【配信講義】CMSI計算科学技術特論C (2015) 第9回
2015年11月26日

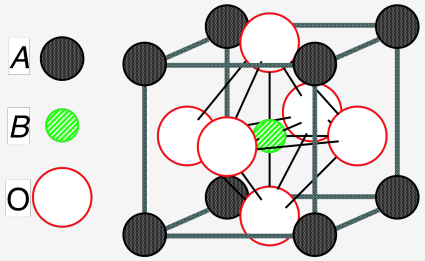
feramと強誘電体①

東北大学金属材料研究所 西松毅
t-nissie@imr.tohoku.ac.jp

自己紹介とferamの開発の経緯

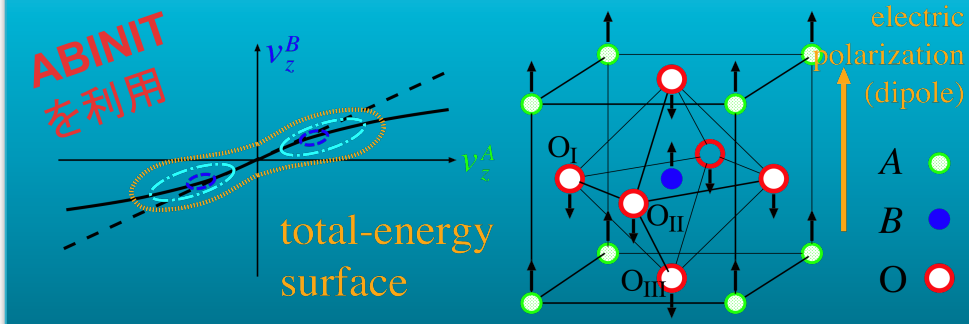
- 東北大理学部物理　crd2txt.exeをakiu.gwから公開
  - ftp.tohoku.ac.jp:/pub/Windows/Util/crd2txt/crd2tx10.lzh
- 2000年　阪大吉田博研で博士→東北大金研で助教
- 2003年～　強誘電体の研究を始める
  - NEC/TOKINからの国内留学生と → 谷底線法を開発（後述）
- 2004年12月～2005年2月（3ヶ月間）
  - インド・バンガロールの Jawaharlal Nehru Centre for Advanced Scientific Research (JNCASR) に滞在
  - Umesh V. Waghmare教授らと強誘電体薄膜の研究
  - **feram**の開発を開始、キャパシタのモデル化のアイデア
- 2006年3月～2008年2月（2年間）
  - 米国Rutgers大学に滞在
  - David Vanderbilt教授らと強誘電体について研究
- 2008年9月　やっと自分が書いた論文を出版！

feramを1枚のスライドで紹介:
誘電体のマルチスケールシミュレーション
http://loto.sourceforge.net/feram/



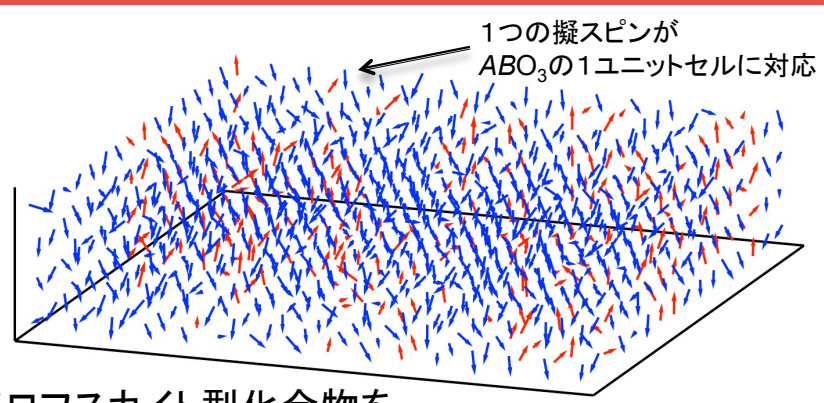
ペロブスカイト型
酸化合物ABO_3

第一原理計算による有効ハミルトニアンH_{eff}の構築



[T. Hashimoto, T. Nishimatsu *et al.*: Jpn. J. Appl. Phys. **43**, 6785 (2004)]

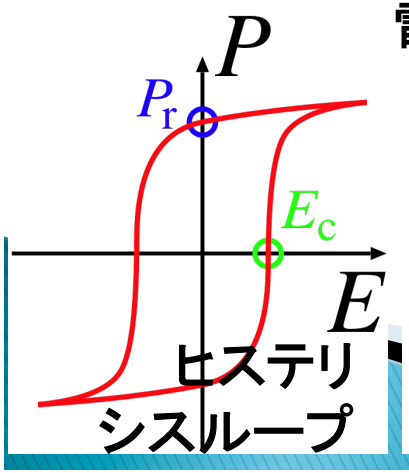
feramによる分子動力学計算

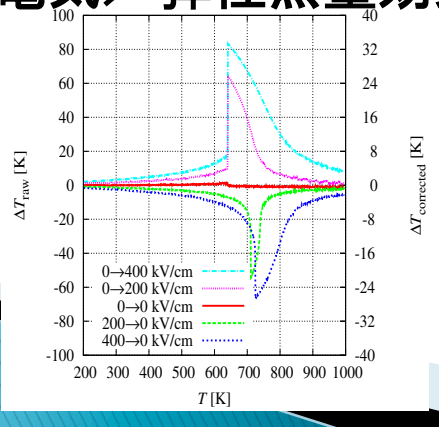


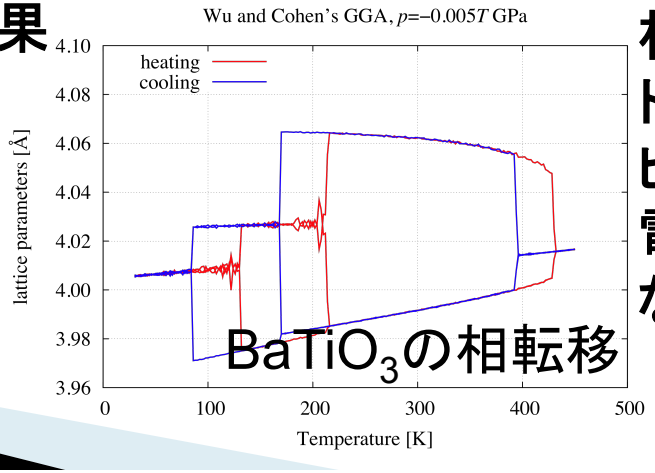
ペロフスカイト型化合物を
双極子相互作用のある擬スピン系としてH_{eff}を
時間発展させ，その有限温度の性質をシミュレート

大規模(> 100 nm), 長時間 (> 100 ns) の計算が可能

電気／弾性熱量効果







**相転移や
ドメイン構造，
ヒステリシスループ，
電気／弾性熱量効果
などのシミュレーション**

PRL **99**, 077601 (2007),
PRB **78**, 104104 (2008),
PRB **82**, 134106 (2010),
JPSJ **84**, 024716 (2015), …

feramで使われている主な理論の発展

- R. D. King-Smith and David Vanderbilt: **“First-principles investigation of ferroelectricity in perovskite compounds”**, Phys. Rev. B **49**, 5828 (1994). 全エネルギ表面.
- W. Zhong, D. Vanderbilt, and K. M. Rabe: **“First-principles theory of ferroelectric phase transitions for perovskites: The case of $BaTiO_3$”**, Phys. Rev. B **52**, 6301 (1995). MC.
- U.V. Waghmare, E. J. Cockayne, and B. P. Burton: **“Ferroelectric Phase Transitions in Nano-scale Chemically Ordered $PbSc_{0.5}Nb_{0.5}O_3$ using a First-principles Model Hamiltonian”**, Ferroelectrics **291**, 187 (2003). FFTでMD.
- Takeshi Nishimatsu, Umesh V. Waghmare, Yoshiyuki Kawazoe and David Vanderbilt: **“Fast molecular-dynamics simulation for ferroelectric thin-film capacitors using a first-principles effective Hamiltonian”**, Phys. Rev. B **78**, 104104 (2008). キャパシタ.

feramの概要(1)

- 強誘電体に特化した高速分子動力学シミュレーター
- 開発ポリシー
  - 論文を読んだ人がシミュレーションを再現できるように
  - KISS (Keep It Short and Simple)
  - 拝借できるものは拝借
    - ライブラリ (FFTW(計算時間の1/3), LAPACK)
    - Autotools (autoconf+automake) → 来週
    - SourceForge.org (Subversionリポジトリ、Webページ、メーリングリスト)
- ライセンス GNU GPLv3
  - だれでも自由に使ってもらえたらうれしい
  - ついでに論文とURLを引用してくれたらもっとうれしい

MD

feramの概要(2)

- 現在$BaTiO_3$, $PbTiO_3$, $KNbO_3$のパラメータを同梱
- 長距離の双極子−双極子相互作用をFFTで高速計算
  - 高速フーリエ変換 (FFT) ライブラリFFTWを利用
- OpenMPで並列化
  - 基本的に1ノードの計算機で高速に走る
- バルクだけでなく薄膜キャパシタのシミュレーションが可能
- ./configure && make で簡単なコンパイル
- Fortram 95/2003 で Object oriented programming (OOP)

feramの行数、開発者数、ユーザ数

2015年11月現在

分類	概数
本体 (Fortran: *.F, *.f, *.h)	5,500行
テスト (Fortran, Shell, Ruby)	1,000行
ツール(Fortran, Shell, Ruby, PostScript, Gnuplot)	1,000行
ドキュメント、他 (configure.ac, Makefile.am, etc.)	5,000行
time (./configure && make -j4) # Core i5, SSD, gfortran6	11.4 秒
開発者	1名
ユーザ(なぜか女性が多い。よいところに就職できる。)	10名
ソースパッケージ feram-0.24.02.tar.xz の大きさ	18.5 MB
リリースしたバージョンの数	40
ダウンロード数	2,000
feramを用いて書かれた論文の数	20本

ソフトウエアの命名法
なぜ feram という名前にしたのか

- 将来的に強誘電体メモリ (FeRAM) の設計にまで使えるようにしたい
- Search Engine Optimization (SEO) でFeRAMの検索で上位にくるように
- ぜんぶ小文字、できればCourierで feram ← FeRAM, FRAM®と差別化
- ツール類はferam_cross_section_q.shとferam_で始まるようにしてインストール先の名前空間に配慮

絶縁体の誘電性による分類

誘電体 (dielectrics)

圧電体 (piezoelectrics)

焦電体 (pyroelectrics)

強誘電体 (ferroelectrics)

高

対称性

低

$BaTiO_3$系や$PbZr_xTi_{1-x}O_3$系 (PZT) などの強誘電体セラミックスが工業的に優れた性質を持っているので、上記すべての誘電性についての応用製品がある。強誘電体セラミックスは日本のメーカーのシェアが高い割合を占めるものが多い、日本の得意分野。

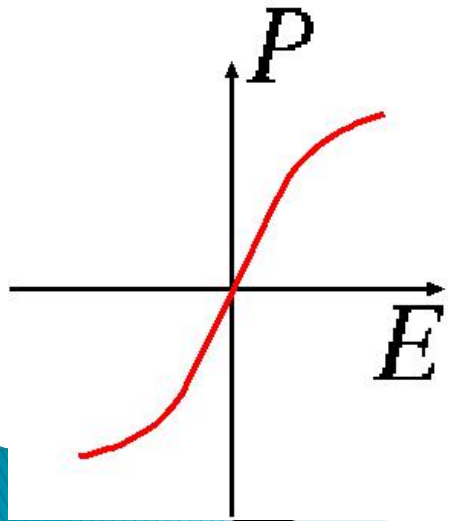
結晶の対称性と圧電性・焦電性

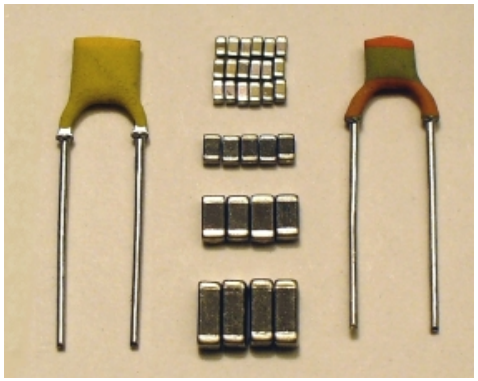
極性	反転対称性	晶族数	結晶系										
			立方		六方		正方		三方（菱面体）		斜方	単斜	三斜
非極性結晶 (22)	有 (11)	11	O_h	T_h	D_{6h}	C_{6h}	D_{4h}	C_{4h}	D_{3d}	C_{3i}	D_{2h}	C_{2h}	C_i
	無 (21)	11	O T_d	T	D_6 D_{3h}	C_{3h}	D_4 D_{2d}	S_4	D_3		D_2		
極性（焦電性）結晶 (10)		10			C_{6v}	C_6	C_{4v}	C_4	C_{3v}	C_3	C_{2v}	C_2 C_6	C_1

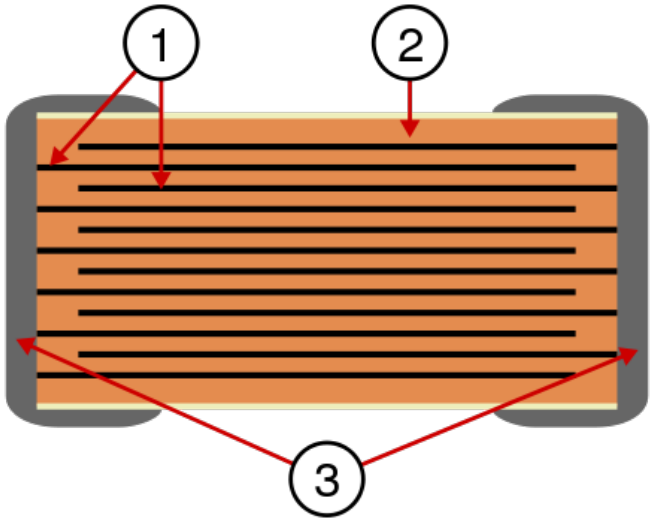
白地：圧電性結晶　赤字：焦電性結晶

誘電体 (dielectrics) とは？

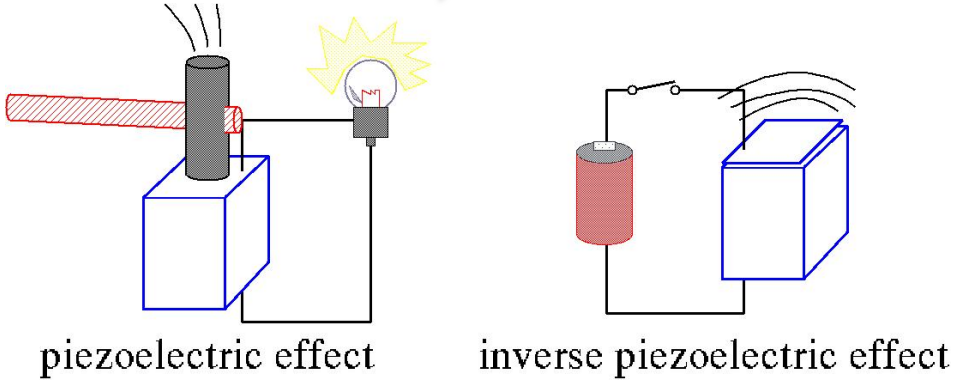
- 電場をかけると分極する
- 電場をゼロにすると分極もゼロになる
- 応用：普通のコンデンサー (capacitor)
  セラミックコンデンサーは$BaTiO_3$系など
- 応用：コンデンサーマイク（電極間にあるのは空気）







圧電体 (piezoelectrics) とは？



- 圧電効果（と逆圧電効果）をもつ
- GaAs（ZnS構造 T_d）など単純な構造でも圧電性はある
- 応用：圧力センサー（ランガサイト $La_3Ga_5SiO_{14}$）
- 水晶振動子（quartz SiO_2）
- 圧電スピーカー（PZT、携帯電話に多用されている）
- セラミック振動子、加速度センサー、ジャイロ（PZT）
- 超音波画像診断装置、魚群探知機、ソナー
  超音波モーター、etc…

