Intl. Conf. on Parallel Processing & Applied Mathematics (PPAM 2007)
September 9-12, Gdańsk, Poland

Cloth Simulation in the SILC Matrix Computation Framework: A Case Study

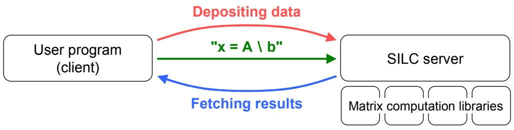
Tamito KAJIYAMA (JST / University of Tokyo, Japan)
Akira NUKADA (JST / University of Tokyo, Japan)
Reiji SUDA (University of Tokyo / JST, Japan)
Hidehiko HASEGAWA (University of Tsukuba, Japan)
Akira NISHIDA (University of Tokyo / JST, Japan)

Outline

- The SILC matrix computation framework
  - Easy-to-use interface for matrix computation libraries
  - Proposed application styles for numerical simulations in SILC
- Case study: Cloth simulation in SILC
- Experimental results
- Concluding remarks

Overview of the SILC framework

- **S**imple **I**nterface for **L**ibrary **C**ollections
  - Independent of libraries, environments & languages
  - Easy to use
- Three steps to use libraries
  - **Depositing data** (matrices, vectors, etc.) to a server
  - **Making requests for computation** by means of mathematical expressions
  - **Fetching the results of computation** if necessary



Example: Using SILC in C

```
silc_envelope_t A, C, u;
/* create matrices A, C and vector u0 */
SILC_PUT("A", &A);
SILC_PUT("C", &C);
SILC_PUT("u", &u); /* u0 */
for (k = 1; k <= n_steps; k++)
{
   SILC_EXEC("b = C * u");
   SILC_EXEC("u = A \\ b");
   SILC_GET(&u, "u"); /* uk */
   /* output solution uk at time tk */
}
```

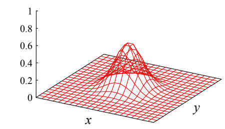
Solve the initial value problem of 2D diffusion equation below using the Crank-Nicolson method:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \quad x, y \in (0, 1),$$

$$u(x, y, t) = 0, \quad t > 0,$$

$$u(x, y, 0) = \begin{cases} 1 & \text{if } x, y \in (0.4, 0.6) \\ 0 & \text{otherwise} \end{cases}$$

$u(x, y, 0.004)$



Main characteristics of SILC

- Independence from programming languages
  - User programs for SILC in your favorite languages
- Independence from libraries and environments
  - Using alternative libraries and environments requires no modification in user programs
  - Flexible combinations of client & server environments

User program (client)	SILC server
Sequential	Sequential
Sequential	Shared-memory parallel (OpenMP)
Sequential	Distributed parallel (MPI)
Distributed parallel (MPI)	Distributed parallel (MPI)

Proposed application styles

- **Limited application style**
  - Use SILC only in the most time-consuming, computationally intensive part of a program
- **Comprehensive application style**
  - Move all relevant data to a SILC server, and implement overall simulations using SILC's mathematical expressions

Abbreviations:
- **LTD** for the limited application style
- **CMP** for the comprehensive application style

Comparison of LTD & CMP styles

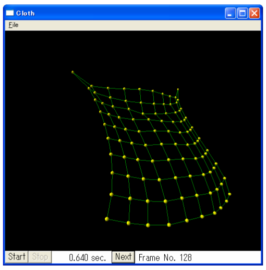
	LTD style	CMP style
Pros	• Easy to apply	• Less data transfer • More parallelizable computations
Cons	• Frequent data transfer to/from the SILC server	• May require a major rewrite of programs

Purposes of the present research

- To verify the feasibility of numerical simulations in SILC
- To examine the pros and cons of the two application styles

A case study: Cloth simulation

- Time-dependent simulation of cloth motion
  - Mass-spring model
  - The implicit Euler method
  - Solving a sparse linear system is necessary for each time step
- Original code
  - Sequential program in C
  - The CG method in the Lis iterative solvers library
  - Visualization via OpenGL



The original code

Do some initialization (defining cloth geometry, etc.)
For each time step:

1. Calculate force f and its derivatives $\partial f / \partial x$ and $\partial f / \partial v$ (Jacobian matrices).

2. Solve a linear system $A\Delta v = b$, where

$$A = \left\{ M - \Delta t^2 \frac{\partial f}{\partial x} - \Delta t \frac{\partial f}{\partial v} \right\}$$

$$b = \left\{ f_0 + \Delta t \frac{\partial f}{\partial x} v_0 \right\} \Delta t$$

