CMSI計算科学技術特論C (2015/10/01)

ALPS

ALPS と量子多体問題 ①

藤堂眞治

東京大学大学院理学系研究科 / 物性研究所 / NIMS

wistaria@phys.s.u-tokyo.ac.jp

CMSI

MateriApps

自己紹介 - 藤堂眞治 (とうどうしんじ)

- 1968年 愛媛県松山市に生まれる
- 1996年 Ph.D @ 東京大学大学院理学系研究科物理学専攻 [鈴木(増)研]
- 1996年 東京大学物性研究所 (六本木 & 柏)
- 2000年 スイス連邦工科大学 (Zürich)
- 2002年 東京大学大学院工学系研究科物理工学専攻
- 2011年 東京大学物性研究所 (神戸)
- 2014年 東京大学大学院理学系研究科物理学専攻
  - 物性研究所 (兼)、NIMS (兼)

- 専門：計算物理、量子統計物理、物性基礎論
- 趣味：学生のプログラムのデバッグ

「CMSI計算科学技術特論C」のねらい

- 対象
  - アプリ開発のスペシャリストまたはスペシャリストを目指したい人
- 目標
  - 開発・普及が進んでいるアプリに着目し、そのアプリの周辺事情を概観すると同時に、本格的なアプリ開発のための総合的な技術、普及•運用における知識やノウハウを学ぶ
- 「プログラムを書いた後」のこと
  - アプリの開発・公開
  - 普及、運用
  - ユーザビリティ向上、メンテナンス性向上、など
- 方法論・カリキュラムの確立

「CMSI計算科学技術特論C」の講義項目 (1/2)

- (A) アプリの背景
  - A-1 どのような問題に使用されるか
  - A-2 物理モデル、基礎理論(数学的定式化)、アルゴリズム
  - A-3 分野(業界)の歴史、(競合する他アプリを含む)アプリ発展の歴史
- (B) アプリ実行イメージ
  - B-1 入力、計算、出力まで
  - B-2 データ解析や他アプリとの連携
- (C) プログラム
  - C-1 使用言語について
  - C-2 プログラムの全体的な構造、設計方針など
  - C-3 コーディングの方針(作法、書式)など
  - C-4 入出力データ、入出力の仕方など
  - C-5 ユーザーのため、開発者のため、性能のため、の工夫
  - C-6 「今後こうしたい」「こうしておけばよかったと思う事」など

「CMSI計算科学技術特論C」の講義項目 (2/2)

- (D) 開発体制
    - D-1 開発の歴史
    - D-2 開発メンバー、人数
    - D-3 バージョン管理ソフト等の利用
    - D-4 新機能追加やデバッグの事例
    - D-5 新人教育
- (E) 公開、運営
    - E-1 ドキュメント
    - E-2 チュートリアル
    - E-3 公開方法、普及活動
    - E-4 ユーザーサポート
- (F) 今後

「CMSI計算科学技術特論C」の講義予定

- 10月1日、10月8日:「ALPSと量子多体問題」 藤堂眞治 (東大)
- 10月15日、10月22日:「OpenMXとDFT」尾崎泰助 (東大)
- 10月29日:「xTAPPをはじめとしたDFTコードとユーザーインターフェース」
  吉澤香奈子 (RIST)
- 11月5日:「可読性と性能の両立を目指して」渡辺宙志 (東大)
- 11月12日:「ライセンスについて」五十嵐 亮 (東大)
- 11月19日:「みずほの業務事例から(仮)」渡辺尚貴 (みずほ総研)
- 11月26日、12月3日:「feramと強誘電体」西松 毅 (東北大)
- 12月10日、12月17日:「MODYLASと古典MD」吉井範行 (名古屋大)
- 1月7日、1月21日:「ソフトウェア工学の視点から」千葉 滋 (東大)
- 1月14日:「SMASHと量子化学」石村和也 (分子研)

「ALPS と量子多体問題」Agenda (1/2)

- CMSI計算科学技術特論Cの全体像
  - 対象と目標、講義項目、プログラム
- ALPS (Applications and Libraries for Physics Simulations) とは?
  - 量子格子模型とは? 強相関量子多体系のためのシミュレーション手法
  - ALPS の生まれた背景、歴史と活用事例
- ALPS の設計
  - シミュレーションのワークフロー、ALPS階層構造
  - XML / HDF5 による入出力
  - C++言語のテンプレートとジェネリックアルゴリズム
  - Pythonによる後処理・可視化
  - アプリケーション・フレームワークとしての利用

「ALPS と量子多体問題」Agenda (2/2)

- ALPS の開発
  - ALPLS開発の経緯と開発体制、ライセンスと「ビジネスモデル」
  - オープンソースのツールの利用 (ビルドシステム、バージョン管理、バグ追跡、ドキュメント作成、GUI・可視化)
- アプリケーション普及のために
  - ユーザサポート (ドキュメント、講習会、ワークショップ)
  - 公開と普及 (MateriApps と MateriApps LIVE!)
  - プロジェクトの普及と継続に不可欠なものは?

ALPS

ALPS とは？

参考文献:

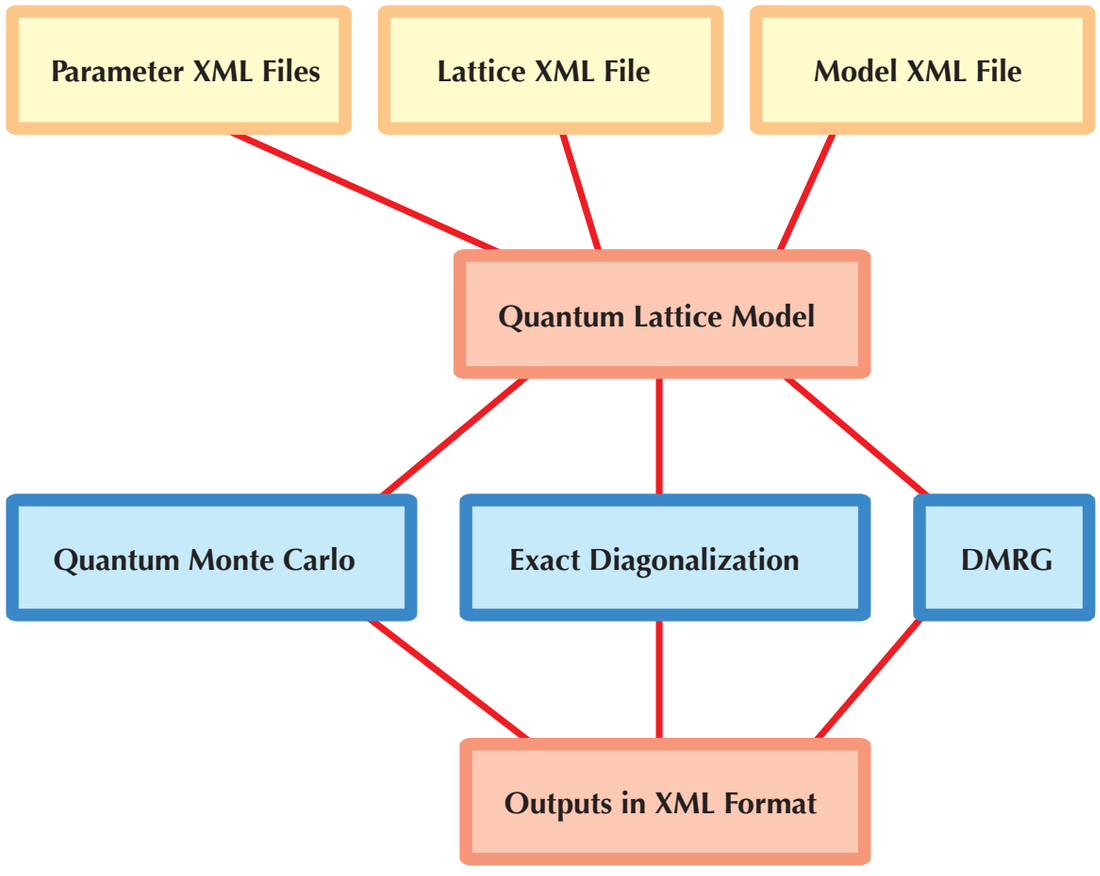
藤堂「“実験技術”としての量子多体系シミュレーションソフトウェア ALPS」

日本物理学会誌 70, 275-282 (2015)

