International Journal of Parallel Programming, Vol. 33, Nos. 2/3, June 2005 (© 2005)
DOI: 10.1007/s10766-005-3577-3

The Component Structure of a Self-Adapting Numerical Software System

Victor Eijkhout,[1] Erika Fuentes,[1] Thomas Eidson,[2] and Jack Dongarra[1]

Self-Adapting Numerical Software (SANS) systems aim to automate some of the laborious human decision making involved in adapting numerical algorithms to problem data, network conditions, and computational platform. In this paper we describe the structure of a SANS system that tackles automatic algorithm choice, based on dynamic inspection of the problem data. We describe the various components of such a system, and their interfaces.

KEY WORDS: Linear system solving; component frameworks; adaptive systems.

1. INTRODUCTION

The process of arriving at an efficient numerical solution of problems in applied physics, chemistry, etc., involves numerous decision by a numerical expert. Attempts to automate such decisions (see Ref. 1 for a recent overview) distinguish three levels:

- algorithmic decision;
- management of the parallel environment;
- processor-specific tuning of kernels.

This paper addresses the top level, where algorithm choices are made dynamically based on the problem data. We describe the architecture of

[1] University of Tennessee, Knoxville TN, USA.
[2] Old Dominion University, Norfolk, VA, USA.

such a Self-Adapting Numerical Software (SANS) system for algorithm choice, paying particular attention to the formalization of various interfaces between modules in the system. We will not go into the modeling techniques that build up the heuristics of the 'intelligence' of the system. An introduction to this subject can be found in Ref. 1.

2. SYSTEM COMPONENTS

A SANS system has the following large scale building blocks:

- Application,
- Analysis Modules,
- Intelligent switch,
- Numerical libraries or components,
- Database,
- Modeler.

We will discuss each of these, devoting particular attention to their interfaces.

2.1. The Application

The problem to be solved by a SANS system typically derives from a physics, chemistry, etc., application. This application would normally call a library routine, picked and parametrized by an application expert. Absent such an expert, the application calls the SANS routine that solves the problem.

For maximum ease of use, then, the API of the SANS routine should be largely similar to the library call it replaces. However, this ignores the issue that we may want the application to pass application metadata to the SANS system. Other application questions to be addressed relate to the fact that we may call the SANS system repeatedly on data that varies only a little between instances. The paradigmatic example here is the sequence of linear systems to be solved in the course of a nonlinear (Newton) process. In such cases we want to limit the effort expended by the Analysis Modules.

The solution to both problems is to extend the notion of problem data to a 'dataset', which can contain application metadata, as well as knowledge about the context of the system call. Components accepting such datasets as input are said to have an 'extended interface'. We will go into the matter of this below; see Section 3.3.

2.2. Analysis Modules

Analysis modules have a two-level structure of categories and elements inside the categories. Categories are mostly intended to be conceptual, but they can also be dictated by practical considerations.

In the case of linear algebra problems, conceptual categories are

- pertaining to the nonzero structure of matrices (for sparse problems);
- norm-like properties (including diagonal dominance);
- spectral properties.

As an example of a nonconceptual categories, one can imagine the set of elements required by a certain algorithm.

An analysis element can either be computed exactly or approximately. For instance, the nonzero structure of a matrix can be computed exactly at very little cost, but bounds on the spectrum will in practice only be approximated. For some approximations, the degree of confidence can be quantified, but in other cases one can at best indicate what algorithm was used to compute them.

The output interface of the modules is defined by our standard for numerical metadeta.

The input interface is slightly more complicated. Here we remark that modules have to accept the same kind of data as the numerical components do, so we can adopt the extended interface here too.

2.3. Intelligent Switch

The intelligent switch determines which library code to apply to the problem. However, the method choice can be a composite decision, where certain stages can be considered preliminary transformations of the problem. Since such a transform maps the original problem to another, for which other numerical metadata applies, the switch can choose to rerun the analysis modules. This approach is expensive but likely to be accurate.

The alternative is to use only the initial metadata, and decide all transforms together. Of course, certain transforms leave certain categories of metadata invariant. For instance, scaling a matrix leaves the sparsity structure intact.

2.4. Numerical Components

In order to make numerical library routines more managable, we embed them in a component framework. This will also introduce a level of abstraction, as there need not be a one-to-one relation between library routines and components. In fact, we will define two kinds of components:

- “library components” are uniquely based on library routines, but they carry a specification in the numerical adaptivity language (Section 3.2) that describes their applicability;

- 'numerical components' are based one or more library routines, and having an extended interface (Section 3.3) that accomodates passing numerical metadata.

This distinction allows us to make components corresponding to the specific algorithm level ('Incomplete LU with drop tolerance') and generic ('preconditioner').

2.4.1. Transform Components

In certain problem domains it may be possible to pick a routine (or component in our framework) that solves the stated problem by itself. However, in other cases the solution is effected by the interplay of various pieces of software, such as the preconditioner and iterative method in iterative linear system solution.

