計算科学技術 特論B

第2回
アプリケーション性能最適化1
（高並列性能最適化）

2026年4月

株式会社 waveZ
テクノロジーディレクター
南 一生

minami@wave-z.com

講義全体の概要

waveZ

- スーパーコンピュータとアプリケーションの性能
- アプリケーションの性能最適化１（高並列性能最適化）
- スレッド並列・スレッド性能最適化概要
- スレッド並列・スレッド性能最適化詳細（1）
- スレッド並列・スレッド性能最適化詳細（2）

今回の講義内容

waveZ

- 反復法(CPU/GPU)
- アプリケーション並列度の拡張(CPU/GPU)
- 高並列化手法の手順

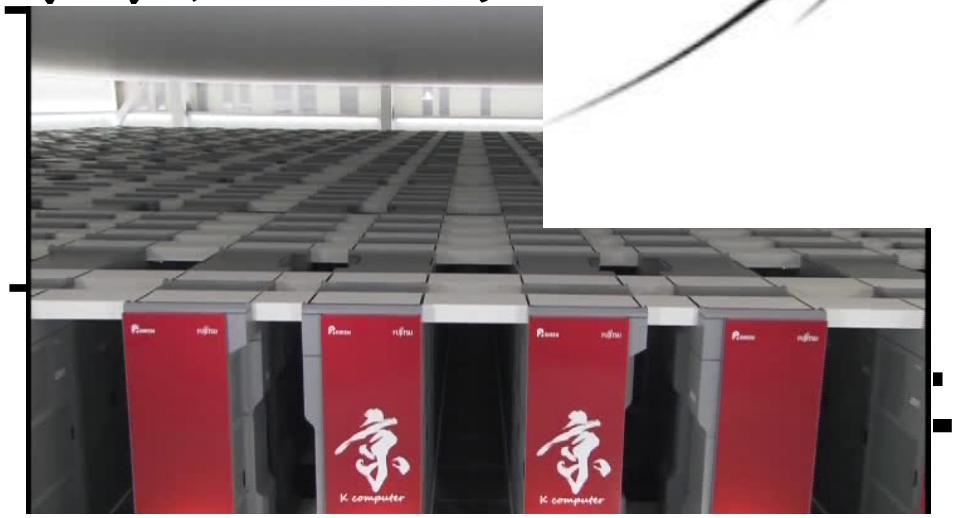
スーパーコンピュータを使うため

waveZ

(1) 定式化 (

(2) 離散化 (

(3) アルゴリ

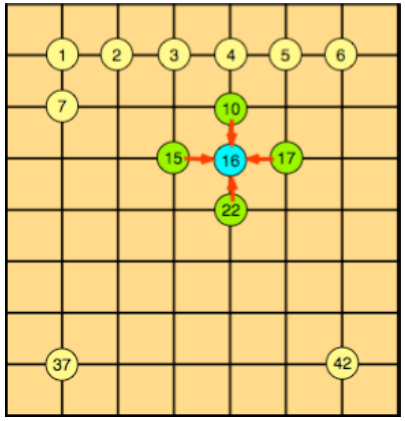


ング

- ✦前回の講義でCPU/GPU両方について「並列処理」を利用しないと性能向上しないことを話しました
- ✦「並列処理」できるアルゴリズムを採用することが重要とお伝えしました
- ✦最初の話題は「並列処理」できるアルゴリズムの話題です

(CPU/GPU共通)

2次元偏微分方程式-離散化



- 左図のような2次元の領域で微分方程式を解くとする.
- 5点差分で差分化すると自分自身の差分点と周りの4つの差分点の式ができる.

$$C_{1,3}u_1 + C_{1,4}u_2 + C_{1,5}u_7 = \alpha_1 \quad \text{①の差分点について}$$

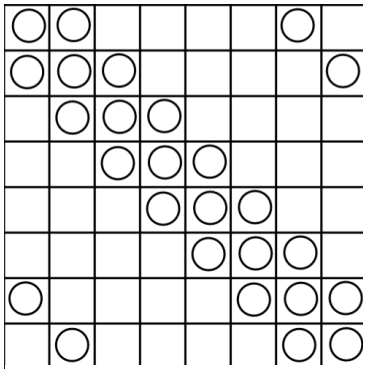
$$C_{2,2}u_1 + C_{2,3}u_2 + C_{2,4}u_3 + C_{2,5}u_8 = \alpha_2 \quad \text{②の差分点について}$$

. . . . . .

$$C_{16,1}u_{10} + C_{16,2}u_{15} + C_{16,3}u_{16} + C_{16,4}u_{17} + C_{16,5}u_{22} = \alpha_{16} \quad \text{⑯の差分点について}$$

. . . . . .

行列の形（疎行列）



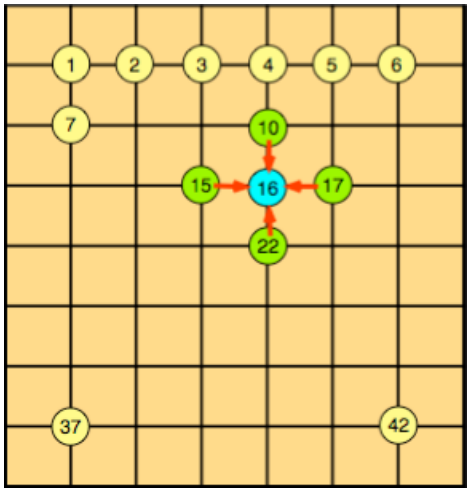
2次元偏微分方程式−離散化

$$-(p(x,y)u_x)_x-(p(x,y)u_y)_y+\sigma(x,y)u(x,y)=f(x) \quad (x,y)\in R$$

離散化
(+−×÷の世界)

$$C_{ij}\begin{cases} D_{ij}u_{ij}-L_{ij}u_{i-1,j}-R_{ij}u_{i+1,j}-T_{ij}u_{i,j+1}-B_{ij}u_{i,j-1} \\ \qquad =S_{ij} \\ L_{ij}=p_{i-1/2,j}\left(\dfrac{k_{j-1}+k_j}{2h_{i-1}}\right) \quad T_{ij}=p_{i,j+1/2}\left(\dfrac{h_{i-1}+h_i}{2k_j}\right) \\ R_{ij}=p_{i+1/2,j}\left(\dfrac{k_{j-1}+k_j}{2h_i}\right) \quad B_{ij}=p_{i,j-1/2}\left(\dfrac{h_{i-1}+h_i}{2k_{j-1}}\right) \\ D_{ij}=L_{ij}+R_{ij}+B_{ij}+T_{ij}+\sigma_{ij}\left(\dfrac{h_{i-1}+h_i}{2}\right)\left(\dfrac{k_{j-1}+k_j}{2}\right) \\ S_{ij}=f_{ij}\left(\dfrac{h_{i-1}+h_i}{2}\right)\left(\dfrac{k_{j-1}+k_j}{2}\right) \end{cases}$$

離散化−離散化と連立一次方程式



- 左図のような2次元の領域で微分方程式を解くとする.
- 5点差分で差分化すると自分自身の差分点と周りの4つの差分点の式ができる.

$$C_{1,3}u_1 + C_{1,4}u_2 + C_{1,5}u_7 = \alpha_1 \quad \text{①の差分点について}$$

$$C_{2,2}u_1 + C_{2,3}u_2 + C_{2,4}u_3 + C_{2,5}u_8 = \alpha_2 \quad \text{②の差分点について}$$

. . . . . .

$$C_{16,1}u_{10} + C_{16,2}u_{15} + C_{16,3}u_{16} + C_{16,4}u_{17} + C_{16,5}u_{22} = \alpha_{16} \quad \text{⑯の差分点について}$$

. . . . . .

- 全部で42点の差分点が存在するので42元の連立方程式となる.
- したがって元の微分方程式は，離散化により以下のような連立一次方程式を解く事に帰着させる事ができる.

$$\mathbf{Ax} = \mathbf{b}$$

反復法

反復法の一般的な議論

- 左のような連立一次方程式を考える.

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \cdots ①$$

- Aを右のように書き直す. I は単位行列.

$$\mathbf{A} = \mathbf{I} - \mathbf{B} \quad \cdots ②$$

- ②を①に代入し整理する.

$$(\mathbf{I} - \mathbf{B})\mathbf{x} = \mathbf{b}$$

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{b} \quad \cdots ③$$

- ③を満たすベクトルの列: $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3 \cdots$ を考える.

$$\mathbf{x}^{(m+1)} = \mathbf{B}\mathbf{x}^{(m)} + \mathbf{b} \quad \cdots ④$$

- ここで④の両辺の極限を取れば⑤のようになる

$$\lim_{m\to\infty} \mathbf{x}^{(m+1)} = \mathbf{B} \lim_{m\to\infty} \mathbf{x}^{(m)} + \mathbf{b} \quad \cdots ⑤$$

- 極限ベクトルをxとおけば⑤は⑥の様になる

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{b} \quad \cdots ⑥$$

- ⑥は③と全く同一となる. つまり連立一次方程式:①を解く事は④を満たすベクトルの列: $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3 \cdots$ を見つける事に帰着する.

