Distributed SILC: An Easy-to-Use Interface for MPI-Based Parallel Matrix Computation Libraries

Tamito Kajiyama[1,2], Akira Nukada[1,2], Reiji Suda[1,2],
Hidehiko Hasegawa[3], and Akira Nishida[1,4]

[1] CREST, Japan Science and Technology Agency, Saitama 332–0012, Japan
[2] The University of Tokyo, Tokyo 113–8656, Japan
kajiyama@is.s.u-tokyo.ac.jp
[3] University of Tsukuba, Tsukuba 305–8550, Japan
[4] 21st Century COE Program, Chuo University, Tokyo 112–8551, Japan

Abstract. The present paper describes the design and implementation of distributed SILC (Simple Interface for Library Collections) that gives users access to a variety of MPI-based parallel matrix computation libraries in a flexible and environment-independent manner. Distributed SILC allows users to make use of MPI-based parallel matrix computation libraries not only in MPI-based parallel user programs but also in sequential user programs. Since user programs for SILC are free of a source-level dependency on particular libraries and computing environments, users can easily utilize alternative libraries and computing environments without any modification in the user programs. The experimental results of two test problems showed that the implemented SILC system achieved speedups of 2.69 and 7.54 using MPI-based parallel matrix computation libraries with 16 processes.

1 Introduction

The traditional way of using matrix computation libraries based directly on library-specific application programming interfaces usually leads to a source-level dependency on the libraries in use. This source-level dependency is the primary reason why users (i.e., application programmers) are often required to make a considerable amount of modifications to their user programs, for example when porting them to other computing environments or when trying out other libraries having different sets of solvers, matrix storage formats, arithmetic precisions, and so on. To address this issue inherent in the traditional programming style, we have been proposing an easy-to-use application framework named Simple Interface for Library Collections (SILC) [1,2]. A user program in the SILC framework first deposits data such as matrices and vectors into a separate memory space. Next, the user program makes requests for computation by means of mathematical expressions in the form of text. These requests are translated into calls for appropriate library functions, which are carried out in the separate

B. Kågström et al. (Eds.): PARA 2006, LNCS 4699, pp. 860–870, 2007.

```
double *A, *B;
int desc_A[9], desc_B[9], *ipiv, info;
/* create matrix A and vector B */
PDGESV(N, NRHS, A, IA, JA, desc_A, ipiv, B,
       IB, JB, desc_B, &info);
/* solution X is stored in B */
```

(a)

```
silc_envelope_t A, b, x;
/* create matrix A and vector B */
SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("x = A \\ b"); /* call PDGESV() */
SILC_GET(&x, "x");
```

(b)

Fig. 1. A comparison of two C programs (a) in the traditional programming style and (b) in the SILC framework, both making use of ScaLAPACK to solve a system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$

memory space independently of the user program. Finally, the user program fetches the results of computation from the separate memory space.

Figure 1 shows two user programs written in C, one in the traditional programming style and the other in the SILC framework. The traditional user program shown in Fig. 1 (a) prepares matrix A and vector $\boldsymbol{b}$ using library-specific data structures and makes a call for a library function in ScaLAPACK [3] to solve a system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$. The user program for SILC shown in Fig. 1 (b) realizes the same computation using the following three routines: `SILC_PUT` to deposit A and $\boldsymbol{b}$ into a separate memory space, `SILC_EXEC` to issue a request for solution of the linear system by means of a mathematical expression in the form of text, and `SILC_GET` to retrieve the solution $\boldsymbol{x}$. The mathematical expression specified as the argument of `SILC_EXEC` is translated into a call for the library function in ScaLAPACK for example, and carried out in the separate memory space.

We have developed a SILC system for sequential and shared-memory parallel computing environments [1,2]. The current implementation of SILC is based on a client-server architecture, in which a user program is a client of a SILC server running in a remote computing environment. Since a user program for SILC does not contain any library-specific code, no modification to the user program is required to utilize alternative matrix computation libraries. Moreover, users can automatically gain the advantages of parallel computation by using a SILC server that runs in a parallel computing environment. The main overhead in using SILC, on the other hand, is the cost of data communications between a user program and a SILC server. However, it is not difficult to reduce the relative amount of communication overhead, since the time complexities of matrix computations tend to be larger than their space complexities. For instance, solving a dense linear system with N unknowns takes $O(N^3)$ time, while the time necessary for data communications is of $O(N^2)$. Consequently, in many cases the use of a faster matrix computation library and computing environment results in good speedups even at the cost of data communications.

2 SILC for Distributed Parallel Computing Environments

We have been developing a SILC system for distributed parallel computing environments that allows users to make use of MPI-based matrix computation