SILC: 行列計算ライブラリの利用を簡単化するフレームワーク

SILC: A SIMPLE AND FLEXIBLE FRAMEWORK FOR MATRIX COMPUTATION LIBRARIES

梶山民人[1], 額田彰[2], 須田礼仁[3], 長谷川秀彦[4], 西田晃[5]
Tamito KAJIYAMA, Akira NUKADA, Reiji SUDA, Hidehiko HASEGAWA and Akira NISHIDA

[1]科学技術振興機構 戦略的創造研究推進事業（〒 113−0033 文京区本郷 7−3−1, kajiyama@is.s.u-tokyo.ac.jp）
[2]科学技術振興機構 戦略的創造研究推進事業（〒 113−0033 文京区本郷 7−3−1, nukada@is.s.u-tokyo.ac.jp）
[3]東京大学 情報理工学系研究科（〒 113−0033 文京区本郷 7−3−1, reiji@is.s.u-tokyo.ac.jp）
[4]筑波大学 図書館情報メディア研究科（〒 305−8550 つくば市春日 1−2, hasegawa@slis.tsukuba.ac.jp）
[5]東京大学 情報理工学系研究科（〒 113−0033 文京区本郷 7−3−1, nishida@is.s.u-tokyo.ac.jp）

User programs for matrix computation libraries typically include a number of library calls, which lead to a source-level dependency upon the libraries. The dependency imposes heavy burden when porting the user programs, since high-performance computing environments usually have their own libraries to make the best use of computational resources. To allow flexible use of existing matrix computation libraries, we have been proposing an environment-independent framework named SILC (Simple Interface for Library Collections). In this framework, independence between user programs and libraries is achieved by (1) separating a library call into the delivery of arguments and the request of computation, (2) requesting the computation by means of mathematical expressions, and (3) using a separate memory space for the libraries independently of the user programs. This paper describes the design and implementation of the framework, and shows that the usability of matrix computation libraries is improved with SILC in the context of diverse computing environments, libraries, and programming languages in which user programs are written.
Key Words : *Matrix Computation, Application Programming Interface, Middleware*

1. はじめに

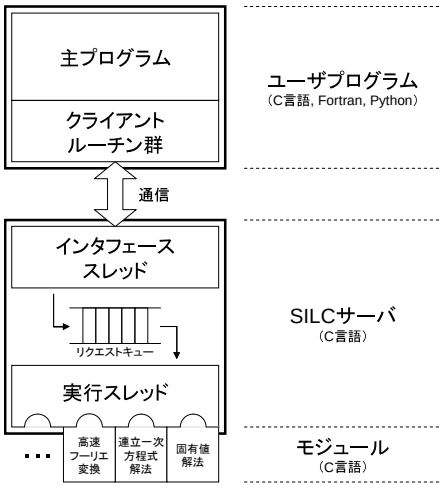
連立一次方程式の求解をはじめとする行列計算は科学技術計算に不可欠の基本演算であり，多様な行列計算ライブラリが開発されている [1][2][3][4]．これらのライブラリが提供する関数をユーザプログラムから呼び出すには，関数名と引数を所定の順序で記述する．関数呼び出しが実行されると，引数に指定したデータがライブラリ関数に渡されて所定の計算が行なわれる．このライブラリ関数の呼び出し方法は直感的で分かりやすく，Fortran や C 言語など多くのプログラミング言語で採用されている．

その一方で，従来のライブラリ関数の呼び出し方法には，ユーザプログラムが特定のライブラリに依存するという問題がある．あるライブラリ向けに作成されたユーザプログラムには，しばしばそのライブラリに特化した関数呼び出しが多数含まれるため，別のライブラリを用いるように修正するのは容易でない．また，科学技術計算に用いられる各種の計算環境では環境毎に最適化された行列計算ライブラリが用意されており，ある計算環境において作成したユーザプログラムは別の計算環境ではライブラリの違いにより実行できないことがある．利用するライブラリを変更してユーザプログラムを別の計算環境向けに書き直すことはユーザの負担であり，ライブラリに依存しないユーザプログラムの開発手法が求められている．

これに対して我々は，ライブラリ呼び出しを計算の指示とデータの授受に分けることによりユーザプログラムとライブラリの独立性を高める枠組み SILC（Simple Interface for Library Collections）を提案している [5]．SILC では，ユーザプログラムは次の 3 つのステップによりライブラリの機能を利用する．