各反復法

- ④を解くための反復法について説明する　$\mathbf{x}^{(m+1)} = \mathbf{B}\mathbf{x}^{(m)} + \mathbf{b}$　・・・④
- Aは正則，Aの対角要素は全て0でない数であると仮定する．
- 解くべき方程式　$\mathbf{A}\mathbf{x} = \mathbf{k}$　・・・⑦
- Aを⑧のように変形．DはAの対角成分のみの行列．EはAの狭義下三角行列の符号を変えた行列．FはAの狭義上三角行列の符号を変えた行列．　$\mathbf{A} = \mathbf{D} - \mathbf{E} - \mathbf{F}$　・・・⑧
- ⑧を⑦に代入し⑨を得る　$\mathbf{D}\mathbf{x} = (\mathbf{E} + \mathbf{F})\mathbf{x} + \mathbf{k}$　・・・⑨
- ⑨を④の反復法にすると⑩を得る．　$\mathbf{x}^{(m+1)} = \mathbf{D}^{-1}(\mathbf{E} + \mathbf{F})\mathbf{x}^{(m)} + \mathbf{D}^{-1}k$　・・・⑩
- この反復法を**点ヤコビ法**という．
- 具体的には⑪の計算法となる．
- 注目すべきは (m+1) 世代の解を得るために (m) 世代の解しか使用しない点である．

$$\left.\begin{aligned} a_{ii}x_i^{(m+1)} &= -\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^{(m)} + k_j \\ x_i^{(m+1)} &= -\sum_{\substack{j=1 \\ j\neq i}}^{n} \left(\frac{a_{ij}}{a_{ii}}\right) x_j^{(m)} + \frac{k_i}{a_{ii}} \end{aligned}\right\} \quad \cdots ⑪$$

各反復法

- ⑨を⑫のように変形する.

$$(\mathbf{D}-\mathbf{E})\mathbf{x}=\mathbf{F}\mathbf{x}+\mathbf{k} \quad \cdots ⑫$$

$$(\mathbf{D}-\mathbf{E})\mathbf{x}^{(m+1)}=\mathbf{F}\mathbf{x}^{(m)}+\mathbf{k} \quad \cdots ⑬$$

- ⑫を④の反復法にすると⑬を得る.
- この反復法を**点ガウス・ザイデル法**という.

- 具体的には⑭の計算法となる.

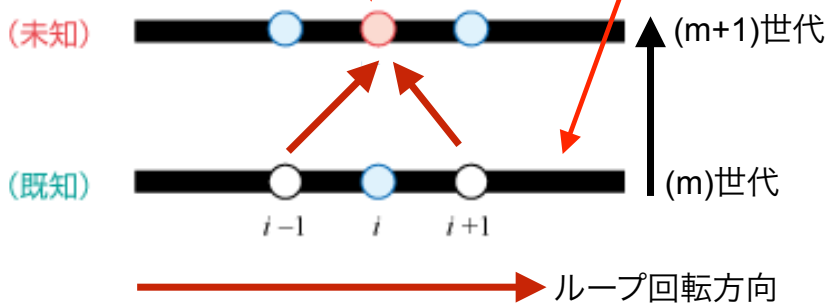
$$a_{ii}x_i^{(m+1)}=-\sum_{j=1}^{i-1}a_{ij}x_j^{(m+1)}-\sum_{j=1+1}^{n}a_{ij}x_j^{(m)}+k_j \quad \cdots ⑭$$

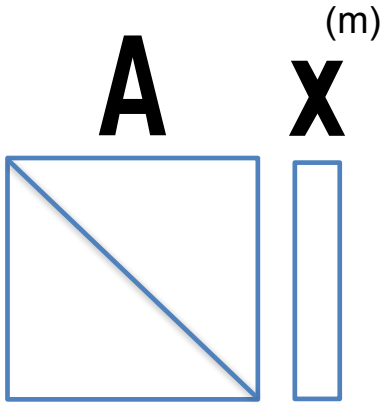
点ヤコビ法の参照関係

$$x_i^{(m+1)} = -\sum_{\substack{j=1 \\ j\neq i}}^{n} \left(\frac{a_{ij}}{a_{ii}} \right) x_j^{(m)} + \frac{k_i}{a_{ii}}$$

同じ(m+1)世代の別

参照しない.



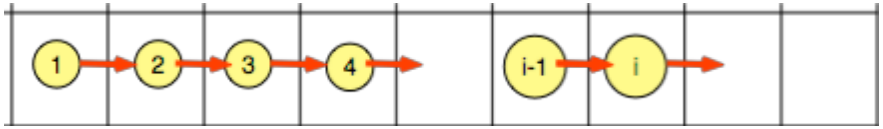


- (例) 右図のように, iを定義するのにi−1の値を参照するコーディングは下図のような参照関係となり,リカレンス (回帰参照) が発生していない！

do i=1,n

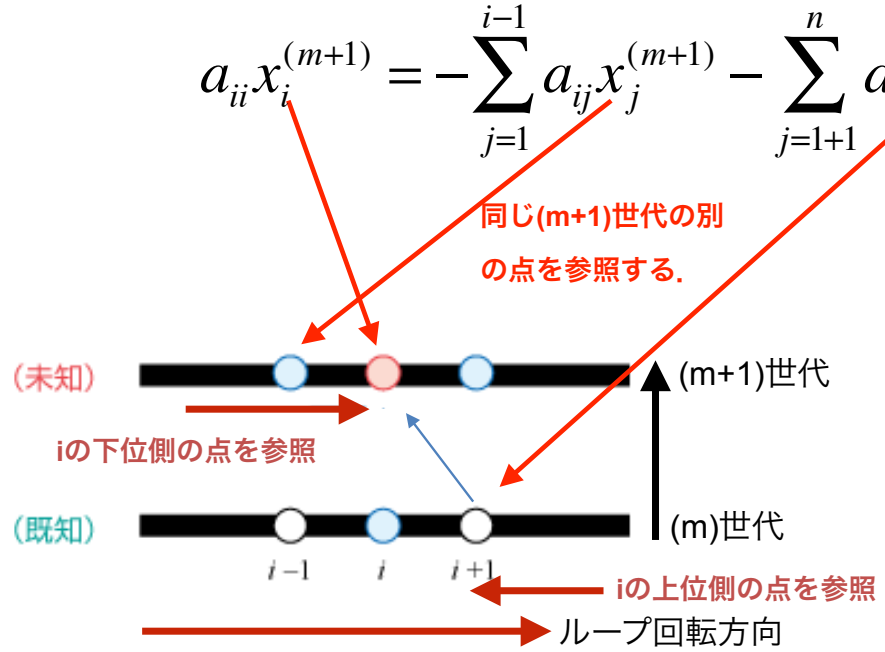
$$x(i) = -a(i) * x_{old}(i) + b(i)$$

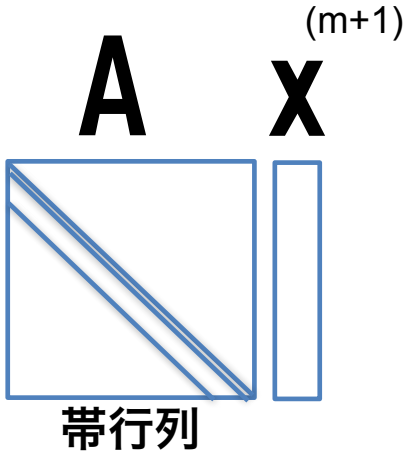
- リカレンスがあると, iについては依存関係があり並列処理ができない.



点ガウス・ザイデル法の参照関係

$$a_{ii}x_i^{(m+1)} = -\sum_{j=1}^{i-1} a_{ij}x_j^{(m+1)} - \sum_{j=1+1}^{n} a_{ij}x_j^{(m)} + k_i$$



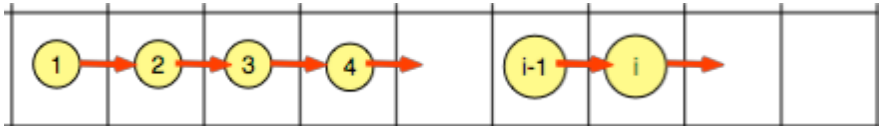


- (例) 右図のように, iを定義するのにi−1の値を参照するコーディングは下図のような参照関係となり,リカレンス(回帰参照)が発生している.

