, such as the IBM SP2s combined together to create ASCI BLUE/pacific (`www.llnl.gov/asci/platforms/bluepac/`), the SGI Power Challenges connected via HIPPI in ASCI blue mountain (`www.lanl.gov/projects/asci/bluemtn/`
and finally the clusters of fully connected Myranet nodes, which are themselves connected with a Myranet topology to form C-plant (`www.cs.sandia.gov/cplant/`) present this kind of problem.

These machines utilize different hardware and software when they communicate within a building block than when they communicate between building blocks. In this case, differing optimization techniques are usually required in order to maximize performance, and theoretically a SANS-aware MPI could detect these differences and provide for truly portable message passing performance.

6 Related Work

We list here, briefly, a number of existing projects and their relations to our proposed SANS systems.

LSA The University of Indiana's *Linear System Analyzer (LSA)* (`http://www.extreme.indiana.edu/pseware/lsa/index.html`; [27]) is building a problem solving environment (PSE) for solving large, sparse, unstructured linear systems of equations. It differs from our proposed systems in that it mostly provides a testbed for user experimentation, instead of a system with intelligence built in. A proposed LSA intelligent component (`www.extreme.indiana.edu/pseware/lsa/LSAfuture.html`) is more built on Artificial Intelligence techniques than numerical analysis.

ESI The *Equation Solver Interface (ESI) Standards Multi-lab Working Group & Interface Design Effort* (`http://z.ca.sandia.gov/esi/`) aims to develop an integral set of standards for equation-solver services and components. These standards are explicitly represented as an interoperable set of interface specifications.
While the ESI standard gives a much more detailed interface to equation solver libraries than we aim to provide in our scripting language, its existence will make it easier for us to integrate libraries that have an ESI interface into our systems.

CCA The *Common Component Architecture Forum (CCA Forum)* (`http://www.acl.lanl.gov/cca/`) has as its objective to define a minimal set of standard features that a High-Performance Component Framework has to provide, or can expect, in order to be able to use components developed within different frameworks.

ILU Tuning There is ongoing work at Boeing [41] in choosing the many parameters determining an ILU decomposition to optimize a either time or space, depending on the class of matrices (aerodynamics, structures, etc.).

Tune The TUNE project (`http://www.cs.unc.edu/Research/TUNE/`) seeks to develops a toolkit that will aid a programmer in making programs more memory-friendly.

References

[1] Ali-Reza Adl-Tabatabai, Michał Cierniak, Cuei-Yuan Lueh, Vishesh M. Parikh, and James M. Stichnoth. Fast, effective code generation in a Just-in-Time Java compiler. In *Proceedings of the ACM SIGPLAN'98 Conference on Programming Language Design and Implementation (PLDI)*, Montreal, Canada, June 1998.

[2] R. Agarwal, F. Gustavson, and M. Zubair. A High-Performance Algorithm Using Preprocessing for the Sparse Matrix-Vector Multiplication. In *Proceedings of the International Conference on SuperComputing*, pages 32–41, 1992.

[3] Glenn Ammons, Thomas Ball, and James R. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. In *Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation*, Las Vegas, NV, June 1997.

[4] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Second Edition.* SIAM, Philadelphia, PA, 1995.

[5] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide (third edition).* SIAM, Philadelphia, 1999.

[6] J. Auslander, M. Philipose, C. Chambers, S. Eggers, and B. Bershad. Fast, effective dynamic compilation. In *ACM SIGPLAN '98 Conference on Programming Language Design and Implementation (PLDI)*, May 1996.

[7] Thomas Ball and James R. Larus. Efficient path profiling. In *Proceedings of MICRO 96*, pages 46–57, Paris, France, December 1996.

[8] Richard Barrett, Michael Berry, Jack Dongarra, Victor Eijkhout, and Charles Romine. Algorithmic bombardment for the iterative solution of linear systems: a poly-iterative approach. *J. Comp. Appl. Math.*, 74:91–109, 1996.

[9] A. Bik, P. Brinkhaus, P. Knijnenburg, and H.Wijshoff. The automatic generation of sparse primitives. *ACM Transactions on Mathematical Software*, 24(2):190–225, 1998.

[10] A. Bik and H. Wijshoff. On Automatic Data Stucture Selection and Code Generation for Sparse Computation. In *Languages and Compilers for Parallel Computing*, number 768 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

[11] A. Bik and H. Wijshoff. Advanced Compiler Optimizations for Sparse Computations. *Journal of Parallel and Distributed Computing*, 31:14–24, 1995.

[12] J. Bilmes, K. Asanović, C.W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. In *Proceedings of the International Conference on Supercomputing*, Vienna, Austria, July 1997. ACM SIGARC. see `http://www.icsi.berkeley.edu/~bilmes/phipac`.

[13] S. Browne, J. Dongarra, J. Horner, P. McMahan, and S. Wells. Technologies for repository interoperation and access control. Technical Report ut-cs-98-395, University of Tennessee.

[14] S. Browne, P. McMahan, and S. Wells. Repository in a box toolkit for software and resource sharing. Technical Report ut-cs-99-424, University of Tennessee.

[15] Michael Burke, Jong-Deok Choi, Stephen Fink, David Grove, Michael Hind, Vivek Sarkar, Mauricio Serrano, V. C. Sreedhar, and Harini Srinivasan. The Jalapeno dynamic optimizing compiler for Java. In *Proceedings of the 1999 ACM Java Grande Conference*, San Francisco, CA, June 1999.

[16] H Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 1997.

