高速フーリエ変換ライブラリ FFTSS

「大規模シミュレーション向けの基盤ソフトウェアの開発」プロジェクト　高速関数変換グループ

FFTSS ライブラリとは

現在高速フーリエ変換(FFT)は様々な分野で高速化に用いられている代表的なアルゴリズムのひとつである。「大規模シミュレーション向けの基盤ソフトウェアの開発」プロジェクトの高速関数変換グループの主たるソフトウェア成果物としてFFTSSライブラリを公開している。

目標

- □様々なプラットフォームに対応
- □性能において他のライブラリ(ベンダー提供含む)との競争力

対象

- □キャッシュメモリを搭載するスーパースカラ型のCPU

オートチューニングの探索範囲

- □各種拡張命令の使用／不使用
- □FFTカーネルの基底の優先度
- □各基底のカーネルの計算順序
- □階層メモリへの最適化(多次元FFT)
- □最適なMPI関数を選択(MPI版)

オープンソースなライブラリとして以下のサイトからダウンロード可能。

http://www.ssisc.org/fftss/

※公式サイトのURLが変更になりました

更新履歴

平成17年9月　Version 1.0 公開
- これまでの研究成果をまとめてライブラリの形態で公開。
- 自動チューニング機構を搭載
- 最適なカーネル、基底(順序含む)を選択
- 改良型8基底カーネルを使用
- Intel SSE2/3 命令, 積和演算命令(FMA)に対応
- FFTWとの互換性

平成18年5月　同 update 20051231 公開
- Blue Gene 用のFFTカーネルの追加
- FFTWとの互換性の強化(関数の追加)
- Microsoft Visual Studio のサポート(Windowsに対応)
- タイマーの精度向上(パフォーマンスカウンタ等)

平成18年11月 Version 2.0 公開
- 2次元FFT、3次元FFTのサポート
- 多次元FFTのOpenMP並列化
- マニュアルを公開

平成19年10月 Version 3.0 公開
- **2次元FFTのMPI並列化 (OpenMPのハイブリッド並列化)**
- **日本語版マニュアルを公開**

コードの実装方針

コンパイラが最適化しやすいコードを用意。

特にベンダー提供コンパイラの高度なループ最適化に期待。

□ポインタ間のエイリアシング問題を解決
- ◆restrict (C99 キーワード)
- ◆disjoint (XLC用のpragma)
- ◆-fno-alias, -qansialias 等のコンパイラオプション

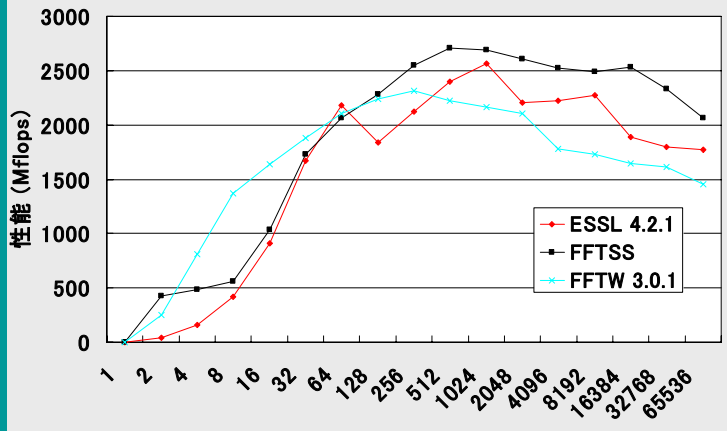
□ループの反復回数が小さい場合のみアンローリングを行う。
- 最内側ループの反復回数が4以下のとき。

□拡張命令に明示的に対応。
- コンパイラオプション付加による自動的な対応には限界あり。

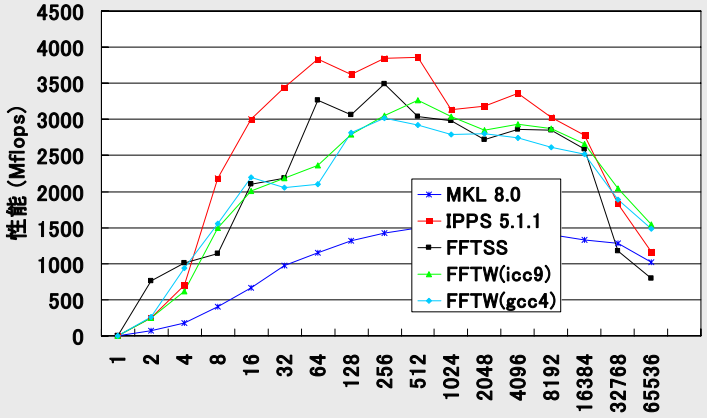
1次元FFTの性能

IBM POWER5 と Intel Pentium4 のシステムでの1次元FFTの性能を他の各種ライブラリと比較.(Flops値は演算量 $5N\log_2 N$ より算出)

1D-FFT on IBM POWER5 1.65GHz (Linux, 32bit)



1D-FFT on Intel Xeon 2.8 GHz (EM64T, Linux)



バージョンと機能の対応表

	Serial	OpenMP	MPI
1-D FFT	○	×	×
2-D FFT	2.0以降	2.0以降	3.0以降
3-D FFT	2.0以降	2.0以降	×

MPI 版ではさらにノード内がOpenMP並列化.

