共有メモリ並列環境における SILC の実現と利用

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

本論文では共有メモリ並列環境における SILC（Simple Interface for Library Collections）の実現と利用について述べる．SILC は，行列計算ライブラリに対してプログラミング言語および計算環境に依存しない一様なインタフェースを付与する枠組みである．本枠組みを利用することにより，多様な行列計算ライブラリを C や Fortran などのコンパイル言語および Python などの対話型スクリプト言語から同じ枠組みで扱うこと，パソコンや共有メモリ並列計算機など種々の計算環境を柔軟かつ容易に組み合わせることができる．

Implementation and Use of SILC in Shared Memory Multiprocessor Environments

Tamito KAJIYAMA (JST, University of Tokyo), Akira NUKADA (JST, University of Tokyo),
Reiji SUDA (University of Tokyo), Hidehiko HASEGAWA (University of Tsukuba)
and Akira NISHIDA (University of Tokyo)

This paper describes the implementation of SILC (Simple Interface for Library Collections) and its use in shared memory multiprocessor environments. SILC is a framework that allows user programs to easily utilize matrix computation libraries in a language and computing environment independent manner. Using this framework, users can develop their programs with compiler-based languages (e.g., C and Fortran) as well as interactive scripting languages (e.g., Python) in the same way, while making a flexible combination of various computing environments such as sequential personal computers and high-performance shared memory parallel computers.

1　はじめに

連立一次方程式の求解に代表される行列計算は，大規模科学技術計算の実行時間とメモリ使用量の大半を占める主要な処理であり，行列計算を必要とするユーザプログラムの実現を容易にするために多数の行列計算ライブラリが開発されている [2, 4, 7]．行列計算ライブラリの多くは C や Fortran などのコンパイル言語で実現されており，ユーザプログラムにおいては関数呼び出しにより利用する．また，行列計算ライブラリのインタフェースはライブラリ毎に異なるため，ユーザは決められたデータ構造で行列データを用意して所定の引数の順序でライブラリ関数を呼び出さなければならない．

関数呼び出しによる行列計算ライブラリの利用は直感的な反面，ユーザプログラムが特定のライブラリに依存するため，別のライブラリを利用する場合にユーザプログラムを大幅に書き換える必要が生じる．ライブラリの変更は，主として（1）計算環境を変更する場合，および（2）異なる解法および行列の格納形式を採用する場合に必要となる．大規模科学技術計算に用いられる高性能計算環境では特定環境用の行列計算ライブラリが用意されており，ユーザプログラムを別の計算環境に移植する際にライブラリの変更が必要になることがある．また，最も効率のよい解法および格納形式は問題（行列の性質）と計算環境に応じて変わる [1]．そのため，種々のライブラリが提供している解法および格納形式を利用するためにユーザプログラムの修正が必要になる．

関数呼び出しによる行列計算ライブラリの利用は手軽であり，少数の限られたライブラリを特定の計算環境で利用する場合には都合がよい．しかし，多様な計算環境で実行する場合，複数のライブラリの解法および格納形式を利用する場合には，ユーザプログラムの修正が必要であり，ユーザの負担が大きい．

この問題を解決するために，われわれは行列計算ライブラリに対してプログラミング言語と計算環境に依存しない一様なインタフェースを付与する枠組み SILC（Simple Interface for Library Collections）を提案している [3, 6]．SILC の主な特徴を以下に示す．

- 複数の行列計算ライブラリをライブラリ固有のインタフェースによらず統一的に利用できる．
- 多くのプログラミング言語でユーザプログラム

を作成できる．コンパイル言語（C，Fortran 等）と対話型スクリプト言語（Python 等）で同じ枠組みが利用できる．

- 多様な計算環境を利用できる．パソコンでユーザプログラムを実行して並列計算機における計算を遠隔制御するなど，計算環境を柔軟に組み合わせられる．

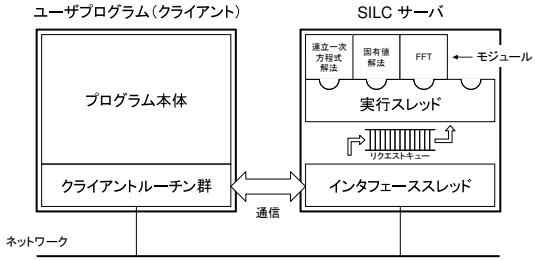


図 1: 共有メモリ並列環境向け SILC のシステム構成

2 SILC の設計と実装

SILC では，(1) 従来の関数呼び出しをデータの授受と計算の指示に分け，(2) 計算を文字列（数式）で指示し，(3) ユーザプログラムから独立したメモリ空間で計算を行なうことにより，ユーザプログラムと行列計算ライブラリの間の依存関係を解消する．SILC のユーザプログラムは，次の 3 つのリクエストを通じて SILC が管理する行列計算ライブラリを利用する．