ALPS プロジェクト

ALPS = Algorithms and Libraries for Physics Simulations

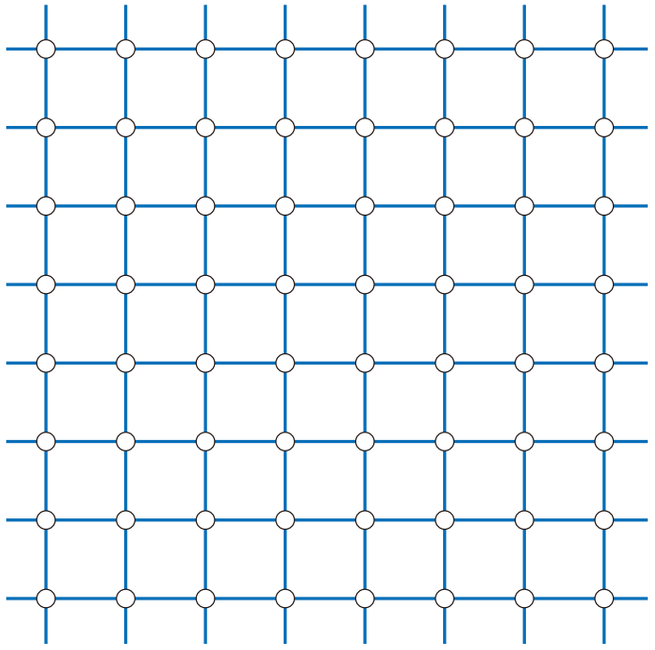
- 量子スピン系、電子系など強相関量子格子模型のシミュレーションのためのオープンソースソフトウェアの開発を目指す国際共同プロジェクト
- ALPS ライブラリ = C++による格子模型のための汎用ライブラリ群
- ALPS アプリケーション = 最新のアルゴリズムに基づくアプリケーション群: QMC、DMRG、ED、DMFT 等
- ALPS フレームワーク = 汎用入出力形式、解析ツール、スケジューラなど、大規模並列シミュレーションのための環境



量子格子模型 (Quantum Lattice Models)

- 量子スピン模型 (XXZ 模型)

$$\mathcal{H} = \frac{J^{xy}}{2} \sum_{\langle i,j \rangle} (S_i^+ S_j^- + S_i^- S_j^+) + J^z \sum_{\langle i,j \rangle} S_i^z S_j^z$$



- Hubbard 模型 (fermionic / bosonic)

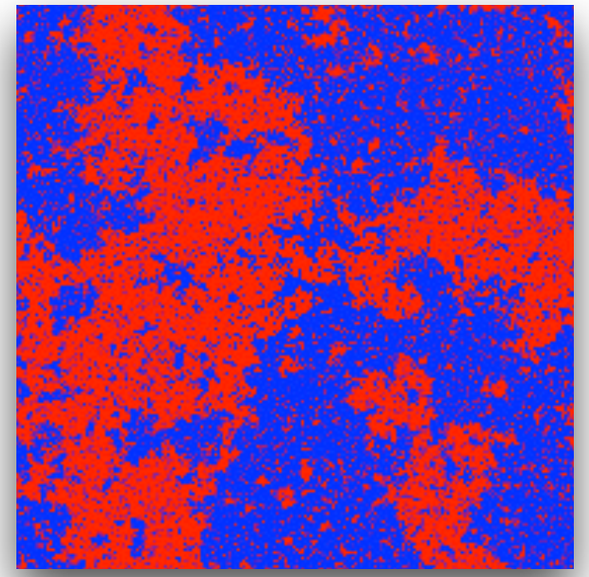
$$\mathcal{H} = -t \sum_{\langle i,j \rangle \sigma} (c_{i\sigma}^\dagger c_{j\sigma} + \text{h.c.}) + U \sum_i n_{i\uparrow} n_{i\downarrow}$$

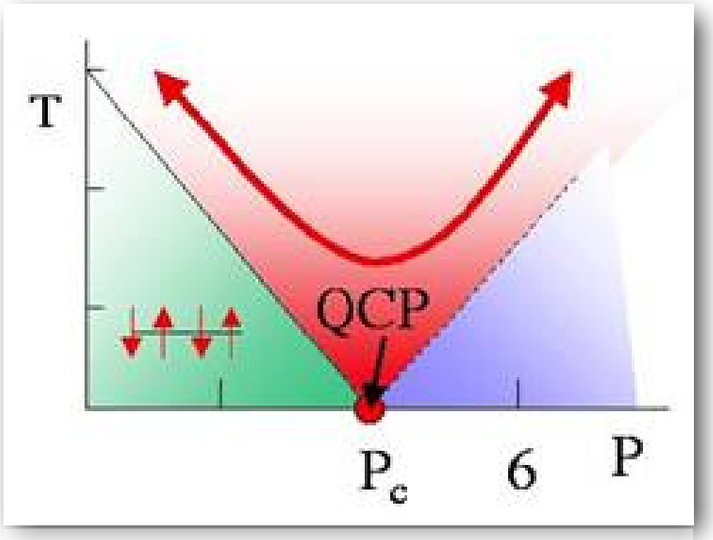
- t-J 模型

$$\mathcal{H} = -t \sum_{\langle i,j \rangle \sigma} (c_{i\sigma}^\dagger c_{j\sigma} + \text{h.c.}) + J \sum_{i,j} (\mathbf{S}_i \cdot \mathbf{S}_j - n_i n_j / 4)$$

なぜ量子格子模型を考えるのか?

- 量子多体系における強相関効果
  - さまざまな秩序状態
  - 量子的に強くゆらいだ相: 量子液体、スピンギャップ相
  - 量子相転移、量子臨界現象
- 量子統計物理におけるユニバーサリティー
  - 量子臨界現象は系の次元、秩序変数の対称性などにのみ依存
  - 量子臨界現象特有のユニバーサリティークラスの探索
- 新しい計算物理学的手法の発展
  - 量子モンテカルロ法、DMRG、DMFT、テンソルネットワーク、など





QLM に対する代表的なシミュレーション手法

- 厳密対角化
  - 完全対角化 (Householder 法)、Lanczos 法
- 古典モンテカルロ法
  - Metropolis 法、クラスターアルゴリズム、etc
- 量子モンテカルロ法
  - ループアルゴリズム、向き付きループアルゴリズム、ワームアルゴリズム、etc
- DMRG とその一般化
  - DMRG (密度行列くりこみ群)、(i)TEBD、(i)PEPS、MERA、etc
- 動的平均場近似
  - QMCソルバ、厳密対角化ソルバ、etc

ALPS の生まれた背景

- 「コミュニティーコード」の不在
  - 個々の研究者が独自のコードを作成・使用
  - 扱うモデル・格子毎にコードを作成
  - コードの再利用がほとんどなされない
- アルゴリズムが複雑化
  - 様々な新しいアルゴリズムの開発
  - プログラム開発の長期化
- 理論家/実験家による使い易く、 信頼性の高いシミュレーション パッケージへの強い要望
- (non-portableな) 独自形式による入力・出力の弊害

ターゲット・オーディエンス

- 実験家
  - 物質のモデリングにソフトウェアパッケージを利用
  - 実験結果とシミュレーション結果のフィッティングにより、相互作用定数などを決定
- 理論家
  - 理論的なアイデアのチェックに使いやすい整備されたコードを利用
  - 自前のコードのデバッグに
  - 新しいコード開発の基盤としての利用
- 計算機科学者、学生、・・・