1. 行列やベクトルなどの入力データに名前（変数名）を付けて預ける．データはユーザプログラムとは別のメモリ空間に転送される．
2. 計算を文字列（数式）で指示する．データに付与した変数名を用いて計算を数式で記述する．計算にはユーザプログラムのメモリ空間を使用しない．
3. 結果を受け取る．結果を保持する変数名を指定してデータをユーザプログラムのメモリ空間に戻す．

従来のライブラリ関数の呼び出し方法では計算の指示とデータの授受が同時に行なわれる．また，関数呼び出しにおける関数名，引数の順序，および授受されるデータの構造はライブラリによって決まるため，ユーザプログラムが特定ライブラリに依存する原因となっ



図–1 SILC のシステム構成

ている．一方，提案手法においては，行列やベクトルなどの入力データを個別に授受し，計算は必要なデータを預け終わった後に数式で指示する．与えられた数式を構成する演算子は，それぞれ適当なライブラリ関数の呼び出しに翻訳されて，ユーザプログラムとは別のメモリ空間で実行される．そのため，SILC のユーザプログラムには特定の行列計算ライブラリに特化した関数呼び出しのコードは現れない．利用するライブラリの変更には，演算子とライブラリ関数の対応関係を変えるだけでよく，ユーザプログラムの大幅な書き換えは不要である．

本論文では，SILC の設計と実現方法について述べる．また，提案手法によって行列計算ライブラリの利便性が向上したことを示す．

2. SILC の設計と実現方法

現在，クライアント・サーバ方式に基づく共有メモリ並列環境向け SILC の開発を進めている．本システムの構成を図–1 に示す．

ユーザが作成するユーザプログラムは，ネットワークを介して SILC サーバに接続して，PUT リクエスト（データの預け入れ），SILC リクエスト（計算指示），および GET リクエスト（データの受け取り）の 3 種のリクエストを送ることにより SILC サーバが管理するライブラリの機能を利用する．C 言語で記述されたユーザプログラムの場合，3 種のリクエストの送信にはそれぞれ以下のクライアントルーチンを用いる．

- `SILC_PUT`(名前，データ)
- `SILC`(数式)
- `SILC_GET`(データ，名前)

ここで,〈名前〉には変数名,〈数式〉には後述の命令記述言語で表された計算指示の文字列を指定する．また,〈データ〉にはデータの授受に用いる `silc_envelope_t` 構造体へのポインタを指定する．

ユーザプログラムから送られたリクエストは，インタフェーススレッドを通じて SILC サーバ内のリクエストキューに追加される．格納されたリクエストは実行スレッドにより順次処理される．インタフェーススレッドはユーザプログラムと同期して動作し，実行スレッドは非同期でリクエストを処理する．

計算指示には SILC の命令記述言語を用いる．計算の実行単位は文であり，文には代入文と手続き呼び出し文がある．データ型にはスカラー，ベクトル，行列，3 次元配列などがある．代入文の左辺には変数名を型宣言なしで指定し，右辺には式を記述する．式を構成する演算子には加減乗除，内積，連立一次方程式の求解，共役転置，複素共役，関数呼び出しなどがある．制御構文はなく，条件分岐や反復はユーザプログラムの記述言語により実現する．

例として，連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を CG 法 [6] で解くユーザプログラムの一部を図–2 に示す．サーバに預ける入力データは Compressed Row Storage（CRS）形式 [6] の疎行列 A とベクトル $\boldsymbol{b}$，サーバから受け取る出力データはベクトル $\boldsymbol{x}$ である．

数式中の演算子に対応する個々のライブラリ関数は演算モジュールを介して呼び出される．演算モジュールは外部ライブラリを SILC サーバに統合するためのラッパーであり，サーバの起動時に動的にリンクされる．また，行列の送受信や複製といった低レベルのデータ処理は，行列の格納形式毎にモジュール化されており，格納形式モジュールと呼ぶ．SILC は現在，密行列と CRS 形式の疎行列に対応している．新しい格納形式を追加するには，格納形式モジュールとその格納形式に対応した演算モジュールの 2 つを新たに設ければよい．