[17] H. Casanova and J. Dongarra. Applying netsolve's network enabled server. *IEEE Computational Science & Engineering*, 5:57–66, 1998.

[18] H. Casanova, MyungHo Kim, James S. Plank, and Jack Dongarra. Adaptive scheduling for task farming with grid middleware. *International Journal of High Performance Computing*, 13:231–240, 1999.

[19] P. P. Chang, S. A. Mahlke, and W. W. Hwu. Using profile information to assist classic code optimizations. *Software – Practice & Experience*, 21(12):1301–1321, December 1991.

[20] R. Das, R. Ponnusamy, J. Saltz, and D. Mavriplis. Distributed memory compiler methods for irregular problems – data copy reuse and runtime partitioning. ICASE Report 91-73, NASA Langley Research Center, Sept 1991.

[21] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[22] J. Dongarra, J. Du Croz, S. Hammarling, and Richard J. Hanson. An Extended Set of FORTRAN Basic Linear Algebra Subroutines. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

[23] U. washington dynamic compilation project. www.cs.washington.edu/research/projects/unisw/DynComp/www, 1999.

[24] D. Engler, W. Hsieh, and M. F. Kaashoek. 'C: A language for high-level, efficient, and machine independent code generation. In *ACM Symposium on Principles of Programming Languages (POPL)*, 1996.

[25] Matteo Frigo. A fast fourier transform compiler. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, Atlanta, Georgia*, May 1999.

[26] Matteo Frigo and Stephen Johnson. Fftw: An adaptive software architecture for the fft. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Seattle, Washington*, May 1998.

[27] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju. Component architectures for distributed scientific problem solving. *IEEE CS&E Magazine on Languages for Computational Science and Engineering*. to appear.

[28] Roman Geus and Stefan Rollin. Towards A Fast Parallel Sparse Matrix-Vector Multiplication, Institute Of Scientific Computing. ETH Zurich, Submitted to World Scientific, 1999.

[29] S. L. Graham, P. B. Kessler, and M. K. McKusick. gprof: A call graph execution profiler. *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices*, 17(6):120–126, June 1982.

[30] E. Im and K. Yelick. Model-Based Memory Hierarchy Optimizations for Sparse Matrices. In *Workshop on Profile and Feedback-Directed Compilation*, Paris France, 1998.

[31] E. Im and K. Yelick. Optimizing sparse matrix-vector multiplication on SMPs. In *In Ninth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, 1999.

[32] Eun-Jin Im. *Automatic Optimization of Sparse Matrix - Vector Multiplication*. PhD thesis, University of California at Berkeley, May 2000. To Appear.

[33] Eun-Jin Im and Katherine Yelick. Optimizing sparse matrix vector multiplication on SMPs. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, TX, March 1999.

[34] Eun-Jin Im and Katherine Yelick. Optimization of sparse matrix kernels for data mining. submitted to First SIAM Conf. on Data Mining, 2000.

[35] Kazuaki Ishizaki, Motohiro Kawahito, Toshiaki Yasue, Mikio Takeuchi, Takeshi Ogasawara, Toshio Suganuma, Tamiya Onodera, Hideaki Komatsu, and Toshio Nakatani. Design, implementation, and evaluation of optimizations in a just-in-time compiler. In *Proceedings of the 1999 ACM Java Grande Conference*, San Francisco, CA, June 1999.

[36] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of Supercomputing '96*. Sponsored by ACM SIGARCH and IEEE Computer Society, 1996.

[37] P. Knijnenburg and H. Wijshoff. On Improving Data Locality in Sparse Matrix Computations. Personal Communication.

[38] V. Kotlyar, K. Pingali, and P. Stodghill. Compiling Parallel Sparse Code for User-Defined Data Structures. In *Proceedings of 8th SIAM conference on Parallel Processing for Scientific Computations*. SIAM, 1995.

[39] V. Kotlyar, K. Pingali, and P. Stodghill. A Relational Approach to the Compilation of Sparse Matrix Programs. In *Proceedings of EuroPar Conference*, 1997.

[40] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic Linear Algebra Subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5:308–323, 1979.

[41] John Lewis. Cruising (approximately) at 41,000 feet - Iterative methods at Boeing. talk presented Seventh SIAM Conference on Applied Linear Algebra, 2000. `www.siam.org/meetings/la00`.

[42] A. Lumsdaine and J. Siek. The Matrix Template Library. `http://www.lsc.nd.edu/research/mtl`.

[43] L. McInnes, S. Balay, W. Gropp, and B. Smith. PETSc 2.0 Users Manual. Technical Report ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, 1998.

[44] L. Oliker, X. Li, G. Heber, and R. Biswas. Ordering unstructured meshes for sparse matrix computations, on leading parallel systems. to appear in Irregular 2000 Intern. Workshop, 2000.

[45] Jakob Ostergaard. OptimQR - A Software-package to create near-optimal solvers for sparse systems of linear equations. `http://ostenfeld.dk/ jakob/OptimQR`.

[46] See homepage for a complete list of the people involved. TUNE - Mathematical Models, Transformations and System Support for Memory-Friendly Programming. `http://www.cs.unc.edu/Research/TUNE`.

[47] S. Toledo. Improving memory-system performance of sparse-matrix vector multiplication. In *In Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, 1997.

[48] R. Whaley and J. Dongarra. Automatically tuned linear algebra software. Computer Science Department CS-97-366, University of Tennessee, Knoxville, TN, December 1997. (LAPACK Working Note #131; see `http://www.netlib.org/utk/projects/atlas/index.html`).