MPI ライブラリは MT-Unsafe であると仮定.

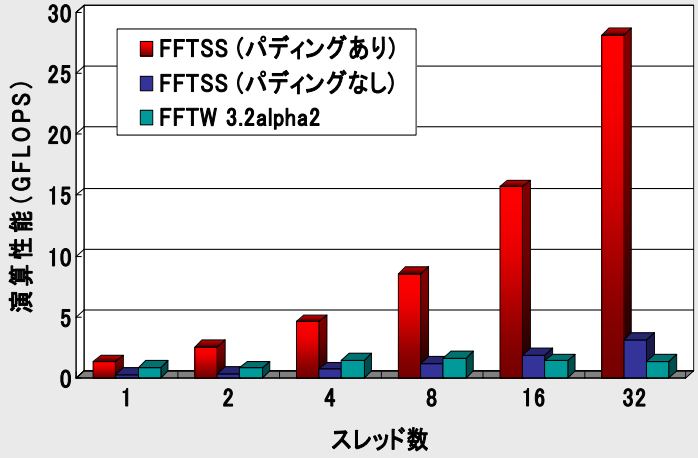
Altix 3700 上での性能評価

Intel Itanium 2 1.3 GHz ×32, 32GB Main Memory, cc-NUMA.

Linux 2.4, SGI Pro Pack 2.1, Intel Compiler 9.1.42.

OpenMP と MPIの両方の並列化が可能.

OpenMP版2次元FFT (4096²)



32PE のスピードアップは 21.1 倍. (最大 28.1 GFLOPS)

FFTSS の場合パディングの有無によって大きな性能差.

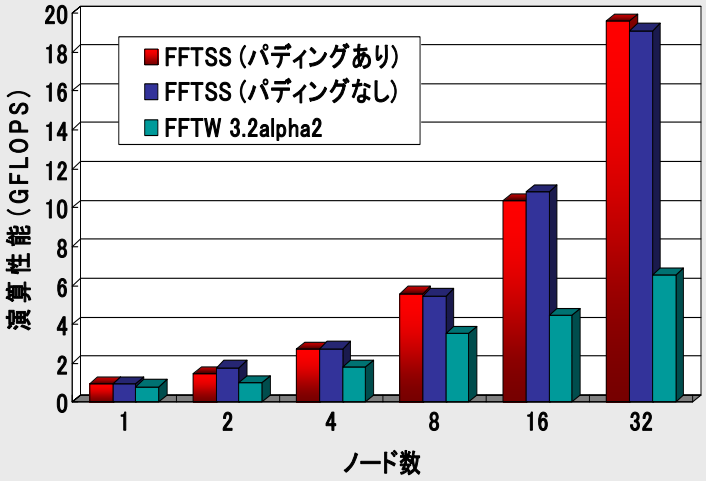
パディングありの場合には4096×4096の入出力データを

C言語: double complex V[4096][4096+1];

Fortran: DOUBLE COMPLEX V(4096+1,4096)

のような配列に格納.

MPI版2次元FFT (4096²)



32PE のスピードアップは 21.2 倍. (最大 19.6 GFLOPS)

入出力データの格納方式でパディングの有無の影響は少ない.

(ライブラリ内部で利用するバッファは常にパディングあり)

OpenMP版並列2次元FFT

2次元FFTではX,Y軸方向にそれぞれ1次元FFTを計算.

```
#pragma omp parallel for
  for (i = 0; i < NY; i++)
    X軸方向の変換
#pragma omp parallel for schedule(※1)
  for (i = 0; i < NX; i++)
    Y軸方向の変換
```

L2 共有／非共有とスケジューリング(※1)の関係(数値はGFLOPS)

	block	cyclic
Intel Itanium 2 (L2非共有)	28.11	2.78
Sun UltraSPARC III (L2非共有)	0.69	0.49
Intel Core 2 Duo (L2共有)	2.46	2.75

Y軸方向のデータを主記憶から読む場合, その転送効率はL2キャッシュが共有か否かの影響を受ける.

L2キャッシュがコア間で共有 ⇒ サイクリック分割

L2キャッシュがコア間で独立 ⇒ ブロック分割

外部発表

[1] 額田彰, 西田晃, 小柳義夫,「分散共有メモリを用いた並列 FFT とその最適化」, 情報処理学会論文誌「コンピューティングシステム」, Vol. 44, No. SIG 6 (ACS 1), pp.1-8, 2003.

[2] 額田彰, 西田晃, 小柳義夫,「積和演算命令に適した新しい8基底 FFT カーネル」, 2004年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論文集, pp.17-24, 2004.

[3] 額田彰, 高橋大介, 須田礼仁, 西田晃,「多様な計算環境で高性能を実現するFFTライブラリFFTSS」, 2006年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論文集, pp.33, ポスター, 2006.