3. 行列計算ライブラリの利用

SILC から行列計算ライブラリを利用することにより得られる利点を，計算環境，ライブラリ，およびユーザプログラムの記述言語の観点から述べる．

(1) 計算環境

SILC のユーザプログラムは SILC サーバの実行環境から独立しており，ユーザはサーバの実行環境について何ら仮定することなくユーザプログラムを作成できる．そのため，サーバの実行環境を逐次環境から並列環境に移してもユーザプログラムを変更する必要はなく，自動的に並列環境の恩恵を享受できる．

また，SILC では計算指示とデータの授受を分けており，PUT，SILC，GET の各リクエストを複数のユーザプログラムから個別に実行できる．例えば，行列生成，求解，可視化の 3 ステップから成る計算は，行列を生成してサーバに預けるプログラム，求解を指示するプログラム，計算結果を受け取って可視化するプログラムに分けて実現できる．

データの送受信と文字列による計算指示はいずれも比較的負荷の少ない処理であり，ユーザプログラムは処理能力の低い計算機でも実行できる．そのため，パソコンでユーザプログラムを実行し，スーパーコンピュー

```
int silc_cg(int n, int nnz,
  double *value, int *index, int *row,
  double *b, double *x)
{
  silc_envelope_t object;
  int i;
  double nrm2;

  /* connect to SILC server */
  SILC_INIT();

  /* put matrix A */
  object.type = SILC_MATRIX_TYPE;
  object.format = SILC_FORMAT_CRS;
  object.precision = SILC_DOUBLE;
  object.m = object.n = n;
  object.nnz = nnz;
  object.v = value;
  object.row = row;
  object.index = index;
  SILC_PUT("A", &object);

  /* put vector b */
  object.type = SILC_COLUMN_VECTOR_TYPE;
  object.precision = SILC_DOUBLE;
  object.length = n;
  object.v = b;
  SILC_PUT("b", &object);

  /* solve "Ax = b" with CG method */
  SILC("rho_old = 1.0");
  SILC("n = length(b)");
  SILC("p = zeros(n, 1)");
  SILC("x = zeros(n, 1)");
  SILC("r = b");
  SILC("bnrm2 = 1.0 / norm2(b)");
  for (i = 0; i < n; i++) {
    SILC("rho = r' * r");
    SILC("beta = rho / rho_old");
    SILC("p = r + beta * p");
    SILC("q = A * p");
    SILC("alpha = rho / (p' * q)");
    SILC("x = x + alpha * p");
    SILC("r = r - alpha * q");

    /* check convergence */
    SILC("nrm2 = norm2(r) * bnrm2");
    object.v = &nrm2;
    SILC_GET(&object, "nrm2");
    printf("iter: %d, res: %e\n", i+1, nrm2);
    if (nrm2 <= EPSILON) {
      /* get vector x */
      object.v = x;
      SILC_GET(&object, "x");
      break;
    }
    SILC("rho_old = rho");
  }

  /* close connection */
  SILC_END();

  return i; /* number of iterations */
}
```

図–2 SILC のユーザプログラムの例（CG 法）

タ上の SILC サーバにデータを送って計算するといった計算環境の柔軟な利用が可能になる．

(2) ライブラリ

行列計算ライブラリには，大別して密行列を扱うものと疎行列を扱うものの 2 種がある．

a) 密行列

密行列を対象とする行列計算においては，汎用の行列計算ライブラリ BLAS (Basic Linear Algebra Subprograms) [3] が広く利用されている．BLAS には計算環境毎に最適化された多数の実装があり，Fortran では共通のインタフェースで利用できる．

しかしながら，BLAS を C 言語から呼び出すためのインタフェースは実装毎に異なっている．例えば，Sun Performance Library [7] と Intel Math Kernel Library [8] にはそれぞれ別の BLAS の実装が含まれており，両者の C 言語インタフェースは関数名，ヘッダファイル名，複素数の扱いなどに違いがある．そのため，C 言語から BLAS を呼び出すユーザプログラムを作成すると，計算環境を変更する際にユーザプログラムを書き換える必要が生じる．

これに対して SILC では，ライブラリ呼び出しに相当するコードを数式で記述するので，SILC のユーザプログラムには BLAS の C 言語インタフェースに依存したコードは一切現れない．すなわち，SILC においては BLAS 固有のインタフェースはユーザプログラムから完全に隠されている．そのため，計算環境を変更してもユーザプログラムを書き換えることなく当該環境用に最適化された BLAS を利用できる．