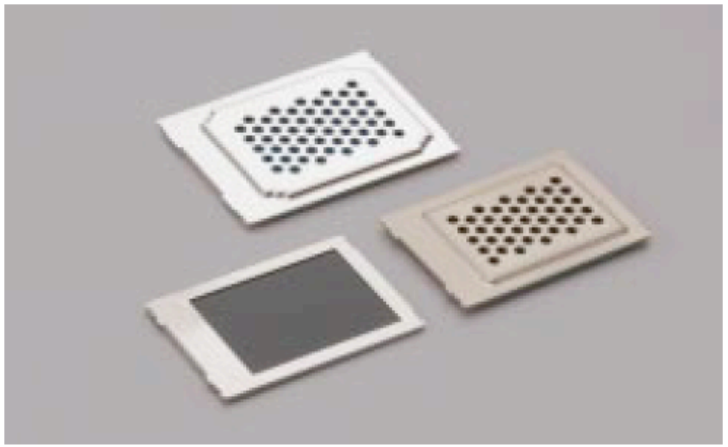
圧電体の応用: 圧電スピーカー

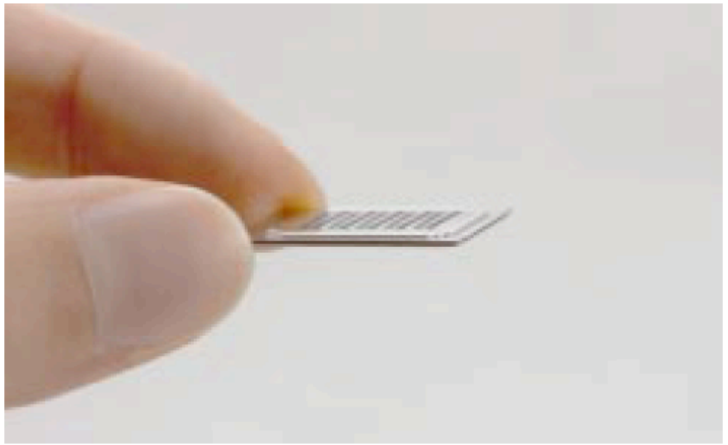
圧電スピーカー

VSLBP/VSLBFシリーズ

■特長
- 高音質・高音圧を実現した薄型スピーカー
- 超薄型パッケージ：携帯機器の薄型化に貢献
  VSLBPシリーズ：厚み 1.2mm
  VSLBFシリーズ：厚み 0.5mm
- バッテリー持続時間の向上に貢献
- 圧電素子駆動のため、電磁ノイズがなく、また砂塵 (砂鉄) 吸い込みによる音圧の低下もなし
- 防水対応可能 (IPX7相当)※

■用途
携帯電話 / PDA / 携帯音楽プレーヤ / ICレコーダ / DSC 等





村田製作所のカタログ P82J.pdf
2011-06-30 より。下線は西松。

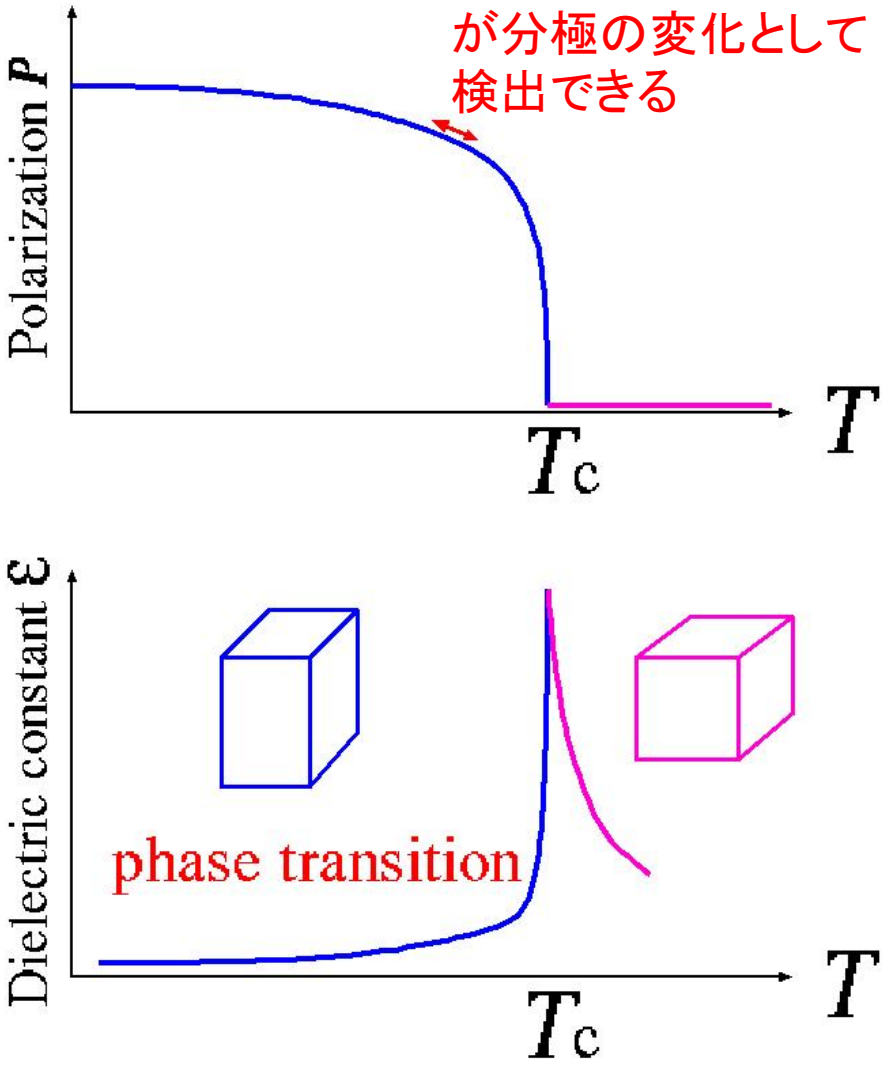
焦電体 (pyroelectrics) とは？

- 極性（電気分極）のある結晶
- 光をあてるとその分極がわずかに変化する
- 応用：人感センサー (PZT)

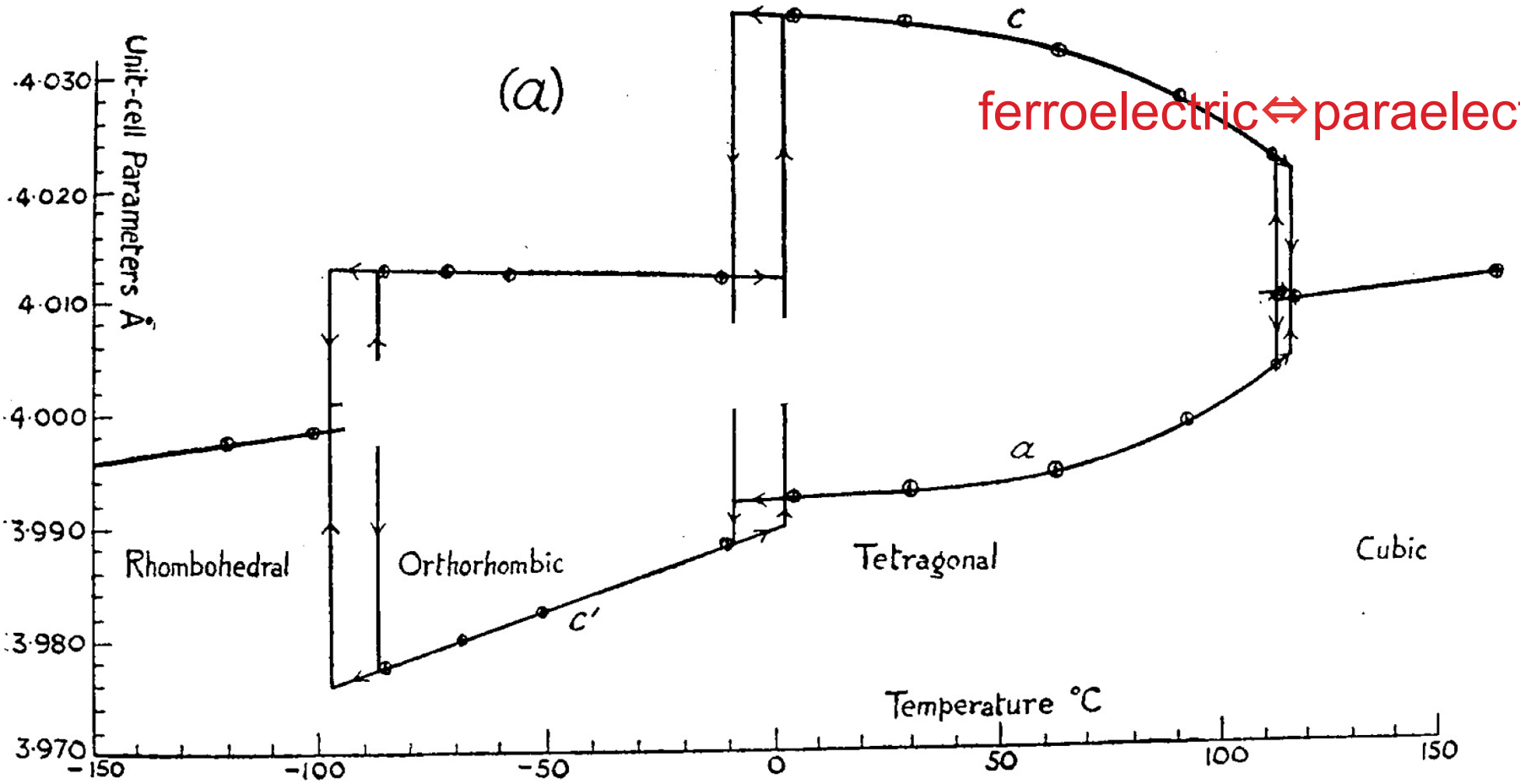
東北大金研の2階の廊下天井の照明点灯用人感センサー

- さらに電場により分極の反転が可能なものを強誘電体と呼ぶ（焦電体と強誘電体との区別は微妙）

赤外線の照射などのごく微小な温度変化が分極の変化として検出できる



$BaTiO_3$ (実験値)

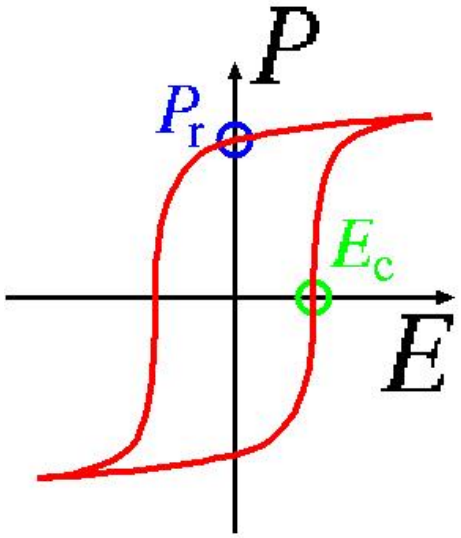


ferroelectric⇔paraelectric

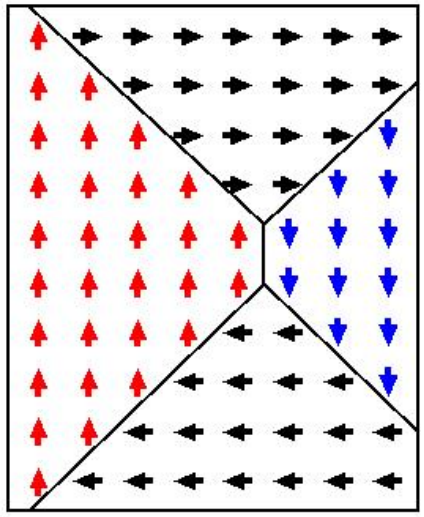
Experimentally observed temperature dependence of lattice constants for $BaTiO_3$. There are three first-order phase transitions. the cubic to tetragonal phase transition is nearly second-order one. After [H. E. Kay and P. Vousden: Philos. Mag. 40, 1019 (1949)].

強誘電体とは？
ドメイン構造とは？

- 電場により
  分極の反転が可能
- 電場をゼロにし
  ても自発分極が残る



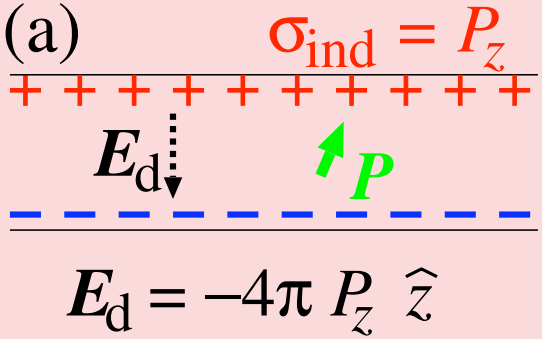
spontaneous polarization



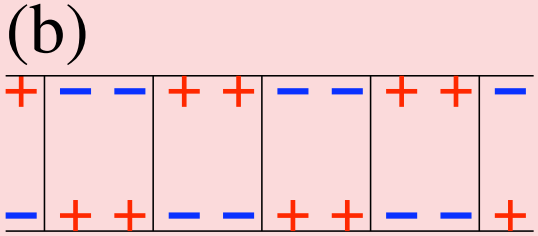
domain structure

- 双極子相互作用のある有限系ではドメイン構造をとることにより反分極場を小さくして系を安定化させる
  （ここで「有限系」とは、無限に広がった端のない強誘電体や強磁性体ではないという意味）

薄膜の場合
（紙面に垂直）



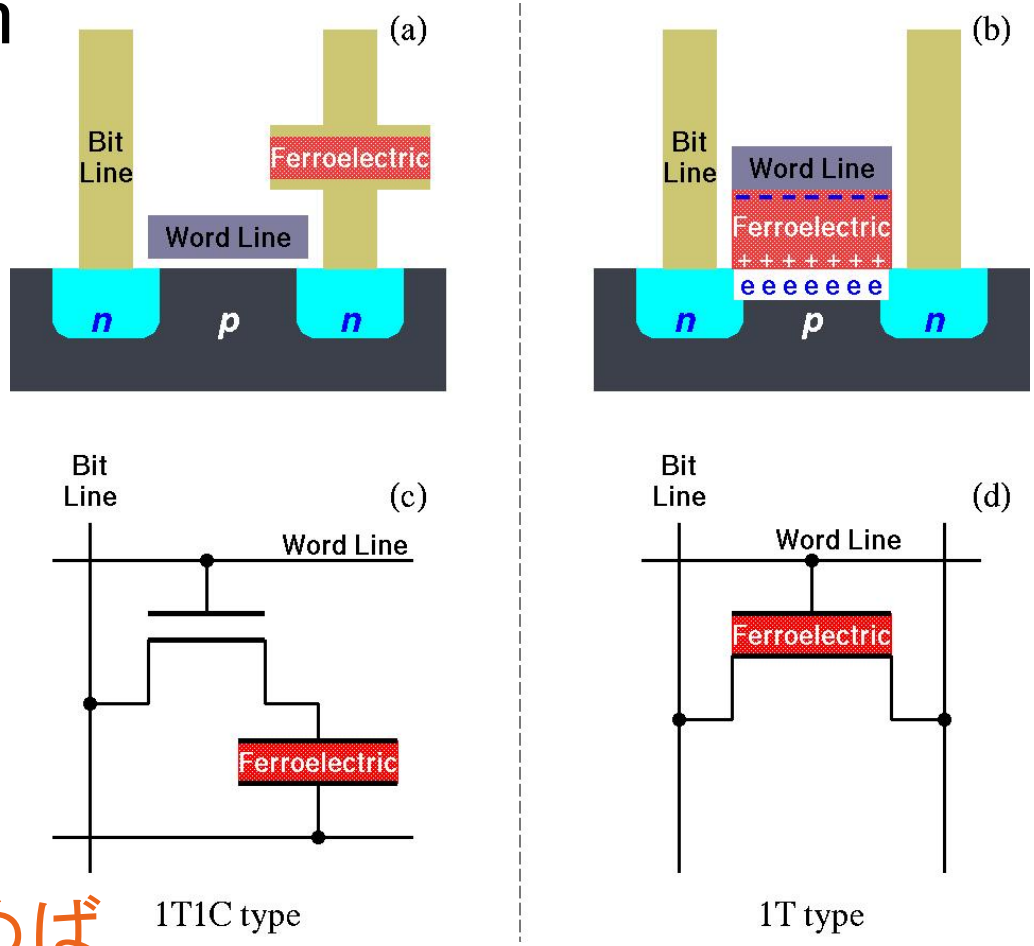
$E_d = -4\pi P_z \hat{z}$



forming domains, reducing E_d

強誘電体メモリー (FeRAM) とは？

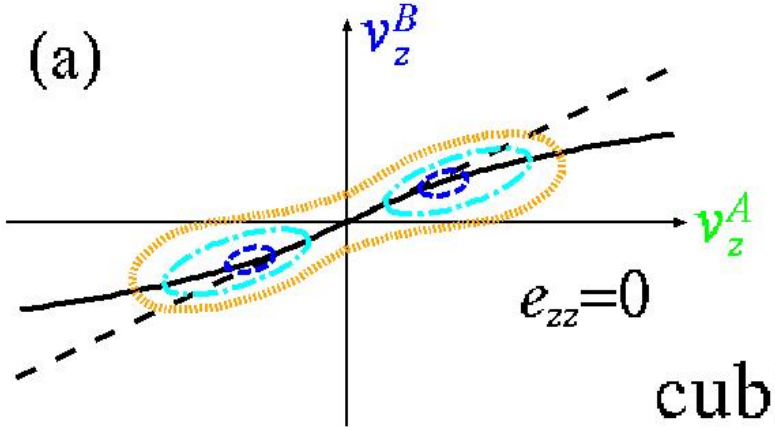
- Ferroelectric Random Access Memory (FeRAM) とは強誘電体薄膜キャパシタのヒステリシスを利用し正負の自発分極を1と0に対応させた不揮発性の半導体メモリー
- フラッシュ・メモリーより低電圧で動作
- 高集積・大容量化が進めば DRAM→FeRAM（高速・不揮発性・リフレッシュ不要）