3. Update particle motion.

$$v = v_0 + \Delta v$$

$$x = x_0 + \Delta t\, v$$

New code in the LTD style

Original code using Lis*

```
LIS_MATRIX A; LIS_VECTOR b, dv;

for (k = 1; k <= n_steps; k++)
{
  /* 1. Calculate f, ∂f/∂x and ∂f/∂v */
     :
  /* 2. Solve AΔv = b */
  lis_solve(A, b, dv, /* Δv */
            lis_params,
            lis_options,
            lis_status);
  /* 3. Update particle motion */
     :
}
```

New code using SILC

```
silc_envelope_t A, b, dv;

for (k = 1; k <= n_steps; k++)
{
  /* 1. Calculate f, ∂f/∂x and ∂f/∂v */
     :
  /* 2. Solve AΔv = b */
  SILC_PUT("A", &A);
  SILC_PUT("b", &b);
  SILC_EXEC("dv = A \\ b");
  SILC_GET(&dv, "dv"); /* Δv */
  /* 3. Update particle motion */
     :
}
```

* An iterative solvers library written in C.

Force and its derivatives

- Force

$$f_i = \sum_{j \in P_i} (f_{ij} + d_{ij})$$

$$f_{ij} = b_k \left(|x_j - x_i| - l_k \right) \frac{x_j - x_i}{|x_j - x_i|} \quad \text{(spring force)}$$

$$d_{ij} = -h_k (v_i - v_j) \quad \text{(damping)}$$

- Derivatives (Jacobian matrices)

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{f}_1}{\partial \mathbf{x}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{f}_n}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{f}_n}{\partial \mathbf{x}_n} \end{pmatrix}, \quad \frac{\partial \mathbf{f}}{\partial \mathbf{v}} = \begin{pmatrix} \frac{\partial \mathbf{f}_1}{\partial \mathbf{v}_1} & \cdots & \frac{\partial \mathbf{f}_1}{\partial \mathbf{v}_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{f}_n}{\partial \mathbf{v}_1} & \cdots & \frac{\partial \mathbf{f}_n}{\partial \mathbf{v}_n} \end{pmatrix}$$

Elements of the derivatives

- Off-diagonal blocks (3×3 submatrices)

$$\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_j} = b_k I - \frac{b_k l_k}{|\boldsymbol{x}_j - \boldsymbol{x}_i|}\left(I - \frac{(\boldsymbol{x}_j - \boldsymbol{x}_i)(\boldsymbol{x}_j - \boldsymbol{x}_i)^T}{|\boldsymbol{x}_j - \boldsymbol{x}_i|^2}\right), \quad \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{v}_j} = h_k I$$

- Diagonal blocks (3×3 submatrices)

$$\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_i} = \sum_{j \in P_i}\left(-\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_j}\right), \quad \frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{v}_i} = \sum_{j \in P_i}\left(-\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{v}_j}\right)$$

- Computing these blocks one after another is not a good idea in SILC (too fine-grain to parallelize)

Exploiting data parallelism

Original

For each spring k connecting particles i and j :

$\boldsymbol{p}_k = \boldsymbol{x}_i - \boldsymbol{x}_j$

$z_k = \mathrm{sqrt}(\mathrm{dot}(\boldsymbol{p}_k, \boldsymbol{p}_k))$

CMP style

n : number of particles, s : number of springs

Y_1, Y_2 : linear maps from $\mathbf{R}^{3n}$ to $\mathbf{R}^{3s}$

X : another linear map from $\mathbf{R}^{3s}$ to $\mathbf{R}^{s}$

$\boldsymbol{p} = (Y_1 - Y_2)\,\boldsymbol{x}$

$z = \mathrm{sqrt}(X\,(\boldsymbol{p}\ \ *@\ \ \boldsymbol{p}))$

*@ : elementwise multiplication operator in SILC

All coarse-grain, parallelizable matrix computations

```
silc_envelope_t v, x;

/*  1. Calculate f, ∂f/∂x and ∂f/∂v  */
SILC_EXEC("p = Y * x");
SILC_EXEC("z = sqrt(X_T * (p *@ p))");
SILC_EXEC("fij = p *@ (X * (K_stiff *@ (z - L)
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. Solve AΔv = b */
SILC_EXEC("A = M + (dt * dt) * DfDx + dt * DfDv");
SILC_EXEC("b = dt * (f - dt * (DfDx * v))");
SILC_EXEC("dv = A \\ b");

/* 3. Update particle motion */
SILC_EXEC("v += dv *@ fixed");
SILC_EXEC("x += dt * v");
SILC_GET(&v, "v");
SILC_GET(&x, "x");
```

New code in the CMP style: All expressions are **coarse-grain matrix computations** to be efficiently parallelized by a parallel SILC server.