また，BLAS を Fortran や C 言語から利用する場合，引数に与えるデータの精度に応じて異なる名前のライブラリ関数を呼び出さなければならない．例えば，行列ベクトル積を計算するには単精度では SGEMV，倍精度では DGEMV を用いる．一方，SILC を利用する場合は，行列ベクトル積を “y = A * x” のように指示すれば，サーバが A と x の精度に応じて適切なライブラリ関数を選択する．A と x の精度が異なる場合は自動的に精度を合わせてライブラリ関数を呼び出す．また，計算用の一時メモリが必要な場合はサーバが自動的に確保する．

b) 疎行列

大規模疎行列を対象とする計算は科学技術計算の実行時間の多くの部分を占め，多数の疎行列計算ライブラリが開発されている [2][4]．これらのライブラリは，利用可能な解法，前処理，および疎行列の格納形式に違いがある．また，ライブラリ関数のインタフェースはライブラリ毎に異なっており，一般に互換性はない．

大規模疎行列計算では，計算環境に応じて適切な疎行列の格納形式を選ぶ必要がある．しかしながら，使用するライブラリを変更して別の格納形式を採用するにはユーザプログラムの大幅な書き換えが必要となる．また，計算環境毎に異なる格納形式を採用した複数の

ユーザプログラムを作成することは負担が大きい．

一方，SILC においては，ユーザプログラムとサーバのメモリ空間は独立しており，サーバは受け取った疎行列の格納形式をユーザプログラムに影響を与えることなく変更できる．したがって，例えばユーザプログラムでは行列の格納に CRS 形式を利用し，サーバ側の行列計算には Jagged Diagonal Storage (JDS) 形式 [6] を用いるといった格納形式の変更が容易に実現できる．

格納形式の変換には疎行列の非零要素数に応じた時間がかかるが，計算環境によってはユーザプログラムが採用した格納形式のまま計算するよりも別の格納形式に変換してから計算した方が速いことがある [9]．また，格納形式を変換してから計算するという方法には，疎行列計算に利用できるライブラリの選択肢が増えるという利点がある．

(3) ユーザプログラムの記述言語

SILC のクライアントルーチン群は小規模で実現が容易であるため，ユーザプログラムの開発には多くのプログラミング言語が利用できる．ユーザプログラムの記述言語に対する主な要件は，(a) 数値計算と文字列処理を記述できること，(b) ソケット通信を利用できることの 2 点であり，主要なプログラミング言語はこれらの要件を満たしている．現時点では，C 言語，Fortran，および Python [10] のクライアントルーチンが利用できる．

4. 関連研究

行列計算ライブラリの利便性を高める手法には，遠隔手続き呼び出し（RPC）に基づく方法 [11][12] や行列言語コンパイラを用いる方法 [9][13] がある．Ninf–G [11] はグリッドと呼ばれる広域分散環境において RPC を実現するためのミドルウェアである．このシステムでは，ライブラリ関数の名前，引数および戻り値を Ninf–G IDL と呼ぶインタフェース記述言語により記述して，RPC の実行時に必要となるデータ変換と通信を自動化する．Ninf–G では従来のライブラリ呼び出しとほぼ同一の記法で RPC を実行する．一方，SILC では数式で計算を指示する点が大きく異なる．CMC [9] は MATLAB [14] の言語で記述されたユーザプログラム（関数）を Fortran 90 のサブルーチンに変換するコンパイラである．複数の疎行列の格納形式に対応し，ソースコードレベルの最適化機能を備えている．CMC と SILC の大きな差は，SILC ではクライアント・サーバ方式で遠隔の計算環境を利用できること，数式による計算指示を関数単位ではなく文単位で行なうこと，および数式をユーザプログラム中に埋め込む用法を前提にしていることである．

5. おわりに

本論文では，共有メモリ並列環境向け SILC の設計と実現方法について述べた．また，本システムにより行列計算ライブラリの利便性が向上することを計算環境，ライブラリ，およびユーザプログラムの記述言語の観点から示した．今後の課題としては，既存ライブラリの SILC への統合，単独利用可能なスクリプト言語への命令記述言語の拡張，数式に基づく実行時最適化，分散メモリ並列環境への対応などが挙げられる．