PUT 行列やベクトルなどのデータに名前を付けて預ける．データはユーザプログラムから独立した別メモリ空間に転送され，指示されるまで削除されない．

EXEC 計算を数式（後述）で指示する．計算は別メモリ空間において非同期に実行され，計算結果は別メモリ空間に保持される．

GET データを受け取る．データはユーザプログラムのメモリ空間に転送される．転送元のデータは指示がない限りそのまま保持される．

C で書かれたユーザプログラムの場合，`SILC_PUT`，`SILC_EXEC`，`SILC_GET` の 3 つのクライアントルーチンを用いて上記のリクエストを送る．

共有メモリ並列環境向け SILC のシステム構成を図 1 に示す．本システムはクライアントサーバモデルで実現されている．ユーザプログラムはネットワークを通じて SILC サーバに接続する．ユーザプログラムから送られたリクエストは SILC サーバ内のインタフェーススレッドにより受信される．PUT リクエストと GET リクエストはインタフェーススレッドにより処理される．また，EXEC リクエストはキューに追加され，実行スレッドにより非同期に処理される．

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，代入文と手続き呼び出し文がある．代入文の左辺には変数名，右辺には式を記述する．変数の型宣言は不要である．式の構成要素には変数名，四則演算，連立一次方程式の求解（“`A\b`” は $A\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x}$ を求める），転置，関数呼び出しなどがある．演算子，関数および手続きは，実行スレッドにおいて適当なライブラリ関数の呼び出しに翻訳されて実行される．ライブラリ関数には予め，インタフェースの差異を吸収するためのラッパーを付与しておく．ラッパーはモジュールとしてまとめられ，SILC サーバの起動時に読み込まれる．

3 SILC の利用

従来の関数呼び出しによるライブラリ関数の利用例を図 2，対応する SILC のユーザプログラムの例を図 3 に示す．図 2 のプログラム中の `ssi_cg` は，連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を CG 法 [1] で解くライブラリ関数 [4] である．関数呼び出しに先立って行列 A とベクトル $\boldsymbol{b}$ の 2 つの入力データをライブラリ固有のデータ構造で準備し，解 $\boldsymbol{x}$ を受け取るための配列および解法を制御するためのパラメタと共に引数に与えている．一方，SILC のユーザプログラムでは，まず `SILC_PUT` で A と $\boldsymbol{b}$ を個別に預け，次に `SILC_EXEC` で求解を指示している[1]．この計算指示は適当なライブラリ関数（例えば `ssi_cg`）の呼び出しに翻訳されて処理される．最後に `SILC_GET` で解 $\boldsymbol{x}$ を受け取る．

図 3 に示した SILC のユーザプログラムには特定ライブラリに特化したコードは一切現れない．使用するライブラリを変更するには，SILC サーバにおける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修正は不要である．

ユーザプログラムの記述言語として，現在，C，Fortran，対話型スクリプト言語 Python が利用できる．どのプログラミング言語でも，PUT，EXEC，GET

[1] 連立一次方程式の求解を表すバックスラッシュは C の文字列中では特殊文字を表記するためのエスケープ文字であるため，文字列中でバックスラッシュを表すには “\\” と記述する．

```
SSI_MATRIX A;
SSI_SCALAR *b, *x, work[N * 6], params[2];
int options[6], status;

行列 A，ベクトル b および x の作成

status = ssi_cg(b, x, work, params, options,
                &A, NULL);
```

図 2: 従来のライブラリ関数の利用例

```
silc_envelope_t A, b, x;

行列 A，ベクトル b および x の作成

SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("x = A \\ b");
SILC_GET(&x, "x");
```

図 3: 図 2 に対応する SILC のユーザプログラム

の 3 つのリクエストを通じてサーバ上のライブラリを利用する仕組みは同じである．また，SILC の命令記述言語はすべてのプログラミング言語から一様に利用でき，記述された数式には可搬性がある．

SILC サーバの実行には逐次計算機および共有メモリ並列環境が利用できる．並列環境における SILC サーバの場合，計算は自動的に並列実行される．計算の並列化には OpenMP を用いている．また，データ通信および文字列による計算指示は比較的負荷の小さい処理なので，SILC のユーザプログラムは廉価な計算機やモバイル環境でも実行できる．

4 実験

従来のライブラリ利用法と比較して，SILC には通信が必要である，メモリ使用量が増える，メモリコピーが増える等のデメリットがある．たとえば，パソコン（Intel Pentium M 733 1.1GHz，主記憶 768MB）で次数 32,000 の三重対角行列を係数とする連立一次方程式を解く場合，図 2 のプログラムの実行時間は 38.568 秒なのに対して図 3 のプログラムは 39.708 秒であり，数パーセントのオーバーヘッドがある．しかし，サーバ側が並列計算機などの高速な計算環境の場合，クライアント側のユーザプログラムを一切書き換えることなく高速性を享受できる．また，SILC サーバはマルチスレッド化されているため計算中にデータを送受信できる．したがって計算時間と通信時間を重ね合わせることにより通信のオーバーヘッドをある程度隠すことができる．