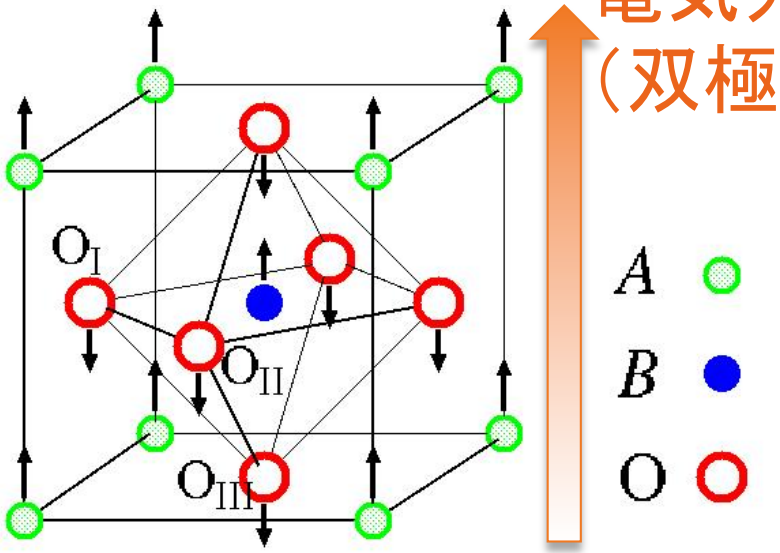


強誘電体の第一原理有効ハミルトニアンにもとづいた分子動力学シミュレーションの手順

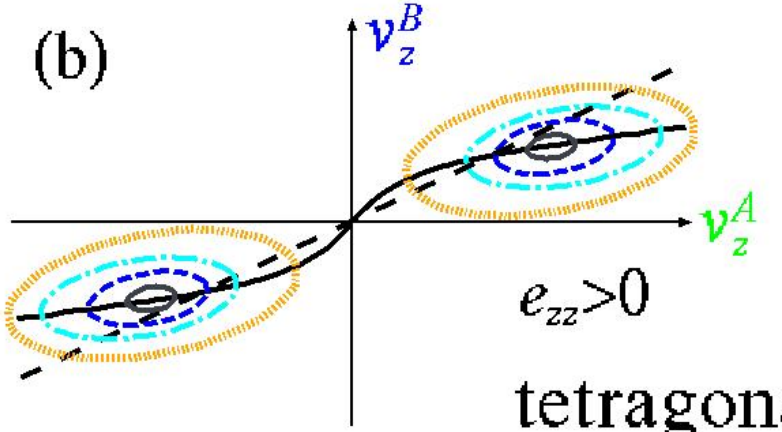
- 強誘電体$BaTiO_3$や$PbTiO_3$を**第一原理計算**で調べて有効ハミルトニアン（25個のパラメータを持つ）を構築
  - **ABINIT**を改造して利用 http://www.abinit.org/
    - 平面波展開: E_{cut}=60 Hartree, on 8x8x8 *k*-points
    - Rappeのノルム保存擬ポテンシャル http://opium.sf.net/
    - GGA (Wu and Cohen), LDAやGGA (PBE) ではダメ
  - 絶対0度の物性しかわからない
- その有効ハミルトニアンを**分子動力学法**を使っていろいろな条件下で時間発展して物性を予測
  - 独自開発した**feram**を利用 http://loto.sf.net/feram/
  - 大規模（32x32x512ユニット・セル、～100nm）な系の長時間（～100ns）のシミュレーションが可能
  - 温度、圧力、ひずみ、バルクか薄膜か、外部電場

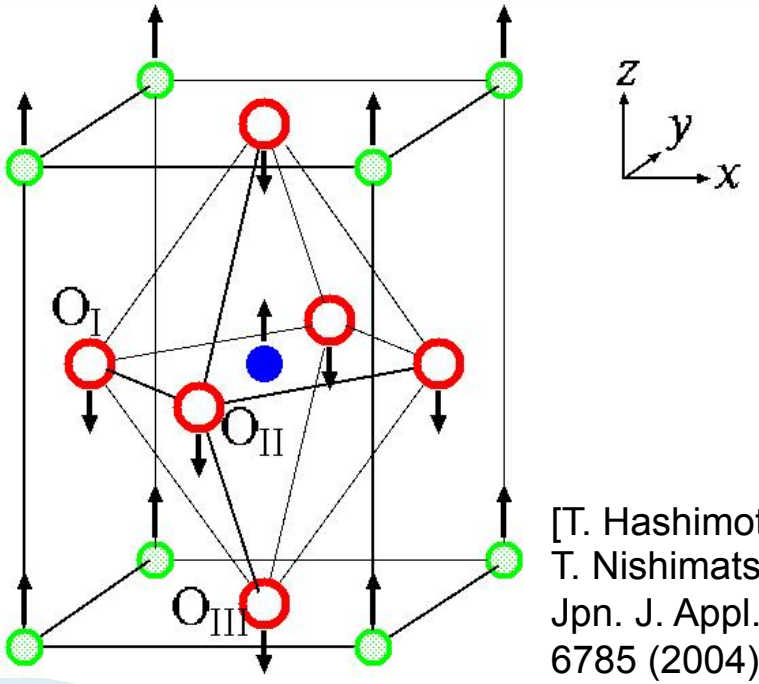
ペロブスカイト型強誘電体ABO_3の全エネルギー表面





電気分極
（双極子）

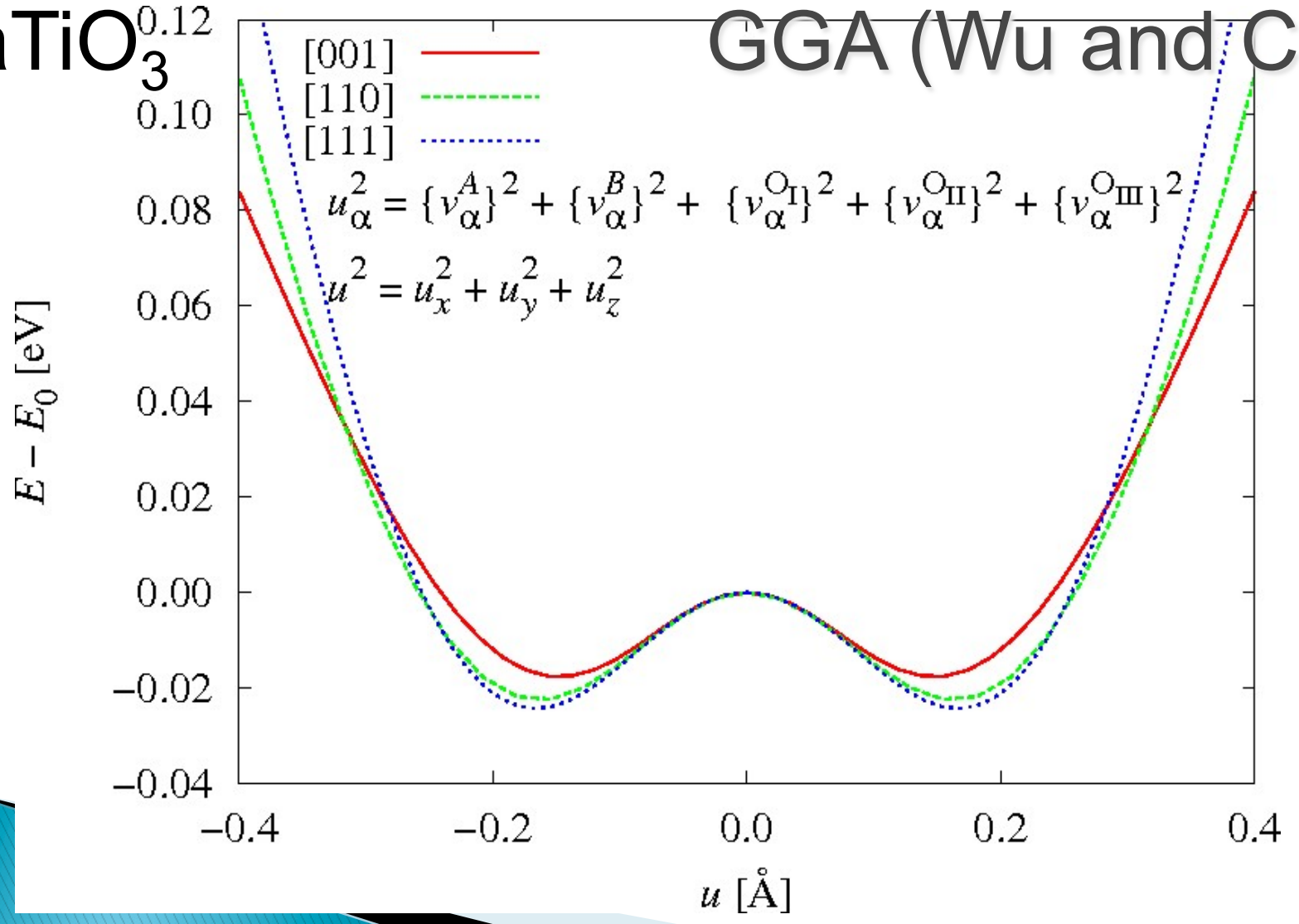




[T. Hashimoto, T. Nishimatsu *et al.*: Jpn. J. Appl. Phys. **43**, 6785 (2004)] より

全エネルギー表面を第一原理計算（ABINITパッケージを改造して利用）

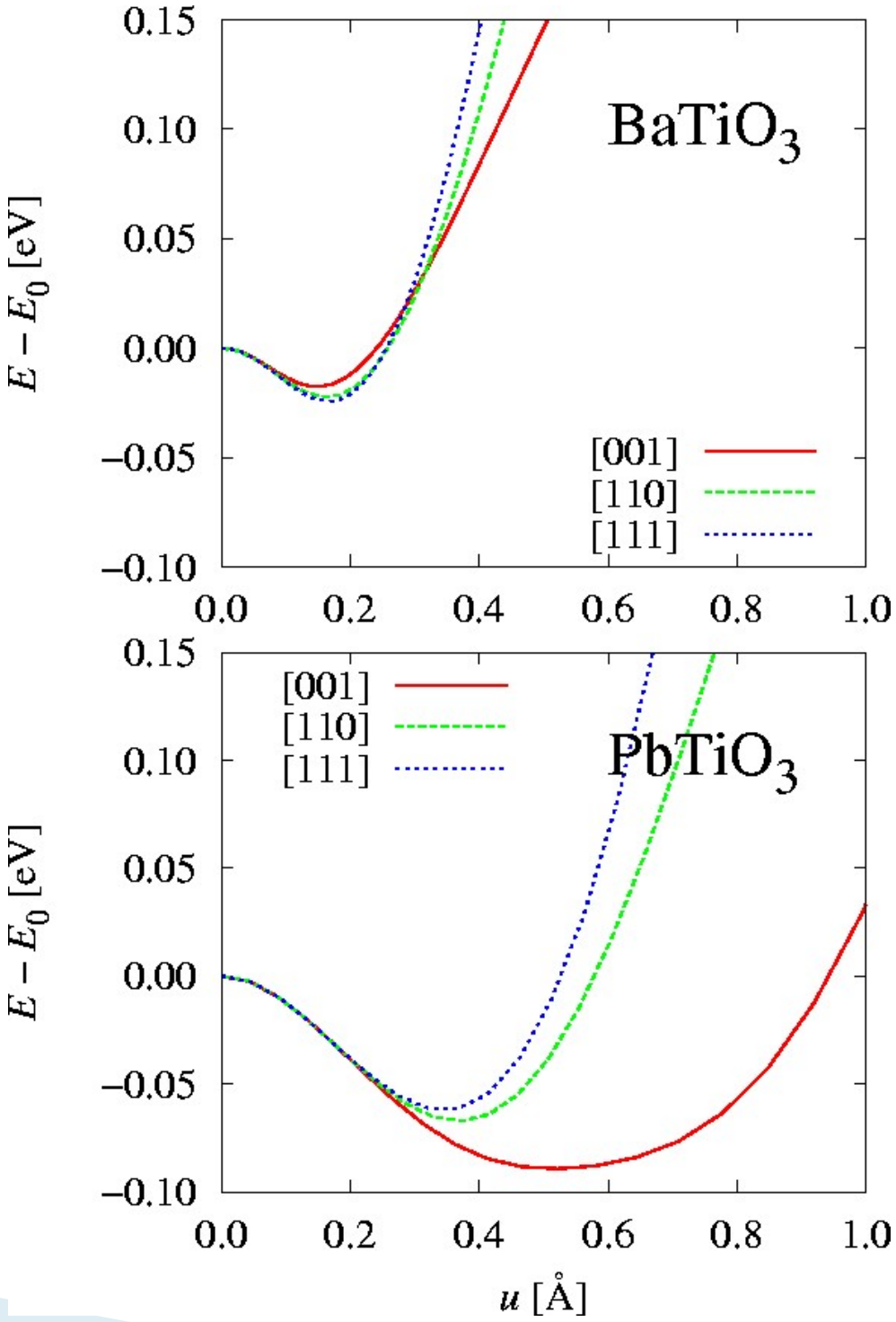
$BaTiO_3$ GGA (Wu and Cohen)



$BaTiO_3$と$PbTiO_3$との比較

- $BaTiO_3$の全エネルギー
  表面は浅くて原点に近い
- [111]方向に歪むのが
  最安定

- $PbTiO_3$の全エネルギー
  表面は深くて原点から
  遠い
- [001]方向に歪むのが
  最安定

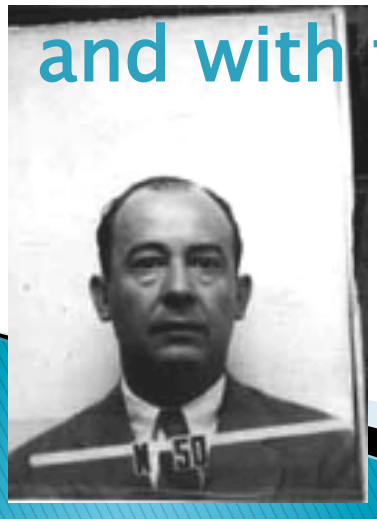


その他に必要な第一原理計算

- 格子定数、弾性定数を求める計算
- フォノンの分散関係の計算（線形応答で）
  より正確にはフォノンの dynamical matrix $D(k)$ ではなく、原子の質量で割る前の interatomic force constant matrix $\Phi(k)$を計算し、その固有値から擬スピン間の相互作用を見積もる

With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.