ALPS の歴史

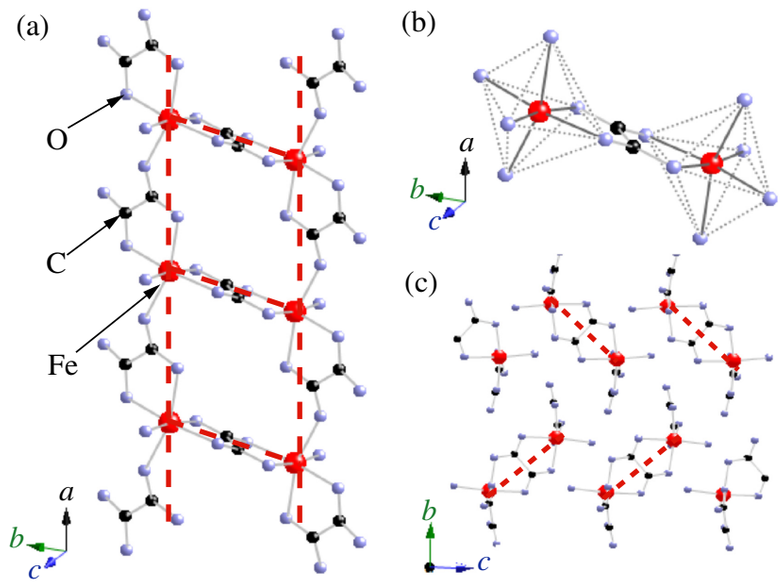
- 1994年～
  - PALM++ C++ ライブラリの開発、量子モンテカルロコード、DMRG コード
- 1997年～
  - looper ライブラリ(量子モンテカルロ)の開発
- 2001年~
  - SSE 量子モンテカルロコード
- 2001年～
  - ALPS プロジェクト
- 2002年
  - 第1回開発者ワークショップ
- 2004年
  - ALPS バージョン1.0公開、第1回ユーザワークショップ
- 2015年現在、バージョン1.3を公開中

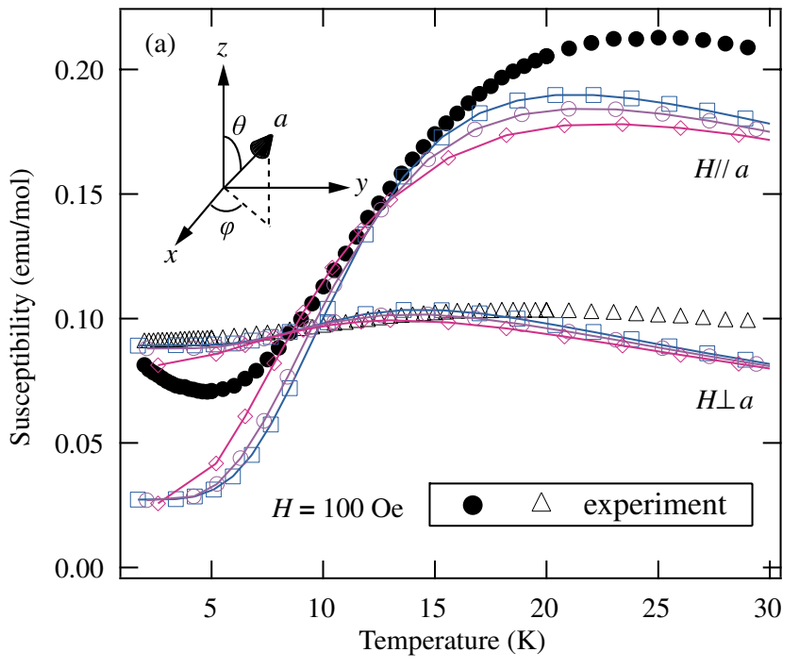
活用事例: Spin ladder material $Na_2Fe_2(C_2O_4)_3(H_2O)_2$

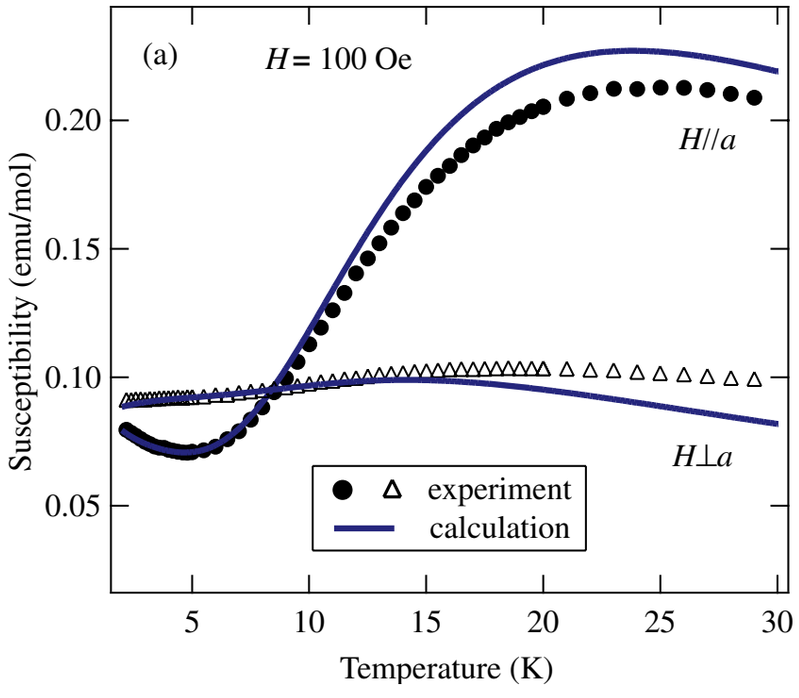
- Fe^{2+} ions in octahedral crystal field

  ⇒ effective S=1 spins at low T

- Fitting experimental data by QMC results for several theoretical models (ladders, dimers, etc)



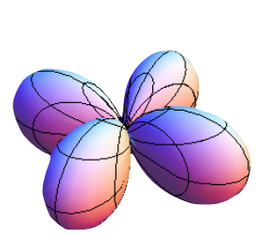




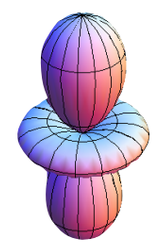
Yamaguchi, Kimura, Honda, Okunishi, Todo, Kindo, Hagiwara (2009)

活用事例: Orbital ordering in e_g orbital systems

- Mott insulators with partially filled d-shells
- Non-trivial interplay of charge, spin, and orbital degrees of freedom



