The Sixth International Conference on
Parallel Processing and Applied Mathematics
(PPAM 2005)

SILC: a Flexible and Environment Independent Interface to Matrix Computation Libraries

Tamito KAJIYAMA [1, 2] Akira NUKADA [1, 2]
Hidehiko HASEGAWA [3, 1] Reiji SUDA [2, 1] Akira NISHIDA [2, 1]

1. CREST, Japan Science and Technology Agency (JST), Japan
2. The University of Tokyo, Japan
3. University of Tsukuba, Japan

Outline

- Background
  - Matrix computation libraries
  - Traditional programming style based on function calls
- Proposal of SILC
  - Simple Interface for Library Collections
  - How SILC works
- Design and Implementation of SILC
- Experimental Results
- Future work

Background

- Matrix computations
  - Fundamental components in large-scale scientific applications
    - Taking a major proportion of execution time and memory resources
    - Long computation time with relatively small data
  - **Matrix computation libraries**
    - Facilitating rapid development of **user programs**
    - A few examples of libraries: LAPACK, IMSL, NAG

The traditional way of using libraries

1. Preparation of matrices and vectors using library-specific data structures
2. Function calls with a function's name and its arguments in a prescribed order

As a result...

- User programs will depend on a specific library
  - Not easy to replace the library by another

You need to use other libraries

- When user programs need to be ported to other computing environments
  - Required to use environment-specific libraries
- When solvers and matrix storage formats in other libraries are necessary
  - The best solver and matrix storage format depend on:
    - The problem to be solved
    - The computing environment in use

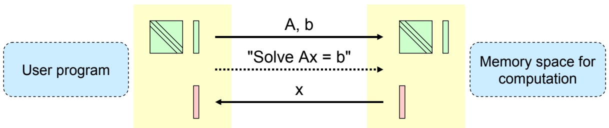
An example in the traditional way

```
SSI_MATRIX A;
SSI_SCALAR *b, *x, work[N*6], params[2];
int options[6], status;
/* Create matrix A and vector b, allocate buffer for x */
status = ssi_cg (b, x, work, params, options, &A, NULL);
```

- A user program to solve $Ax = b$
- Using a library-specific function and data structures
- A source-level dependency upon the library
  - Switch of libraries requires a number of modifications to the user program

Proposal of SILC

- Simple Interface for Library Collections
  - Separating a function call into data transfer and a request of computation
  - Requesting the computation by means of mathematical expressions in the form of text
  - Using separate memory space to carry out the requested computation



An example in SILC

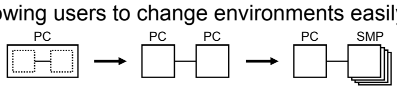
```
silc_envelope_t A, b, x;  /* as in Ax = b */
/* Create matrix A and vector b, allocate buffer for x */
SILC_PUT ("A", &A);
SILC_PUT ("b", &b);
SILC_EXEC ("x = A \ b"); /* Call a solver (e.g., ssi_cg) */
SILC_GET (&x, "x");
```

- Data transfer and a request of computation
- Mathematical expressions in the form of text
- Computation in separate memory space

→ Independent of any specific library and environment

Main benefits of using SILC

- User programs are independent of libraries
  - Allowing users to change environments easily



- Only the smallest amount of data is needed
  - Temporary buffers for computation are automatically allocated in separate memory space
- Mathematical expressions are well-defined and language-independent
  - Fit for use in many computing environments with various programming languages (C, Fortran, Python)

Functionalities

- Data types: scalar, vector, matrix, cubic array
- Precisions: integer, real, complex (single/double)
- Matrix storage formats: dense, band, CRS
- Mathematical expressions
  - Statements: assignments, procedure calls
  - Components of a statement
    - Binary arithmetic operators (+, −, *, /, %)
    - Solution of systems of linear equations (A \ b)
    - Transposition (A'), complex conjugate (A~)
    - Functions (e.g., "sqrt(b' * b)" is the 2-norm of vector b)
    - Subscript (e.g., "A[1:5, 1:5]" is a 5×5 submatrix of A)