-- John von Neumann

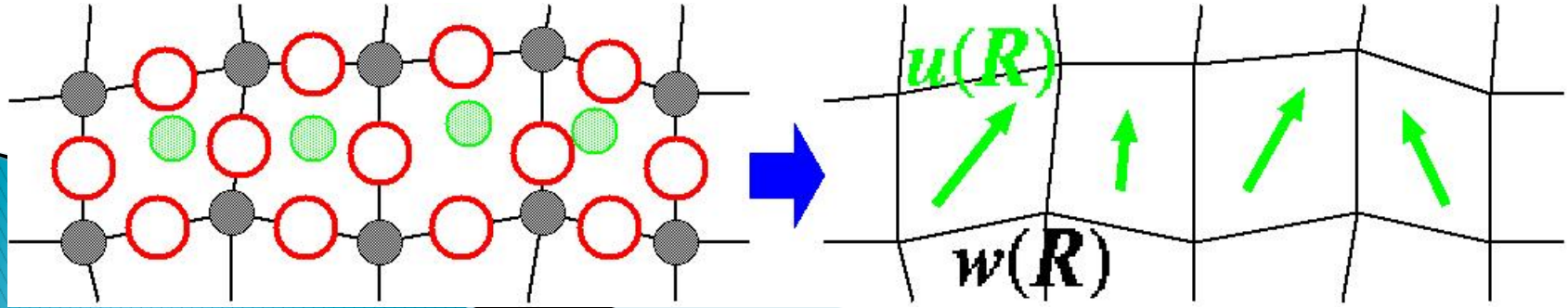


25個ものパラメータを第一原理計算だけで決めるのはわりとしんどい

-- 西松毅

強誘電体の分子動力学計算のための粗視化: 系の自由度を簡単化

- 実際のペロブスカイト型ABO_3: $15N+6$ 自由度
  - 単位胞中5個の原子
  - 各原子はx, y, zの3方向に動く
  - スーパーセル中N個の単位胞
  - 歪みの6成分
- 粗視化したモデル: $6N+6$ 自由度
  - 単位胞に1つの双極子ベクトル$Z^* u(R)$
  - 単位胞に1つの「音響変位」ベクトル$w(R)$
  - 計 6 自由度/単位胞



有効ハミルトニアン（25個のパラメータは第一原理計算により決める）

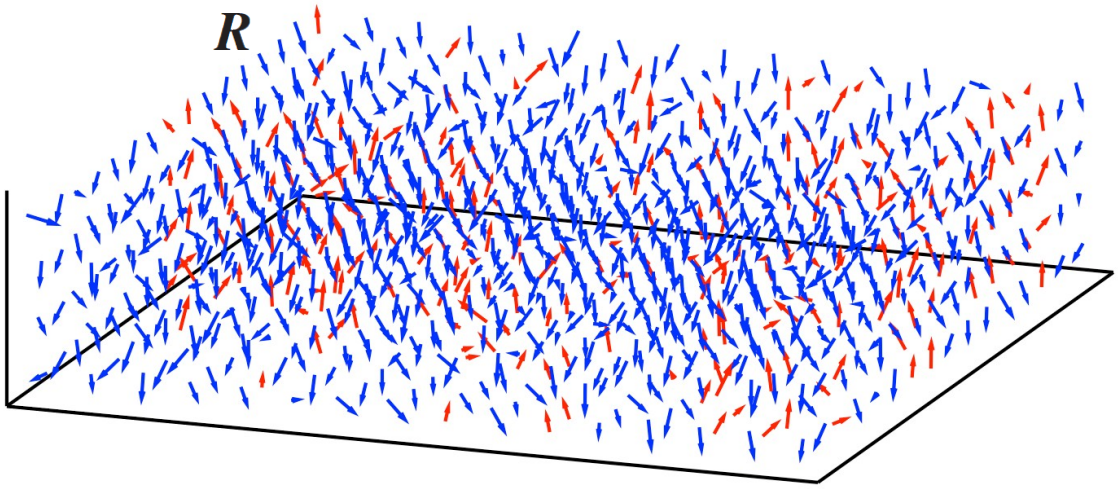
$$H^{\mathrm{eff}} = \frac{M^*_{\mathrm{dipole}}}{2}\sum_{\boldsymbol{R},\alpha}\dot{u}^2_\alpha(\boldsymbol{R}) + \frac{M^*_{\mathrm{acoustic}}}{2}\sum_{\boldsymbol{R},\alpha}\dot{w}^2_\alpha(\boldsymbol{R}) + V^{\mathrm{self}}(\{\boldsymbol{u}\})$$

$$+ V^{\mathrm{dpl}}(\{\boldsymbol{u}\}) + V^{\mathrm{short}}(\{\boldsymbol{u}\}) + V^{\mathrm{elas,homo}}(\eta_1, \cdots, \eta_6)$$

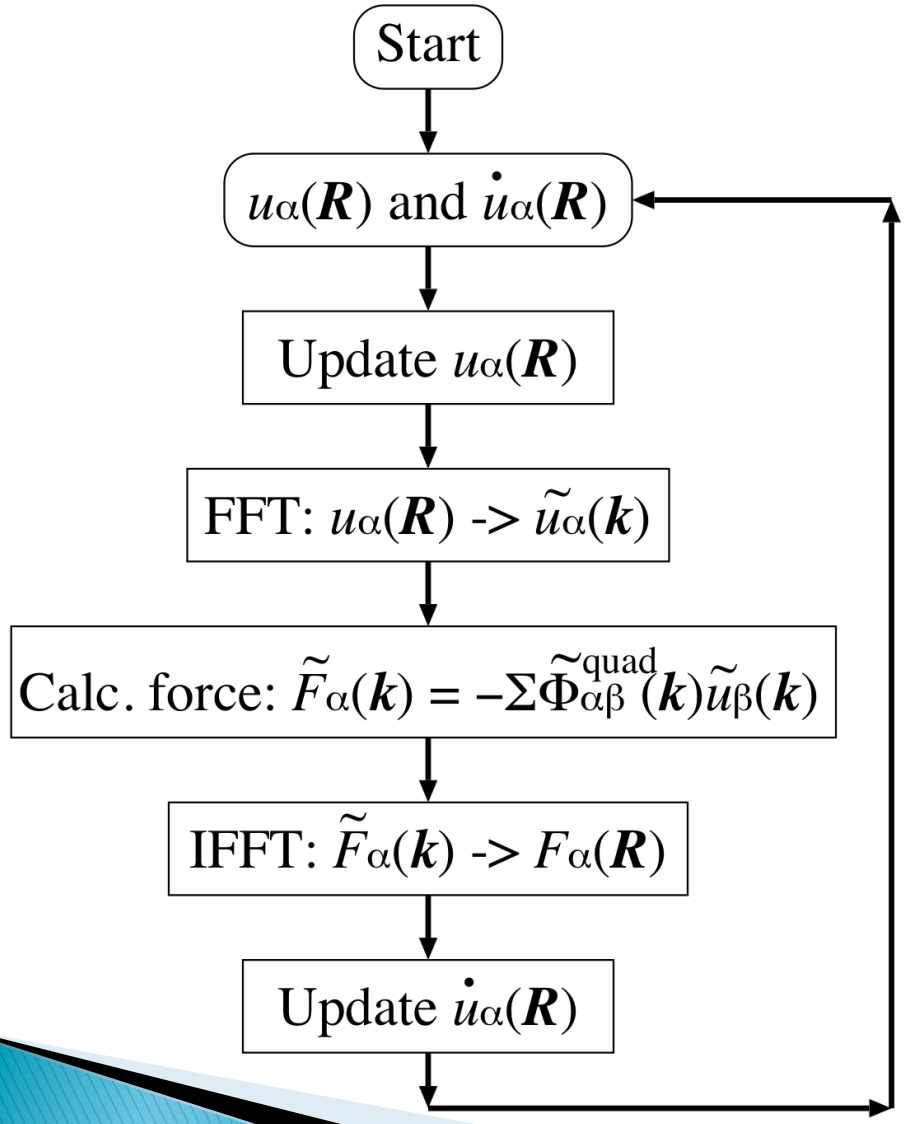
$$+ V^{\mathrm{elas,inho}}(\{\boldsymbol{w}\}) + V^{\mathrm{coup,homo}}(\{\boldsymbol{u}\}, \eta_1, \cdots, \eta_6)$$

$$+ V^{\mathrm{coup,inho}}(\{\boldsymbol{u}\},\{\boldsymbol{w}\}) - Z^*\sum_{\boldsymbol{R}} \mathcal{E}\cdot\boldsymbol{u}(\boldsymbol{R}),$$

すなわち、双極子をたくさんならべたスーパーセル（周期的境界条件）で計算



長距離双極子間相互作用の高速計算



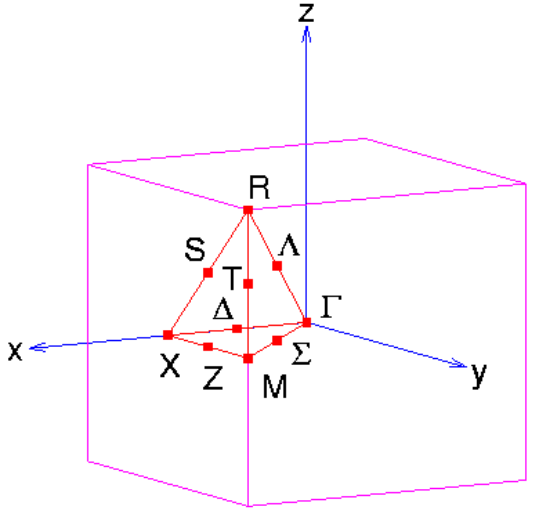
$$V^{\mathrm{dpl}}(\{\boldsymbol{u}\}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{\alpha}\sum_{j=1}^{N}\sum_{\beta} u_{\alpha}(\boldsymbol{R}_i)\Phi_{\alpha\beta}(\boldsymbol{R}_{ij})u_{\beta}(\boldsymbol{R}_j),$$

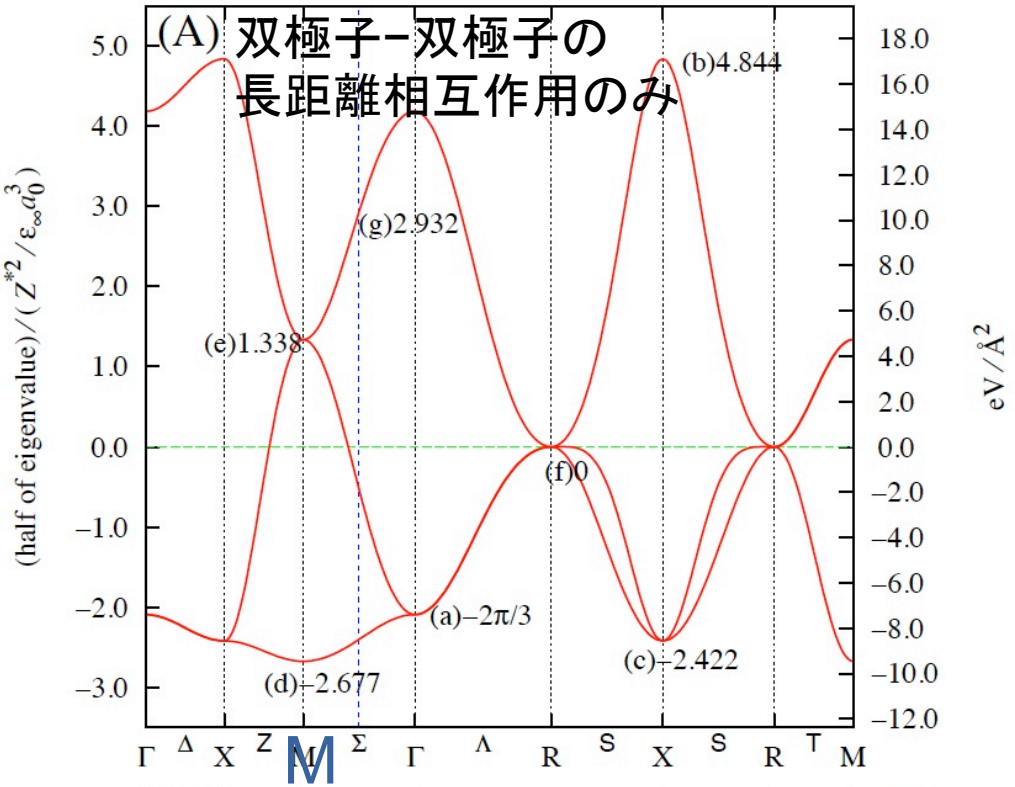
$$\Phi_{\alpha\beta}(\boldsymbol{R}_{ij}) = \frac{Z^{*2}}{\epsilon_{\infty}}\sum_{\boldsymbol{n}}{}' \frac{\delta_{\alpha\beta} - 3\widehat{(\boldsymbol{R}_{ij}+\boldsymbol{n})}_{\alpha}\widehat{(\boldsymbol{R}_{ij}+\boldsymbol{n})}_{\beta}}{|\boldsymbol{R}_{ij}+\boldsymbol{n}|^3},$$

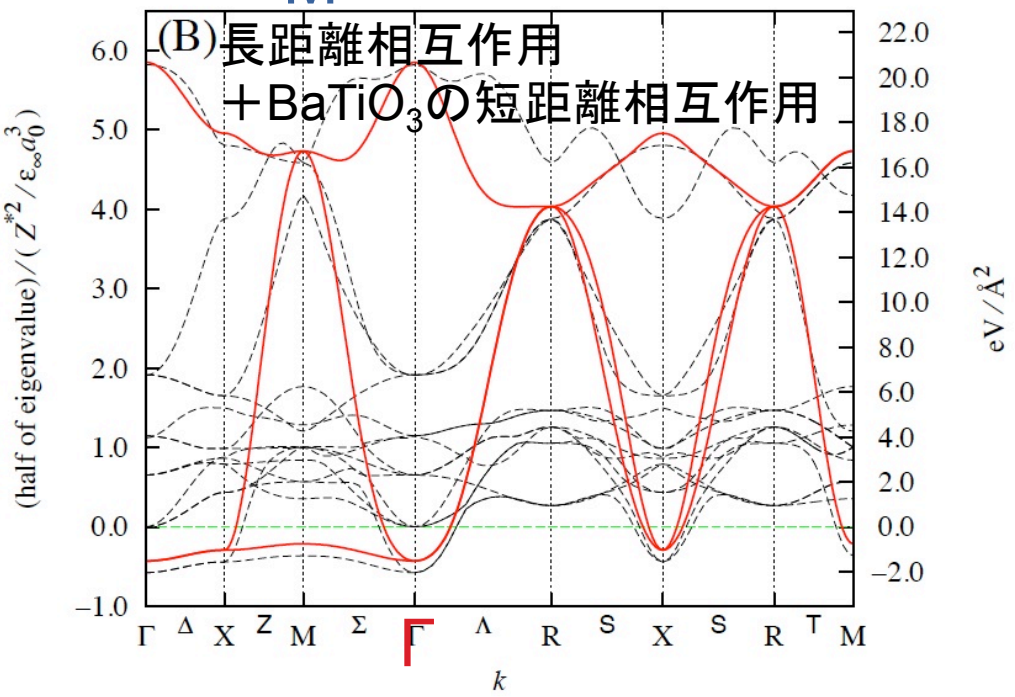
双極子に働く力の計算方法の簡単化したフローチャート．高速フーリエ変換(FFT) と逆FFT (IFFT) とが長距離双極子−双極子相互作用の高速計算を可能にし，実空間で計算していたら$O(N^2)$の計算時間がかかるものが$O(N\log N)$になる．

単純立方格子上の双極子uの相互作用

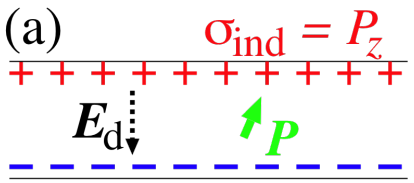
- 双極子－双極子の長距離相互作用のみではM点の反強誘電状態が最安定
- 短距離相互作用を入れてはじめてΓ点の強誘電状態が最安定になる