[4] A. Nukada. FFTSS: A HIGH PERFORMANCE FAST FOURIER TRANSFORM LIBRARY, In Proceedings of the 2006 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2006), Vol. III, pp.980-983, IEEE, 2006.

[5] A. Nishida, H. Kotakemori, T. Kajiyama, and A. Nukada. Scalable Software Infrastructure Project. In Proceedings of the IEEE/ACM SC06 Conference, poster, November 11-17, 2006.

[6] A. Nishida, H. Kotakemori, A. Fujii, and A. Nukada, Development of Scalable Software Infrastructure on Blue Gene Systems, In Report on the Jülich Blue Gene/L Scaling Workshop 2006 (FZJ-ZAM-IB-2007-02), John von Neumann Institute for Computing, Research Centre Juelich, pp.13-14, February, 2007.

[7] 額田彰, 西田晃,「地球シミュレータを用いた高性能2次元 FFT」, 2007年先進的計算基盤システムシンポジウム論文集, pp.137-144, 2007.

[8] 蓬来祐一郎, 額田彰,「Performance Evaluation of CONV3D with nVidia CUDA」, 2007年先進的計算基盤システムシンポジウム論文集, pp.156-157, ポスター, 2007.

[9] A. Nukada, Y. Hourai, A. Nishida and Y. Akiyama. High Performance 3D Convolution for Protein Docking on IBM Blue Gene, In Proceedings of the Fifth International Symposium on Parallel and Distributed Processing and Applications (ISPA07), Lecture Notes in Computer Science 4742, pp.958-969, Springer, 2007.

[10] A. Nukada, D. Takahashi, R. Suda, and A. Nishida. High Performance FFT on SGI Altix 3700, In Proceedings of the Third International Conference on High Performance Computing and Communications (HPCC 2007), Lecture Notes in Computer Science 4782, pp.396-407, Springer, 2007.

反復解法ライブラリ Lis

「大規模シミュレーション向けの基盤ソフトウェアの開発」プロジェクト　反復解法研究グループ

Lis（a Library of Iterative Solvers for linear systems）とは

- 大規模実疎行列係数の線型方程式 Ax = bに対する反復解法ライブラリ
- C言語とFortran 90で記述
- 逐次，OpenMP共有メモリ並列，MPI単独，あるいはOpenMP＋MPIのハイブリッドで利用可能

特徴

- 20通りの反復解法，11 通りの前処理が組合わせて利用できる
- 11通りの疎行列格納形式が利用できる
- 逐次環境から共有メモリまたは分散メモリ環境への移行はプログラムの変更なし（あるいはごくわずかの変更）でよい
- 4倍精度演算に対応

ソフトウェアの公開

- 動作環境
  - Linux，Windows，MacOS X, Solaris
  - Intel Compiler, IBM XL Compiler, Sun ONE Studio, GNU
  - X86互換, Intel64/AMD64, IA-64, POWER, SPARC
- 更新履歴
  - Lis-1.1.0　　2007年10月31日
    - 4倍精度演算の追加
    - FORTRANユーザインターフェイスの追加
    - 非対称SA-AMG前処理（行列構造は対称）の追加
    - Crout版ILU前処理の追加
    - additive Schwarz前処理の追加
    - ユーザ定義前処理の追加
    - Harwell-Boeingファイルフォーマットの追加
    - Lis-AMGをLisへ統合
    - Autoconfの導入
  - Lis-1.0.2　　2006年04月17日
  - Lis-1.0.1　　2005年11月06日
  - Lis-1.0.0　　2005年09月20日
- URL　http://www.ssisc.org/lis/

lis　検索

実装されている解法・前処理・格納形式

解法

1.0.x	CG	1.1.0	CR
	BiCG		BiCR
	CGS		CRS
	BiCGSTAB		BiCRSTAB
	BiCGSTAB(l)		GPBiCR
	GPBiCG		BiCRSafe
	Orthomin(m)		FGMRES(m)
	GMRES(m)		
	TFQMR		
	Jacobi		
	Gauss-Seidel		
	SOR		

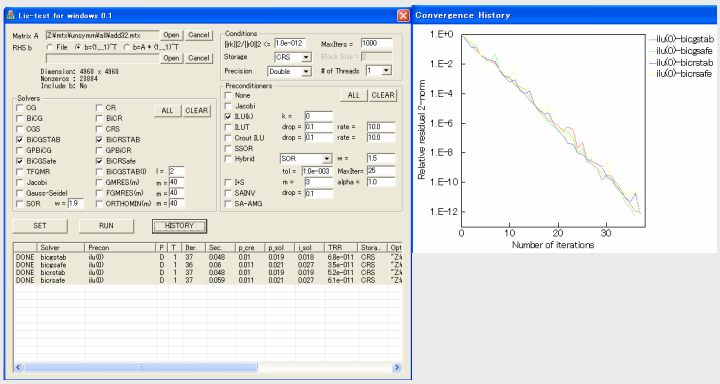
前処理

1.0.x	Jacobi
	ILU(k)
	SSOR
	Hybrid
	I+S
	SA-AMG
	SAINV
1.1.0	Crout版ILU
	ILUT
	Additive schwarz
	ユーザ定義前処理

