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

Simple Interface for Library Collections

長谷川 秀彦 (筑波大)、須田 礼仁 (東大)、
櫛田 彰 (JST)、梶山 民人 (JST) 、中島 研吾 (東大) 、
高橋 大介 (筑波大) 、小武守 恒 (JST)、
藤井 昭宏 (工学院大)、西田 晃 (東大)

われわれの立場

科学技術振興機構戦略的創造研究推進事業 (CREST)
研究領域「シミュレーション技術の革新と実用化基盤の構築」
(平成14年度 - 平成19年度)

「大規模シミュレーション向け基盤ソフトウェアの開発」の一部

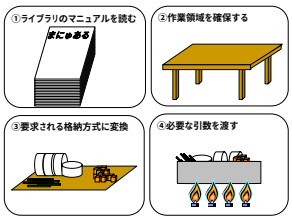
昔) Development of Software Infrastructure for Large Scale Scientific Simulation

ちょっと前) Simulation Software Infrastructure?

今) Scalable Software Infrastructure

これまでのライブラリ呼び出し

by Nukada



(例) ライブラリを用いて Ax = b を解く

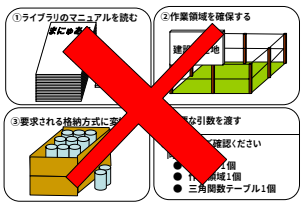
```
      REAL *8  A(LDA,N), B(N)
      INTEGER IP(N)
C     データ生成
C     計算:ライブラリコール
      CALL LU(LDA, N, A,IP,IERR)
      CALL SOLVE(LDA, N, A, B,IP, IERR)
C     結果の利用
      END
```


問題点

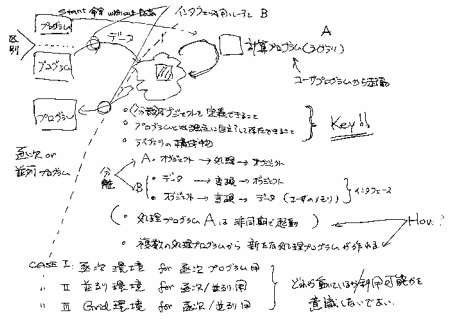
- データ格納形式はユーザプログラムに依存
- 使って良いのは与えられた作業領域のみ
- 呼び出し方法は個々のライブラリに依存
- 分散並列用ライブラリが作成困難
- プログラムが肥大化
- Reverse Communication に頼る必要性
- 言語ごとに異なるメモリ配置

できることなら

by Nukada



思いつき



Simple Interface for Library Collections のポイント

- データ受け渡しと「演算処理の指定」を分離
- 演算処理の指定は文字列で
  (処理はライブラリシステムに任せる)
- 処理はライブラリシステムのメモリ空間で
  (ユーザプログラムのメモリ空間を使わない)

SILC での記述例: Solve Ax=b

```
C*  これまでの方法
C
      REAL *8  A(LDA,N), B(N)
      INTEGER IP(N)
C     データ生成
C     計算:ライブラリコール
      CALL LU(LDA, N, A, IP,IERR)
      CALL SOLVE(LDA,N,A,B,IP,IERR)
C     結果の利用
      END
```

```
C*  SILC では
C
      REAL  *8  A(LDA, N), B(N), SOL(N)
C  (1) データを預ける
      CALL PUT('A', LDA, N, A, IERR)
      CALL PUT('b', N, B, IERR)
C  (2) 計算依頼
      CALL SILC(' x = A¥b ')
C  (3) 結果の受け取り
      CALL GET('x', N, SOL, IERR)
      END
```


SILC における流れ

- 預け – 計算 – 受け取りの 3 ステップ
- 3つのステップは分離可能
  (分割できる・組み合わせられる:メッシュ生成 – 計算 – 可視化など)
- ステップ2では配列不要
  (ユーザプログラムを小さくできる)
- ステップ2で使うのは文字列のみ
  (プログラム言語から独立;非同期処理が可能;自由な文法)
- ステップ1, 3のみがユーザのデータ構造に依存
  (数値データの格納形式に対する依存性を解消できる)

メリット

- 使い方の容易さ
- 記述の容易性
- 高性能技術の利用
- などなど

使い方の容易さ

- メモリ管理を考えなくてよい
  - メモリを自動的に確保・解放
  - 一方で、ある程度の制御も可能
- ハードウェアを考えなくてよい
  - 自動的に使うシステムに対応
  - 単体、並列、ネットワーク・・・

記述の容易性

- 記述が明快、読みやすい
  - よく知られた数学の表現方法に近い
- 書き方が共通（入出力を除く）
  - 呼び出すプログラミング言語によらない
  - 計算機構成（並列、ネットワーク）によらない
- 便利な機能を簡単に書ける
  - プログラミング言語の制約から解放される
  - 「引数」などは最小限でよい

高性能技術の利用

- バージョンアップが簡単
  - 呼び出し方は変わらない
- 新技術の導入が容易
  - 新しいアルゴリズム
  - 拡張精度、超高精度
  - 精度保証
  - フォールトトレラント
  - 自動性能最適化

その他

- 使い方に広がり
  - 異なるプロセスから呼び出し可能
    - 複数のプログラムを組み合わせて処理できる
    - プログラムをコンパクトにできる
  - 遠隔地の計算能力・メモリを利用可能
- 既存技術を活用できる
  - 既存のライブラリを利用できる
- デバッグが少し容易
  - サーバ・クライアントなので、切り分け可能

デメリットなど

- CCA などの Wrapper 方式との比較は？
- データ転送が必要
  （ネットワークはより高速化する）
- ユーザ側とシステム側に2重のメモリ
  （1時的か常駐か）
- 「ライブラリシステム」と呼んではいるが、なに？
- 本当にできるの？？？

おわりに

- いろいろな可能性がみえてきた
- 逐次 – 逐次版を実装中
  → **HPCS2005 ポスターセッションで！**
- ＦＦＴへの利用
  → データ分散の自由化！
- 疎行列に対する反復法
  → データ格納形式の自由化！