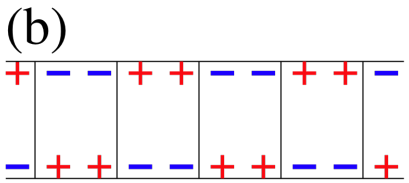




キャパシターのための周期境界条件

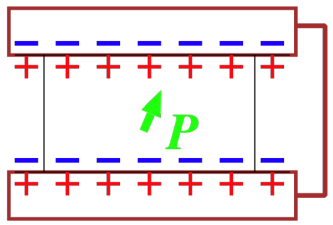


$E_d = -4\pi P_z \hat{z}$

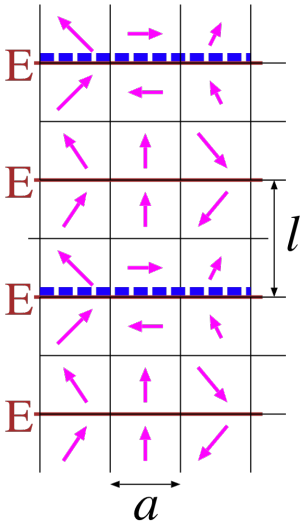


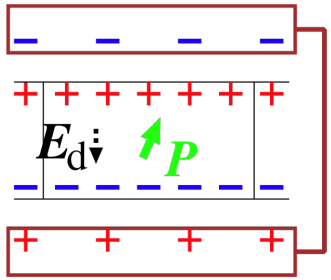
反分極場E_dが
一様分極を妨げる

forming domains, reducing E_d

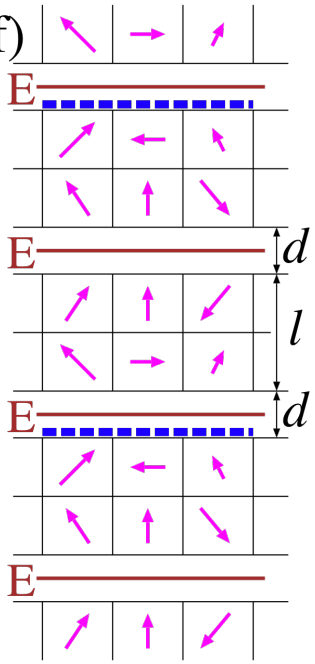


$E_d = 0$





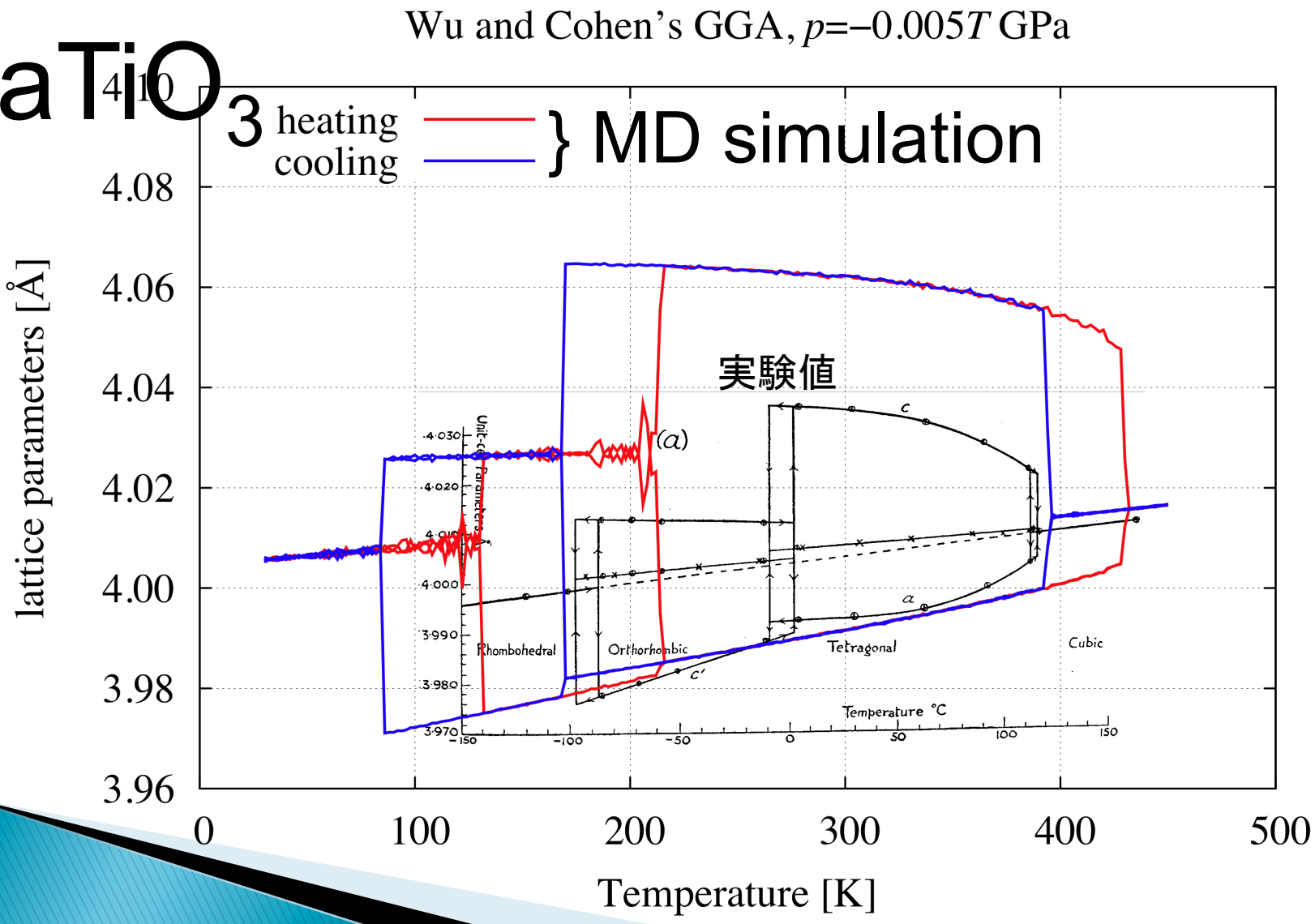
$E_d = -4\pi \frac{d}{l+d} P_z \hat{z}$



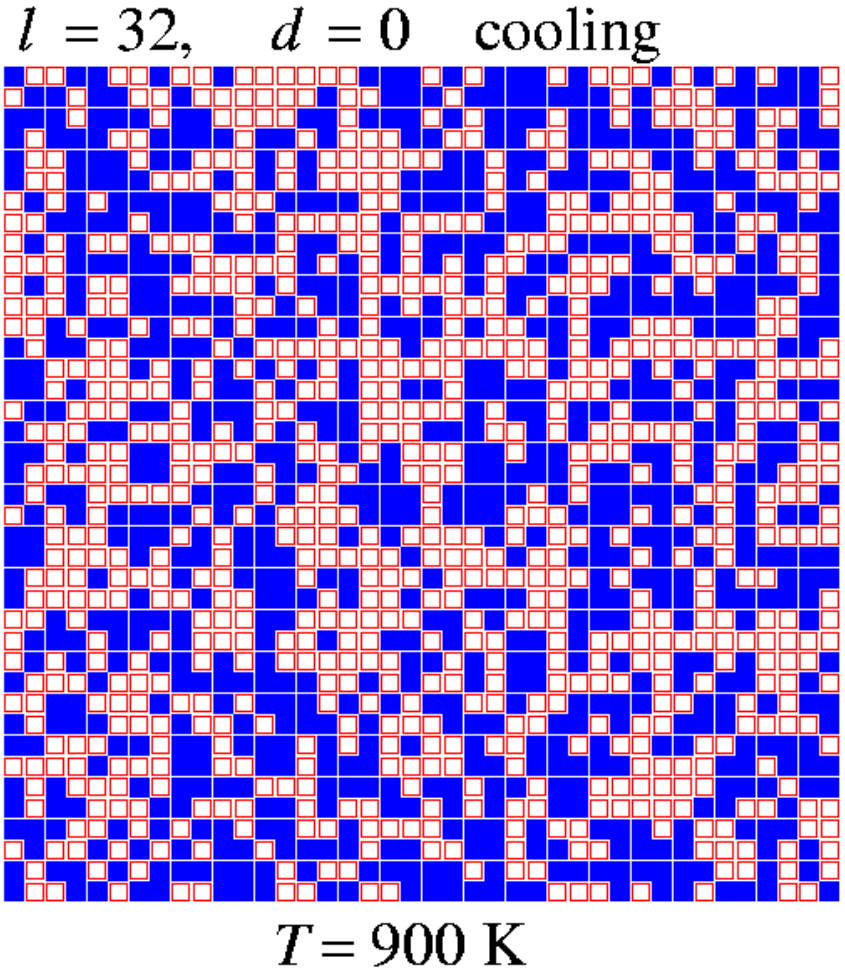
電極は静電的な鏡と見なすことができる

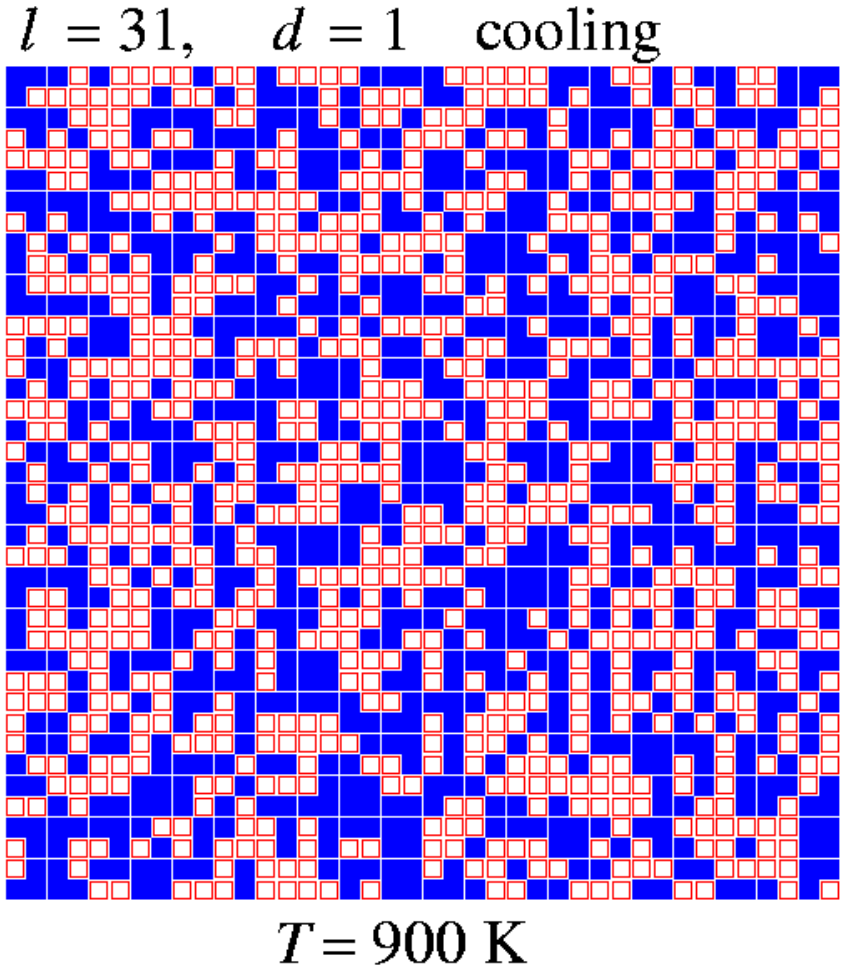
どんな計算ができるか: 相転移

$BaTiO_3$



$BaTiO_3$強誘電体キャパシタの電極依存性（断面の分極のアニメーション）



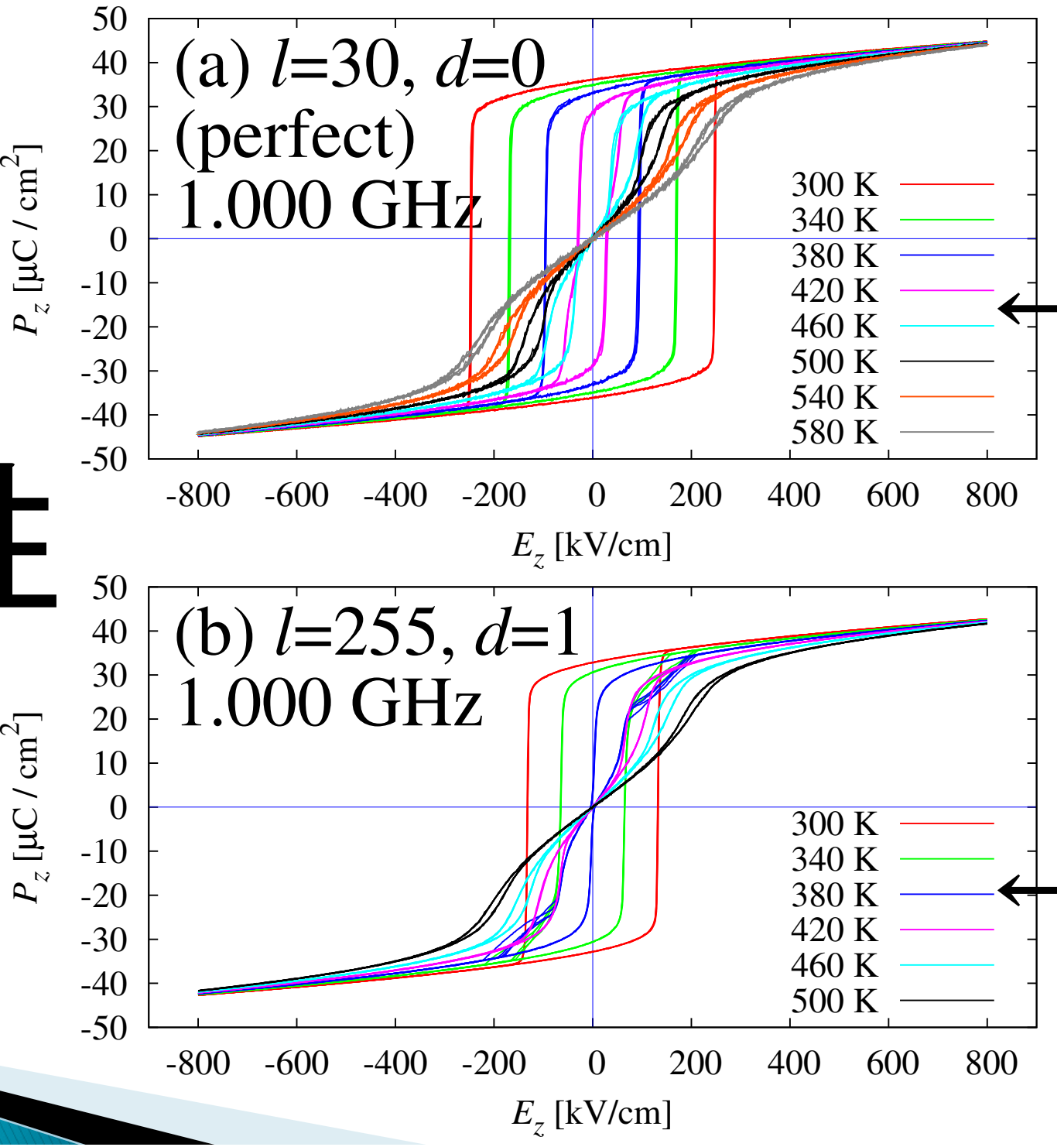


完全な電極（アニメーション）

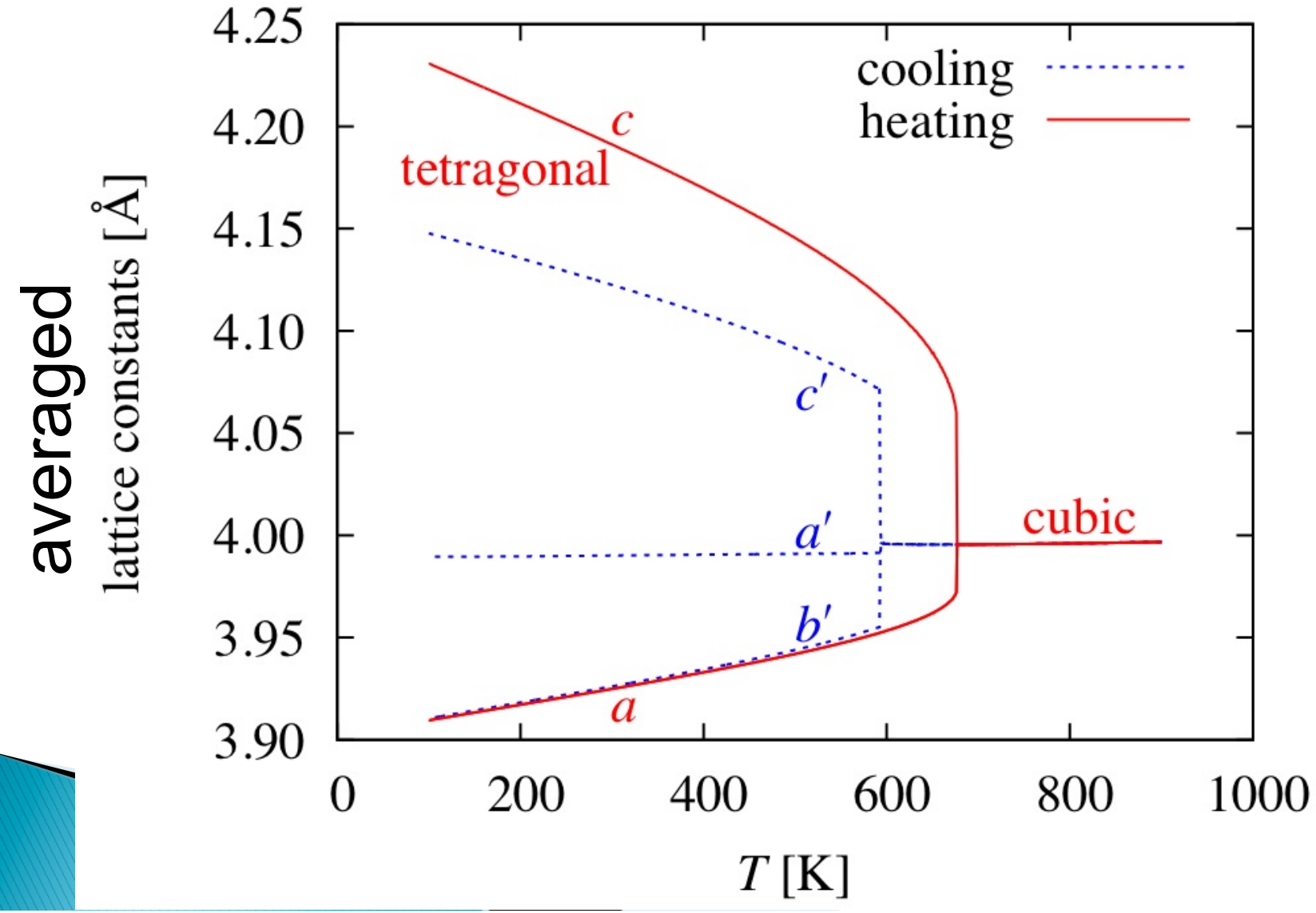
不完全な電極（アニメーション）

温度電極依存性

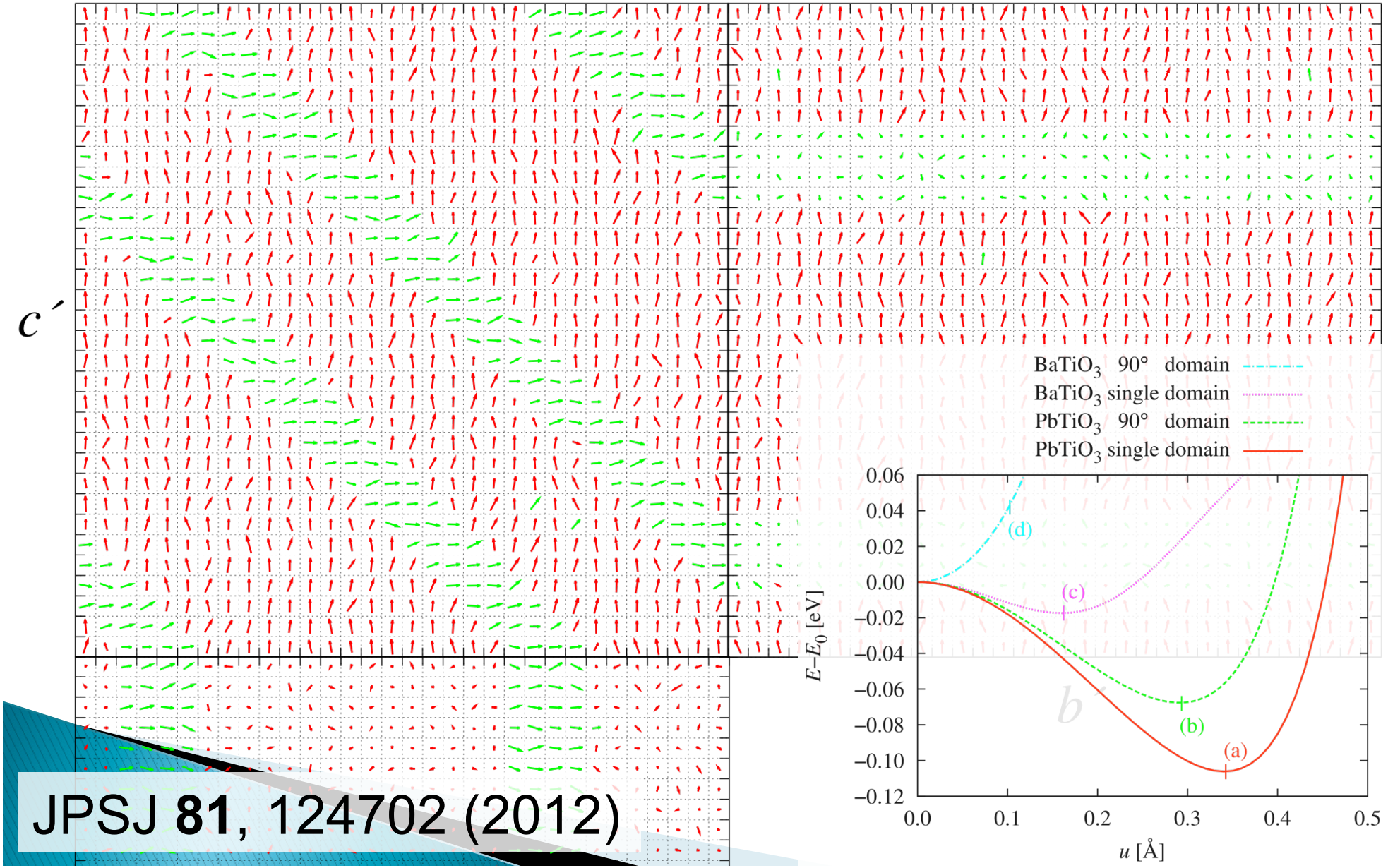
不完全な電極
→T_Cを下げる
→T_C付近での応用に課題



$PbTiO_3$は降温シミュレーションで90°ドメイン構造が凍結される



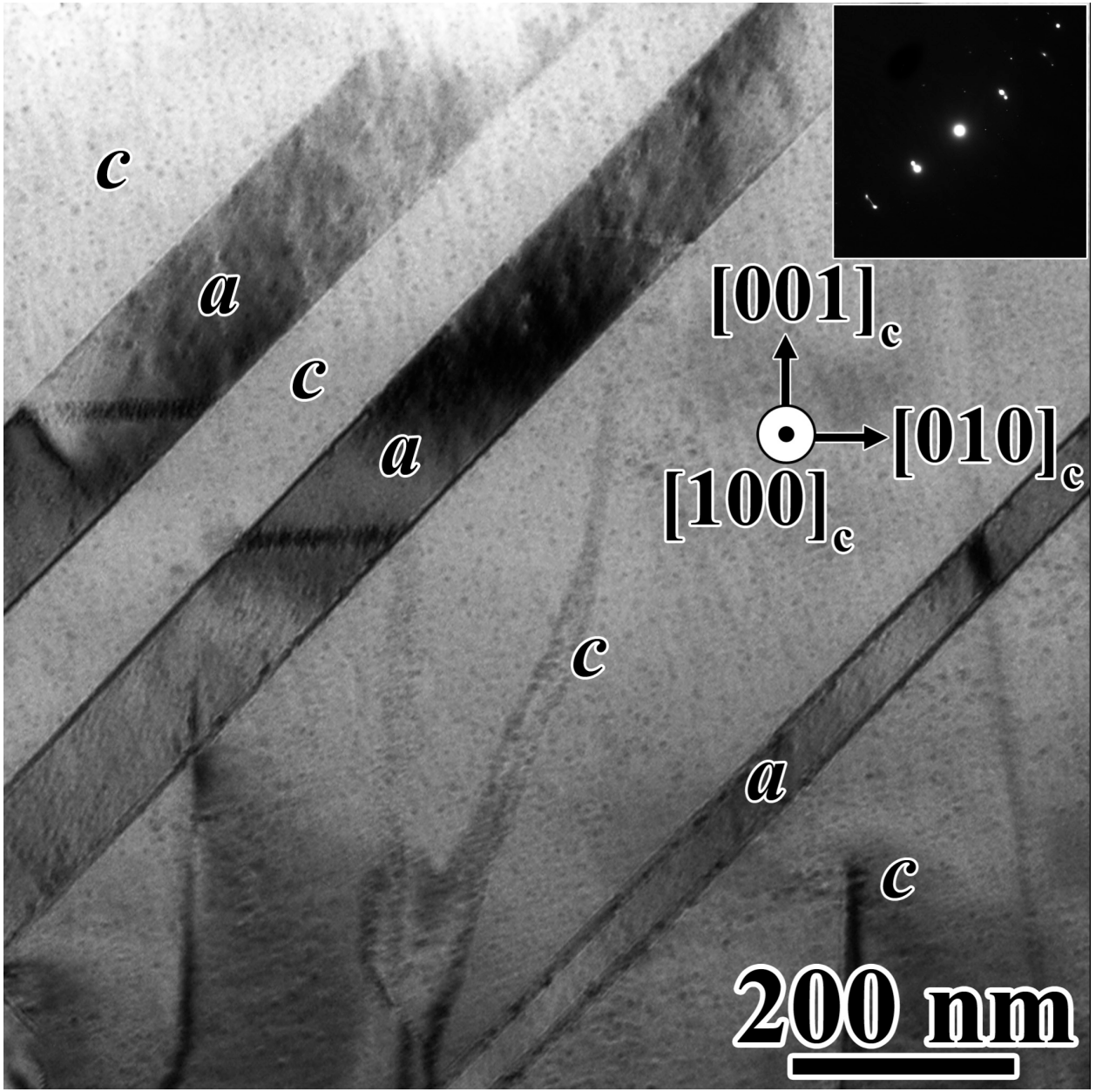
凍結された$PbTiO_3$の90°ドメイン構造



JPSJ **81**, 124702 (2012)

$PbTiO_3$の90°ドメイン

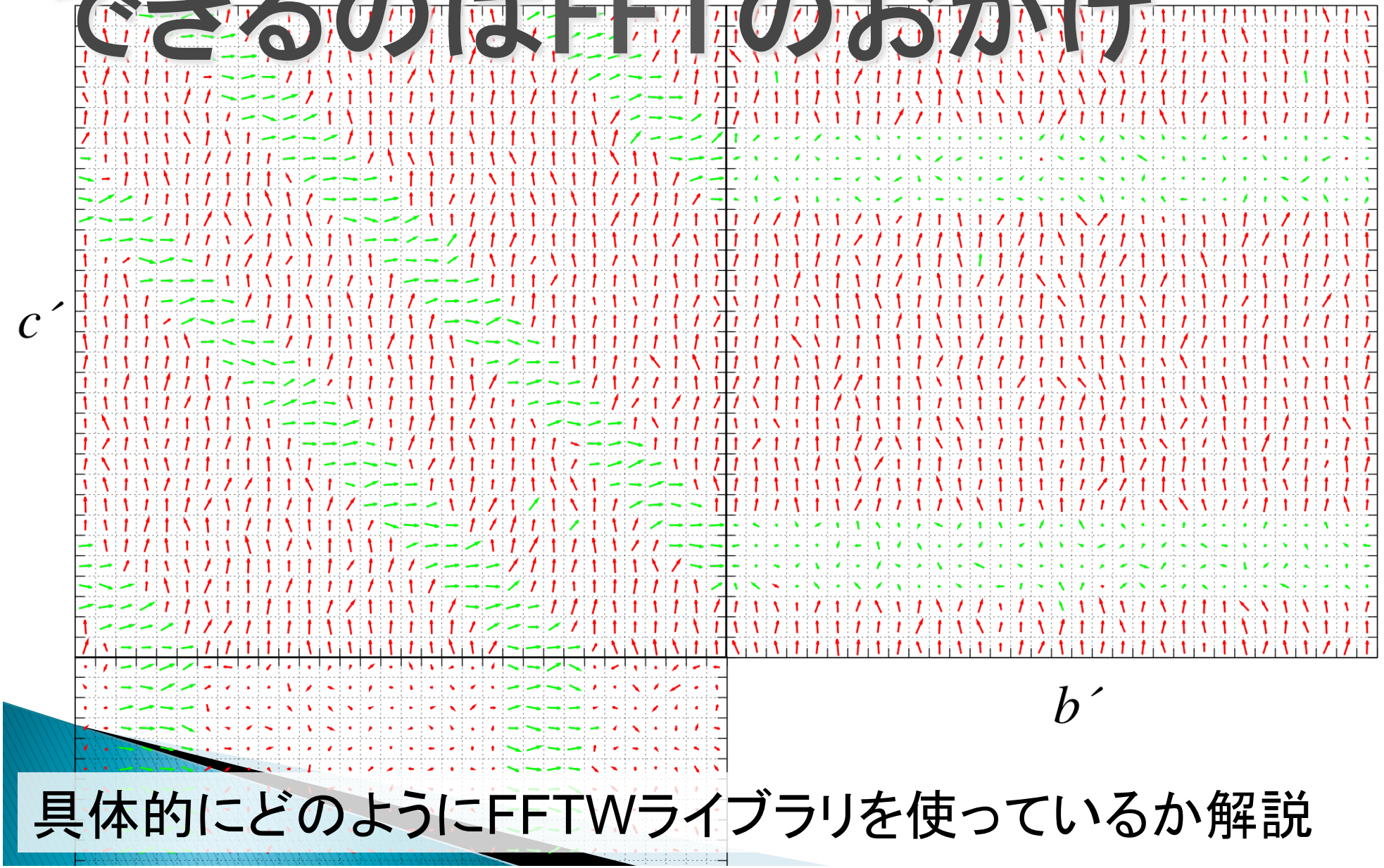
- $SrTiO_3$基板に成長させた$PbTiO_3$厚膜の明視野TEM像
- 基板に垂直な *c* ドメインの中に *a* ドメインを形成することで基板との miss-fit strain を緩和



前半のまとめ

- ペロブスカイト型強誘電体ABO_3のための第一原理有効ハミルトニアンに基づく高速な分子動力学計算コード**feram**を開発。フリーソフトウェアとして公開。
- 長距離の双極子－双極子相互作用をFFTで扱うことにより大規模で長時間のMDシミュレーションが可能になってきた。
- 強誘電体の相転移、ヒステリシスループ、ドメイン構造のシミュレーション
- 電気熱量効果と弾性熱量効果のシミュレーション（来週）

このような大きな系の計算ができるのはFFTのおかげ



具体的にどのようにFFTWライブラリを使っているか解説

離散フーリエ変換 (DFT) とは
一般的な定義（1次元）

- *N*個の（周期的な）**複素数**から、*N*個の（周期的な）**複素数**への写像
- 1 / *N* はどちらに付けてもよいし、それぞれに 1 / sqrt(*N*) を付けてもよい
- 計算量は約$8N^2$ FLOP

$$x = 0, 1, 2, \ldots, (N-1)$$

$$s = 0, 1, 2, \ldots, (N-1)$$