行列格納形式

Point	Compressed Row Storage	Point	Dense
	Compressed Column Storage		Coordinate
	Modified Compressed Sparse Row	Block	Block Sparse Row
	Diagonal		Block Sparse Column
	Ellpack-Itpack generalized diagonal		Variable Block Row
	Jagged Diagonal Storage		

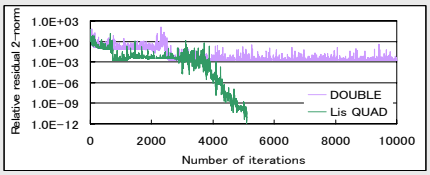
Lis-test for Windows



- いろいろな組み合わせが手軽に実行できる（結果はファイルに）
- 収束履歴グラフもボタン一つで表示できる
- GUIと実行形式の2つのファイル

4倍精度演算

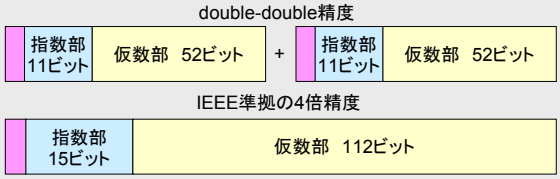
- 4倍精度演算の収束性



デバイスシミュレータで現れる行列（37,054次元）をILUC-BiCGSafe法で解く

倍精度では収束しないが、4倍精度なら収束

- Lisでの実装方針
  - 倍精度と同一のインタフェース
    - 入力（係数行列A，右辺ベクトル b，初期ベクトルx0 ）は倍精度、出力 解xは倍精度
    - 解法中の解 x，補助ベクトル,スカラーは4倍精度
    - 前処理行列Mの生成部分は倍精度演算
    - 反復中のMu = v の求解は4倍精度
- double-double精度演算を採用
  - 倍精度浮動小数を2個用いて倍精度の四則演算の組み合わせで4倍精度を実現
  - FORTRAN REAL*16より高速
  - 仮数部がIEEE準拠より8ビット少ない
  - 有効桁数　double-double精度　約31桁
    IEEE準拠4倍精度　約33桁



- 実装方法
  - 反復解法を4倍精度演算に置き換える
    - 行列ベクトル積（matvec）
    - ベクトルの内積（dot）
    - ベクトルおよびその実数倍の加減（axpy）
  - SSE2を活用した高速化
    - 組み込み関数を利用
    - matvec, dot, axpyのループを2段のループアンローリング

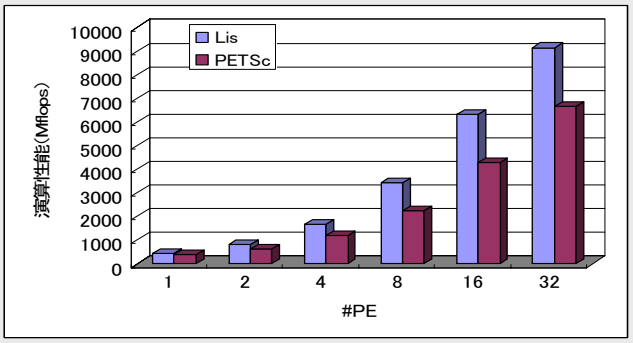
PETSc との比較

	Lis	PETSc(2.3.0)
反復解法	20	15
直接解法	×	○
前処理	10	10(外部+10)
演算精度	実のみ	実、複素
環境	逐次, MPI OpenMP MPI+OpenMP	逐次, MPI
API	C, FORTRAN	C,C++,FORTRAN
その他	**4倍精度**	非線形解法

Lisの性能評価

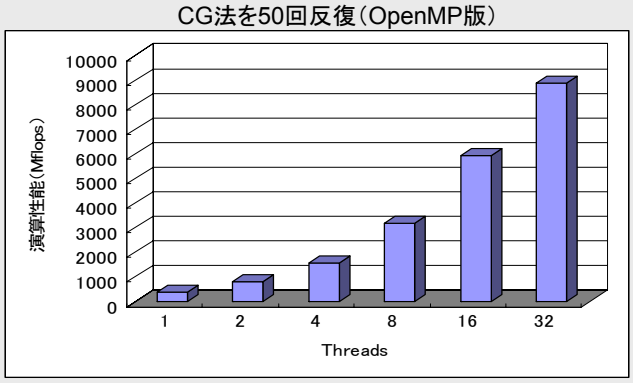
- 計算環境
  - SGI Altix 3700　　Itanium2 1.3GHz x 32
  - Compiler　　Intel Compiler 9.1
- 行列
  - 3次元ポアソン方程式を有限要素法で離散化
    - 次数100万, 非零要素数26,207,180
  - 行列格納形式はCRS

CG法を50回反復(MPI版)



