社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－HPC－107（43）
2006／8／2

行列計算ライブラリインタフェースSILCの分散並列環境への実装

梶山民人[†1]　額田彰[†1]　須田礼仁[†2]
長谷川秀彦[†3]　西田晃[†4]

本論文では，行列計算ライブラリを計算環境やプログラミング言語に依らない方法で利用するためのインタフェース SILC（Simple Interface for Library Collections）の分散並列環境向けの設計および実装方法について述べる．本システムを利用することで，MPI ベースの行列計算ライブラリを分散並列プログラムからだけでなく逐次プログラムからも利用できる．SILC を用いるユーザプログラムには特定のライブラリや計算環境に依存したコードは現れないため，ユーザプログラムを書き換えることなく容易に別のライブラリや計算環境を利用できる．2 つの例題を用いた評価実験では，本システムを介して MPI ベースの行列計算ライブラリを 16 プロセスで利用したとき 6.46 倍から 9.12 倍の性能向上が得られた．

Implementation of the Matrix Computation Library Interface SILC in Distributed Parallel Environments

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

This paper presents the design and implementation of distributed SILC (Simple Interface for Library Collections) that allows users to easily utilize a variety of MPI-based parallel matrix computation libraries in a language- and computing environment-independent manner. Distributed SILC makes it possible to employ 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 do not contain any code that is specific to particular libraries and computing environments, users can easily switch libraries and computing environments without modifications to the user programs. Experimental results with two test problems showed that the present system achieved speedups of 6.46 and 9.12 by using an MPI-based parallel library with 16 processes via SILC.

1. はじめに

行列計算ライブラリは科学技術計算プログラムの作成に不可欠な構成要素であり，その重要性から多くのライブラリが開発されている[1)~5)]．これらのライブラリは一般に，個々のライブラリが提供するインタフェース（API）を通じて利用される．例えば，代表的な行列計算の 1 つである連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の求解の場合，ユーザはまず行列 A とベクトル $\boldsymbol{b}$ をライブラリ固有のデータ構造で準備する．次に，特定のライブラリ関数（求解ルーチン）を所定の引数の順序に従って呼び出す．このようなライブラリ固有のデータ構造と関数呼び出しに基づく従来のライブラリ利用法では，作成したユーザプログラムが特定のライブラリに依存する．ライブラリ間のインタフェースには互換性がないことが多いため，他のライブラリが提供する求解ルーチンを利用するにはユーザプログラムの修正が必要である．別の前処理や演算精度を利用する場合も同様である．また，特定の計算環境でのみ利用可能なライブラリが存在するため，ユーザプログラムを他の環境に移す際にライブラリの変更とそれに伴うユーザプログラムの修正が必要になる．しかしながら，ライブラリや計算環境を変更する都度ユーザプログラムを修正することは負担であり，より柔軟なライブラリ利用法が求められている．

この要求に応えるために，我々は計算環境に依存しない行列計算ライブラリインタフェース SILC（Simple Interface for Library Collections）を提案してい

†1 科学技術振興機構戦略的創造研究推進事業
CREST, Japan Science and Technology Agency
†2 東京大学情報理工学系研究科
Grad. Sch. of Info. Sci. & Tech., the Univ. of Tokyo
†3 筑波大学図書館情報メディア研究科
Grad. Sch. of Lib., Info. & Media Stud., U. of Tsukuba
†4 中央大学 21 世紀 COE プログラム
The 21st Century COE Program, Chuo University

る[6),7)]．ライブラリ固有のインタフェースに基づく従来のライブラリ利用法とは異なり，SILC では（1）入力データの預入れ，(2）文字列（数式）による計算指示，(3）計算結果の受取りの 3 つのステップで行列計算ライブラリの機能を利用する．行列やベクトルなどの入力データは，まずユーザプログラムから独立したメモリ空間に転送される．また，数式の形で与えられた計算指示は，適当なライブラリ関数の呼び出しに翻訳されて，独立したメモリ空間内で実行される．最後に，ユーザプログラムからの要求に応じて計算結果がユーザプログラムのメモリ空間に戻される．例として，SILC のインタフェースを介して行列計算ライブラリ LAPACK[3)] の連立一次方程式の求解ルーチンを呼び出す C プログラムを以下に示す．

```
silc_envelope_t A, b, x;
/* 行列 A とベクトル b の作成 */
SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("x = A \\ b"); /* 求解 */
SILC_GET(&x, "x");
```