$$\tilde{c}(s) = \frac{1}{N} \sum_x c(x) e^{-i\frac{2\pi sx}{N}}$$

$$c(x) = \sum_s \tilde{c}(s) e^{i\frac{2\pi sx}{N}}$$

周期 T で N 回サンプリング

音波の解析など

$$t = \frac{T}{N}0, \quad \frac{T}{N}1, \quad \frac{T}{N}2, \quad \ldots, \quad \frac{T}{N}(N-1)$$

$$\omega = \frac{2\pi}{T}0, \quad \frac{2\pi}{T}1, \quad \frac{2\pi}{T}2, \quad \ldots, \quad \frac{2\pi}{T}(N-1)$$

$$\tilde{c}(\omega) = \frac{1}{N}\sum_{t} c(t)e^{-i\omega t}$$

$$c(t) = \sum_{\omega} \tilde{c}(\omega)e^{i\omega t}$$

周期aでN回サンプリング
結晶中の電子の波動関数(「点の)

$$x = \frac{a}{N}0,\quad \frac{a}{N}1,\quad \frac{a}{N}2,\quad \ldots,\quad \frac{a}{N}(N-1)$$

$$G = \frac{2\pi}{a}0,\ \frac{2\pi}{a}1,\ \frac{2\pi}{a}2,\ \ldots,\ \frac{2\pi}{a}(N-1)$$

$$\tilde{c}(G) = \frac{1}{N}\sum_{x} c(x)e^{-iGx}$$

$$c(x) = \sum_{G} \tilde{c}(G)e^{iGx}$$

フーリエ変換は様々に応用されている

周期NaでN回サンプリング

1次元結晶中のフォノンとか

$$X = \quad a0, \qquad a1, \qquad a2, \quad \ldots, \quad a(N-1)$$

$$k = \frac{2\pi}{a}\frac{0}{N}, \ \frac{2\pi}{a}\frac{1}{N}, \ \frac{2\pi}{a}\frac{2}{N}, \ \ldots, \ \frac{2\pi}{a}\frac{(N-1)}{N}$$

$$\tilde{c}(k) = \frac{1}{N}\sum_{X} c(X) e^{-ikX}$$

$$c(X) = \sum_{k} \tilde{c}(k) e^{ikX}$$

$X \to \boldsymbol{R}, k \to \boldsymbol{k}$ と3次元ベクトルにしたのが
3次元離散フーリエ変換（次ページ）

$N=L_x\times L_y\times L_z$ 3次元離散フーリエ変換

3次元結晶中のフォノンとか

feramはこれの実数⇄複素数を使用

$$\tilde{c}(\boldsymbol{k}) = \frac{1}{N}\sum_{\boldsymbol{R}} c(\boldsymbol{R}) e^{-i\boldsymbol{k}\cdot\boldsymbol{R}}$$

$$c(\boldsymbol{R}) = \sum_{\boldsymbol{k}} \tilde{c}(\boldsymbol{k}) e^{i\boldsymbol{k}\cdot\boldsymbol{R}}$$