オーバーヘッドを詳細に調べるための例として，SILC の命令記述言語で実現した CG 法のプログラムを図 4 に示す．`silc_envelope_t` は行列やベクトルなどのデータの送受信に用いる構造体であり，メンバ `v` はデータのメモリ領域を指すポインタである．CRS 形式 [1] の疎行列の場合は `row` と `index` にもデータのメモリ領域を設定する．このプログラムと図 3 のプログラムの実行時間を図 5 に示す．後者のプログラムにおける連立一次方程式の求解には図 2 のライブラリ関数 `ssi_cg` を用いた．サーバとユーザプログラムは共に SGI Altix3700（Intel Itanium2 1.3GHz × 32 基，主記憶 32GB）で実行した．係数行列として三重対角行列を CRS 形式で与えた．サーバへの接続およびデータ授受に要した通信時間はすべての場合において 0.1 秒前後であった．そのため，行列の次数が小さく計算時間が短い場合には相対的に通信のオーバーヘッドが顕著になっている．また，2 つのユーザプログラムの実行時間の比は 7.0 倍（次数 1,000 の場合）から 49.9 倍（同 256,000）であった．これらのオーバーヘッドと引き換えに，ユーザは特定のライブラリや格納形式に依存しない可搬性の高い解法プログラムを実現できる．別の格納形式を採用するには `SILC_PUT` の呼び出し部分を変更するだけでよく，数式で記述された解法プログラムの修正は不要である．

5 おわりに

本論文では，行列計算ライブラリインタフェース SILC の設計と実現方法について述べた．SILC を利用することにより，特定の行列計算ライブラリに依存しないユーザプログラムを容易に開発できるようになり，種々の計算環境やプログラミング言語を同じ枠組みの中で柔軟に利用できる．

今後の課題としては，SILC の命令記述言語の単独利用可能なスクリプト言語への拡張，数式に基づく実行時最適化，MPI に基づく分散メモリ並列環境向け SILC の実現が挙げられる．

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」プロジェクトの一部として実施した．

```
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_EXEC("rho_old = 1.0");
  SILC_EXEC("n = length(b)");
  SILC_EXEC("p = zeros(n, 1)");
  SILC_EXEC("x = zeros(n, 1)");
  SILC_EXEC("r = b");
  SILC_EXEC("bnrm2 = 1.0 / norm2(b)");
  for (i = 0; i < n; i++) {
    SILC_EXEC("rho = r' * r");
    SILC_EXEC("beta = rho / rho_old");
    SILC_EXEC("p = r + beta * p");
    SILC_EXEC("q = A * p");
    SILC_EXEC("alpha = rho / (p' * q)");
    SILC_EXEC("r = r - alpha * q");
    SILC_EXEC("nrm2 = norm2(r) * bnrm2");
    SILC_EXEC("x = x + alpha * p");

    /* check convergence */
    object.v = &nrm2;
    SILC_GET(&object, "nrm2");
    printf("iter: %d, res: %e\n", i+1, nrm2);
    if (nrm2 <= EPSILON)
      break;
    SILC_EXEC("rho_old = rho");
  }

  /* get vector x */
  object.v = x;
  SILC_GET(&object, "x");

  /* close connection */
  SILC_END();

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

図 4: SILC の命令記述言語で実現した CG 法のプログラム

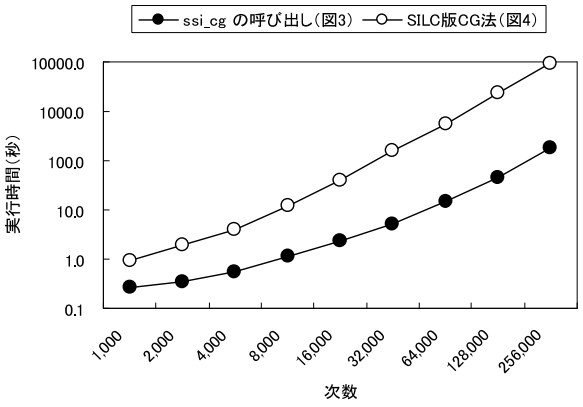


図 5: 2 つのユーザプログラムの実行結果

参考文献

[1] R. Barrett *et al. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994.

[2] J. Dongarra. Freely available software for linear algebra on the Web. http://www.netlib.org/utk/people/JackDongarra/la-sw.html.

[3] 長谷川, 須田, 額田, 梶山, 中島, 高橋, 小武守, 藤井, 西田. 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, Dec. 2004.

[4] H. Kotakemori, H. Hasegawa, T. Kajiyama, A. Nukada, R. Suda, and A. Nishida. Performance evaluation of parallel sparse matrix–vector products on SGI Altix3700. In *IWOMP 2005*, to appear.

[5] P. Luszczek and J. Dongarra. Design of interactive environment for numerically intensive parallel linear algebra calculations. In *ICCS2004*, LNCS 3039, pp. 270–277, 2004.

[6] 西田. 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, Apr. 2004.

[7] K. Wu and B. Milne. A survey of packages for large linear systems. Technical Report LBNL–45446, Lawrence Berkeley National Laboratory, 2000.

第34回数値解析シンポジウム
（2005年6月28日）

共有メモリ並列環境における SILC の実現と利用

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
- 関数呼び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

- ユーザプログラムを修正せずに様々な行列計算ライブラリを利用したい
- SILC の提案
  （Simple Interface for Library Collections）
- 基本アイデア
  - 関数呼び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

SILC によるライブラリの利用例