このプログラムは，LAPACK のデータ構造を用いて行列 A とベクトル $\boldsymbol{b}$ を作成した後，SILC が提供する SILC_PUT, SILC_EXEC, SILC_GET の 3 つのルーチンを介して LAPACK の求解ルーチンを呼び出す．SILC_PUT は 2 つの入力データをユーザプログラムから独立したメモリ空間に転送するルーチン，SILC_EXEC は入力データに対する計算を指示するルーチン，SILC_GET は計算結果を受け取るためのルーチンである．SILC_EXEC の引数は文字列で表された数式であり，数式中のバックスラッシュは連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x}$ を得る演算子である．この演算子は LAPACK の求解ルーチンの呼び出しに翻訳されて，ユーザプログラムから独立したメモリ空間において実行される．

本論文では分散並列環境向け SILC（分散型 SILC）の設計と実装方法について述べる．本システムはクライアント・サーバ方式に基づいて実現されている．ユーザプログラム（クライアント）は逐次プログラムまたは MPI ベースの分散並列プログラムである．ユーザプログラムはソケット通信を用いて SILC サーバに接続し，サーバの管理する行列計算ライブラリを利用する．SILC のユーザプログラムには利用するライブラリに特化したコードは含まれないため，ユーザプログラムを書き換えることなく容易に別のライブラリを利用できる．また，サーバの実行環境を逐次環境から並列環境に移すことでユーザプログラムは並列計算のメリットを自動的に享受できる．例えば，LAPACK を従来法で用いている逐次のユーザプログラムを分散並列環境に移植する場合，LAPACK の代わりに ScaLAPACK[4)] を呼び出すにはユーザプログラムの大幅な修正が必要となる．一方，上述の SILC のユーザプログラムにおいて LAPACK の代わりに ScaLAPACK を用いるには逐次環境で動作するサーバの代わりに分散並列環境で動作するサーバを利用するだけでよく，ユーザプログラムの修正は不要である．

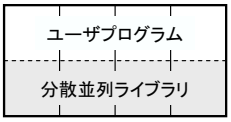


図 1 従来の分散並列プログラムの構成

2. 分散並列環境向け SILC

科学技術計算に用いられる種々の分散並列環境では，プロセス間通信に MPI を採用したさまざまな行列計算ライブラリが使われている．分散型 SILC の目的は，これらの MPI ベースのライブラリをプログラミング言語や計算環境に依らない方法で手軽に利用できるようにすることである．SILC はユーザプログラムと計算環境の間に位置して計算環境の違いを吸収するシステムであり，できるだけ多くの分散並列環境に対応することを目標にしている．現在は次の 3 種のシステム構成を検討している．

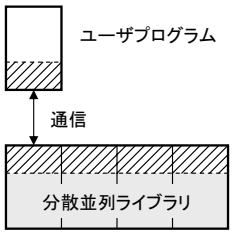
（A） 逐次クライアント＋分散並列サーバ
（B） 分散並列クライアント＋分散並列サーバ
（C） SILC のインタフェース機能をリンクした分散並列プログラム

比較のために MPI ベースのライブラリを従来法で利用するユーザプログラムの構成を図 1 に示す．ライブラリはユーザプログラムにリンクされて用いられる．ユーザプログラムは複数のプロセスから成る．図 1 は 4 プロセスの場合を表す．

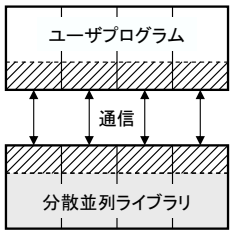
一方，分散型 SILC のシステム構成を図 2 に示す．斜線部分は SILC の行列計算ライブラリインタフェースの構成要素を表す．（A）と（B）はクライアント・サーバ方式に基づく構成である．（A）のクライアントは逐次プログラムであり，SILC サーバと（B）のクライアントは MPI に基づく分散並列プログラムである．

（A）の構成では，ユーザプログラムは SILC サーバを構成するプロセスの 1 つに接続してデータ転送と計算指示を行なう．ユーザプログラムからサーバに転送されたデータは，サーバが備えるデータ再分散機構によりライブラリの要求するデータ分散方式に変換され，複数のサーバプロセスに分散して保持される．また，ユーザプログラムに返される計算結果は，データ再分散機構により転送前にユーザプログラムの要求するデータ分散方式に戻される．