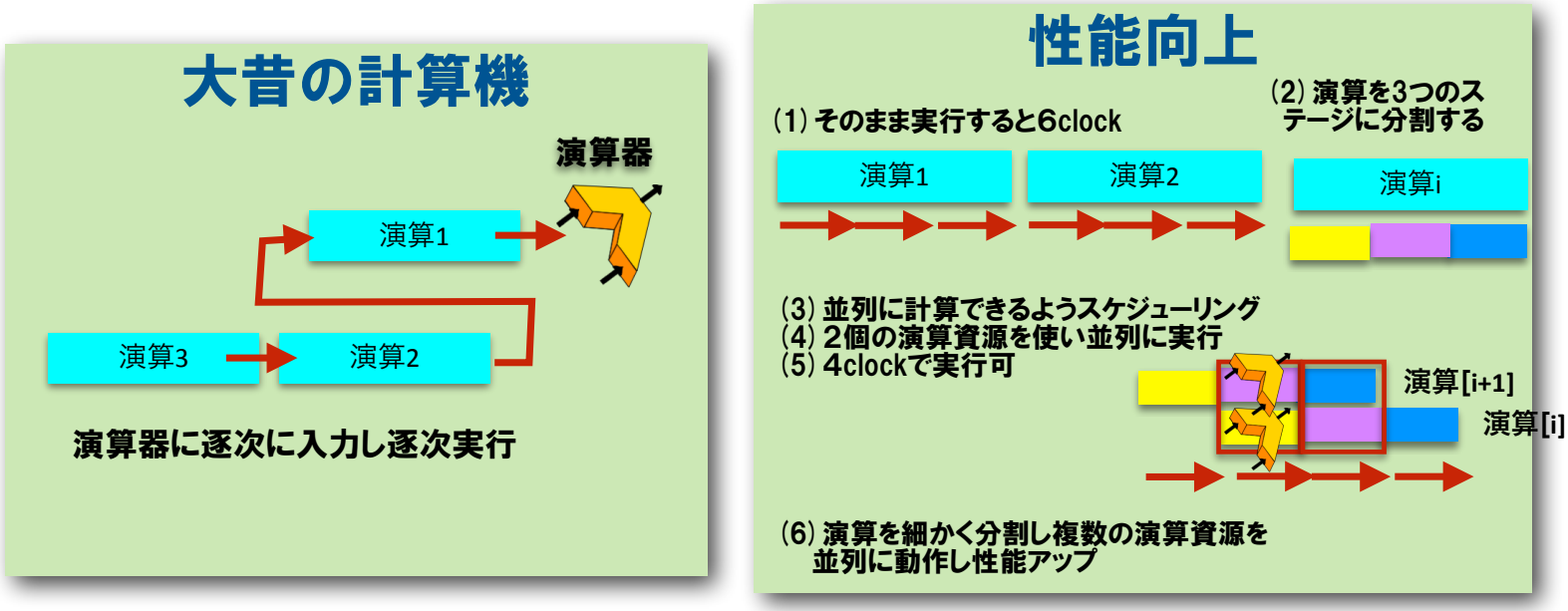
do j=1,i-1

$$a(ii) * x(i) = -a(ij) * x(j) - x_{old} + k(i)$$

- リカレンスがあると, iについては依存関係があり並列処理ができない.

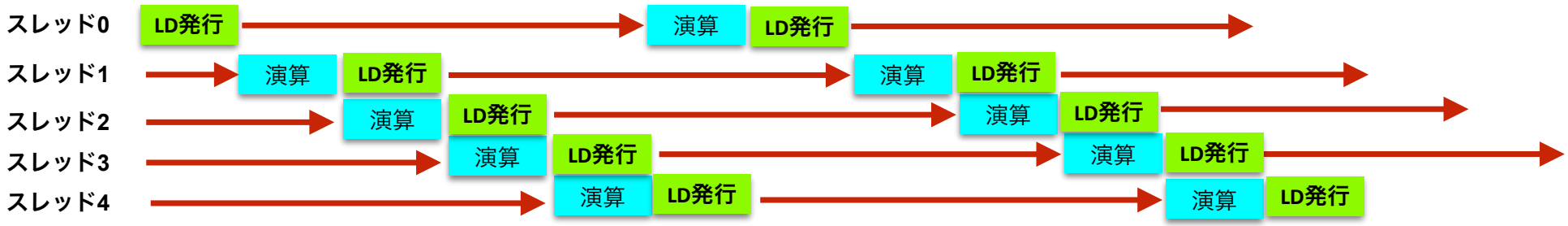


リカレンスがあると．．．（CPUのSIMD）



リカレンスにより並列スケジューリングができない
→処理のリカレンスをなくすチューニング
→演算器を多く使うことが可能となる
→現代のコンピュータを使う上で必須事項!

リカレンスがあると．．．（GPUのSIMT）

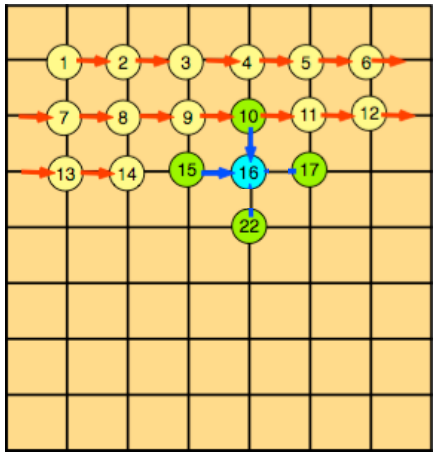


リカレンスがあるとスレッドを並列で計算することができない
→処理のリカレンスをなくすチューニング
→スレッドの並列処理が可能となる
→現代のコンピュータを使う上で必須事項!

反復法のリカレンスの解消

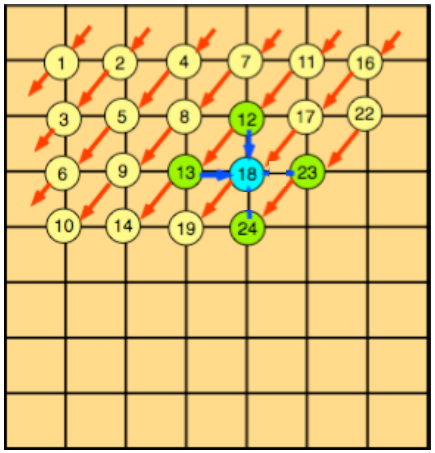
- ヤコビ法やガウス・ザイデル法はメッシュの**計算順序を変更しても収束する**事が数学的に証明されている.
- ガウス・ザイデル法オリジナルの計算順序は前述のようにリカレンスのために並列化できない.
- そこで(b)のハイパープレーン法や(c)のRED-BLACK法を用いループ内のリカレンスによる依存関係をなくす。
- 依存関係をなくす事で**前回講義のようにハードウェアの並列計算機構を使い高速に計算可能**となる. またコア間の並列化そのものにも使用可能.

(a)オリジナルスイープ



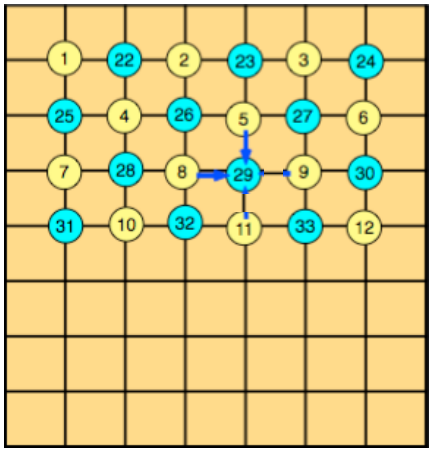
参照関係
計算順序(スイープ順序)

(b)ハイパープレーンスイープ



- 斜めのハイパープレーン内はリカレンスがない.
- ハイパープレーン間のみにリカレンスが発生.

(c)RED-BLACKスイープ



- 黄と青の2つのループに再構成する.
- それぞれのループ内はリカレンスがない.

反復法に関する参考書

「計算機による大型行列の反復解法」

R. S. バーガ／著　渋谷政昭／〔ほか〕訳

共役勾配法（CG法）

- ①の2次の関数を考える.
- この関数の最小値問題を解くためにf（x）の微分を取り0とおく.
- ②より①の最小値問題を解く事がax=bの解を得る事と同等であることが分かる.

$$f(x)=\frac{1}{2}ax^2-bx \qquad \cdots ①$$

$$f'(x)=ax-b=0 \qquad \cdots ②$$

- 上記と同様な事を多次元のベクトルで行う.
- ①と同等な関数として③を考える.
- ③を成分で標記すると④のようになる.（Aは実対称行列であるとする）

$$f(\mathbf{x})=\frac{1}{2}(\mathbf{x},\mathbf{Ax})-(\mathbf{b},\mathbf{x}) \qquad \cdots ③$$

$$f(\mathbf{x})=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ij}x_ix_j-\sum_{i=1}^{n}b_ix_i \qquad \cdots ④$$

- ④をxで偏微分すると⑤になる.
- 行列Aの対称性を使い0とおくことにより⑥が得られる.
- ⑥は連立一次方程式⑦の成分表示である.