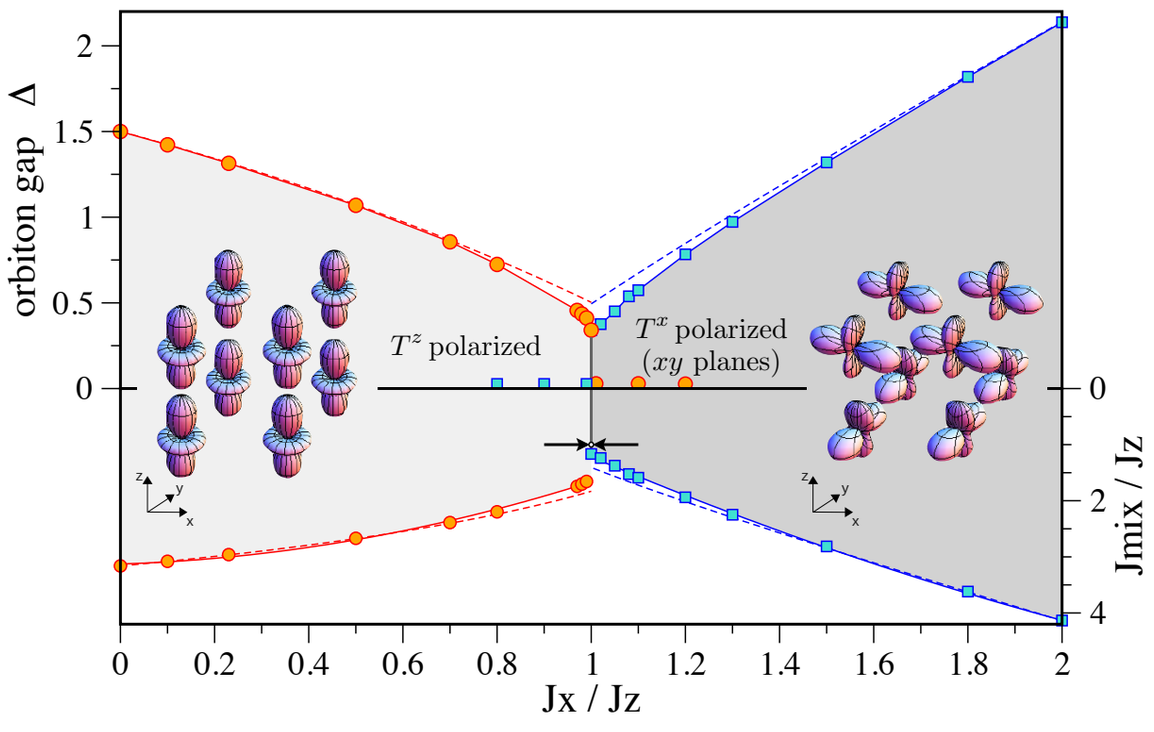
$|x^2 - y^2\rangle$



$|3z^2 - r^2\rangle$

- Effective Hamiltonian for orbital degrees of freedom (120° model)

$$H_{120} = -\sum_{i,\gamma=x,y} \frac{1}{4}\Big[J_z T_i^z T_{i+\gamma}^z + 3J_x T_i^x T_{i+\gamma}^x \pm\sqrt{3}J_{\rm mix}(T_i^z T_{i+\gamma}^x + T_i^x T_{i+\gamma}^z)\Big] - \sum_i J_z T_i^z T_{i+z}^z$$



van Rynbach, Todo, Trebst (2010)

活用事例: Supersolid in extended Bose-Hubbard model

- Interacting soft-core bosons

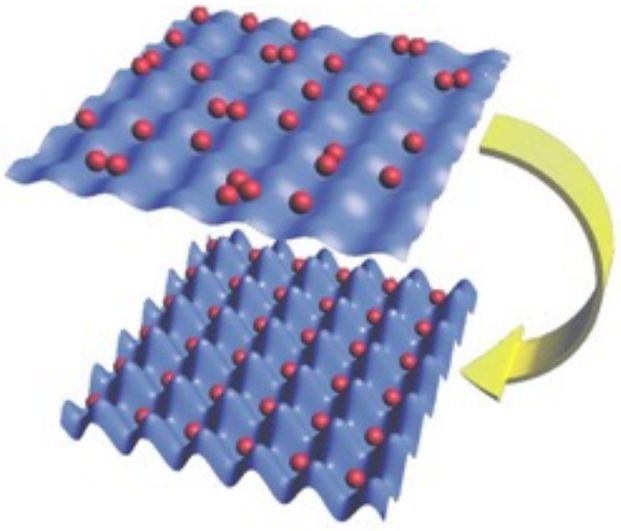
$$\mathcal{H} = -t\sum_{\langle ij\rangle}(a_i^\dagger a_j + a_i a_j^\dagger) + V\sum_{\langle ij\rangle} n_i n_j + \frac{1}{2}U\sum_i n_i(n_i-1) - \mu\sum_i n_i,$$

- Supersolid = co-existence of

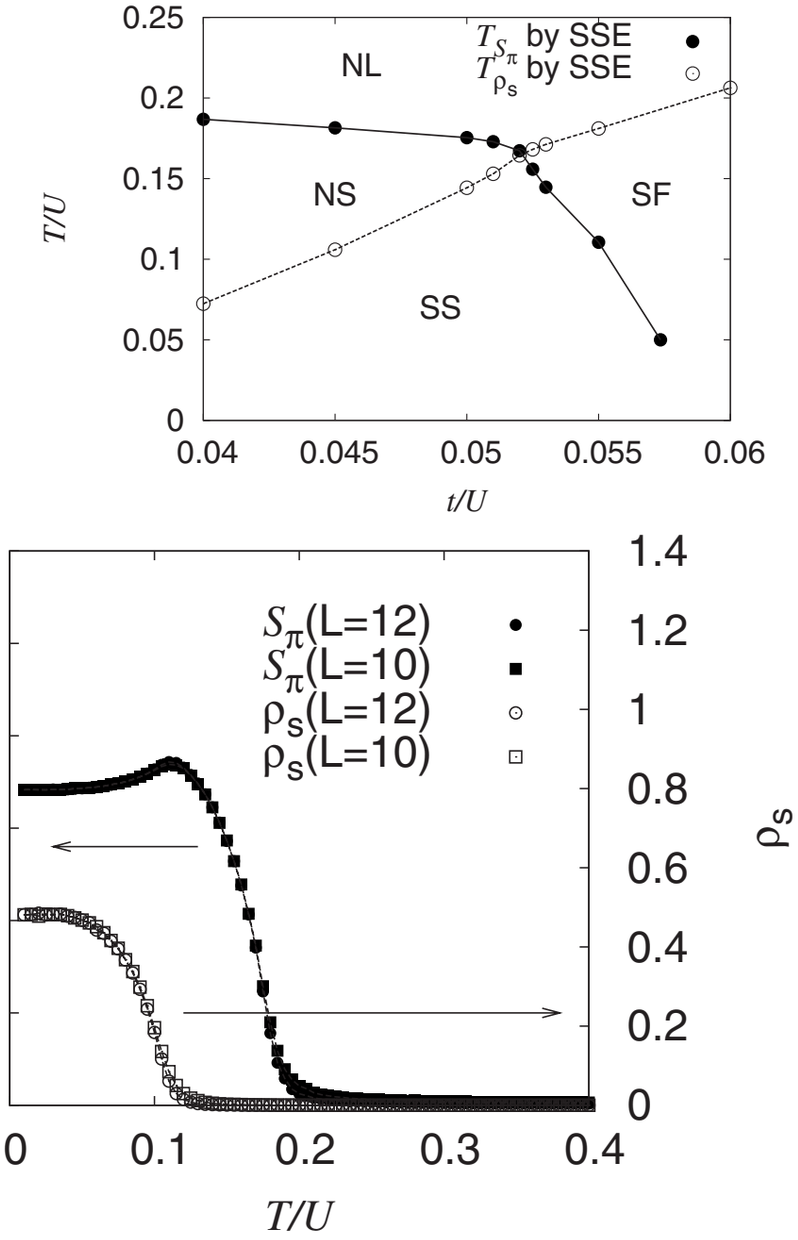
  diagonal long-range order (=solid) and

  off-diagonal long-range order (=superfluid)

- Experimental realization: optical lattice?



http://www.uibk.ac.at/th-physik/qo



Yamamoto, Todo, Miyashita (2009)

ALPS を利用した研究

- 600 以上の論文で ALPS が利用されている (論文タイトルの例 2015年8月以降)
  - First-order commensurate-incommensurate magnetic phase transition in the coupled FM spin-1/2 two-leg ladders
  - Critical behavior of the site diluted quantum anisotropic Heisenberg model in two dimensions
  - Dimerized ground states in spin-S frustrated systems
  - Practical, Reliable Error Bars in Quantum Tomography
  - Fulde-Ferrell-Larkin-Ovchinnikov pairing as leading instability on the square lattice
  - Mott Transition for Strongly-Interacting 1D Bosons in a Shallow Periodic Potential
  - Pair Correlations in Doped Hubbard Ladders
  - Theoretical investigation of the behavior of CuSe2O5 compound in high magnetic fields
  - FLEX+DMFT approach to the d -wave superconducting phase diagram of the two-dimensional Hubbard model
  - Transport properties for a quantum dot coupled to normal leads with a pseudogap
  - ・ ・ ・

http://alps.comp-phys.org/mediawiki/index.php/PapersTalks

ALPS

ALPS の設計 (デザイン)