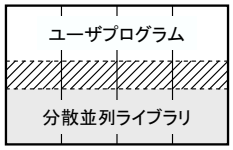
（B）の構成では，ユーザプログラムを構成する複数のプロセスがそれぞれ別のサーバプロセスに接続す



（A）逐次クライアント＋分散並列サーバ



（B）分散並列クライアント＋分散並列サーバ



（C）SILC のインタフェース機能をリンクした分散並列プログラム

図 2　分散型 SILC のシステム構成

る．データはユーザプログラムとサーバの双方において複数のプロセスに分散して保持され，両者の間で並列に授受される．データの分散方式はデータ再分散機構によりデータ転送の前後に適宜変更される．一方，計算指示はユーザプログラムのプロセスの 1 つが代表して行なう．なお，現時点ではユーザプログラムのプロセス数が SILC サーバのプロセス数以下の場合のみをサポートしている．

（C）の構成では，SILC のインタフェース機能をライブラリとして実現し，MPI ベースの行列計算ライブラリと共にユーザプログラムにリンクして用いる．このシステム構成は，クライアント・サーバ方式の採用が困難な分散並列環境に対応するために設けた．この場合，ライブラリ関数はユーザプログラムのメモリ空間で実行される．ユーザプログラムの記述方法は（B）の場合と同じであり，ユーザプログラムを書き換えることなく（B）の構成と（C）の構成を切り替えて利用できる．

2.1　クライアント・サーバ間通信とデータ再分散

ユーザプログラムと SILC サーバの間の通信とデータ再分散の処理の流れを図 3 に示す．図 3（a）および（b）は逐次クライアント（システム構成（A））における SILC_PUT と SILC_GET の動作を表し，図 3（c）および（d）は分散並列クライアント（システム構成（B））における SILC_PUT と SILC_GET の動作を表す．この図では，分散並列クライアントは 3 プロセス，SILC サーバは 6 プロセスから成る．

逐次クライアントの場合，SILC_PUT により送信されたデータは最初にサーバプロセスの 1 つにより受信され，さらにデータ再分散機構により全サーバプロセスに分散される．一方，SILC_GET によりクライアントに送り返されるデータは，一旦サーバプロセスの 1 つに集められた後，ネットワークを通じてクライアントに転送される．

分散並列クライアントの場合，データは分散型のデータ構造で表され，クライアントプロセス間に分散した形で保持される．分散データは SILC_PUT および SILC_GET によりサーバとの間で並列に授受される．SILC_PUT でサーバに送られたデータは，まずクライアントプロセス数と同数のサーバプロセスにより受信され，次にデータ再分散機構により全プロセスに分散される．同様に SILC_GET でクライアントに送り返されるデータは，最初にクライアントプロセス数と同数のサーバプロセスに集められ，複数の TCP 接続を通じて並列に転送される．

2.2　格納形式モジュールと演算モジュール

分散型 SILC で利用できる行列の格納形式（データ構造）を以下に示す．

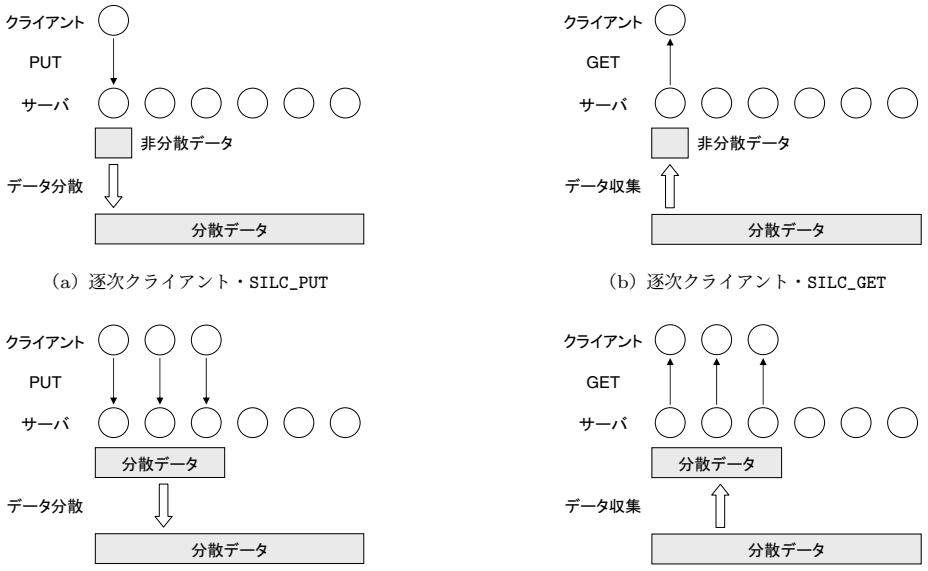
- 2 次元ブロックサイクリック分割の密行列
- 1 次元ブロック分割の密行列および帯行列
- 1 次元ブロック分割の CRS 形式の疎行列