高速フーリエ変換 (FFT) とは

- 和の中のe^{-ikx}に同じものや同じものの積が何度も出てくることを使う（？）、離散フーリエ変換を高速に行うアルゴリズム
- １次元でも2次元でも3次元でも何次元でも計算量はほぼ $5N\log_2 N$ FLOP

離散フーリエ変換 (DFT) ・高速フーリエ変換 (FFT) の演習問題

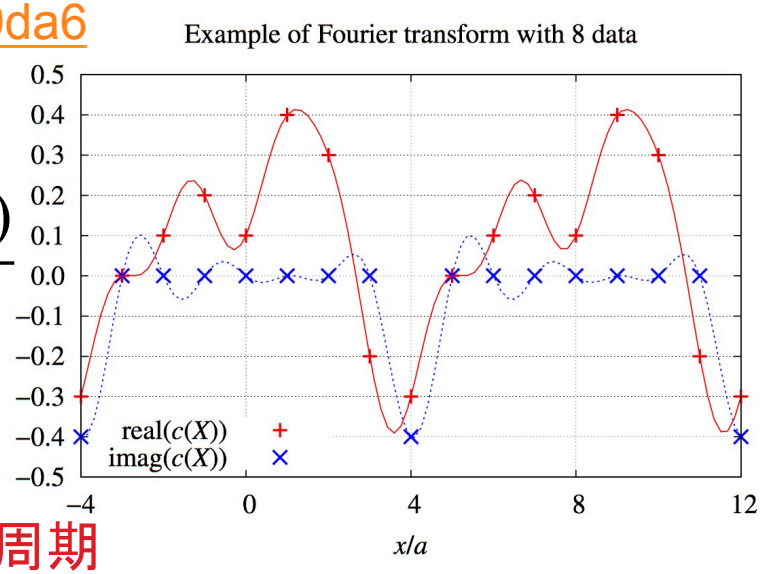
だいぶ昔に書いたN=8の演習問題をGistに置きました．kの周期性が勉強できます．
https://gist.github.com/t-nissie/831940161039db8a9da6

$$X = \quad a0, \quad a1, \quad a2, \quad \ldots, \quad a(N-1)$$

$$k = \frac{2\pi}{a}\frac{0}{N}, \ \frac{2\pi}{a}\frac{1}{N}, \ \frac{2\pi}{a}\frac{2}{N}, \ \ldots, \ \frac{2\pi}{a}\frac{(N-1)}{N}$$

$$\tilde{c}(k) = \frac{1}{N}\sum_{X} c(X) e^{-ikX}$$

$$c(X) = \sum_{k} \tilde{c}(k) e^{ikX}$$



周期

配列として確保する範囲 0〜7

X					$0a$	$1a$	$2a$	$3a$	$4a$	$5a$	$6a$	$7a$	$8a$	…
$k/(2\pi/a)$	-4/8	-3/8	-2/8	-1/8	0/8	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8	…

等価

FFTWライブラリとは

- MITのNinjaたち (Matteo Frigo and Steven G. Johnson) が開発・公開しているFFTライブラリ
- フリーソフトウェア　最新: fftw-3.3.4.tar.gz
- 高速（とんでもなく）
- 高速（**デファクトスタンダード**、MKLやcuFFTにはラッパ）
- 高速（「FFTWより高速」とかういベンチマークの9割はFFTWの使い方を間違えている）
- どんな*N*（係数の数）にも対応
- 単精度、倍精度、拡張精度にはリンクするライブラリを変えて対応する
- 初期段階にいくつかの試行計算を行って、最適なアルゴリズムを選択するのが特徴→plan（後述）

FFTWのインストール方法

- http://loto.sourceforge.net/feram/INSTALL.html
- Debian GNU/Linux系（Ubuntuなど）
  - `# apt-get install libfftw3-dev`
- RedHat系（CentOSなど）
  - `# yum install fftw-devel`
- Cygwin on Windows（gnupackなど）
  - `# apt-cyg install libfftw3-devel`
- Mac OS X
  - Homebrew: `# brew install fftw`
  - MacPorts（ompはなし？）: `# port install fftw-3`
  - Xcode付属の/usr/bin/gcc（実はApple LLVMのclang）で GCC (GNU Compiler Collection) をコンパイル・インストールして、さらにそれでFFTWをコンパイル・インストール（次ページ）

FFTWのコンパイルとインストール(倍精度)

```
$ wget http://www.fftw.org/fftw-3.3.4.tar.gz
$ tar xf fftw-3.3.4.tar.gz
$ mkdir fftw-3.3.4/build-with-gcc
$ cd $_
$ ../configure --prefix=/usr/local --libdir=/usr/local/lib64 \
 --enable-openmp --enable-threads --enable-sse2 --enable-avx \
 --enable-mpi --enable-shared
$ make -j --max-load=10.0
$ make check
$ su
# make install
# ldconfig   # 必要なら
# exit
$ /usr/local/bin/fftw-wisdom --version
fftw-wisdom tool for FFTW version 3.3.4.
```

- --prefix=, --libdir=, --max-load=の値は適宜システムに合わせる
- MPI版が不要なら--enable-mpiは不要

FFTWの基本的な使い方

- 初期化（FortranでもCの関数を直接使う）
  - use, intrinsic :: iso_c_binding; include 'fftw3.f03'; type(C_PTR) :: plan
  - ireturn = fftw_init_threads()
  - call fftw_plan_with_nthreads(OMP_GET_MAX_THREADS())
- 配列のallocateとファーストタッチ
- 準備段階でplanを作成
  - plan = fftw_plan_many_dft_r2c(..., r1, ..., c1, ..., FFTW_MEASURE)
- 本番ではplanの実行を繰り返す
  - call fftw_execute_dft_r2c(plan, r2, c2)
  - r2, c2のサイズとファーストタッチがr1, c1と同じなら同じplanが使える
- 配列のdeallocate
- 終了処理（プログラムがすぐ終わるなら、なくてもよい）
  - call fftw_destroy_plan(plan)
  - call fftw_cleanup_threads()

SoA or AoS用（後述）

フラグ（次ページ）

planの作成で複数のアルゴリズムなどが試され最速のcodeletが選ばれる。

fftw_planのフラグについて
FFTW_MEASURE or FFTW_PATIENT

- FFTW_ESTIMATEで作られるplanは一般的に遅いので初期化などだけで用いる。配列の中身を壊さないという利点はある。
- プログラム内ではFFTW_MEASUREを用いる。planを作っているときに配列の中身が壊れることに注意。
- FFTW_PATIENTを用いてwisdom（後述）を書き出せるようにしておく。
- FFTW_EXHAUSTIVEはplanを作るのにとても時間がかかる。

FFTWを高速に使う方法（1ノードOpenMP篇）

- 配列サイズやシステムによる
- よって、最適な使い方をするにはベンチマークを取る
- メモリバンド幅が律速; Xeonなどメモリチャネル数が多いCPUに速いメモリを積む
- 実行効率は10%前後
- CPUをちゃんと冷やしておかないとクロックが上がらない（下がってしまう、TBが効かない）ことがある？
- FFTWのAPIをそのまま使ってプログラミングすべき Intel MKLやCUDAにはラッパが用意されているから
- 以下のスライドで、用語の説明の後、FFTWを高速かつ高効率に使用する方法をいくつか説明

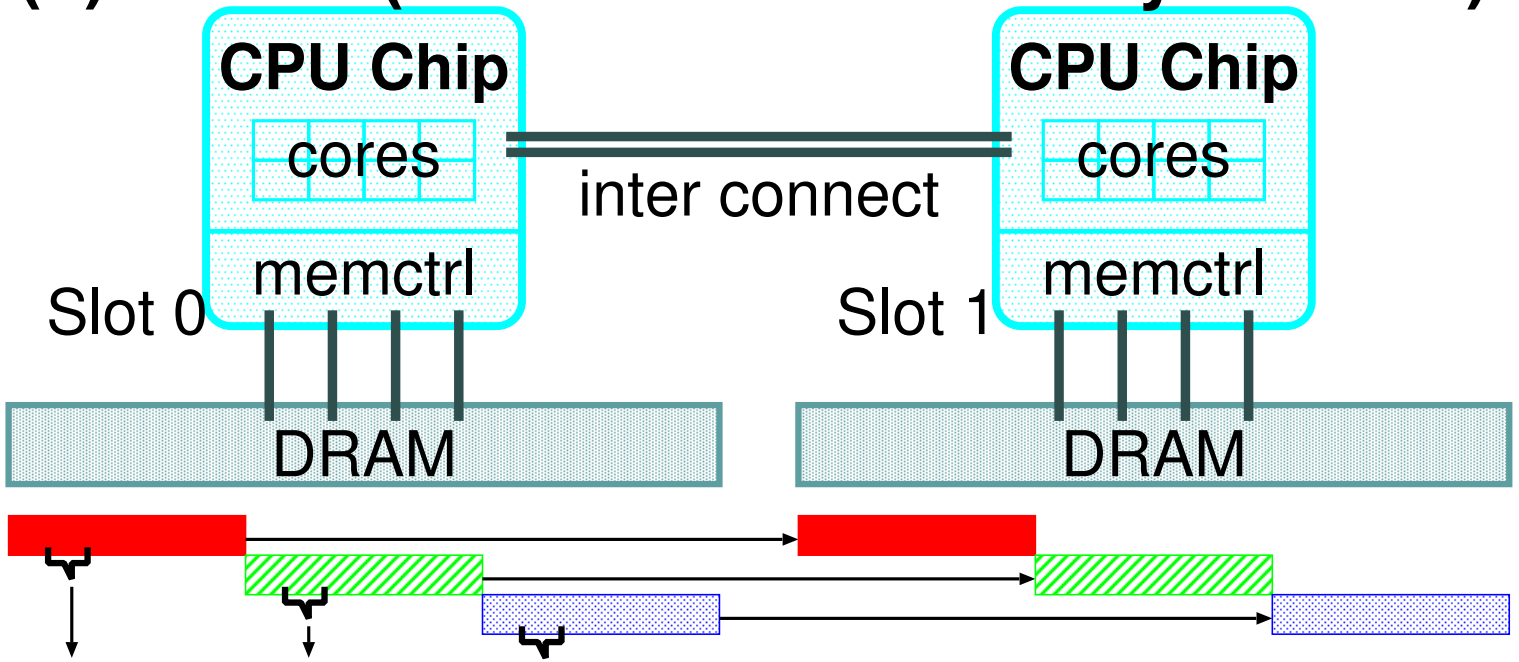
in-place and out-of-place FFT

- in-place
  - FFTの元と先の配列が同じ。
  - real⇄complexの場合、配列の確保が少し面倒。Fortranの場合、C_PTRを駆使するとrealとcomplexの2つの別の配列のようにアクセスができる。
  - メモリ使用量を削減できる。feramの0.23.02→0.24.00のin-place化などで64x64x1024なら3.13GB→2.73GB と (3+6)x33x65x1024x16≒316MB 以上削減。
- out-of-place
  - FFTの元と先の配列が異なる。
  - real⇄complexの場合、それぞれN, N/2+1と、配列の大きさが異なっていてもよい。コードはシンプル。**とりあえずこっちで書く。**
  - メモリ使用量が多くなる。
- FFTWでは**APIは同じで自動判定**してくれる
- 配列サイズやシステムによるが**FFTWなら計算速度はほぼ同じ**
- **feramではどちらも使っている**

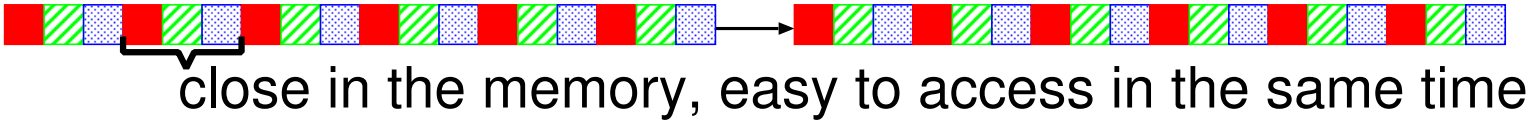
複素数⇄複素数 と 実数⇄複素数

- 複素数⇄複素数
  - c(Lx, Ly, Lz) ⇄ c(Lx, Ly, Lz)
- 複素数⇄複素数と比べると**実数⇄複素数**のFFTは計算量とメモリ量をそれぞれ約半分にできる
  - out-of-place: r(Lx, Ly, Lz) ⇄ c(Lx/2+1, Ly, Lz)
  - in-place: ary(2*(Lx/2+1), Ly, Lz)
  - FFTWではplanを作るときにc2rまたはr2cの付いた特別な関数を使う
- 以下のベンチマークはすべて**倍精度**の3次元**実数⇄複素数**のFFTの結果（feramでは複素数⇄複素数を使っていない）

(a) NUMA (Non-Uniform Memory Access)



(b) before: `SoA ary(Lx,Ly,Lz,3)`



(c) after: `AoS ary(3,Lx,Ly,Lz)`

(d) padding: `ary(3,2*(Lx/2+1),Ly+1,Lz)`

for in-place real↔complex FFT

padding to avoid bank conflicts

feramのパッケージに同梱されているFFTWのベンチマークの実行例

- $L_x \times L_y \times L_z = N$の実数⇄複素数の3次元FFT
- 計算量を$5N \log_2 N/2$としてGFLOPS値を概算
- SR16000, ¼ node = 1 chip, SMT on

```
$ export MALLOCMULTIHEAP=true
$ export XLSMPOPTS="spins=0:yields=0:parthds=16:stride=1:startproc=0"
$ ./feram_fftw_SoA    50 128 128 128 1 FFTW_PATIENT
feram_fftw_SoA.F:66: flags = FFTW_PATIENT                          GFLOPS
      50  1  128  128  128    1    2097152  in  16   0.240  45.9
      50  1  128  128  128    1    2097152 out  16   0.240  45.9

      50  3  128  128  128    1    2097152  in  16   0.770  42.9
      50  6  128  128  128    1    2097152  in  16   1.540  42.9
      50  3  128  128  128    1    2097152 out  16   0.780  42.3
      50  6  128  128  128    1    2097152 out  16   1.580  41.8
$ ./feram_fftw_wisdom 50 128 128 128 1 FFTW_PATIENT
feram_fftw_wisdom.F:60: flags = FFTW_PATIENT
      50  3  128  128  128    1    2097152  in  16   0.890  37.1
      50  6  128  128  128    1    2097152  in  16   1.830  36.1
      50  3  128  128  128    1    2097152 out  16   0.780  42.3
      50  6  128  128  128    1    2097152 out  16   1.590  41.5
```

Paddingについて

- ary(2*(Lx/2+1), Ly, Lz) → ary(2*(Lx/2+1), Ly+1, Lz)
- Lx, Ly, Lzが2の冪乗のとき、paddingを入れてバンクコンフリクトを回避すると速くなることもある。
- 配列サイズやシステムによる。
- paddingにより比較的容易な変更でまずまずの高速化が見込める。
- *ただし、コードが汚くなるので、*