Solving $A\Delta\boldsymbol{v} = \boldsymbol{b}$ is done in the same way as the LTD style.

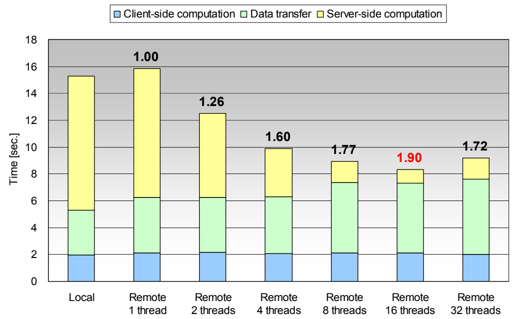
Experimental results

- 10^4 particles (7.998 × 10^8 springs), 20 time steps
- 3 user programs on the same PC
- A SILC server on the same PC
- Another server on SGI Altix 3700 in a GbE LAN

User program	Original	LTD version		CMP version
SILC server	—	PC (sequential)	Altix (16 threads)	Altix (32 threads)
Execution time [sec]	11.80	15.28	8.33	29.87
Speedup	—	×1.29 slower	×1.42 faster	×2.53 slower

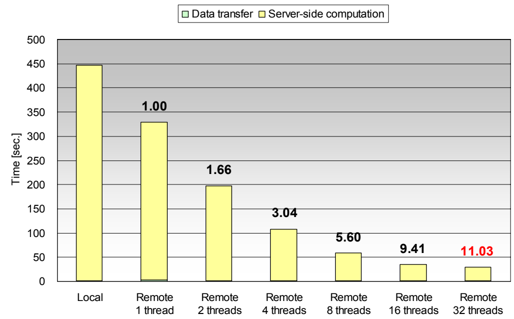
PC: Intel Pentium 4 3.40 GHz, 1 GB RAM, Microsoft Windows XP SP2
SGI Altix 3700: Intel Itanium 2 1.3 GHz × 32, 32 GB RAM (cc-NUMA), Red Hat Linux AS 2.1

Performance of the LTD version



✓ Amount of data transfer per time step: 17.7 MB

Performance of the CMP version



✓ Amount of data transfer per time step: 0.458 MB (2.59%)

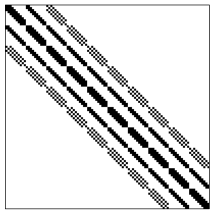
Performance of the CMP version (cont'd)

- The CMP version is slow because there are ***lots of non-floating point operations*** in sparse matrix computations in the Compressed Row Storage (CRS) format
- In fact, the FLOP count is about 20% fewer than the original and LTD versions

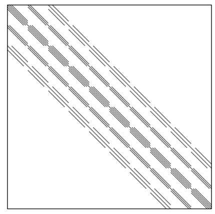
Sparse matrix-matrix products	53.57 %
Sparse matrix transpositions	20.04 %
Calls for the "sparse" function	11.48 %
Sparse matrix-matrix additions	7.13 %
Calls for the linear solver (CG)	4.59 %
Others	3.19 %

Breakdown of the server-side computation (32 threads)

Non-zero blocks in the derivatives



$\frac{\partial f}{\partial x}$



$\frac{\partial f}{\partial v}$

✓ Use of a block matrix storage format may accelerate the CMP version

Summary

- A feasibility study of numerical simulations in SILC
  - LTD and CMP versions of an existing cloth simulation code were developed
  - Pros and cons of the application styles were verified
- Future work
  - Performance improvements of the CMP version by means of a block matrix storage format
  - Further case studies with other types of numerical simulations such as CFD

Advertisement

- A short demo of SILC and copies of our papers are available
- SILC v1.2 is freely available at

http://ssi.is.s.u-tokyo.ac.jp/silc/

- Source (Unix/Linux, Windows, Mac OS X)
- Precompiled binary package for Windows
- Documentation, sample programs