格納形式毎に異なる処理は共有ライブラリの形で別個の格納形式モジュールとして実現され，SILC サーバの起動時に動的にリンクされる．例えば，上述の SILC_PUT と SILC_GET におけるデータ分散とデータ収集の 2 つの処理は格納形式によって処理内容が異なるので，格納形式モジュールにおいて実装される．これらの処理はデータ再分散機構により随時実行される．この他にも，格納形式モジュールにはメモリ管理，クライアント・サーバ間通信，演算精度の変換（キャスト）などの格納形式依存の処理が含まれる．

また，SILC を介して呼び出される個々のライブラリ関数にはインタフェースの違いを吸収するための手続き（ラッパー）が付与される．ラッパーは，ライブラリ関数の呼び出しに必要なデータ構造の変換や作業用メモリ領域の確保，開放などの処理を行なう．ラッパーが付与されたライブラリ関数は，ライブラリ別に演算モジュールとしてまとめられ，格納形式モジュールと同様に動的に SILC サーバにリンクされる．

3.　評 価 実 験

実現したシステムを通じて遠隔の高速な分散並列環境を利用することにより，従来法で書かれたユーザプ



（a）逐次クライアント・SILC_PUT　（b）逐次クライアント・SILC_GET

（c）分散並列クライアント・SILC_PUT　（d）分散並列クライアント・SILC_GET

図 3　クライアント・サーバ間通信とデータ再分散の処理の流れ

ログラムと比較して性能向上が得られることを確かめるため，次の 2 つの例題を用いて実験を行なった．

例題 1　連立一次方程式の求解
例題 2　偏微分方程式の初期値問題の求解

実験に用いた計算環境を表 1 に示す．これらの計算環境はすべて同じ Gigabit Ethernet の LAN に接続されている．計算はすべて倍精度実数で行なった．

3.1　例題 1: 連立一次方程式の求解

ScaLAPACK の PDGESV ルーチンを用いて $N \times N$ の密行列を係数とする連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く．実験では次の 2 つのユーザプログラムを用意して実行時間を比較した．いずれも C 言語で記述された MPI ベースの分散並列プログラムである．

従来法のユーザプログラム P_1　まず，行列 A とベクトル $\boldsymbol{b}$ を 2 次元ブロックサイクリック分割で作成する．A の要素は乱数で与え，$\boldsymbol{b}$ は解 $\boldsymbol{x}$ のすべての要素が 1 になるように設定する．次に，A と $\boldsymbol{b}$ を引数にして PDGESV ルーチンを呼び出す．このルーチンの実行時間を gettimeofday 関数で計測した．

SILC のユーザプログラム P_2　行列 A とベクトル $\boldsymbol{b}$ を P_1 と同様に作成する．次に，表 1 の Xeon クラスタまたは Altix で動作する SILC サーバに A と $\boldsymbol{b}$ を送り，連立一次方程式の求解を指示する．2 つの SILC サーバは連立一次方程式の求解演算子を PDGESV の呼び出しに翻訳するように設定した．P_2 の実行時間はサーバへの接続開始から解 $\boldsymbol{x}$ の受信完了までの経過時間とし，計時には gettimeofday 関数を用いた．

実験結果を図 4 に示す．横軸は係数行列の次数 N，縦軸は実行時間（単位は秒）である．P_1 と P_2 は OpenPower 上で 4 プロセスで実行した．P_2 が利用する SILC サーバは Xeon クラスタでは 8 プロセス，Altix では 16 プロセスで実行した．このシステム構成は図 2（B）に相当する．