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 [15] の一部として実施した．

参考文献

1) Jack Dongarra. Freely available software for linear algebra on the Web, May 2004. http://www.netlib.org/utk/people/JackDongarra/la-sw.html.
2) K. Wu and B. Milne. A survey of packages for large linear systems. Technical Report LBNL-45446, Lawrence Berkeley National Laboratory, 2000.
3) BLAS. http://www.netlib.org/blas/.
4) 小武守恒, 藤井昭宏, 中島研吾, 長谷川秀彦, 西田晃. 複数の前処理と格納形式を持つ反復解法ライブラリ. ハイパフォーマンスコンピューティングと計算科学シンポジウム (HPCS2005) ポスター論文集, p. 4, January 2005.
5) 長谷川秀彦, 須田礼仁, 額田彰, 梶山民人, 中島研吾, 高橋大介, 小武守恒, 藤井昭宏, 西田晃. 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. 情報処理学会研究報告, 2004–HPC–100, pp. 37–42, December 2004.
6) R. Barrett *et al.* *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994.
7) Sun Microsystems. *Sun Performance Library User's Guide for Fortran and C*, July 2001.
8) Intel Corporation. Intel Math Kernel Library. http://www.intel.com/software/products/mkl/.
9) 湯之上康一, 川端英之, 北村俊明. 行列言語コンパイラ CMC の JDS 形式への対応と Matrix Market を用いた評価. 情報処理学会研究報告, 2005–HPC–101 (HOKKE2005), pp. 103–108, March 2005.
10) Python. http://www.python.org/.
11) Ninf Project. http://ninf.apgrid.org/.
12) NetSolve. http://icl.cs.utk.edu/netsolve/.
13) L. De Rose and D. Padua. Techniques for the translation of MATLAB programs into Fortran 90. *ACM Trans. on Programming Languages and Systems*, Vol. 21, No. 2, pp. 286–323, 1999.
14) MathWorks. http://www.mathworks.com/.
15) 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, April 2004.

第10回計算工学講演会
（2005年6月1日）

SILC: 行列計算ライブラリの利用を簡単化するフレームワーク

梶山民人（JST CREST） 額田彰（JST CREST）
須田礼仁（東京大学） 長谷川秀彦（筑波大学）
西田晃（東京大学）

研究の背景

- 行列計算
  - 科学技術計算に不可欠の基本演算
  - 多数の行列計算ライブラリ
- 関数呼出しによるライブラリの利用
  - ユーザプログラムが利用するライブラリに依存
    - ライブラリ固有のデータ構造、関数名、引数の順序
- 別のライブラリの利用が困難
  - ライブラリ変更のためにユーザプログラムの大幅な書き換えが必要

関数呼出しによるライブラリの利用例

- 連立一次方程式 $Ax = b$ の求解

```
SSI_MATRIX  A;
SSI_SCALAR  *b, *x, work[N*6], params[2];
int options[6], status;
行列 A とベクトル b を作成し、解 x のメモリ領域を確保
status = ssi_cg (b, x, work, params, options, &A, NULL);
```

- データ構造、関数名、引数の順序が利用する行列計算ライブラリに依存
  - ⇒ 別ライブラリを利用するにはソースコードの修正が必要

別の行列計算ライブラリを用いる動機

- 別の計算環境を利用する場合
  - 多様な計算環境（逐次、SMP並列、クラスタなど）
  - 各々の計算環境は固有のライブラリを有する
- 解法や行列の格納形式を変更する場合
  - 最適な解法と格納形式は、問題（行列の性質）と計算環境によって変わる
  - 用意されている解法と格納形式はライブラリ毎に異なる

提案手法

```
silc_envelope_t  A, b, x;
行列 A とベクトル b を作成し、解 x のメモリ領域を確保
SILC_PUT ("A", &A);          行列 A を預ける
SILC_PUT ("b", &b);          ベクトル b を預ける
SILC_EXEC ("x = A \ b");     Ax = b の求解を指示
SILC_GET (&x, "x");          解 x を受け取る
```

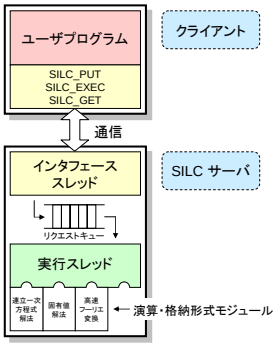
- 関数呼出しをデータ授受と計算指示に分離
- 計算は文字列（数式）で指示
- ユーザプログラムとは別のメモリ空間で計算を実行