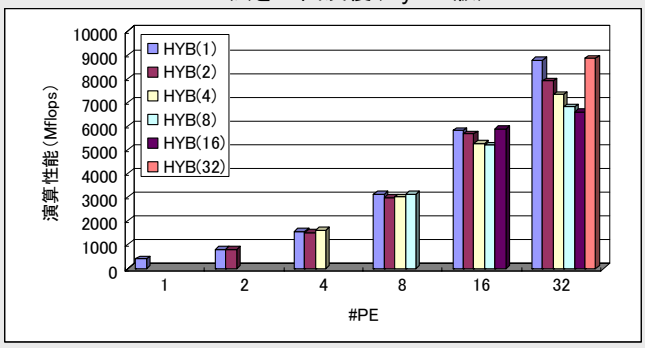
- 32PEのときLisのほうがPETScより27%高速
- 32PEのスピードアップは21.4倍

CG法を50回反復(OpenMP版)



- 32PEのスピードアップは22.0倍

CG法を50回反復(Hybrid版)

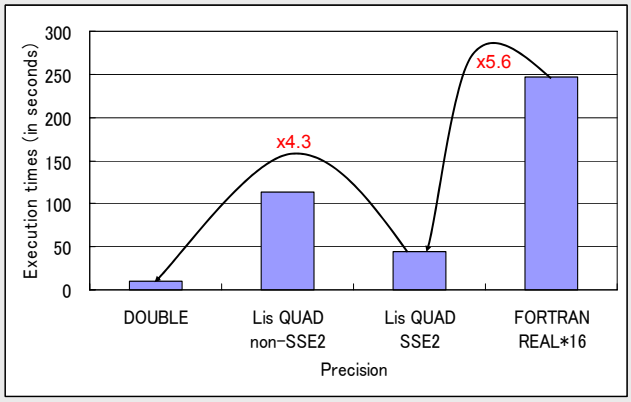


- AltixではMPI+OpenMPは10～25%性能低下

4倍精度演算の性能

- 計算環境
  - CPU　　Core 2 Duo 2.4GHz
  - Compiler　　Intel Compiler 9.1
- 行列
  - 3次元ポアソン方程式を有限要素法で離散化
    - 次数100万, 非零要素数26,207,180
  - 行列格納形式はCRS

BiCG法を50回反復

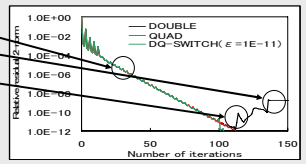


DQ-SWITCHアルゴリズム

- 倍精度と高速な4倍精度の組み合わせ
- 計算時間の短縮のため4倍精度演算の削減
  - 倍精度の解(あるいは途中の値)を初期値として4倍精度で反復
  - リスタート時に渡すのは倍精度演算で生成された解x_kのみ
  - 適切な restart_tol の選択が問題

```
for(k=0;k<最大反復回数;k++){
   倍精度演算の反復解法
   if( nrm2<restart_tol ) break;
}
x以外の作業変数をゼロクリア
for(k=k+1;k<最大反復回数;k++) {
   4倍精度演算の反復解法
}
```

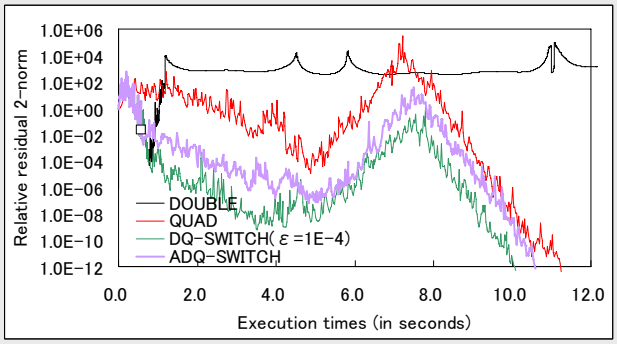
- リスタートのパラメータを自動的に定める
  - 残差の振る舞いはある地点での近傍で
    - (C)Convergent
    - (S)Stagnate
    - (D)Divergent
  - DまたはSを検出後リスタート
  - ある地点での残差ノルムを基点とした偏差を計算



$$\nu = \frac{1}{p}\sum_{i=1}^{p}\left(\frac{nrm(i)-nrm(1)}{nrm(1)}\right)^2$$

D 発散 $\nu \geq 10^2$　　S 停滞 $\nu \leq 10^{-1}$

- 数値例
  - ホール効果やイオンスリップといった効果を含めた一般化されたオームの法則から生じる電場ポテンシャル問題(次数:23,994)



- 停滞または発散を検出し、QUADよりも実行時間を短縮

研究成果

[1] 小武守, 藤井, 長谷川, 西田. 倍精度と4倍精度の混合型反復法の提案, 2007年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, pp.9-16, 2007.

[2] A. Nishida, H. Kotakemori, T. Kajiyama, A. Nukada. Scalable Software Infrastructure Project, In SC06, poster, 2006.

[3] 小武守, 藤井, 長谷川, 西田. SSE2を用いた反復解法ライブラリLis 4倍精度版の高速化, 情報処理学会研究報告2006-HPC-108, pp.7-12, 2006.