+1でなく`padding_y`など変数を用いる。

Padding なし vs. あり

128x128x128のFFTのpadding_y=0 or 1のGFLOPS値を比較

SR16000 (Power7) 3.83 GHz,¼ node = 1 chip, 8 core, SMT on, 16 thread

SoA or AoS	1		SoA				AoS			
in-place or out-of-place	in	out	in	in	out	out	in	in	out	out
structure length	1	1	3	6	3	6	3	6	3	6
padding_y=0	39.0	38.3	32.1	34.5	38.9	37.4	33.1	30.9	35.6	32.6
padding_y=1	45.9	45.9	42.9	42.9	42.3	41.8	37.1	36.1	42.3	41.5

Intel Xeon X5650, max 3.1 GHz, 6 core x 2 chip, HT off, 12 thread

SoA or AoS	1		SoA				AoS			
in-place or out-of-place	in	out	in	in	out	out	in	in	out	out
structure length	1	1	3	6	3	6	3	6	3	6
padding_y=0	36.6	25.7	24.2	23.3	20.1	20.7	27.3	26.2	23.3	22.7
padding_y=1	52.7	36.5	40.7	39.5	31.5	31.3	32.9	27.8	26.1	25.3

libfftw3_omp か libfftw3_threads か

どちらでも同じAPIが使えるが、libfftw3_ompをリンクする方が速いので、リンクオプションは -lfftw3 -lfftw3_omp とする。
64x64x64, padding_y=1, FFTW_PATIENT, Xeon X5650,
6 core x 2 chip, max 3.1 GHz, HT off でGFLOPS値を比較。

SoA or AoS	1		SoA				AoS			
in-place or out-of-place	in	out	in	in	out	out	in	in	out	out
structure length	1	1	3	6	3	6	3	6	3	6
omp	42.9	41.2	43.5	43.0	41.2	33.2	38.7	40.1	41.2	33.8
threads	11.4	10.9	17.0	21.5	15.5	18.3	15.7	18.7	16.2	16.8

オリジナルFFTW vs Intel MKLのラッパ

- Intel MKLにはFFTWのラッパが用意されているのでFFTWのほとんどのAPIがそのまま使える
- wisdomファイルのI/Oやfftw_alloc_*()はない
- fftw_malloc()はある
- 現行のMKL version 11.2.3では特にSoA・AoSが遅い
- 計算速度を求めるならオリジナルのFFTWを使う
- 64x64x64, padding_y=1, FFTW_PATIENT, Xeon X5650, 6 core x 2 chip, max 3.1 GHz, HT off でGFLOPS値を比較

SoA or AoS	1		SoA				AoS			
in-place or out-of-place	in	out	in	in	out	out	in	in	out	out
structure length	1	1	3	6	3	6	3	6	3	6
FFTW3	42.9	41.2	43.5	43.0	41.2	33.2	38.7	40.1	41.2	33.8
MKL	35.1	29.5	12.8	24.3	10.2	17.2	0.9	0.8	4.9	7.7

NUMA (Non-Uniform Memory Access)

Xeon X5650 (2010), 6 core x 1 or 2 chip, max 3.1 GHz
Xeon E5-2680 v3 (2014), 12 core x 1 or 2 chip, max 3.3 GHz
96x96x96, padding_y=1, HT off, FFTW_PATIENT でGFLOPS値を比較

SoA or AoS	1		SoA				AoS			
in-place or out-of-place	in	out	in	in	out	out	in	in	out	out
structure length	1	1	3	6	3	6	3	6	3	6
X5650 1 chip	25.7	21.7	20.5	20.2	16.8	16.5	18.7	18.1	15.9	14.9
X5650 2 chip	40.1	39.4	38.1	34.2	28.9	27.8	37.9	31.2	29.7	24.7
E5-2680 v3 1	77.3	76.0	79.7	48.0	45.7	34.0	64.4	45.8	47.8	24.8
E5-2680 v3 2	107.9	112.0	117.6	117.6	114.0	63.3	100.1	117.0	104.0	66.7

128x128x128, padding_y=1, HT off, FFTW_PATIENT でGFLOPS値を比較

SoA or AoS	1		SoA				AoS			
X5650 1 chip	26.2	19.5	22.9	22.8	18.2	18.1	17.2	17.8	15.6	15.8
X5650 2 chip	52.7	36.5	40.7	39.5	31.5	31.3	32.9	27.8	26.1	25.3
E5-2680 v3 1	135.9	104.4	75.2	59.0	40.3	44.7	48.1	43.8	36.2	28.3
E5-2680 v3 2	161.9	160.7	152.9	97.0	80.4	69.6	111.8	80.4	76.7	57.1

Simultaneous Multithreading（SMT、同時マルチスレッディング、HT）はonかoffか

- 配列サイズやシステムによる
- 128x128x128の実数⇄複素数の3次元FFT
  SR16000 (Power7) 3.83 GHz
  ¼ node = 1 chip, 8 core
  SMT off, 8 thread vs. SMT on, 16 thread

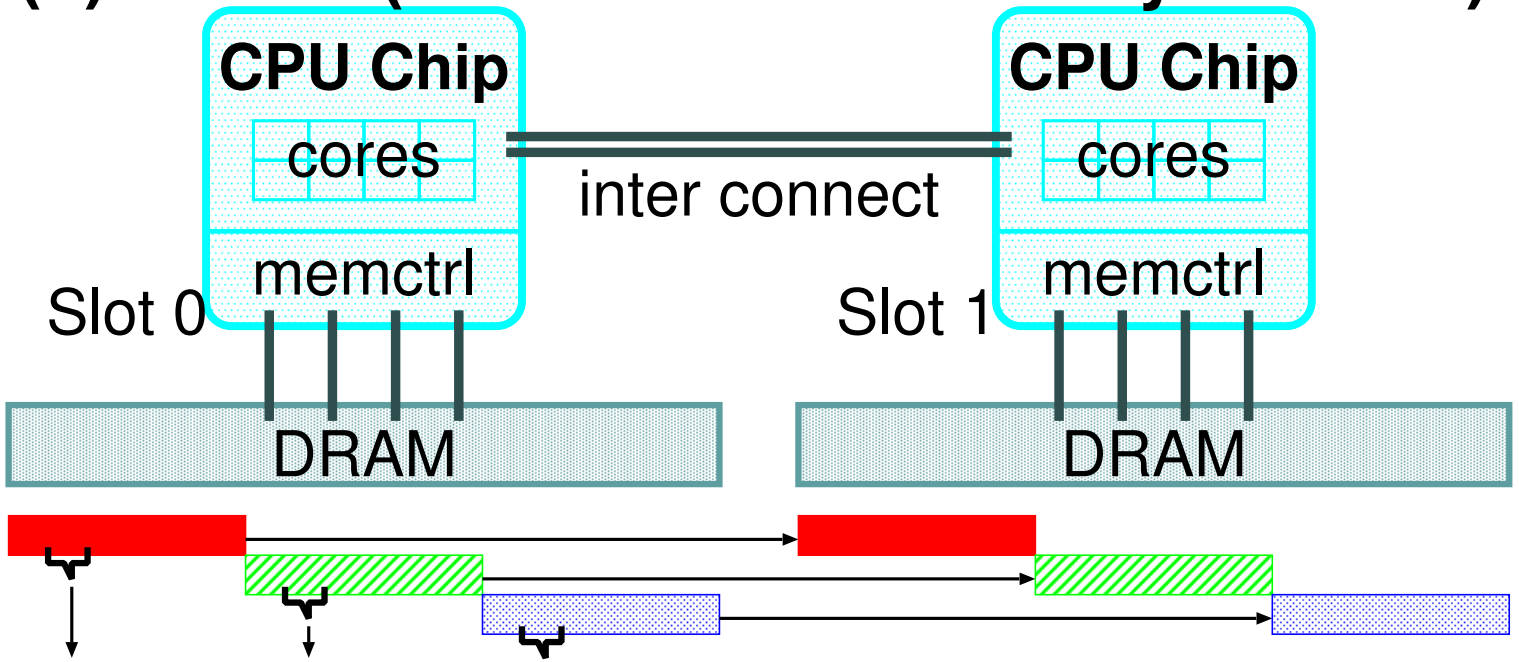
SoA or AoS	1		SoA				AoS			
in-place or out-of-place	in	out	in	in	out	out	in	in	out	out
structure length	1	1	3	6	3	6	3	6	3	6
SMT off 8 th.	46.4	46.9	43.7	42.8	42.5	45.1	34.6	36.1	42.8	42.1
SMT on 16 th.	45.9	45.9	42.9	42.9	42.3	41.8	37.1	36.1	42.3	41.5

単位はGFLOPS。あまりかわらない。

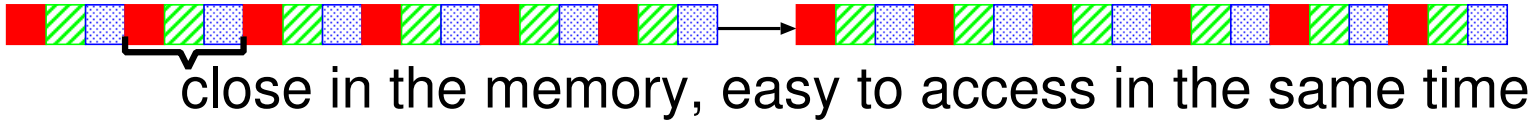
今年 Structure of Array (SoA) から Array of Structure (AoS) に変更した

- 配列サイズやシステムにもよるが、SR16000ではFFTWだけなら計算時間はたいして変わらない
- 他の部分でAoSにするとSR16000では高速になる→SR16000に最適化しすぎた
- Intel系ではSoAにして1配列づつ実行するのが最速かもしれない（旧バージョンではそうしていた）が、コードが汚くなるし、将来のFFTWのバージョンアップで1配列のFFTとSoAがほぼ同じ速さになる可能性はある
- Intel系でわりとAoSは遅かった…
- GPGPU（NDIVIA K20X, CUDA 5.5, cuFFT, cuBLAS）でもAoSは遅かった…

(a) NUMA (Non-Uniform Memory Access)



(b) before: `SoA ary(Lx,Ly,Lz,3)`



(c) after: `AoS ary(3,Lx,Ly,Lz)`

(d) padding: `ary(3,2*(Lx/2+1),Ly+1,Lz)`

for in-place real↔complex FFT

padding to avoid bank conflicts

0.24.00で一部の配列をin-place化。

新旧バージョンを比較

SMT, HTはoff

input file: 64x64x1024.feram, 8,200 MD steps, padding_y=1

feram version	¼ node of SR16000# (Power7) 8 core 3.83 GHz	1 node of SR16000# (Power7) 32 core 3.83 GHz	1 chip of Xeon X5650 6 core 2.66 GHz	2 chip of Xeon X5650 12 core 2.66 GHz	1 chip of Xeon E5-2680 v3 12 core 2.87* GHz	2 chip of Xeon E5-2680 v3 24 core 2.87@ GHz
TDP	250 W	1000 W	95 W	190 W	120 W	240 W
0.22.06	2818 sec	1023 sec	6361 sec	3716 sec	2093 sec	1686 sec
0.23.01	1980 sec	601 sec				
0.23.02	1866 sec	557 sec			2072 sec	1731 sec
0.24.00	1819 sec	553 sec	5663 sec	3179 sec	2050 sec	1650 sec

並列化率が改善

高速化

2011年のSandy Bridge以降並列化率がよくない？

SR16000ではスレッドをコアにバインドするなど工夫が必要

* turbostatコマンドでTurbo Boostの周波数を実測

@ 24 coreで実行中はヒマにしているcoreもある

FFTW: wisdomの利用

- wisdomファイルを通して、FFTWがplanを作るときに行ったベンチマークの結果の書き出し／読み込みができる
- S式で書かれている
- feram_fftw_wisdom: feram の中で使うFFTのベンチマークをして、wisdom_newを出力する
- FFTW_MEASURE と FFTW_PATIENT で feram_fftw_wisdom を数回づつ実行して最速のwisdomを使う

GFLOPS

```
FFTW_MEASURE
 100  3   48   48  960    1    2211840  in    8   3.130  22.341
 100  6   48   48  960    1    2211840  in    8   5.230  26.741
 100  3   48   48  960    1    2211840 out    8   2.490  28.083
 100  6   48   48  960    1    2211840 out    8   4.600  30.403

FFTW_PATIENT
 100  3   48   48  960    1    2211840  in    8   2.900  24.113
 100  6   48   48  960    1    2211840  in    8   4.800  29.137
 100  3   48   48  960    1    2211840 out    8   2.400  29.137
 100  6   48   48  960    1    2211840 out    8   4.060  34.447
```

- 48x48x960、SR16000の1チップ＝¼ノード、SMT on、MD5万ステップ　9393秒→9091秒　約3%高速化
- wisdomを読み込むと使われるcodeletが固定される

FFTWのMPI並列化機能

- flat MPIも，OpenMP+MPIのハイブリッド並列も可能
  - flat MPIの実行効率は5%前後
  - ハイブリッド並列は1～2%前後
- 1D (or slab) decomposition にのみ対応
- Fortranの配列c(*Lx*, *Ly*, *Lz*)ならMPI並列は*Lz*まで
- 最後の順序変換（final transpose、多量の通信を伴う）をoffにもできるが、プログラミングが大変
- プログラムとベンチマークの例: https://github.com/t-nissie/fft_check_mpi

後半のまとめ

- **feram**のFFTにはFFTWライブラリを使っている
- FFTWはplanを用いるなど特徴がある
- FFTWをより高速に使うためにはいくつか調節すべきパラメータがある→プログラムのFFT以外の部分の寄与も考慮しながらそれらのパラメータを調節する
  - padding
  - リンクすべきライブラリ
  - NUMA
  - SMT (HT)
  - SoA or AoS
  - wisdomファイル
- さらにFFTWを使い倒したい→guruやgenfftと検索

補足資料

Xeon X5650 vs Core i7 3770K

FFTはメモリバンド幅が律速; Xeonなどメモリチャンネル数が多いCPUに速い（クロック周波数の高い）メモリを積むべき

Xeon X5650 (2010), 6 core, max 3.06 GHz, HT off, 1 chip, チャンネル数 3
Core i7 3770K (2012), 4 core, max 3.9 GHz, HT on, 1 chip, チャンネル数 2
96x96x96, padding_y=1, FFTW_PATIENT でGFLOPS値を比較

SoA or AoS	1		SoA				AoS			
in-place or out-of-place	in	out	in	in	out	out	in	in	out	out
structure length	1	1	3	6	3	6	3	6	3	6
Xeon 6 core	25.7	21.7	20.5	20.2	16.8	16.5	18.7	18.1	15.9	14.9
Xeon 4 core	18.3	16.1	15.0	14.8	13.3	13.4	13.5	12.8	11.9	12.0
Core i7	16.4	14.7	13.5	13.4	12.9	12.8	6.2	6.2	13.1	13.1

SoAからAoSに変更したら GPU（CUDA#, cuFFT, cuBLAS, Fortranだけで）だとFFTが2倍遅くなってしまった with K20X*

- gfortran -fopenmp -ffree-form -c cufft_module.f
  （feramに同梱予定、cuFFTライブラリのAPIへのinterfaceが書いてある）
- gfortran -fopenmp -ffree-form -c cufft_check.F
- gfortran -fopenmp -o cufft_check cufft_check.o cufft_module.o -L/usr/local/cuda/lib64 -lcublas -lcufft -lcudart
- ./cufft_check 100 64 64 1024
  100 64 64 1024 4194304 0.551 秒 83.7 GFLOPS
  100 64 64 1024 4194304 3.056 秒 45.3 GFLOPS ←遅い
- ./cufft_check 100 128 128 1024
  100 128 128 1024 16777216 2.114 秒 95.2 GFLOPS
  100 128 128 1024 16777216 11.891 秒 50.8 GFLOPS ←遅い

CUDA 5.5を使用。CUDA 7でも速くはなっていない。

* K20XのTDPは235 W。

crd2txt.exeのその後

- crd2txt.exeはWindows 3.1に付属していたカードファイルアプリcardfile.exeのファイルフォーマットをテキストファイルに変換するプログラム
- 1995年に公開
- 現在はEvernote export format (.enex) に変換するプログラムを公開している
  https://gist.github.com/t-nissie/9771048