$$\frac{\partial f(\mathbf{x})}{\partial x_k}=\frac{1}{2}\sum_{i=1}^{n}a_{ik}x_i+\frac{1}{2}\sum_{j=1}^{n}a_{kj}x_j-b_k \qquad \cdots ⑤$$

$$\frac{\partial f(\mathbf{x})}{\partial x_k}=\sum_{i=1}^{n}a_{ki}x_i-b_k=0 \qquad \cdots ⑥$$

- したがって③の最小値問題を解く事が連立一次方程式⑦を解くこととなる.

$$\mathbf{Ax}=\mathbf{b} \qquad \cdots ⑦$$

共役勾配法（CG法）

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x}, \mathbf{Ax}) - (\mathbf{b}, \mathbf{x}) \quad \cdots ③$$

③を求めるためのアルゴリズムについては
例えば

＜シリーズ新しい応用の数学 17＞
「共役勾配法」戸川隼人著

共役勾配法（CG法）

$$r_0 = b - A x_0$$

$$p_0 = r_0$$

for $i = 0, 1, 2, \cdots$

$$\alpha_i = \frac{(r_i, r_i)}{(p_i, A p_i)}$$

$$x_{i+1} = x_i + \alpha_i p_i$$

$$r_{i+1} = r_i - \alpha_i A p_i$$

$$\beta_i = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)}$$

$$p_{i+1} = r_{i+1} + \beta_i p_i$$

- CG法アルゴリズムの基本形を左図に示す．
- この他に色々なアルゴリズムの派生形がある．
- ③式のf（x）が最小値が得られるようにxiの列を求めて行く．
- ri は残差ベクトルであり互いに直交し一次独立であることが証明されている．
- したがって残差ベクトルはN元の連立一次方程式にはN個しか存在しない．
- CG法を使うとN元の連立一次方程式は高々N回の反復で収束する．
- またAの固有値が縮重しているか密集していれば収束が速くなる性質を持つ．

前処理付き共役勾配法（CG法）

- Aに近い正値対称行列:Mを考えコレスキー分解する（①式）.
- ここで②のようなxの置き換えを行う.
- これを使ってもとのAx=bを変形する（③④式）
- （Ax=bに②を代入すれば③となる．③にU^{-T}を左辺からかければ④になる）
- ここで⑤で置き換えたAの性質をみてみる.
- MはAに近いので⑥を満たす.
- したがって①より⑦を満たす.
- ⑤に⑦を適用すると⑧をみたし置き換えたAの性質は単位行列に近いことが分かる.
- したがって⑨によって置き換えた行列Aの固有値は1の周りに密集しているものと考えられる
- 置き換えた行列とベクトルに対する連立一次方程式：⑩の収束は早いものと期待できる.

$$\mathbf{M} = \mathbf{U}^T\mathbf{U} \quad \cdots ①$$

$$\tilde{\mathbf{x}} = \mathbf{U}\mathbf{x} \quad \cdots ②$$

$$\mathbf{A}\mathbf{U}^{-1}\tilde{\mathbf{x}} = \mathbf{b} \quad \cdots ③$$

$$\mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1}\tilde{\mathbf{x}} = \mathbf{U}^{-T}\mathbf{b} = \tilde{\mathbf{b}} \quad \cdots ④$$

$$\mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1} = \tilde{\mathbf{A}} \quad \cdots ⑤$$

$$\mathbf{A} \approx \mathbf{M} \quad \cdots ⑥$$

$$\mathbf{A} \approx \mathbf{U}^T\mathbf{U} \quad \cdots ⑦$$

$$\mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1} \approx \mathbf{U}^{-T}(\mathbf{U}^T\mathbf{U})\mathbf{U}^{-1} = \mathbf{I} \quad \cdots ⑧$$

$$\tilde{\mathbf{A}} = \mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1} \quad \cdots ⑨$$

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \quad \cdots ⑩$$

前処理付き共役勾配法（CG法）

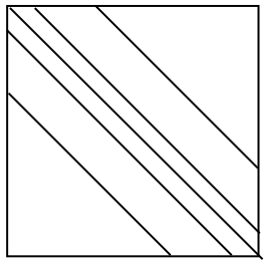
- Aは例えば先例に示した5点差分で得られる疎行列であるとする.
- ここでAのコレスキー分解を考える.
- Aを完全にコレスキー分解すると方程式を解くことになる.
- Aの完全なコレスキー分解を行った行列は密行列となる.
- Aの完全なコレスキー分解のための演算量は大きい.
- そこでAを完全でなく不完全なコレスキー分解を行うこととする.

$$A \approx U^T U$$

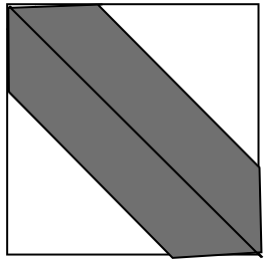
- 不完全コレスキー分解とは元の行列:Aが持つ要素のみ出現するようにコレスキー分解を行う.
- このような不完全コレスキー分解を実施することを前処理と呼ぶ.
- 前処理した行列について共役勾配法を使用して連立一次方程式を解く方法を前処理付き共役勾配法という.

$$\tilde{A} = U^{-T} A U^{-1} \qquad \tilde{A}\tilde{x} = \tilde{b}$$

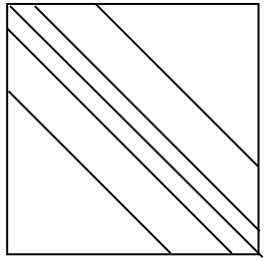
行列:A



Aの完全コレスキー分解



Aの不完全コレスキー分解



前処理付き共役勾配法（CG法）

- 不完全コレスキー分解による前処理CG法アルゴリズムの基本形を下図に示す．
- 不完全コレスキー分解は元の行列要素の位置のみを計算する．
- 0要素に対するフィルインは計算しない．
- 計算は行列ベクトル積・ベクトルスカラ積・ベクトルの和・内積・除算で構成される．
- 赤線で示した前処理部分は**前進代入・後退代入**で計算される．

ステップ1: $\alpha^k = (r_i^k \bullet (LL^T)^{-1} r_i^k) / (Ap_i^k \bullet p_i^k)$

ステップ2: $x_i^{k+1} = x_i^k + \alpha^k p_i^k$

ステップ3: $r_i^{k+1} = r_i^k - \alpha^k Ap_i^k$

ステップ4: $\beta^k = (r_i^{k+1} \bullet (LL^T)^{-1} r_i^{k+1}) / (r_i^k \bullet (LL^T)^{-1} r_i^k)$

ステップ5: $p_i^{k+1} = (LL^T)^{-1} r_i^{k+1} + \beta^k p_i^k$

前進代入・後退代入処理

$$(LL^T)^{-1}r = x$$
$$(L^T)^{-1}L^{-1}r = x$$

$y = L^{-1}r$ とする

$Ly = r$ と変形する

L は下三角行列であるので
以下のように書ける.

$$L_{11}y_1 = r_1$$
$$L_{21}y_1 + L_{22}y_2 = r_2$$
$$L_{31}y_1 + L_{32}y_2 + L_{33}y_3 = r_3$$
$$\cdots \quad \cdots$$

この式は右のように逐次に
解くことができる.

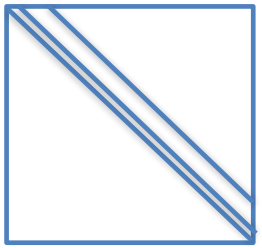
(1)これを前進代入処理という

$$y_1 = \frac{r_1}{L_{11}}$$
$$y_2 = \frac{1}{L_{22}}(r_2 - L_{21}y_1)$$
$$y_3 = \frac{1}{L_{33}}(r_3 - L_{31}y_1 - L_{32}y_2)$$
$$\cdots \quad \cdots$$

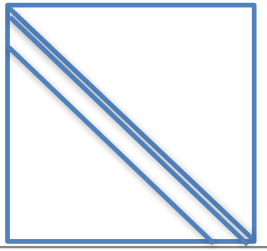
$$(L^T)^{-1}y = x$$

L^T は上三角行列である. Lと同様に
書き下す. 前進代入処理と逆方向に
逐次に計算することによりxを求める
ことができる.