[4] 小武守, 西田. キャッシュを考慮した疎行列格納形式CJDSの提案と評価, 2006年先進的計算基盤システムシンポジウム論文集, pp.167-173, 2006.

SILC: Simple Interface for Library Collections

「大規模シミュレーション向けの基盤ソフトウェアの開発」プロジェクト　実装手法研究グループ

概要

行列計算ライブラリを使いやすくするインタフェース SILC（Simple Interface for Library Collections）を提案し，逐次・共有メモリ並列環境向けおよび分散並列環境向けシステムをクライアント・サーバ方式で実現した．本システムにより，ユーザプログラムを行列計算ライブラリや計算環境に依らない方法で作成することが可能になる．

従来法の問題点

ライブラリ固有のインタフェース（API）に基づいてユーザプログラムを作成すると，ユーザプログラムが特定のライブラリに依存する．そのため，別のライブラリや計算環境を利用するにはユーザプログラムの大幅な修正が必要である．

SILC の基本アイデア

1. ライブラリ呼出しを**データ転送**と**計算の指示**に分離
2. 計算を**文字列（数式）**で指示
3. ユーザプログラムから**独立したメモリ空間**で計算を実行

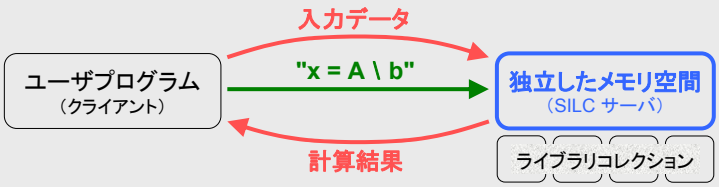


図1　SILC の基本アイデア

◆ 従来のユーザプログラム（図2）

ライブラリ固有のデータ構造で行列やベクトルなどの入力データを用意する．次に，所定の関数名と引数の順序でライブラリ関数を呼び出す．この方法は直感的で手軽な反面，ユーザプログラムが特定のライブラリに依存する．

```
double *A, *b;
int kl, ku, lda, ldb, nrhs, info, *ipiv;

/* 行列 A およびベクトル b の作成 */

dgbtrf (N, N, kl, ku, A, lda, ipiv, &info);  /* LU 分解 */
if (info == 0)
    dgbtrs ('N', N, kl, ku, nrhs, A, lda, ipiv, b, ldb, &info);  /* 求解 */
```

図2　従来のユーザプログラム
（LAPACK による連立一次方程式 $Ax = b$ の求解例）

◆ SILC のユーザプログラム（図3）

SILC のサポートするデータ構造で入力データを用意する．各データに名前を付けて SILC_PUT ルーチンでサーバに預ける．必要なデータを預けたあと，SILC_EXEC により計算を文字列（数式）で指示する．計算終了後，SILC_GET で結果を受け取る．

```
silc_envelope_t A, b, x;

/* 行列 A およびベクトル b の作成 */

SILC_PUT ("A", &A);
SILC_PUT ("b", &b);
SILC_EXEC ("x = A \ b");  /* 適切な求解ルーチンが呼び出される */
SILC_GET (&x, "x");
```

図3　SILC のユーザプログラム
（図2のユーザプログラムと同じ計算を実現する例）

SILC の特徴

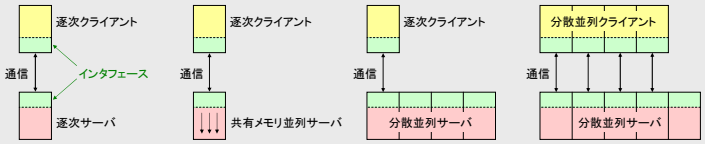
- ◆ ユーザプログラムを行列計算ライブラリや計算環境に依存しない方法で作成できる
  - □ さまざまな行列計算ライブラリを同じ方法で呼び出せる
  - □ ユーザプログラムを修正せずに別の計算環境を利用できる
  - □ 解法，行列の格納形式，演算精度を容易に変更できる
- ◆ 最小限の入力データと記憶領域を用意するだけでよい
  - □ 計算途中で必要な記憶領域は自動的に確保・解放される
- ◆ さまざまなプログラミング言語で利用できる
  - □ C/C++，Fortran，Java，Python，GNU Octave など

SILC のシステム構成

- ◆ クライアント・サーバ方式に基づく2種の実装
  - □ OpenMP 版：逐次・共有メモリ並列環境向け
  - □ MPI 版：分散並列環境向け
- ◆ ユーザプログラムは逐次プログラムまたは分散並列（MPI）の並列プログラム
- ◆ SILC サーバは共有メモリ並列（OpenMP）または分散並列（MPI）の並列プログラム

表1　サポートする計算環境の組み合わせ

構成	ユーザプログラム（クライアント）	SILC サーバ	対応バージョン
（1）	逐次	逐次	OpenMP 版
（2）	逐次	共有メモリ並列	OpenMP 版
（3）	逐次	分散並列	MPI 版
（4）	分散並列	分散並列	MPI 版