ALPS の特徴

- 任意の格子
  - XML フォーマットによる格子構造の定義
  - ユニットセルの繰り返しによる格子生成
  - 格子を任意の有限グラフ (頂点と辺の集合) として定義することも可能
- 任意のハミルトニアン (模型)
  - XML フォーマットによる量子数、演算子の定義
  - 数式によるハミルトニアンの定義
- 様々な最新の解法 (アプリケーション): ED、CMC、QMC、DMRG、DMFT
- 全ての ALPS アプリケーションに共通の入力形式
- 汎用的な出力形式、Python による解析・グラフ作成ツール

ALPS のワークフロー

Parameter File

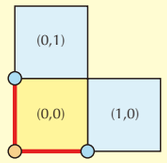
```
LATTICE = "square lattice"
MODEL   = "spin"
L       = 16
Jxy     = 1
Jz      = 2
SWEEPS  = 10000
THERMALIZATION = 1000

{ T = 0.1 }
{ T = 0.2 }
{ T = 0.5 }
{ T = 1.0 }
```

parameter2xml tool

Parameter XML File

Lattice XML File
square lattice

```
<LATTICES>
  <LATTICE name="square lattice" dimension="2">
    <PARAMETER name="a" default="1"/>
    <BASIS><VECTOR>a 0</VECTOR><VECTOR>0 a</VECTOR></BASIS>
  </LATTICE>
  <UNITCELL name="simple2d" dimension="2">
    <VERTEX/>
    <EDGE>
      <SOURCE vertex="1" offset="0 0"/>
      <TARGET vertex="1" offset="0 1"/>
    </EDGE>
    <EDGE>
      <SOURCE vertex="1" offset="0 0"/>
      <TARGET vertex="1" offset="1 0"/>
    </EDGE>
  </UNITCELL>
  <LATTICEGRAPH name="square lattice">
    <FINITELATTICE>
      <LATTICE ref="square lattice"/>
      <EXTENT dimension="1" size="L"/>
      <EXTENT dimension="2" size="L"/>
      <BOUNDARY type="periodic"/>
    </FINITELATTICE>
    <UNITCELL ref="simple2d"/>
  </LATTICEGRAPH>
</LATTICES>
```



Model XML File

$$H = J \sum_{<i,j>} [\, S_i^z S_j^z + (S_i^+ S_j^- + S_i^- S_j^+)/2 \,] - h \sum_i S_i^z$$

```
<MODELS>
  <BASIS name="spin">
    <SITEBASIS name="spin">
      <PARAMETER name="local_S" default="1/2"/>
      <QUANTUMNUMBER name="S" min="local_S" max="local_S"/>
      <QUANTUMNUMBER name="Sz" min="-S" max="S"/>
      <OPERATOR name="Sz" matrixelement="Sz"/>
      <OPERATOR name="Splus" matrixelement="sqrt(S*(S+1)-Sz*(Sz+1))">
        <CHANGE quantumnumber="Sz" change="1"/>
      </OPERATOR>
      <OPERATOR name="Sminus" matrixelement="sqrt(S*(S+1)-Sz*(Sz-1))">
        <CHANGE quantumnumber="Sz" change="-1"/>
      </OPERATOR>
    </SITEBASIS>
  </BASIS>
  <HAMILTONIAN name="spin">
    <PARAMETER name="J" default="1"/>
    <PARAMETER name="h" default="0"/>
    <BASIS ref="spin"/>
    <SITETERM>
      -h * Sz(i)
    </SITETERM>
    <BONDTERM source="i" target="j">
      J * (Sz(i)*Sz(j) + (Splus(i)*Sminus(j)+Sminus(i)*Splus(j)))/2
    </BONDTERM>
  </HAMILTONIAN>
</MODELS>
```

Quantum Lattice Model

application programs

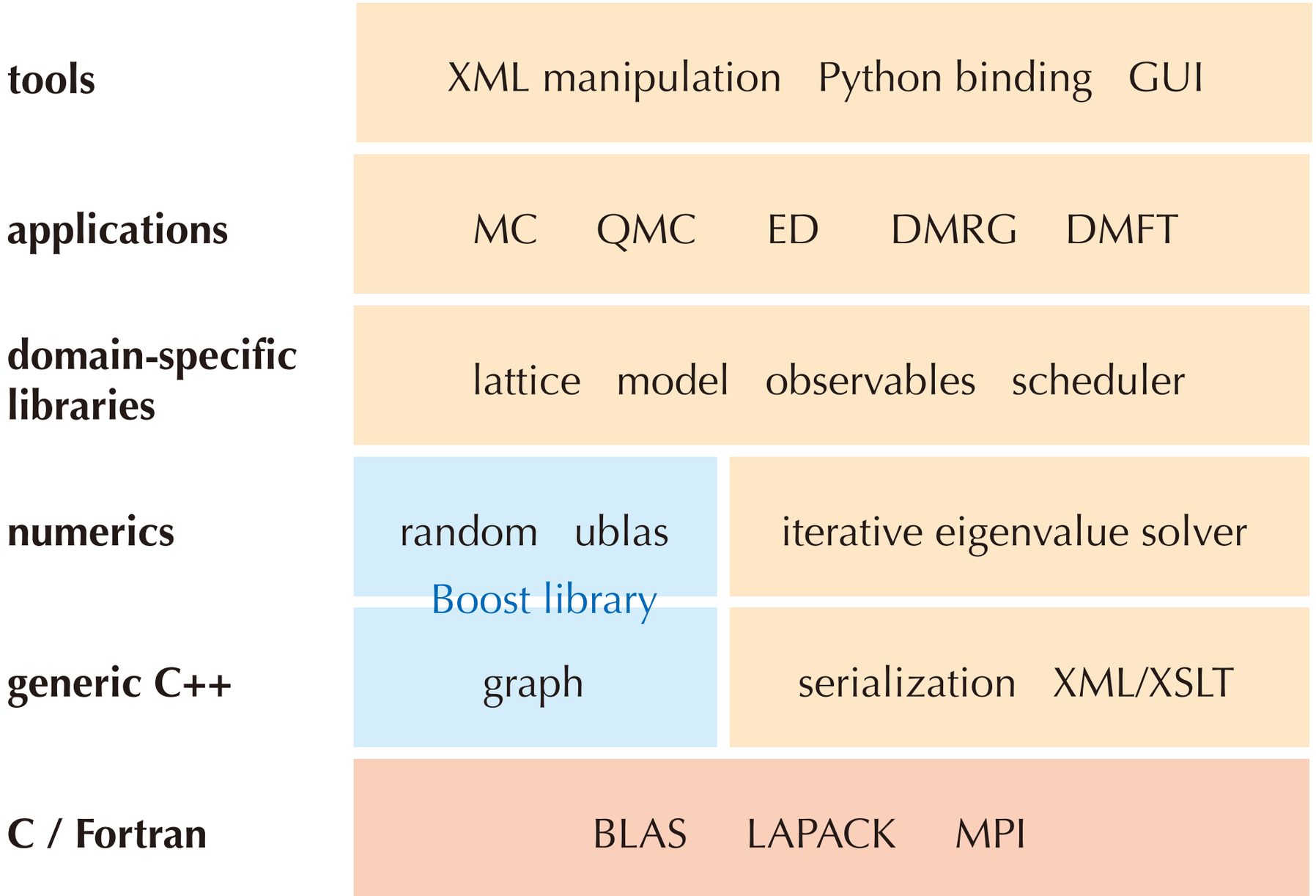
Quantum Monte Carlo | **Exact Diagonalization** | **DMRG** | **DMFT**