(1)これを後退代入処理という

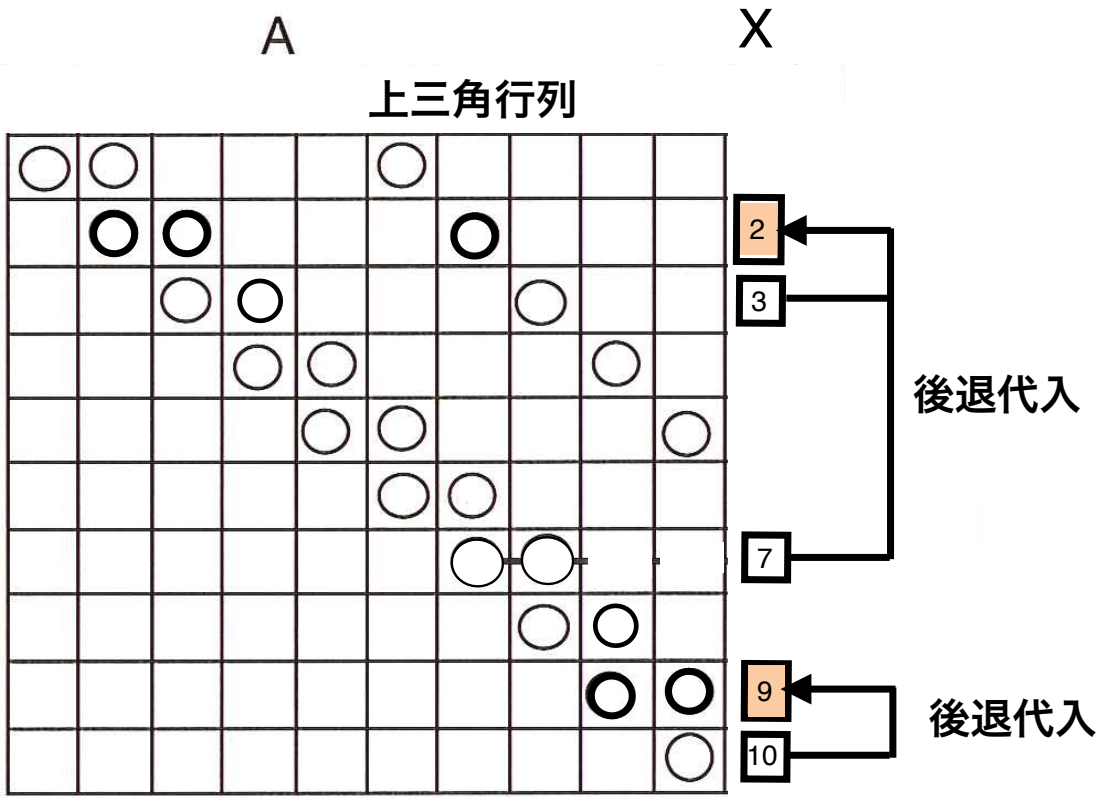


L^T



L

後退代入の例

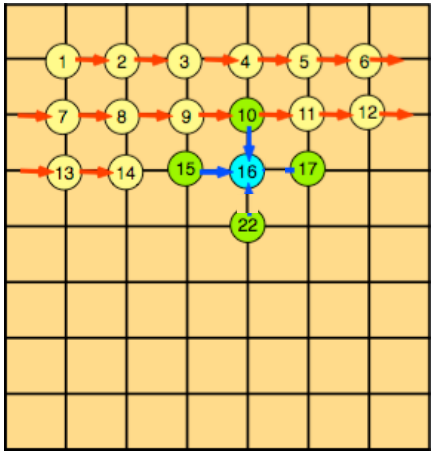


不完全コレスキー分解の前処理は本質的に
リカレンスが発生する！

不完全コレスキー分解のリカレンスの解消

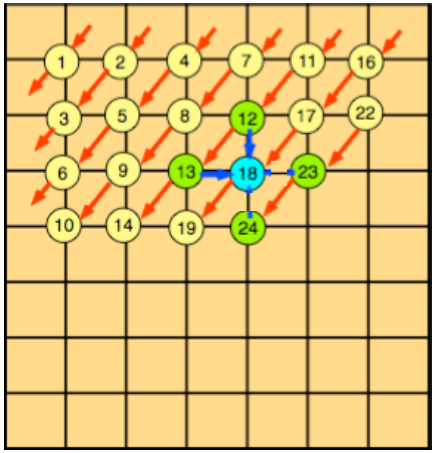
- 不完全コレスキー分解の計算順序は前頁のようにリカレンスのために並列化できない.
- またガウス・ザイデル法と同様な参照関係を持つ(下図は前進代入の例).
- そこで前述の(b)のハイパープレーン法や(c)のRED-BLACK法を用いループ内のリカレンス(依存関係)をなくす手法が使用できる.
- リカレンス(依存関係)をなくす事で**前回講義のようにハードウェアの並列計算機構を使い高速に計算可能**となる. また**スレッド並列化にも使用可能**.

(a)オリジナルスイープ



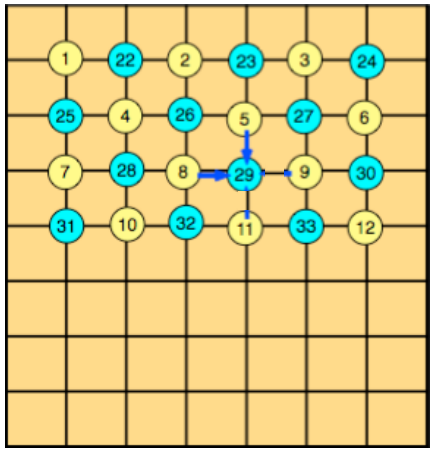
参照関係
計算順序(スイープ順序)

(b)ハイパープレーンスイープ



- 斜めのハイパープレーン内はリカレンスがない.
- ハイパープレーン間のみにリカレンスが発生.

(c)RED-BLACKスイープ



- 黄と青の2つのループに再構成する.
- それぞれのループ内はリカレンスがない.

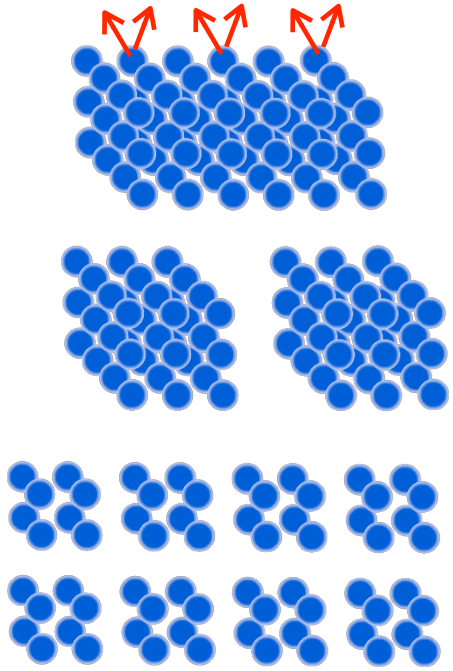
- ✦前回の講義でCPU/GPU両方について現在のスーパーコンピュータは大量の演算器搭載していることを話した
- ✦通常の並列化の方法ではその大量の演算器を使用できない場合がある
- ✦つまり通常のアプリケーションの並列度ではハードウェアの持つ並列度を使いきれない場合がある

アプリケーション並列度の拡張

(CPU/GPU共通)

アプリケーション並列度とハードウェア並列度のミスマッチ

- 波数展開を使う第一原理計算の場合を考える.
- 電子のエネルギーバンド並列のみ実装されていると仮定
- 原子一個に対し最外核の電子2個と考えるとエネルギーバンドの数は，おおよそ原子数：N×2となる.
- N=10000ならエネルギーバンド数は20000程度である.
- この場合，最大でも20000並列までしか到達しない.
- CPUシステムでは通信の増大や計算の粒度を考えると数百並列で限界となる.
- 数万のノードを装備するCPUベースのスパコンとアプリケーションの並列度がミスマッチしている
- GPUシステムでも20000並列では1GPUの並列度を使いきれなくハードウェアとアプリケーションの並列度がミスマッチしている
- 波数等その他の分割を組み合わせた並列度の拡大が必要となる.



RSDFTの並列度拡張の実例

離散化の結果

連立1次方程式 $Ax = b$

A : 行列　　***b*** : 定数ベクトル

x : 解ベクトル

固有値方程式 $Ax = \lambda x$

A : 行列　　λ : 固有値(スカラ)

x : 固有ベクトル

離散化の結果

連立1次方程式 $Ax = b$

A : 行列　　***b*** : 定数ベクトル

x : 解ベクトル

固有値方程式 $Ax = \lambda x$

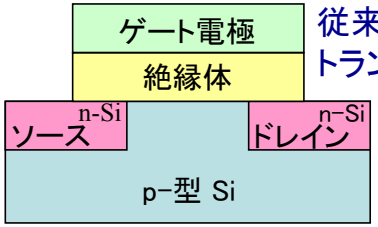
A : 行列　　λ : 固有値(スカラ)

x : 固有ベクトル

RSDFTとは

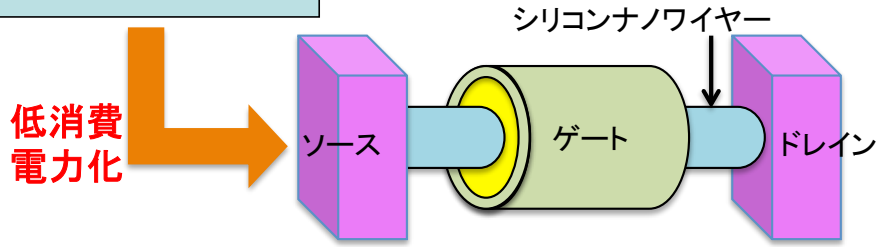
- ナノスケールでの量子論的諸現象を第一原理に立脚して解明し新機能を有するナノ物質・構造を予測
- 例えば・・・