構成（1）　構成（2）　構成（3）　構成（4）

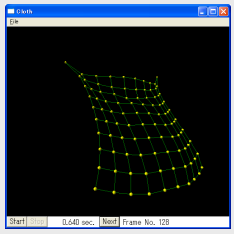
図4　SILC のシステム構成

行列計算ライブラリインタフェースの機能

- ◆ データ構造
  - □ データ型：スカラー，ベクトル，行列，3次元配列
  - □ 行列の格納形式：密，帯，CRS（行圧縮）形式，JDS（Jagged Diagonal Storage）形式
  - □ 演算精度：整数，実数，複素数（単精度および倍精度）
- ◆ 命令記述言語
  - □ 文（statement）
    - – 右辺が式である代入文（単純代入文および累算代入文）
    - – 引数が式である手続き呼出し文
  - □ 式（expression）の構成要素
    - – 変数名，定数（i, e, pi など）
    - – リテラル（すべてのデータ型のリテラルを記述可能）
    - – 四則演算（+, −, *, /, %），連立一次方程式の求解（A \ b）
    - – 要素ごと乗算（*@）および要素ごと除算（/@）
    - – 共役転置（A'），複素共役（A~），転置（A'~）
    - – 関数（sqrt, zeros, ones, sparse, diag など）
  - □ 添字（subscript）による部分参照および制約代入
    - – 例："A[1:5,1:5]" は行列Aの部分行列（5行5列）を表す

応用事例：布のシミュレーション

- 布を質点バネモデルで表して布の動きを陰的オイラー法で計算
  - 時間ステップ毎に疎行列を係数とする連立一次方程式を解く
  - 反復解法ライブラリの Lis の前処理なし共役勾配法（CG法）を利用
  - OpenGL で計算結果を可視化
- 2種の SILC のユーザプログラムを作成
  - **LTD 版**: Lis の呼出しに SILC を利用
  - **CMP 版**: シミュレーション全体を SILC の数式で記述



```
LIS_MATRIX A; LIS_VECTOR b, dv;

for (k = 1; k <= n_steps; k++) {
    ⋮
  /* 2. 連立一次方程式 AΔv = b の求解 */
  lis_solve(A, b, dv,
            lis_params,
            lis_options,
            lis_status);
    ⋮
}
```

図4　従来法のプログラム

```
silc_envelope_t A, b, dv;

for (k = 1; k <= n_steps; k++) {
    ⋮
  /* 2. 連立一次方程式 AΔv = b の求解 */
  SILC_PUT("A", &A);
  SILC_PUT("b", &b);
  SILC_EXEC("dv = A \\ b");
  SILC_GET(&dv, "dv");
    ⋮
}
```

図5　SILC（LTD 版）のプログラム

```
silc_envelope_t v, x;

for (k = 1; k <= n_steps; k++) {
  /* 1. 力の計算 */
  SILC_EXEC("p = Y * x");
  SILC_EXEC("z = sqrt(X_T * (p *@ p))");
  SILC_EXEC("fij = p *@ (X * (K_stiff *@ (z - L) /@ z))");
  SILC_EXEC("dij = (Y * v) *@ (X * K_damp)");
  SILC_EXEC("f = Mg - Y_T * (fij + dij)");

  SILC_EXEC("zhat = ones(s, 1) /@ z");
  SILC_EXEC("pzhat = p *@ (X * zhat)");
  SILC_EXEC("U_L = sparse(U_L_row, U_col, pzhat, 3*n, s)");
  SILC_EXEC("U_R = sparse(U_R_row, U_col, pzhat, 3*n, s)");
  SILC_EXEC("tmp = sqrt(zhat *@ K_stiff *@ L)");
  SILC_EXEC("A2 = Y_T * diag(X * tmp); T2 = A2 * A2'");
  SILC_EXEC("A3 = (U_L - U_R) * diag(tmp); T3 = A3 * A3'");
  SILC_EXEC("DfDx = T1 - T2 + T3");

  /* 2. 連立一次方程式 AΔv = b の求解 */
  SILC_EXEC("A = M + (dt * dt) * DfDx + dt * DfDv");
  SILC_EXEC("b = dt * (f - dt * (DfDx * v))");
  SILC_EXEC("dv = A \\ b");

  /* 3. 速度と位置の更新 */
  SILC_EXEC("v += dv *@ fixed");
  SILC_EXEC("x += dt * v");
  SILC_GET(&v, "v");
  SILC_GET(&x, "x");
}
```

図6　SILC（CMP 版）のプログラム

- 数値実験
  - 質点数 10,000，係数行列の次数 30,000，非零要素数 1,497,780
  - 従来法のプログラムの実行時間 11.80 秒（81.70% が Lis の実行時間）
  - ユーザプログラム，SILC サーバ（Local）：Dell Dimension 8400（Intel Pentium 4 3.4 GHz, メモリ 1GB, Windows XP SP2）
  - SILC サーバ（Remote）：SGI Altix 3700（Intel Itanium 2 1.3 GHz 32基, メモリ 32 GB, Red Hat Linux Advanced Server 2.1）
  - ネットワーク：Gigabit Ethernet LAN