P_2 の実行時間には A，$\boldsymbol{b}$ の転送とデータ分散に要するコスト，および $\boldsymbol{x}$ のデータ収集と転送に要するコストが含まれる．これらのコストは分散型 SILC の利用にともなう主要なオーバヘッドである．このオーバヘッドは N が大きくなると相対的に小さくなる．なぜなら，この例題の計算量は $O(N^3)$ 程度であるのに対してデータ量は $O(N^2)$ 程度であり，計算時間と通信時間の間に N 倍程度の違いがあるためである．SILC を通じて遠隔の高速な計算環境を利用すれば計算時間を大幅に短縮できるので，N が大きい場合には通信コストを上回る速度向上が得られる．このことは図 4 の結果についても当てはまる．$N = 4096$ の場合の速度向上率（P_1 の実行時間を 1 としたときの P_2 の実行時間）は Xeon クラスタを用いたとき 4.88 倍，Altix を用いたとき 6.46 倍であった．

3.2　例題 2: 偏微分方程式の初期値問題の求解

一次元拡散方程式 $\frac{\partial \phi}{\partial t} = \frac{\partial^2 \phi}{\partial x^2}$ $(t \geq 0,\ 0 \leq x \leq \pi)$ を初期条件 $\phi = \sin x$ $(t = 0,\ 0 \leq x \leq \pi)$，境界条件 $\phi = 0$ $(t > 0,\ x = 0)$ および $\phi = 0$ $(t > 0,\ x = \pi)$ の下でクランク・ニコルソン法[8] を用いて解く．初期時刻を $t_0 = 0$ とおき，小さな時間増分 $\Delta t > 0$ を用いて第 k 時刻を $t_k = t_{k-1} + \Delta t$ $(k = 1, 2, 3, \cdots)$ と定める．クランク・ニコルソン法では，各時刻 t_k に

表 1 実験に用いた計算環境（すべて同じ Gigabit Ethernet の LAN に接続）

名称	仕様
T42	IBM ThinkPad T42（Intel Pentium M 735 1.7 GHz，メモリ 512 MB） Fedora Core 4
OpenPower	IBM eServer OpenPower 710（Power5 1.65 GHz × 4 基，メモリ 1 GB） SuSE Linux Enterprise Server 9，LAM/MPI 7.1.2
Xeon クラスタ	IBM eServer xSeries 335（dual Intel Xeon 2.8 GHz，メモリ 1 GB）× 8 ノード Red Hat Linux 8.0，LAM/MPI 7.0
Altix	SGI Altix 3700（Intel Itanium2 1.3 GHz × 32 基，メモリ 32 GB） Red Hat Linux Advanced Server 2.1，SGI MPI 4.4（MPT 1.9.1）

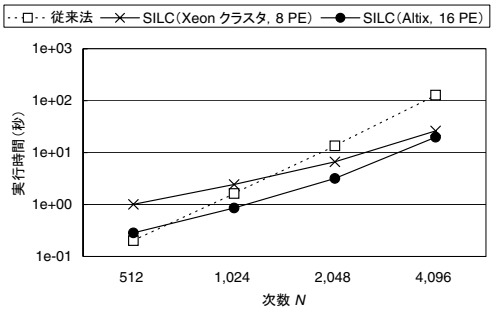


図 4 例題 1（連立一次方程式の求解）の実験結果

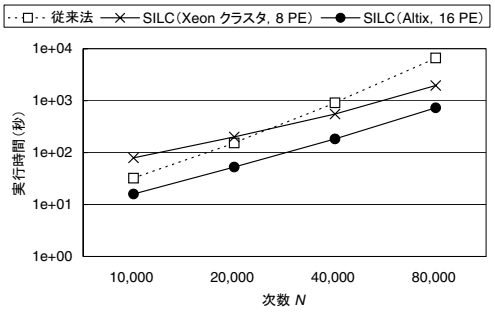


図 5 例題 2（偏微分方程式の初期値問題の求解）の実験結果

おいて $N \times N$ の疎行列を係数とする連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く必要がある．A の非零要素数は $3N-2$ である．実験では次の 2 つのユーザプログラムを用意して実行時間を比較した．これらはいずれも C で書かれた逐次プログラムである