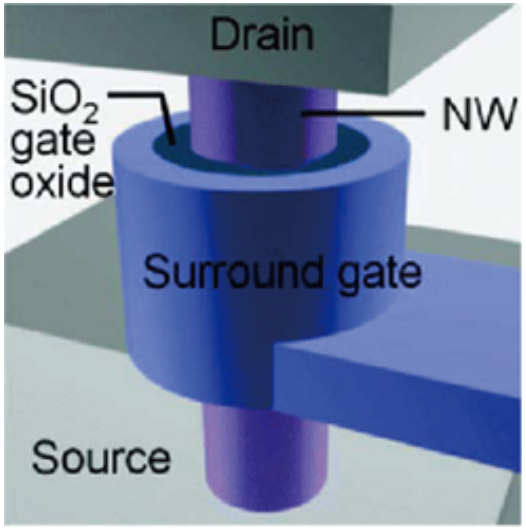
漏れ電流が問題



従来型の電界効果トランジスタ

新しいシリコンナノワイヤ電界効果トランジスタ





漏れ電流を押さえる

RSDFTの原理

waveZ

Kohn-Sham方程式

電子密度 $n(\mathbf{r}) = \sum_i |\varphi_i(r)|^2$

$$\left[-\frac{1}{2}\nabla^2 + v_{\mathrm{nucl}}(\mathbf{r}) + \int \frac{n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}' + \frac{\delta E_{\mathrm{XC}}[n]}{\delta n(\mathbf{r})} \right] \varphi_i(\mathbf{r}) = \varepsilon_i \varphi_i(\mathbf{r})$$

波動関数

ハミルトニアン

φ_i: 電子軌道（=波動関数）

i：電子準位（=エネルギーバンド）

r：空間離散点（=空間格子）

$$H\varphi_i(r) = \varepsilon_i \varphi_i(r)$$

固有値方程式

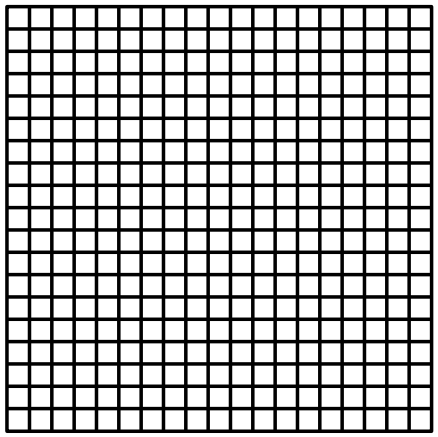
RSDFTの原理

実空間法

$$H\varphi_i(r) = \varepsilon_i\varphi_i(r)$$ 固有値方程式

Kohn-Sham方程式を3次元格子上に
離散化し解く

ML2



ユニットセル（実際は3次元）

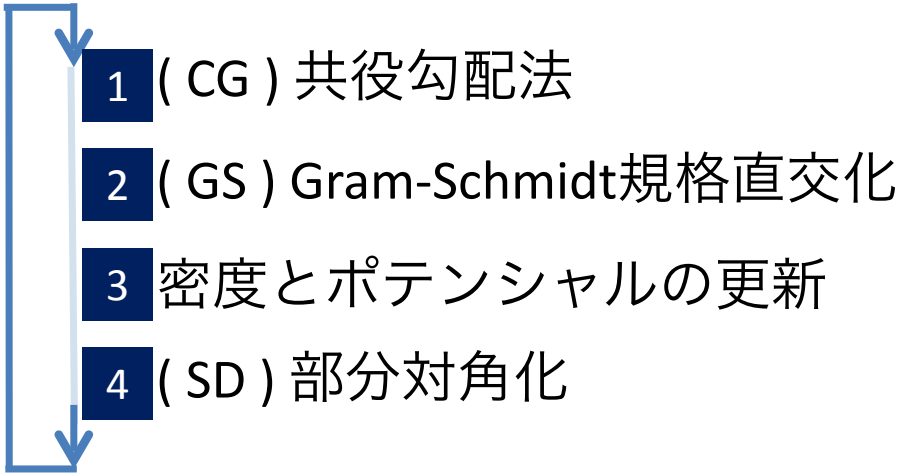
各次元方向をML1,ML2,ML3等分して格子を生成

ML(=ML1×ML2×ML3）次元のエルミート行列の固有値問題

RSDFTの計算フロー

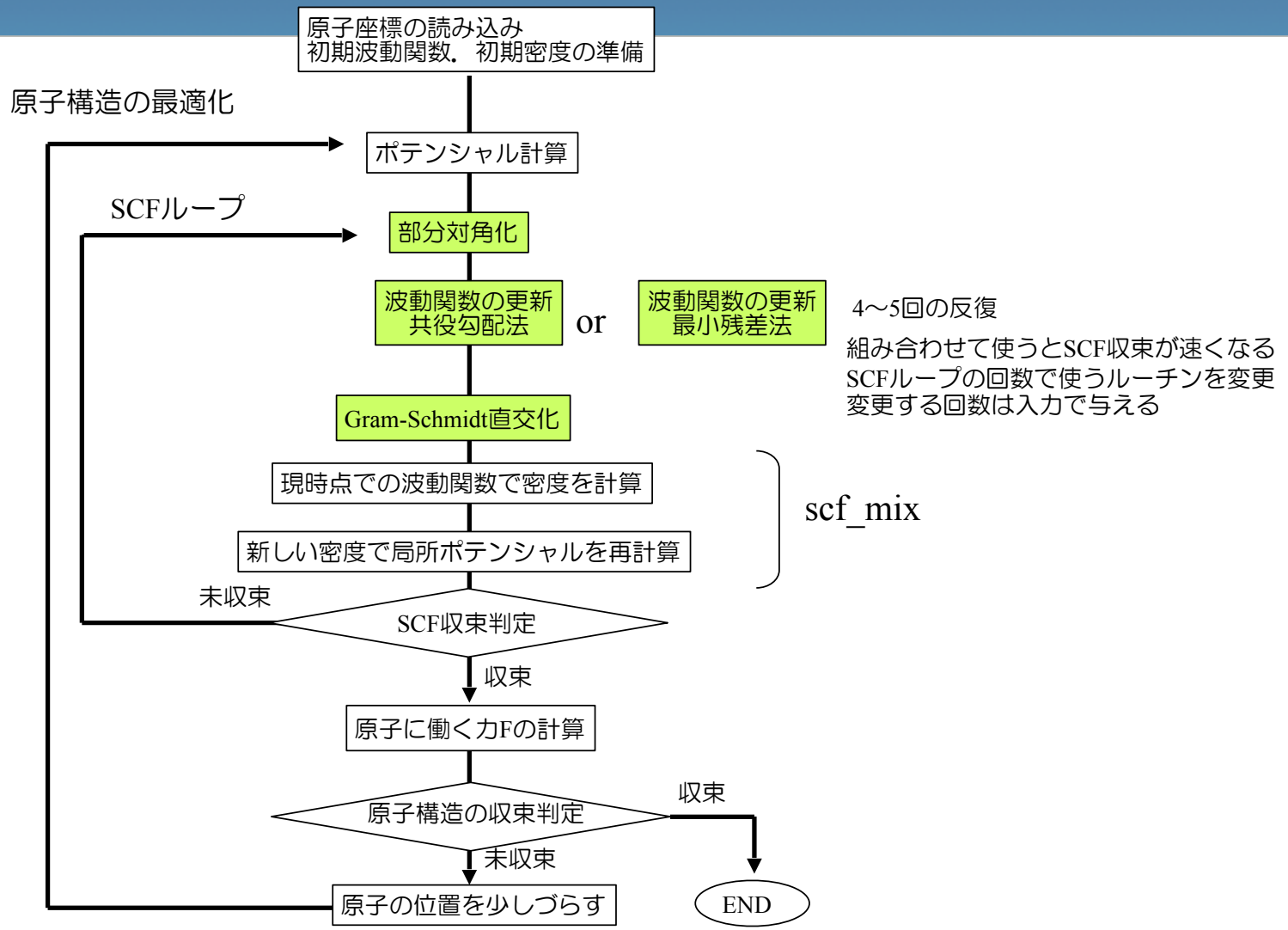
waveZ

Self-Consistent Field procedure



SCF計算

RSDFTの計算フロー

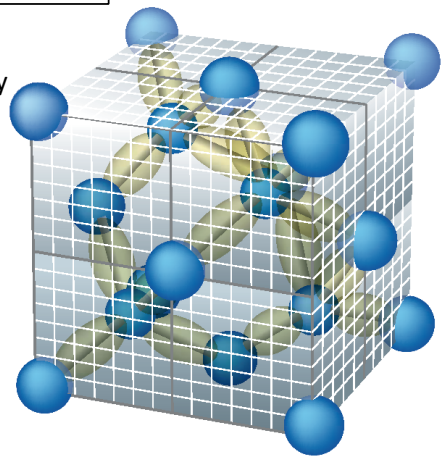


RSDFTの並列化

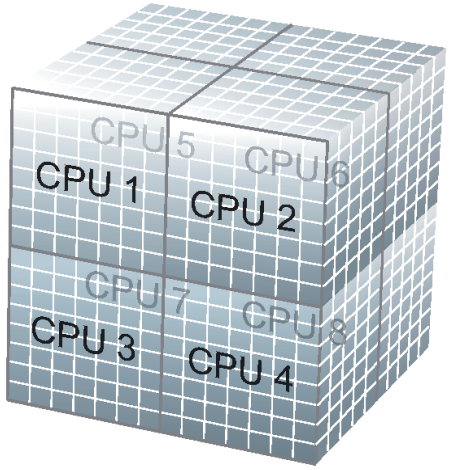
J.-I. Iwata *et al.*, J. Comp. Phys. (2010)