Outputs in HDF5 & XML

Python based evaluation tools

Plots

ALPS の階層構造


tools
XML manipulation
Python binding
GUI
applications
MC
QMC
ED
DMRG
DMFT
domain-specific libraries
lattice
model
observables
scheduler
numerics
random
ublas
iterative eigenvalue solver
Boost library
generic C++
graph
serialization
XML/XSLT
C / Fortran
BLAS
LAPACK
MPI

サードパーティーのライブラリ

- Boost C++ Library
  - 乱数、グラフ、シリアル化、など多くの有用なライブラリ群
- HDF5 (Hierarchical Data Format)
  - プラットフォーム非依存のバイナリファイル格納形式
- MPI (Message Passing Interface)
  - 並列計算のためのメッセー ジ通信
- BLAS、LAPACK
  - 線形演算 (行列対角化、特異値分解など)

ALPS/alea ライブラリ

- マルコフ連鎖における平均値、分散、自己相関を計算するライブラリ

```
alps::RealObservable mag2("Magnetization^2");
...
mag2 << m * m; // at each MC step
```

- ビンニング解析を用いた平均値、エラー、自己相関時間の評価

```
std::cout << mag2 << std::endl;
```

- 出力

```
Magnetization^2: 3.142 +/- 0.001; tau = 10.3
```

- ジャックナイフ法を用いた非線形量のエラー評価

```
alps::RealObsEvaluator binder = mag2 * mag2 / mag4;
std::cout << binder.mean() << ‘ ‘ << binder.error() << ‘\n’;
```

ALPS/lattice ライブラリ

- 「格子構造」は数学的には「グラフ」で表現できる
  - site ⇔ vertex
  - bond ⇔ edge
- Boost Graph Library に対する「ラッパー」を提供
  - XML による「格子構造」の入出力
  - 「ユニットセル」による繰り返し構造の指定
  - 座標、パリティ、逆格子ベクトルなどの属性
- あらかじめ用意されている格子
  - “hain lattice”、“square lattice”、“triangular lattice”、“honeycomb lattice”、“simple cubic lattice”、など

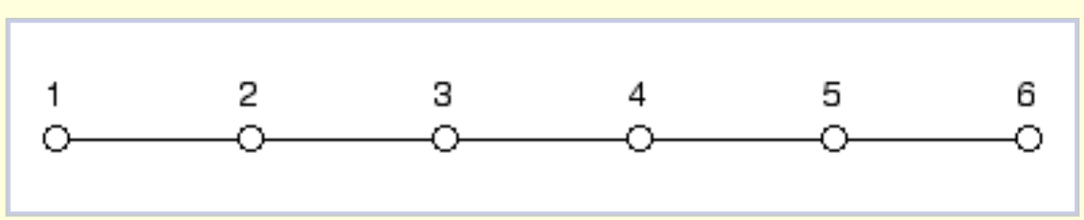
XML による格子の定義

```
<LATTICE name="chain lattice" dimension="1">
  <BASIS><VECTOR> 1 </VECTOR></BASIS>
</LATTICE>
```

```
<UNITCELL name="simple1d" dimension="1" vertices=”1”>
  <EDGE>
    <SOURCE vertex="1" offset="0"/><TARGET vertex="1" offset="1"/>
  </EDGE>
</UNITCELL>
```



```
<LATTICEGRAPH name = "chain lattice">
  <FINITELATTICE>
    <LATTICE ref="chain lattice"/>
    <PARAMETER name=”L”/>
    <EXTENT size ="L"/>
    <BOUNDARY type="periodic"/>
  </FINITELATTICE>
  <UNITCELL ref="simple1d"/>
</LATTICEGRAPH>
```



ALPS/model ライブラリ

- XML を使ってハミルトニアンを定義する
  - 量子数や演算子の定義
  - シンボリックな表現を使って、ハミルトニアンのサイト項やボンド項を定義

```
Jz*Sz(i)*Sz(j)+Jxy/2*(Splus(i)*Sminus(j)+Sminus(i)*Splus(j))
```

- 作成した局所ハミルトニアンは行列の形で取り出せる
- プラケット項などの多サイトにわたる相互作用には未対応
- ALPS Model HOWTO: http://alps.comp-phys.org/mediawiki/ index.php/ Tutorials:ModelHOWTO/ja
- あらかじめ用意されている模型: “spin”、“boson Hubbard”、“hardcore boson”、“fermion Hubbard”、“alternative fermion Hubbard”、“spinless fermions”、“Kondo lattice”、“t-J” 、他

XML によるモデルの定義

```
<HAMILTONIAN name="spin">
  <PARAMETER name="Jz" default="J"/>
  <PARAMETER name="Jxy" default="J"/>
  <PARAMETER name="J" default="1"/>
  <PARAMETER name="Jz'" default="J'"/>
  <PARAMETER name="Jxy'" default="J'"/>
  <PARAMETER name="J'" default="0"/>
  <PARAMETER name="h" default="0"/>
  <BASIS ref="spin"/>
  <SITETERM site="i"> -h * Sz(i) </SITETERM>
  <BONDTERM type="0" source="i" target="j">
    Jz * Sz(i) * Sz(j) + Jxy * (Splus(x)*Sminus(y)+Sminus(x)*Splus(y)) / 2
  </BONDTERM>
  <BONDTERM type="1" source="i" target="j">
    Jz' * Sz(i) * Sz(j) + Jxy' * (Splus(x)*Sminus(y)+Sminus(x)*Splus(y)) / 2
  </BONDTERM>
</HAMILTONIAN>
```

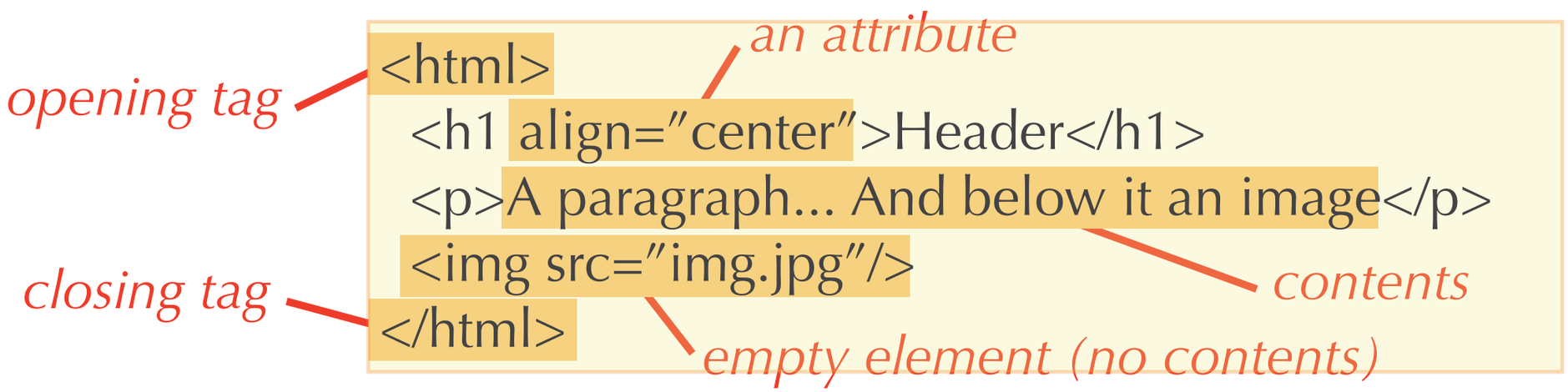
$$\mathcal{H} = \sum_{\langle i,j \rangle} [J_z S_i^z S_j^z + \frac{J_{xy}}{2}(S_i^+ S_j^- + S_i^- S_j^+)] - \sum_i h S_i^z$$