従来法のユーザプログラム P_1

(1) 疎行列 A（CRS 形式）と初期時刻 t_0 における解 $\boldsymbol{x}$ を作成する．
(2) 時刻 t_k $(k = 1, 2, 3, \cdots)$ について：
  (a) $\boldsymbol{x}$ から $\boldsymbol{b}$ を作成する．
  (b) $A\boldsymbol{x} = \boldsymbol{b}$ を解く．

SILC のユーザプログラム P_2

(1) A と $\boldsymbol{x}$ を P_1 と同様に作成する．
(2) `SILC_PUT` で A を SILC サーバに送る．
(3) 時刻 t_k $(k = 1, 2, 3, \cdots)$ について：
  (a) $\boldsymbol{x}$ から $\boldsymbol{b}$ を作成する．
  (b) `SILC_PUT` で b をサーバに送る．
  (c) `SILC_EXEC` で $A\boldsymbol{x} = \boldsymbol{b}$ を解く．
  (d) `SILC_GET` で x を受け取る．

連立一次方程式の求解には反復解法ライブラリ Lis[9] の CG 法を用いた．P_1 と P_2 は T42 上で実行した．P_1 では逐次版の Lis を用いた．SILC サーバは，連立一次方程式の求解演算子を Lis の MPI 版の呼び出しに翻訳するように設定し，例題 1 と同じ計算環境およびプロセス数で実行した．このシステム構成は図 2 (A) に相当する．実行時間は P_1, P_2 共に手順 (2) の最初の 20 ステップ $(1 \le k \le 20)$ の所要時間とした．計時には `gettimeofday` 関数を用いた．

実験結果を図 5 に示す．横軸は次数 N，縦軸は実行時間（単位は秒）である．この例題においても N の増大にともなって速度向上率が高くなる傾向が見られる．非零要素数 $3N-2$ の疎行列を係数とする連立一次方程式の CG 法による求解に必要な浮動小数点演算の回数は $4N + \alpha(18N-1)$ である．ここで α は CG 法の反復回数を表す．一方，手順 (2) の各時刻において P_2 と SILC サーバの間で授受されるデータは $\boldsymbol{b}$ と $\boldsymbol{x}$ の 2 つであり，データ量は $8 \times 2N$（単位はバイト）である．定数 8 は倍精度実数のバイト数を表す．すなわち，各時刻の計算量とデータ量の間には約 α 倍の違いがある．そのため，N が大きい場合には通信コストを上回る速度向上が得られる．$N = 80000$ の場合の速度向上率は Xeon クラスタを用いたとき 3.38 倍，Altix を用いたとき 9.12 倍であった．

4. 関 連 研 究

MPI ベースの行列計算ライブラリの利便性を向上させるための研究は数多くなされている．

Trilinos プロジェクト[5] では，行列計算ライブラリを C++のクラスライブラリとして統合するための枠組みを提案し，多数の行列計算ライブラリを開発している．各ライブラリの API は，(1) 共通の行列とベクトルのデータ構造を用いること，(2) ライブラリの開発言語として C++を採用し，解法のインタフェースを規定する共通の抽象クラスを継承することの 2 点において統一されている．また，(3) 共通のディレクト

リ構造とコンパイル方法を採用し，開発者の異なるライブラリを同じ方法で構築可能な Trilinos パッケージとしてまとめている．ただし，API の詳細はライブラリ毎に大きく異なる．例えば Trilinos に含まれる連立一次方程式の直接解法パッケージと反復解法パッケージの API には互換性がないため，一方から他方に切り替えるにはユーザプログラムの修正が必要である．一方，SILC では計算を数式で指示するので，互換性のないライブラリであっても同じ方法で利用できる．

Amesos[10] は種々の直接解法ライブラリを共通のインタフェースで呼び出せるようにするための C++クラスライブラリである．ScaLAPACK をはじめとして多くの直接解法ライブラリに対応している．Amesos ではインタフェース統合の対象を連立一次方程式の直接解法に限定しているのに対して，SILC ではより広範な行列計算をサポートしている．