Real space

Blue : Si atom
Yellow: electron density



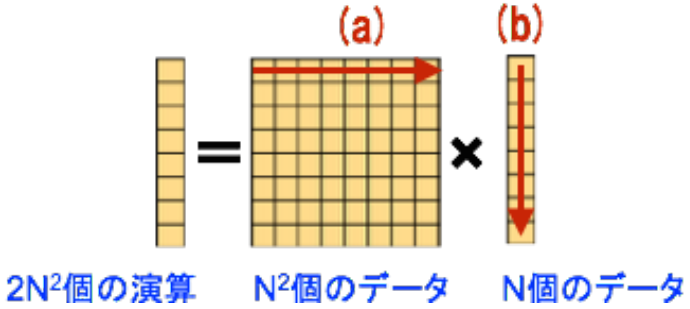
CPU space



RSDFTのCPU単体性能の向上

スレッド並列化
キャッシュの有効利用

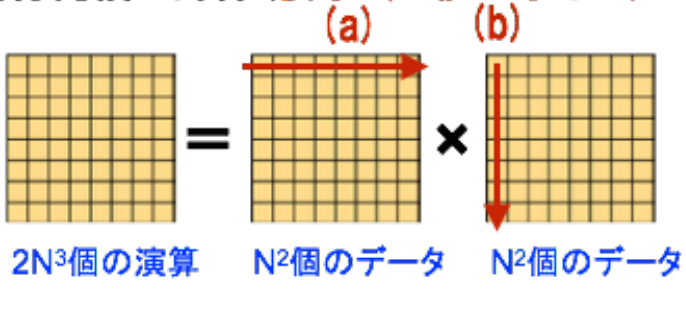
行列ベクトル積の計算（要求B/F値が大きい）



B/F値
=移動量(Byte)/演算量(Flop)
$=(N^2+N)/2N^2$
$\fallingdotseq 1/2$

原理的には1/Nより大きな値

行列行列積の計算（要求B/F値が小さい）



B/F値
=移動量(Byte)/演算量(Flop)
$=2N^2/2N^3$
$=1/N$

原理的にはNが大きい程小さな値

RSDFTのCPU単体性能の向上

waveZ

GramSchmidt直交化の行列積化

$$\psi_1' = \psi_1$$

$$\psi_2' = \psi_2 - \langle\psi_1'|\psi_2\rangle|\psi_1'\rangle$$

$$\psi_3' = \psi_3 - \langle\psi_1'|\psi_3\rangle|\psi_1'\rangle - \langle\psi_2'|\psi_3\rangle|\psi_2'\rangle$$

$$\psi_4' = \psi_4 - \langle\psi_1'|\psi_4\rangle|\psi_1'\rangle - \langle\psi_2'|\psi_4\rangle|\psi_2'\rangle - \langle\psi_3'|\psi_4\rangle|\psi_3'\rangle$$

$$\psi_5' = \psi_5 - \langle\psi_1'|\psi_5\rangle|\psi_1'\rangle - \langle\psi_2'|\psi_5\rangle|\psi_2'\rangle - \langle\psi_3'|\psi_5\rangle|\psi_3'\rangle - \langle\psi_4'|\psi_5\rangle|\psi_4'\rangle$$

$$\psi_6' = \psi_6 - \langle\psi_1'|\psi_6\rangle|\psi_1'\rangle - \langle\psi_2'|\psi_6\rangle|\psi_2'\rangle - \langle\psi_3'|\psi_6\rangle|\psi_3'\rangle - \langle\psi_4'|\psi_6\rangle|\psi_4'\rangle - \langle\psi_5'|\psi_6\rangle|\psi_5'\rangle$$

$$\psi_7' = \psi_7 - \langle\psi_1'|\psi_7\rangle|\psi_1'\rangle - \langle\psi_2'|\psi_7\rangle|\psi_2'\rangle - \langle\psi_3'|\psi_7\rangle|\psi_3'\rangle - \langle\psi_4'|\psi_7\rangle|\psi_4'\rangle - \langle\psi_5'|\psi_7\rangle|\psi_5'\rangle - \langle\psi_6'|\psi_7\rangle|\psi_6'\rangle$$

$$\psi_8' = \psi_8 - \langle\psi_1'|\psi_8\rangle|\psi_1'\rangle - \langle\psi_2'|\psi_8\rangle|\psi_2'\rangle - \langle\psi_3'|\psi_8\rangle|\psi_3'\rangle - \langle\psi_4'|\psi_8\rangle|\psi_4'\rangle - \langle\psi_5'|\psi_8\rangle|\psi_5'\rangle - \langle\psi_6'|\psi_8\rangle|\psi_6'\rangle - \langle\psi_7'|\psi_8\rangle|\psi_7'\rangle$$

$$\psi_9' = \psi_9 - \langle\psi_1'|\psi_9\rangle|\psi_1'\rangle - \langle\psi_2'|\psi_9\rangle|\psi_2'\rangle - \langle\psi_3'|\psi_9\rangle|\psi_3'\rangle - \langle\psi_4'|\psi_9\rangle|\psi_4'\rangle - \langle\psi_5'|\psi_9\rangle|\psi_5'\rangle - \langle\psi_6'|\psi_9\rangle|\psi_6'\rangle - \langle\psi_7'|\psi_9\rangle|\psi_7'\rangle - \langle\psi_8'|\psi_9\rangle|\psi_8'\rangle$$

$\vdots$

オリジナルは行列ベクトル積

RSDFTのCPU単体性能の向上

waveZ

GramSchmidt直交化の行列積化

ベクトル積を行列積に変換

再帰分割法

三角部(DGEMV)

四角部(DGEMM)

$$\varphi_1 = \psi_1$$

$$\varphi_2 = \psi_2 - \varphi_1\langle\varphi_1{}^*|\psi_2\rangle$$

$$\varphi_3 = \psi_3 - \varphi_1\langle\varphi_1{}^*|\psi_3\rangle - \varphi_2\langle\varphi_2{}^*|\psi_3\rangle$$

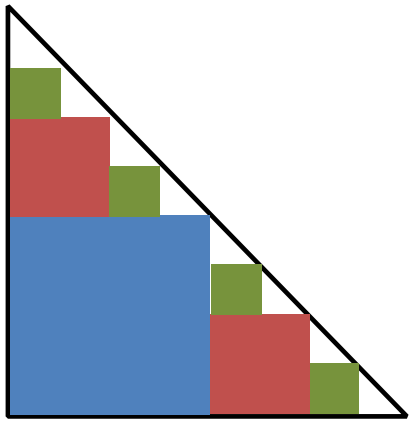
$$\varphi_4 = \psi_4 - \varphi_1\langle\varphi_1{}^*|\psi_4\rangle - \varphi_2\langle\varphi_2{}^*|\psi_4\rangle - \varphi_3\langle\varphi_3{}^*|\psi_4\rangle$$

$$\varphi_5 = \psi_5 - \varphi_1\langle\varphi_1{}^*|\psi_5\rangle - \varphi_2\langle\varphi_2{}^*|\psi_5\rangle - \varphi_3\langle\varphi_3{}^*|\psi_5\rangle - \varphi_4\langle\varphi_4{}^*|\psi_5\rangle$$

$$\varphi_6 = \psi_6 - \varphi_1\langle\varphi_1{}^*|\psi_6\rangle - \varphi_2\langle\varphi_2{}^*|\psi_6\rangle - \varphi_3\langle\varphi_3{}^*|\psi_6\rangle - \varphi_4\langle\varphi_4{}^*|\psi_6\rangle - \varphi_5\langle\varphi_5{}^*|\psi_6\rangle$$

$$\varphi_7 = \psi_7 - \varphi_1\langle\varphi_1{}^*|\psi_7\rangle - \varphi_2\langle\varphi_2{}^*|\psi_7\rangle - \varphi_3\langle\varphi_3{}^*|\psi_7\rangle - \varphi_4\langle\varphi_4{}^*|\psi_7\rangle - \varphi_5\langle\varphi_5{}^*|\psi_7\rangle - \varphi_6\langle\varphi_6{}^*|\psi_7\rangle$$

$$\varphi_8 = \psi_8 - \varphi_1\langle\varphi_1{}^*|\psi_8\rangle - \varphi_2\langle\varphi_2{}^*|\psi_8\rangle - \varphi_3\langle\varphi_3{}^*|\psi_8\rangle - \varphi_4\langle\varphi_4{}^*|\psi_8\rangle - \varphi_5\langle\varphi_5{}^*|\psi_8\rangle - \varphi_6\langle\varphi_6{}^*|\psi_8\rangle - \varphi_7\langle\varphi_7{}^*|\psi_8\rangle$$