主要技術

- XML、HDF5による入出力
  - 可搬性 (ポータビリティ)
  - 出力結果の変換が容易・内容が一目瞭然
- C++ ジェネリックプログラミング
  - 柔軟性・再利用性
  - 高品質なコードの作成
- C++ 標準ライブラリ、Boostライブラリ、サードパーティーライブラリの利用
  - 開発のスピードアップ
  - 様々な標準アルゴリズム
- MPI + OpenMP による並列化
- スクリプト言語(Python)によるポスト処理
- 国際共同作業のためのインフラストラクチャー
  - オープンソースのツールの利用

XML の基礎

- XML ＝ eXtensible Marckup Landuage
- 構造化文書を作成するのに適している
- 「タグ」を使って、文章の構造を記述
- 大文字と小文字は区別される
- XML の例: HTML (XHTML)

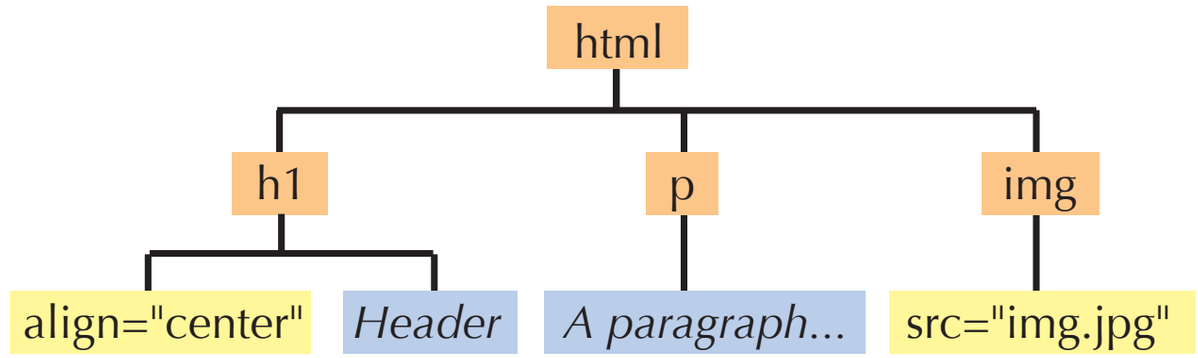


XML の文法

- 「開始タグ」には対応する「終了タグ」が必要
- ある要素の「開始タグ」と終了タグは共通の親ノードに含まれなければならない
- XML の「木」表示

```
<html>
  <h1 align=″center″>Header</h1>
  <p>A paragraph... And below it an image</p>
  <img src=″img.jpg″/>
</html>
```

⇒



- XML では、ユーザが独自のタグ (要素の名前) を定義すること が可能

  (参考: Document Type Definition (DTD)、XML Schema)

なぜ XML を使うのか?

plain text file

```
# parameters
10 0.5 10000 1000
# mean, error
-10.451 0.043
```

XML file

```
<PARAMETER name="L">10</PARAMETER>
<PARAMETER name="T">0.5</PARAMETER>
<PARAMETER name="SWEEPS">10000</PARAMETER>
<PARAMETER name="THERMALIZATION">1000</PARAMETER>
<AVERAGE name="Energy">
  <MEAN> -10.451 </MEAN>
  <ERROR> 0.043 </ERROR>
</AVERAGE>
```

- 人間にはどちらが読みやすい?
- 機械にはどちらがよみやすい?
- 数年後に読んで理解できるのはどっち?

データの拡張性

- 新しいパラメータを追加する

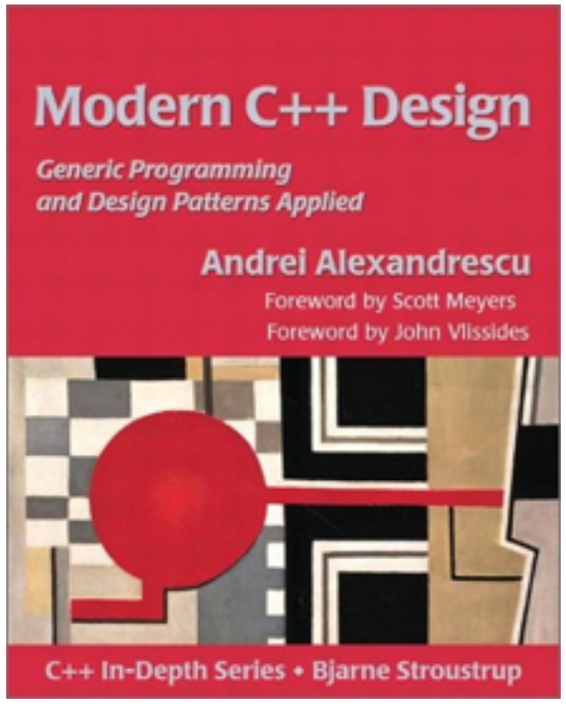
```
# parameters
10 0.5 10000 1000 12
# RNG
“lagged fibonacci”
# mean, error
-10.451 0.043
```

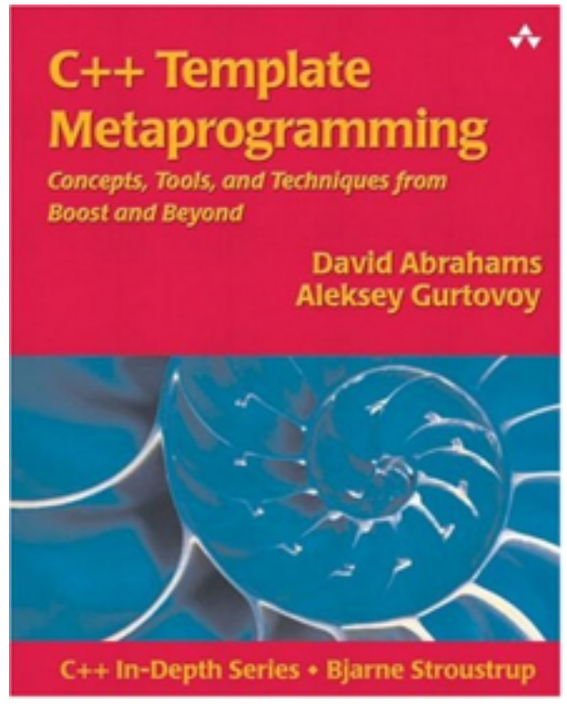
```
<PARAMETER name=“L”>10</PARAMETER>
<PARAMETER name=“T”>0.5</PARAMETER>
<PARAMETER name=“SWEEPS”>10000</PARAMETER>
<PARAMETER name=“THERMALIZATION”>1000</PARAMETER>
<PARAMETER name=”SEED”>12</PARAMETER>
<RNG name=”lagged fibonacci”/>
<AVERAGE name=“Energy”>
  <MEAN> -10.451 </MEAN>
  <ERROR> 0.043 </ERROR>
</AVERAGE>
```

- テキスト形式の場合、これまでのプログラムは動かなくなる
- XML の場合には問題ない (必要のないパラメータは読まれない)

C (Fortran) vs C++

- C++ は C の(ほぼ完全な)スーパーセット
- 厳密な型チェック
- ユーザ定義型 (クラス)
  - 継承 (inheritance)、仮想関数(virtual function)
- 演算子(+、* …)の再定義、関数の多重定義
- テンプレート
- 例外処理、etc...