We can take this modularity one step further by introducing 'transform components' which map one problem into another. Examples here would be permutations or scalings of matrix problems, prior to choosing the preconditioner and iterative method.

Presumably there is a choice of mappings, so we need to pass numerical metadata to a transform component. In fact, a transform will have largely the same extended interface as other numerical components.

Like numerical components, a transform can be queried as to the numerical metadata that is needed for determination of the mapping choice.

Applying the transforms is under control of the intelligent switch.

2.5. Database

The database of a SANS system contains information that couples problem features to method performance. While problem features can be standardized (this is numerical metadata), method performance is very much dependent on the problem area and the actual algorithm.

As an indication of some of the problems in defining method performance, consider linear system solvers. The performance of a direct solver can be characterized by the amount of memory and the time it takes. The amount of memory here is strongly variable between methods, and should perhaps be normalized by the memory needed to store the problem. For iterative solvers, the amount of memory is usually limited to a small multiple of the problem memory, and therefore of less concern. However, in addition to the time to solution, one could here add a measure such as "time to a certain accuracy", which is interesting if the linear solver is used in a nonlinear solver context. There is no counterpart to this measure in direct solvers, other than the trivial measure that the time to any accuracy is the same.

2.6. Modeler

The intelligence in a SANS system resides in two components: the intelligent switch which makes the decisions, and the modeler which draws up the rules that the switch applies. The modeler draws on the database of problem characteristics (as laid down in numerical metadata) to make rules express in an ‘adaptivity specification language’ (Section 3.2).

3. INTERFACES

3.1. Numerical Metadata

Numerical metadata is data associated with the numerical data. This can either be

- derived metadata: information derived from the numerical data, or
- application metadata: facts known a priori and normally not passed from the application to the library routines.

In the NMD library, described in Ref. 2 we have standardized the API to these data. This also defines the interface between the analysis modules and the intelligent switch.

3.1.1. Derived Metadata

Derived numerical metadata comprises such categories as

- structural metadata, relating to the nonzero structure of a sparse matrix;
- norm-like properties, including diagonal dominance; these first two categories are typically cheaply computable up to reasonable round-off;
- spectral information, giving some estimate of the spectrum or singular values of a matrix;
- other measures of a matrix such as departure from normality; these last two measures can not be computed exactly at a reasonable cost, but estimates—though more expensive than for the first two categories—can be obtained at a cost that is still justifiable as part of preprocessing.

In our paper(2) we proposed a core repertoire of numerical metadata categories, but the NMD language definition allows extension. For instance, one could introduce a category to account for the quantities measured in Ref. 3.

3.1.2. Application Metadata

Application metadata is numerical metadata that derives from knowledge of the application. Typical examples are

- grid properties;
- nature of the problem;
- properties of the operator if PDE.

Such information can be useful to an intelligent system (for instance, knowing positive definiteness obviates the need to infer this fact) but is usually dropped because the interface between application and numerics has no way of passing it.

3.2. Adaptivity Specification Language

A language for specifying the rules that control intelligent choice of algorithms is still a topic of research. Such a language will be used as the interface between the modeler and the intelligent switch, where the modeler extracts rules from the database, and the switch applies them to specific data.

An adaptivity specification language can also be used in numerical components. One can envision library components coming equiped with suitably formulated rules describing their applicability. This adds a semantic side to the interface specification of components.

3.3. Extended Interfaces

The extended interface captures more the semantics than the syntax: it contains

- routine parameters if the component is based on a single routines; missing parameters are filled with default or determined values;
- the union of all routine parameters if the component contains more than one routine; in this case parameters can be specified for any and all, only the relevant ones are used;
- numerical metadata: the presence of this makes it possible for the component to be intelligent and choose between the wrapped routines.

We also enhance the output side of components. In the case of numerical components this allows for performance data to be returned; with transform components this allows metadata to be returned, describing the transformed problem.

4. CONCLUSION

We have outlined precise definitions of the modules and interfaces in a SANS system for algorithm choice. Some of these interfaces have been formally defined in our research, others are in a process of being developed.

ACKNOWLEDGMENTS

This research was supported in part by NSF NGS program under grant ACI-0203984 and in part by the AMS/MICS office of the DOE under contract DE-ACO5-00OR22725 with UT-Battelle, LLC., and in part by the Los Alamos Computer Science Institute through the subcontract # R71700J-29200099 from Rice University.

REFERENCES

1. J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. C. Whaley, and K. Yelick, Self Adapting Linear Algebra Algorithms and Software, in: *Proc. of the IEEE* **93**(2):293–312 (February 2005) ISSN 0018–9219.
2. V. Eijkhout and E. Fuentes, A proposed standard for numerical metadata. Technical Report ICL-UT 03-02, Innovative Computing Laboratory, University of Tennessee, 2003. *Poster presentation at Supercomputing* (2003).
3. S. T. Xu, E. J. Lee, and J. Zhang, An Interim Analysis Report on Preconditioners and Matrices, *Technical Report* 388-03, University of Kentucky, Lexington; Department of Computer Science (2003).