表2　LTD 版の実行時間（秒）

	Local	Remote					
Number of threads	-	1	2	4	8	16	32
Client-side computation	1.94	2.13	2.17	2.08	2.11	2.10	2.04
Data transfer	3.36	4.11	4.05	4.21	5.24	5.20	5.59
Server-side computation	9.98	9.58	6.29	3.61	1.57	1.03	1.57
Total execution time	15.28	15.82	12.52	9.90	8.92	8.33	**9.20**
Speedup	-	1.00	1.26	1.60	1.77	1.90	1.72

表3　CMP 版の実行時間（秒）

	Local	Remote					
Number of threads	-	1	2	4	8	16	32
Data transfer	1.66	2.35	1.71	1.23	1.05	1.06	1.40
Server-side computation	434.91	335.09	238.49	114.67	56.46	32.23	23.27
Total execution time	436.57	337.44	240.20	115.90	57.51	33.28	**24.67**
Speedup	-	1.00	1.40	2.91	5.87	10.14	13.68

まとめ

- 計算環境やプログラミング言語に依存しない行列計算ライブラリインタフェース SILC を提案した
- 逐次・共有メモリ並列環境向けおよび分散並列環境向けシステムをクライアント・サーバ方式で実現した
- 本システムにより種々のライブラリ，計算環境，解法，行列の格納形式，演算精度を柔軟かつ容易に利用できる
- 高性能並列計算機を遠隔利用することにより，通信コストを払っても速度向上が得られる

今後の課題

- 主な行列計算ライブラリを呼び出すためのモジュール（ラッパー）
- クライアント・サーバ方式に依らない SILC システムの実装
  - ジョブ管理システムに基づく大規模並列計算環境への対応
- 数式処理（cf. CMP 版のユーザプログラム）の性能向上
- 数式に基づく実行時最適化
- 自動チューニング（automatic performance tuning）機能

ソフトウェア成果物の公開

- OpenMP 版 SILC のソースコード，Windows 用バイナリパッケージ，ドキュメント，サンプルプログラムをフリーソフトウェアとして一般公開している
- ホームページ：**http://www.ssisc.org/silc/**
- 動作環境：Unix, GNU/Linux, Mac OS X, Microsoft Windows
- 公開履歴
  - SILC バージョン1.0（2005年9月20日）
  - SILC バージョン1.1（2005年11月25日）
  - SILC バージョン1.2（2006年11月12日）
  - SILC バージョン1.3（2007年10月31日）

主な研究成果

[1] T. Kajiyama, A. Nukada, R. Suda, H. Hasegawa, and A. Nishida. Toward Automatic Performance Tuning for Numerical Simulations in the SILC Matrix Computation Framework. In Proc. 2nd International Workshop on Automatic Performance Tuning (iWAPT 2007), pp.81-90, 2007.

[2] T. Kajiyama, A. Nukada, R. Suda, H. Hasegawa, and A. Nishida. Cloth Simulation in the SILC Matrix Computation Framework: A Case Study. In Proc. 7th International Conference on Parallel Processing and Applied Mathematics (PPAM 2007), Lecture Notes in Computer Science (LNCS), in press.

[3] T. Kajiyama, A. Nukada, R. Suda, H. Hasegawa, and A. Nishida. Numerical Simulations in the SILC Matrix Computation Framework. In Proc. International Conference on Computational Methods 2007 (ICCM 2007), p.235, 2007.

[4] 梶山. 行列計算ライブラリインタフェース SILC によるアプリケーション開発, 第1回 JST 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」プロジェクトワークショップ, December 27, 2006.

[5] A. Nishida, H. Kotakemori, T. Kajiyama, A. Nukada. Scalable Software Infrastructure Project, In SC06, poster, 2006.

[6] T. Kajiyama, A. Nukada, R. Suda, H. Hasegawa, A. Nishida. A Performance Evaluation Model for the SILC Matrix Computation Framework. In Proc. IFIP International Conference on Network and Parallel Computing (NPC 2006), pp.93-103, 2006.

[7] T. Kajiyama, A. Nukada, R. Suda, H. Hasegawa, A. Nishida. Distributed SILC: An Easy-to-Use Interface for MPI-based Parallel Matrix Computation Libraries. In Proc. 2006 Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'06), LNCS 4699, pp.860-870, 2007.

[8] T. Kajiyama, A. Nukada, H. Hasegawa, R. Suda, A. Nishida. LAPACK in SILC: Use of a Flexible Application Framework for Matrix Computation Libraries. In Proc. 8th International Conference on High Performance Computing in Asia Pacific Region (HPC Asia 2005), pp.205-212, 2005.

[9] T. Kajiyama, A. Nukada, H. Hasegawa, R. Suda, A. Nishida. SILC: A Flexible and Environment Independent Interface to Matrix Computation Libraries. In Proc. 6th International Conference on Parallel Processing and Applied Mathematics (PPAM 2005), LNCS 3911, pp.928-935, 2006.