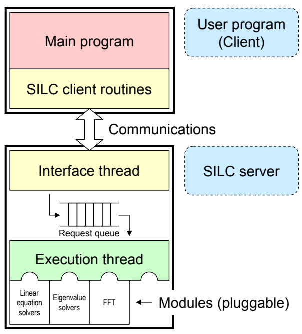
How to use alternative solvers

- Alternative solvers as separate modules
  - One module for each solver
  - The "prefer" statement to specify a preferred module
- An example: a comparison of two solvers

```
SILC_EXEC ("prefer leq_lu");
SILC_EXEC ("x1 = A \ b");   /* solved by LU decomposition */
SILC_EXEC ("prefer leq_cg");
SILC_EXEC ("x2 = A \ b");   /* solved by the CG method */
SILC_EXEC ("d = b − A * x1; norm1 = sqrt(d' * d)");   /* ||b − Ax1|| */
SILC_EXEC ("d = b − A * x2; norm2 = sqrt(d' * d)");   /* ||b − Ax2|| */
```


Implementation

- User program (client)
  - Connects to a SILC server
  - Issues PUT, EXEC and GET requests
- Interface thread
  - For communications
  - Puts EXEC requests into the request queue
- Execution thread
  - For computation
  - Handles EXEC requests asynchronously



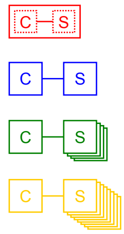
Implementation (continued)

- User programs
  - Sequential programs (at the moment)
  - Written in C, Fortran and Python
- SILC servers
  - Run in sequential and shared-memory (SMP) parallel computing environments
  - OpenMP is used for parallel computation in the execution thread

Experiments with 4 SILC servers in different computing environments

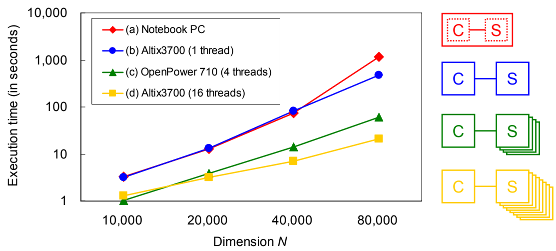
- A user program (client) that solves $Ax = \boldsymbol{b}$
  - Where A is a tridiagonal matrix in the CRS format
  - Run in the notebook PC of Environment (a)
  - In a 100-Base TX local-area network

	Environment	Specification	OpenMP
(a)	A notebook PC	Intel Pentium M 733 1.1GHz, 768MB memory, Fedora Core 3	N/A
(b)	SGI Altix3700	Intel Itanium2 1.3GHz × 32, 32GB memory, Red Hat Linux Advanced Server 2.1	1 thread
(c)	IBM eServer OpenPower 710	IBM Power5 1.65GHz × 2 (4 logical CPUs), 1GB memory, SuSE Linux Enterprise Server 9	4 threads
(d)	SGI Altix3700	Same as (b)	16 threads



Experimental results

- About 0.1 second of data communications over the LAN
  - Data size: 0.46MB (N=10,000) to 4.27MB (N=80,000)
- SILC servers in (c) and (d) achieved better performance because of parallel computation



Observations

- Performance of SILC is not bad
  - Speedups by parallel computation even with a time loss due to data communications
- Communication time will have less impact as dimension *N* increases
  - Communication time is of $O(N)$
  - Computation time is of $O(N^2)$
    - Faster networks and computing environments also reduce communication time in SILC

Future work

- Ready-made modules for existing matrix computation libraries
- MPI-based SILC for distributed-memory parallel computing environments
- Just-in-time (dynamic) optimizations based on mathematical expressions
- Extension of mathematical expressions to an interactive scripting language

For your information

- The first public release of SILC (version 1.0) will be made on September 20
- Please visit our project home page at http://ssi.is.s.u-tokyo.ac.jp/silc/