Simple Interface for Library Collections

- SILC の特徴
  - 多様な行列計算ライブラリを同じ枠組みで利用可
  - 利用するライブラリを容易に変更できる
    - ユーザプログラムの修正は不要
  - 計算環境の柔軟な組み合わせが容易
  - 様々なプログラミング言語で利用できる
    - C, Fortran（コンパイル言語）
    - Python（対話型スクリプト言語）

SILC のシステム構成

- 共有メモリ並列環境向け
- クライアント・サーバ方式
  - ユーザプログラム（逐次）
    - サーバが管理するライブラリをネットワークを介して利用
  - インタフェーススレッド
    - 通信を担当
    - ユーザプログラムと同期して動作
  - 実行スレッド
    - 計算を担当
    - 非同期に計算を実行



SILC の枠組みのメリット

（1）計算環境の柔軟な組み合わせ
（2）多様なライブラリの利用
（3）ユーザプログラムの記述言語

（1）計算環境の柔軟な組み合わせ

- ユーザプログラムと計算環境は独立
  - 計算環境を考慮せずにユーザプログラムを作成可能
  - SILCサーバを逐次環境から並列環境に移せば自動的に並列化のメリットを享受できる
- PUT, EXEC, GET は個別に実行可能
  - サーバ上のデータは指示されるまで消えない
  - 例：行列生成、求解、可視化の3つのプログラムを個別に作成し、SILC を介して統合
- ユーザプログラムは PC でも実行可能
  - 例：PC でスーパーコンピュータ上の計算を制御

（2）多様なライブラリの利用

- BLAS（Basic Linear Algebra Subprograms）
  - 精度による関数名の違い
    - 行列ベクトル積：SGEMV（単精度）、DGEMV（倍精度）など
  - SILC では “y = A * x” のように数式で計算指示
    - 精度に応じて適切なライブラリ関数を選択
- 反復解法ライブラリ [小武守2005]
  - 複数の解法と複数の格納形式をサポート
  - 最適な解法と格納形式は、問題と計算環境により異なる
  - SILC では解法と格納形式を容易に変更可能
    - サーバ側の設定により解法を選択
    - CRS 形式でデータを授受し、JDS 形式に変換して計算

（3）ユーザプログラムの記述言語

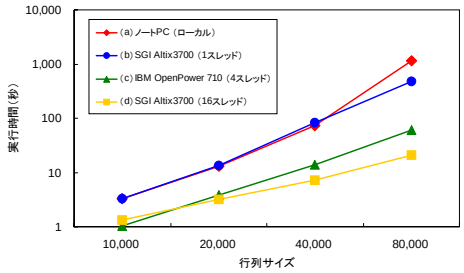
- 様々なプログラミング言語が利用可能
  - 特定言語に依存しない数式による計算指示
  - プログラミング言語に対する要件
    - 数値計算と文字列処理を記述できること
    - ソケット通信を利用できること
- 現在、SILC で利用できるプログラミング言語
  - C, Fortran（コンパイル言語）
  - Python（対話型スクリプト言語）
    - ユーザプログラムの記述が非常に容易
    - Unix でも Windows でも同じスクリプトが動作

Python によるユーザプログラムの例

```
class MatrixVectorProductTestCase (silctest.TestCase):
    def do_test (self, precision):
        data = [[1, 2], [3, 4]]
        self.put_matrix ("A", 2, 2, data, precision)
        data = [1, 1]
        self.put_column_vector ("x", data, precision)
        self.execute ("y = A * x")
        answer = self.get ("y")
        assert answer.type ==  SILC_COLUMN_VECTOR_TYPE
            .
            .
            .
```

実行例

- SILC サーバを(a)～(d)の4つの計算環境で実行
- 前述のユーザプログラムを(a)のノート PC 上で実行
- 行列:3重対角行列、CRS形式、倍精度



まとめと今後の課題

- まとめ
  - 行列計算ライブラリの利便性を向上させる新しいフレームワーク SILC の提案
  - 共有メモリ並列環境向けシステムの実現
- 今後の課題
  - 既存ライブラリの SILC への統合
  - 数式に基づく実行時最適化
  - 数式の“SILC スクリプト言語”への拡張
  - 分散メモリ並列環境への対応

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」
(SSI: Scalable Software Infrastructure)プロジェクトの
一部として実施した。

本プロジェクトのホームページ:
http://ssi.is.s.u-tokyo.ac.jp/