```
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
```

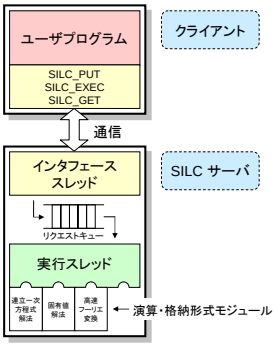
- 関数呼び出しをデータ授受と計算指示に分離
- 計算は文字列（数式）で指示
- ユーザプログラムとは別のメモリ空間で計算を実行

SILC の特徴

- さまざまな行列計算ライブラリを同じ枠組みで呼び出せる
- 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
- 多くの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

実験（1）

- 計算環境：パソコン
  - Intel Pentium M 733 1.1GHz、メモリ 768MB
  - 同じ計算機で SILC サーバを実行
- 次数 32,000 の三重対角行列（CRS 形式）を係数行列とする連立一次方程式を CG 法で解く
  - 反復解法ライブラリの ssi_cg を使用
- 従来のライブラリ呼び出し
  ⇒ 38.568秒
- SILC のユーザプログラム
  ⇒ 39.708秒
  数パーセントの通信オーバーヘッド

SSI Scalable Software Infrastructure

従来のライブラリ関数の利用例（予稿図2）

```
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);
```

対応する SILC のユーザプログラム（予稿図3）

```
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（ssi_cg で求解）
SILC_GET (&x, "x");          解 x を受け取る
```

実験（2）

- 計算環境：SGI Altix3700
  - Intel Itanium2 1.3GHz×32基、メモリ 32GB
  - 同じ計算機で SILC サーバを実行
- SILC の命令記述言語で実現した CG 法のプログラム
- SILC 経由の ssi_cg の呼び出しと比較
- 係数行列：三重対角行列（CRS 形式）

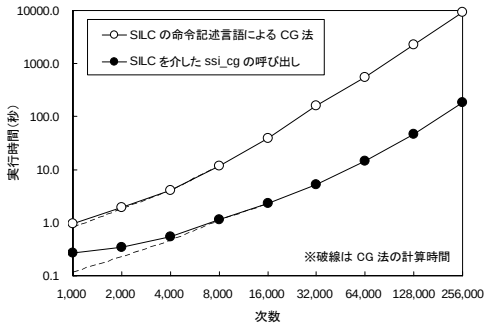
SSI Scalable Software Infrastructure

SILC の命令記述言語で実現した CG 法（予稿図4）

```
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_EXEC("rho_old = 1.0");
  SILC_EXEC("n = length(b)");
  SILC_EXEC("p = zeros(n, 1)");
  SILC_EXEC("x = zeros(n, 1)");
  SILC_EXEC("r = b");
  SILC_EXEC("bnrm2 = 1.0 / norm2(b)");
  for (i = 0; i < n; i++) {
    SILC_EXEC("rho = r' * r");
    SILC_EXEC("beta = rho / rho_old");
    SILC_EXEC("p = r + beta * p");
    SILC_EXEC("q = A * p");
    SILC_EXEC("alpha = rho / (p' * q)");
    SILC_EXEC("r = r - alpha * q");
    SILC_EXEC("nrm2 = norm2(r) * bnrm2");
    SILC_EXEC("x = x + alpha * p");

    /* check convergence */
    object.v = &nrm2;
    SILC_GET(&object, "nrm2");
    printf("iter: %d, res: %e¥n", i+1, nrm2);
    if (nrm2 <= EPSILON)
      break;
    SILC_EXEC("rho_old = rho");
  }

  /* get vector x */
  object.v = x;
  SILC_GET(&object, "x");

  /* close connection */
  SILC_END();

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

SSI Scalable Software Infrastructure

実験（2）の結果



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

SSI Scalable Software Infrastructure

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/