Self-adapting Numerical Software for Next Generation Applications

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

Jack Dongarra, Victor Eijkhout†

December 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.
- System components, including a *runtime adaptive scheduler*, and 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.

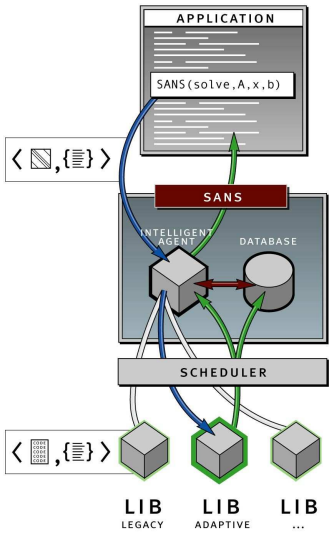
---

* 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 several types 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 application areas such as linear equation solving, eigenvalue computations, and information retrieval. However, the ideas and innovations SANS systems 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
- A *metadata vocabulary* that expresses properties of the user data and of performance profiles, and that will be used to build the performance history database. By considering this as *behavioural metadata*, we are led to *intelligent software components* as an extension of the CCA [2, 10] framework.
- One or more *prototype libraries*, for instance for sparse matrix computations, that accept informa-

tion about the structure of the user's data in order to optimize for execution on different hardware platforms.

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 demands of computational science over the next decade.

2 Optimization modes

The components of a SANS system can operate in several optimization modes, each being more or less appropriate depending on the component's level in the hierarchy and the nature of the data it's dealing with.

Completely off-line optimization This scenario is used in PHIPAC [5] and ATLAS [23], and it works well for 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 [16, 15] 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 to be followed when the available choices depend largely on the nature of the user data. The algorithmic decision making Intelligent Agent follows this protocol of inspecting the data and basing the execution on it. 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 [17] to redistribute it to accelerate subsequent matrix-vector multiplications.

Feedback Directed Optimization This scenario, not disjoint of the last, involves running the program, collecting profile and other information [14, 9, 3, 1] and recompiling with this information, or saving it for future refence when similar problems are to be solved. We will make use of this mode through the explicit incorporation of a database of performance history information.

3 Outline of the structure of self-adaptive software

A Self-adaptive Numerical Software system has three software components: a decision making component, consisting of an Intelligent Agent plus a History Database, a Network Scheduler, 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 the scheduler, which takes into account network conditions, and 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 [22, 5].

Additionally, we have metadata associated with the user input and the library algorithms, plus a simple control language which 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 Intelligent Agent is the decision making component on an algorithmic level. It is that part of the software that uses encoded knowledge of numerical analysis to analyze the data (section 4.1). The System Component (section 6) 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 agent's actions are informed by the History Database (section 4.4) 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.

4 The SaNS Agent

4.1 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 5), but in the most general case the Intelligent Component will do this by means of automated analysis (section 4.2). Moreover, any rules used in analyzing the user data and determining solution strategies are subject to tuning (section 4.3) 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.

4.2 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 system 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.

4.3 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 4.4) 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.

4.4 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.

5 Metadata and the Scripting Language

The operations typically performed by traditional libraries are 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. By introducing metadata, we gain the facility of annotating problem data with information that is typically lost, but which may inform a decision making process. 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.

The syntax with which the user specifies metadata could take the form of simple lists of keywords. However, for increased flexibility we propose that the user interact by means of a simple scripting language.

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.

The metadata passed by the user can not only be of varying levels of detail and sophistication, it can also lie on various points of a scale between purely numerical specification on the one extreme, and user application terms on the other. The former corresponds to the traditional parameter-passing approach of numerical libraries: users who are well-versed in numerics can express guidelines regarding the method to be used. However, most users are not knowledgeable about numerics; they can at most be expected to have expert knowledge of their application area. By building in a – heuristic – translation from application domain concepts to numerical concepts we allow the user to annotate the data in problem-native terms, while still assisting the SANS system in decision making.

6 Scheduler

The System Component of the SANS agent manages 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 4.4.

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.

Part of the System Component is scheduling operations and querying network resources. Software for this part of a SANS system already exists, in the Netsolve [6, 8, 7], GrADS [4, 19], and LFC [21] packages.

7 Libraries

7.1 Optimized libraries

Automation of the process of architecture-dependent tuning of numerical kernels can replace the current hand-tuning process with a semiautomated search procedure. Current limited prototypes for dense matrix-multiplication (ATLAS [23] and PHIPAC [5]) sparse matrix-vector-multiplication (Sparsity [16, 15], and

FFTs (FFTW [12, 11]) show that we can frequently do as well as or even better than hand-tuned vendor code on the kernels attempted.

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. This approach can be extended to a much wider range of computational kernels by using compiler technology to automate the production of these SDCGs.

8 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`; [13]) 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 [18] 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.

Kernel optimization In the preceeding we have already mentioned ATLAS [23], PHIPAC [5], Sparsity [16, 15], FFTW [12, 11]. An interesting project that combines ideas of dynamic optimization and low-level kernel optimization is Spiral [20] which generates optimal implementations of DSP algorithms.

References

[1] 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.

[2] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. C. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceedings of High Performance Distributed Computing*, pages 115–124, 1999.

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

[4] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Applications and Supercomputing*, 15(4):327–344, Winter 2001.

[5] 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`.

[6] 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.

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

[8] 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.

[9] 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.

[10] Common Component Architecture Forum. see `www.cca-forum.org`.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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`.

[19] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerical libraries and the grid: The GrADS experiments with scalapack. Technical Report ut-cs-01-460, University of Tennessee, Computer Science Department, 2001.

[20] Markus Püschel and José Moura. Generation and manipulation of DSP transform algorithms. In *Proc. 10th Digital Signal Processing Workshop*, 2002. http://www.ece.cmu.edu/~spiral/publ.html.

[21] Kenneth J. Roche and Jack J. Dongarra. Deploying parallel numerical library routines to cluster computing in a self adapting fashion, 2002. Submitted.

[22] 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`).

[23] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *To appear in Parallel Computing*, 2001. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (www.netlib.org/lapack/lawns/lawn147.ps).