遠隔の計算環境を利用するという点で分散型 SILC と関連の深いシステムに Ninf-G[11] がある．Ninf-G はグリッド環境において遠隔手続き呼び出し（RPC）を実現するためのミドルウェアであり，MPI ベースの行列計算ライブラリの呼び出しをサポートしている．ただし，RPC を実行するプロセスは 1 つ（逐次プログラム，または分散並列プログラムのプロセスの 1 つ）でなければならない．また，入力データ（ライブラリ関数の引数）はライブラリ側のプロセスの 1 つにのみ転送される．他のプロセスへの入力データの分散と計算結果の収集についてはユーザが IDL ファイル内に記述しなければならない[12]．一方，SILC では複数プロセスからのライブラリ呼び出しをサポートしている．また，SILC がサポートする格納形式を入出力データの授受に使う場合はユーザがデータ分散とデータ収集の処理を記述する必要はない．

5. おわりに

本論文では分散並列計算環境向け SILC の設計と実現方法について述べた．本システムを用いることにより，分散並列環境だけでなく逐次環境からも MPI ベースの行列計算ライブラリを利用できる．ライブラリの実行環境を別の計算環境に移す場合にはユーザプログラムの修正は不要である．また，2 つの例題を用いた評価実験の結果から，問題サイズが十分に大きい場合には，SILC を介して遠隔の高速な計算環境を利用することにより通信コストを上回る速度向上が得られることを確かめた．16 プロセスの SILC サーバを用いたとき，例題 1 では 6.46 倍，例題 2 では 9.12 倍の速度向上率が得られた．

今後の課題は以下の通りである．

- 本システムの利用にともなう通信コストの分析と速度向上率の予測モデルの提案
- 既存の主要な行列計算ライブラリに対応する演算モジュールと格納形式モジュールの提供
- SILC の数式のスクリプト言語への拡張
- 数式に基づく行列計算の実行時最適化

謝辞 本研究は，科学技術振興機構（JST）戦略的創造研究推進事業（CREST）「大規模シミュレーション向け基盤ソフトウェアの開発」プロジェクト[13] の一部として実施した．

参考文献

1) Dongarra, J.: Freely available software for linear algebra on the Web (2004). http://www.netlib.org/utk/people/JackDongarra/la-sw.html.
2) Kotakemori, H., Hasegawa, H., Kajiyama, T., Nukada, A., Suda, R. and Nishida, A.: Performance Evaluation of Parallel Sparse Matrix–Vector Products on SGI Altix3700, *Proc. IWOMP 2005, LNCS* (in press).
3) Anderson, E., *et al.*: *LAPACK Users' Guide*, SIAM, 3rd edition (1999).
4) Blackford, L. S., *et al.*: *ScaLAPACK Users' Guide*, SIAM (1997).
5) Heroux, M. A., *et al.*: An overview of the Trilinos project, *ACM Transactions on Mathematical Software*, Vol. 31, No. 3, pp. 397–423 (2005).
6) Kajiyama, T., Nukada, A., Hasegawa, H., Suda, R. and Nishida, A.: SILC: Flexible and Environment Independent Interface to Matrix Computation Libraries, *Proc. PPAM 2005, LNCS 3911*, pp. 928–935 (2006).
7) Kajiyama, T., Nukada, A., Hasegawa, H., Suda, R. and Nishida, A.: LAPACK in SILC: Use of a Flexible Application Framework for Matrix Computation Libraries, *Proc. HPC Asia 2005*, pp. 205–212 (2005).
8) 登坂宣好, 大西和榮: 偏微分方程式の数値シミュレーション (第 2 版), 東京大学出版局 (2003).
9) SSI プロジェクト: Lis ユーザーズマニュアル (2006). http://ssi.is.s.u-tokyo.ac.jp/lis/.
10) Sala, M.: On the Design of Interfaces to Serial and Parallel Direct Solver Libraries, Technical Report SAND–2005–4239, Sandia National Laboratories (2005).
11) Ninf Project: http://ninf.apgrid.org/.
12) 武宮博, 田中良夫, 中田秀基, 関口智嗣: MPI と GridPRC を利用した大規模 Grid アプリケーションの開発と実行: Hybrid QM/MD シミュレーション, 情報処理学会論文誌：コンピューティングシステム, Vol. 46, No. SIG 12 (ACS 11), pp. 384–395 (2005).
13) 西田晃: SSI: 大規模シミュレーション向け基盤ソフトウェアの概要, 情報処理学会研究報告, 2004–HPC–098, pp. 25–30 (2004).