ユーザ定義複素数型の例

```
class Complex {
public:
  Complex(double r, double i) { re=r; im=i; } // 初期設定

  double real() const { return re; } // メンバ関数

  double imag() const { return im; }
  double abs() const { return sqrt(re*re + im*im); }
  Complex& operator=(const Complex& rhs) {
    re = rhs.re; im = rhs.im;
  } // 代入演算子

  Complex operator+(const Complex& c) const {
    return Complex(re + c.re, im + c.im);
  } // +演算子

  // ...
private:
  double re, im; // メンバ (外部からは直接アクセス不可)

};

Complex a(1,0), b(0,1); // クラスComplexのインスタンスa,bを作成

Complex c = a+b;
cout << c.real() << ' ' << c.imag() << ' ' << c.abs; // 出力
```

クラスを使う利点

- 概念(考え方)の具体化
- データとそれに対する操作の関連付け
- データのカプセル化
  - 複数のデータをひとまとめにする
  - 不変条件の実現
- 継承と仮想関数
  - あるクラス(親クラス、基底クラス)から、子クラス(派生クラス)を作成することができる。 メンバ関数は基底クラスのものが継承される
  - 派生クラスにより(部分的に)上書きも可能
  - 派生クラスのオブジェクトは基底クラスのポイ ンタ(or 参照)へ代入可能
  - 仮想関数の仕組みを使うと基底クラスのポインタを介したメンバ関数の呼び出しは実際のクラスのメンバ関数に置き換えられる

継承と仮想関数

```
// 基底クラス
class Number {
public:
  virtual double abs();
};

// 派生クラス1
class RealNumber : public Number {
public:
  double abs() { return val; }
  double val;
};

// 派生クラス2
class Complex : public Number {
public:
  double abs() { return c.abs(); }
  Complex c;
};
```

```
// 仮想関数の使用例
void output(Number& num) {
  cout << num.abs();
}

RealNumber rn;
ComplexNumber cn;

// ...

// RealNumber::abs()が呼ばれる
output(rn);

// ComplexNumber::abs()が呼ばれる
output(cn);
```

Class/Function Template

- テンプレート: 型のパラメータ化 (一種のマクロ??)
- クラステンプレート

```
template<class T> Complex {
public:
  T real() const { return re; }
  T abs() const { sqrt(re*re + im*im); }
private:
  T re, im;
};

Complex<double> c; // 定義中の T が double で置き変えられる
```

- 関数テンプレート

```
template<class T>
T abs(const Complex<T>& c) {
  return c.abs();
}
```

ライブラリ化とコールバック

- アルゴリズムのライブラリ化
- 「操作」をいかにプログラム化するか?
- 例) ソーティング (bubble sort)
  - ソートされるデータ列とデータの大小を判定する関数、要素を交換する関数が提供されれば、原理的にはどんな型のデータもソーティング可能

```
subroutine sort(n, a)
dimension a(n)
do i = n, 2, -1
  do j = 1, i-1
    if (a(j).lt.a(j+1)) then
      b = a(j)
      a(j) = a(j+1)
      a(j+1) = b
    endif
  enddo
enddo
```

コールバック関数による実装

- ソート関数には、比較、交換を行う関数(コール バック関数)を引数として渡す
- データはcommonブロックを使ってコールバック関数に引き渡す

```
function less(i,j)
logical less
common /sortdata/n,a(n)
return (a(i).lt.a(j))

subroutine swap(i,j)
common /sortdata/n,a(n)
b = a(i)
a(i) = a(j)
a(j) = b

subroutine sort(n, less, swap)
logical less
do i = n, 2, -1
  do j = 1, i-1
    if (less(j,j+1)) then
      call swap(j,j+1)
    endif
  enddo
enddo
```

仮想関数による実装

- メンバー関数lessとswapを持つ基底仮想クラスを定義
- そのクラスから派生した具象クラスを作成。less とswapを実装
- 関数sortの引数は基底仮想クラスとする。仮想関数の仕組みにより、自動的に具象クラスのメンバー関数が呼び出される

```
class sort_array {
public:
  virtual bool less(int i, int j);
  virtual void swap(int i, int j);
};

class sort_a : public sort_array {
public:
  bool less(int i, int j) {
    return a(i) < a(j);
  }
  void swap(int i, int j) { ... }
  double a[];
};

void sort(int n, sort_array& a) {
  for (int i = n-1, i > 0; --i)
    for (int j = 0; j < i; ++j)
      if (a.less(j,j+1))
        a.swap(j,j+1);
}
```

コールバックと仮想関数

- 継承+仮想関数メカニズム = (データのカプセル化)+コールバック関数群のカプセル化
- 欠点
  - [Fortran] ソートする配列毎に common ブロックが必要
  - [C++] ソートする配列は必ず sort_array から派生しなくてはならない
  - [共通] 最も内側のループの中でポインタを介して関数が呼び出される

    ⇒ インライン化不可、大きなオーバヘッド

Templateを用いた実装

- 関数sortの引数をテンプレート化
- lessとswapをメンバー関数として持つクラスのインスタンスであれば、sort可能
- 特定のクラスから継承する必要なし

```
template<class A>
void sort(int n, A& a) {
  for (int i = n-1, i > 0; --i)
    for (int j = 0; j < i; ++j)
      if (a.less(j,j+1))
        a.swap(j,j+1);
}

class myarray {
public:
  bool less(int i, int j) {
    return a(i) < a(j);
  }
  void swap(int i, int j) { ... }
  double a[];
};

myarray a;
//…
sort(n, a);
```

動的 vs 静的コールバック

- クラス継承+仮想関数メカニズム
  - 動的コールバック (動的多様性)
  - どのコールバック関数呼び出されるかは、実行時に決まる。インライン化不可
- テンプレートを用いた実装 (ジェネリックプログラミング)
  - 静的コールバック (静的多様性)
  - コンパイル時にコールバック関数が決定される。インライン展開可能

Template Metaprograming

- テンプレート引数には、class 以外にも定数式 (int)や論理値(bool)も指定可能

```
template<int I, bool b> class myclass { /* ... */ };
```

- 特別バージョンを定義することができる

```
template<class T> myclass { /* ... */ };
template<> myclass<double> { /* ... */ };
```

- テンプレートを用いて、再帰構造や条件文を実現可能

  ⇒ テンプレートメタプログラミング

- 全てコンパイル時に展開、評価される

Static Factorial (N!)

```
// 一般の N 用
template<int N> class Factorial {
public:
  enum {
    value = N * Factorial<N-1>::value
  };
};

// N=1 用特別バージョン
template<> class Factorial<1> {
public:
  enum {
    value = 1;
  }
};

Factorial<5>::value; // コンパイル時に 120 と評価される
```

スクリプト言語

- スクリプト言語とは
  - コンパイルなしでその場で実行できる(簡易)プログラミング言語
  - メジャーなスクリプト言語：Perl、Python、Rubyなど
    - Mac OS X や Linux には、あらかじめインストールされている
    - Windows では、バイナリパッケージをダウンロード & インストール
  - ソースの編集・コンパイル・インストールなどの手間がかからない
    - その場で入力 ⇒ 即実行 ⇒ 出力
    - 電卓代わりにも
  - パッケージ(ライブラリ、モジュール)が充実しているので便利
    - CSV/XML/HDF5の読み書き、DBへのアクセス、ネットワーク、CGI、機器の制御、文字列処理、線形演算・最適化、計算・実験結果の加工、図の作成、GUI、、、
- いろいろなアイデアを手軽に試すことが可能

ALPS Python

- Boost.Python ライブラリの利用
  - C++のクラスや関数を Python から利用可
  - シミュレーション結果の読み込み、モンテカルロデータの非線形処理のための Python モジュール (pyalps) を提供
  - パラメータの準備・シミュレーションの実行を Python から行うことも可

pyalps と matplotlib による結果のプロット

```
import pyalps
import pyalps.plot as alpsplot
import matplotlib.pyplot as pyplot

data = pyalps.loadMeasurements(pyalps.getResultFiles(prefix='parm9a'),
    'Magnetization Density^2')
for item in pyalps.flatten(data):
    item.props['L'] = int(item.props['L'])

magnetization2 = pyalps.collectXY(data, x='T',
    y='Magnetization Density^2', foreach=['L'])
magnetization2.sort(key=lambda item: item.props['L'])

pyplot.figure()
alpsplot.plot(magnetization2)
pyplot.xlabel('Temperture $T$')
pyplot.ylabel('Magnetization Density Squared $m^2$')
pyplot.legend(loc='best')
```