- 依存関係のある三角部とない四角部にブロック化して計算
- 再帰的にブロック化することで四角部を多く確保

※ＳＤも同様に行列積化が可能

RSDFTのCPU単体性能の向上

waveZ

RSDFT

- 実空間差分法
- 空間並列

計算コアの最適化

- 行列積化

ターゲット計算機：PACS-CS, T2K-Tsukuba

スレッド並列の実装

ターゲット計算機：PACS-CS, T2K-Tsukuba

RSDFTの高並列化

waveZ

RSDFT

- 実空間差分法
- ベクトルの内積計算が基本
- 空間並列

※高速固有値ライブラリ

Imamura el al. SNA+MC2010 (2010)

計算コアの最適化

- 行列積化

ターゲット計算機：PACS-CS, T2K-Tsukuba

スレッド並列の実装

ターゲット計算機：PACS-CS, T2K-Tsukuba

超並列向けの実装

- **バンド並列の拡張**
- **EIGENライブラリ※の適用**

ターゲット計算機：K computer

RSDFTの高並列化

アプリケーションとハードウェアの並列度のミスマッチ
（アプリケーションの並列度不足）
の対応

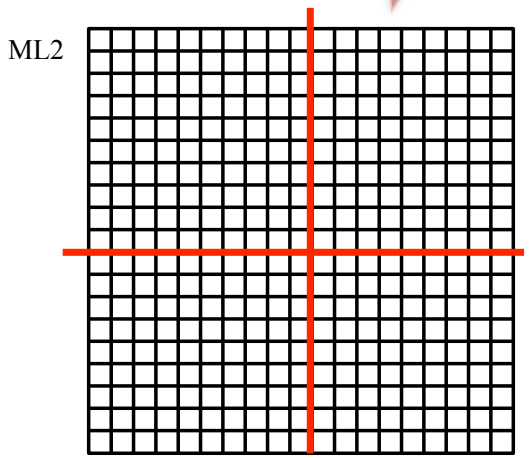
固有値方程式

$$H\varphi_i(r) = \varepsilon_i\varphi_i(r)$$

φ_i: 電子軌道（=波動関数）

i：電子準位（=エネルギーバンド）

r：空間離散点（=空間格子）



ユニットセル（実際は3次元）

RSDFTの高並列化

waveZ

アプリケーションとハードウェアの並列度のミスマッチ
（アプリケーションの並列度不足）
の対応

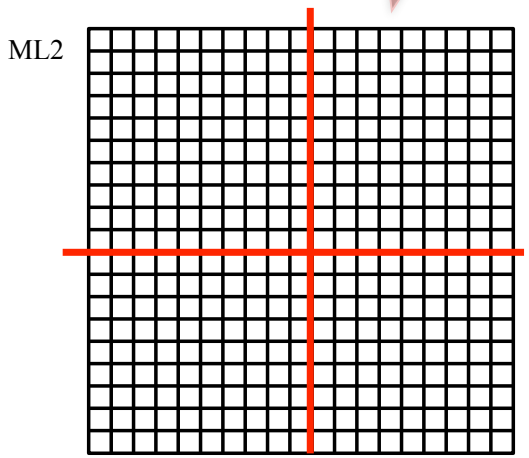
固有値方程式

$$H\varphi_i(r) = \varepsilon_i\varphi_i(r)$$

φ_i: 電子軌道（=波動関数）

i：電子準位（=エネルギーバンド）

r：空間離散点（=空間格子）



ユニットセル（実際は3次元）

- i はエネルギーバンド量子数
- i についての依存関係はない
- **空間(S)に加えエネルギーバンド(B)の完全並列を実装**
- 万を超えるCPUの並列度を確保

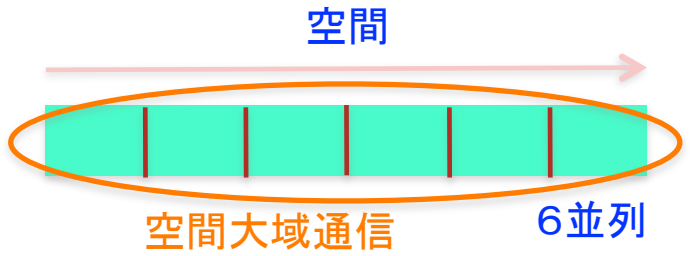
RSDFTの高並列化

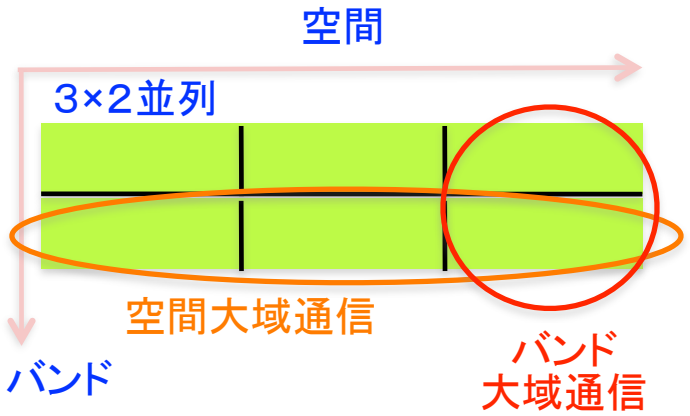
waveZ

並列軸拡張の効果

フルノードにおける大域通信の発生 の対応

- 並列軸を増やす事で空間の分割粒度を増やすことが出来る
- CPUで10万並列レベルに対応可能
- 空間並列のみの場合は全プロセッサ間の大域通信が必要
- 通信時間の増大を招く
- 2軸並列への書換で空間に対する大域通信が一部のプロセッサ間での通信とできる
- バンドに対する大域通信も同様
- 大域通信の効率化が実現可





RSDFTの高並列化

waveZ

総合性能

Table 2. Distribution of computational costs for an iteration of the SCF calculation of the modified code.

Procedure block		Execution time (s)	Computation time (s)	Communication time (s)				Performance (PFLOPS/%)
				Adjacent/grids	Global/grids	Global/orbitals	Wait/orbitals	
SCF		2903.10	1993.89	61.73	823.02	12.57	11.89	5.48/51.67
SD		1796.97	1281.44	13.90	497.36	4.27	–	5.32/50.17
	MatE/SD	525.33	363.18	13.90	143.98	4.27	–	6.15/57.93
	EigenSolve/SD	492.56	240.66	–	251.90	–	–	0.01/1.03
	RotV/SD	779.08	677.60	–	101.48	–	–	8.14/76.70
CG		159.97	43.28	47.83	68.85	0.01	–	0.06/0.60
GS		946.16	669.17	–	256.81	8.29	11.89	6.70/63.10

The test model was a SiNW with 107,292 atoms. The numbers of grids and orbitals were 576 × 576 × 180, and 230,400, respectively. The numbers of parallel tasks in grids and orbitals were 27,648 and three, respectively, using 82,944 compute nodes. Each parallel task had 2160 grids and 76,800 orbitals.

Performance evaluation of ultra-large-scale first-principles electronic structure calculation code on the K computer

Yukihiro Hasegawa et al., *International Journal of High Performance Computing Applications* published online 17 October 2013

- **2011年ゴードン・ベル賞受賞（世界一位）**
  - **ゴードン・ベル賞：アプリケーションの実際の性能と計算科学の成果に対してアメリカ計算機学会が授与する賞**

富岳での性能

出典

富岳コデザイン・レポート

～フラッグシップ2020プロジェクト・テクニカルレポート～

フラッグシップ2020プロジェクト
理化学研究所 計算科学研究センター

2022年3月

ターゲット問題

課題番号	アプリケーション	並列処理のタイプ	ノード数/ジョブ	問題設定
7	RSDFT	多重処理	10368	複数の異種物質から構成されるナノ界面を解明するため、量子力学的第一原理計算に基づき、原子数11万・バンド数22万・SCF200回のSCF計算によるシリコンデバイスの構造最適化の計算を行う。24ケース実行する。

富岳での性能

表 4.106: RSDFT の対京性能倍率と消費電力

アプリケーション	性能倍率（注）	消費電力（注）
RSDFT	38 倍	30 MW

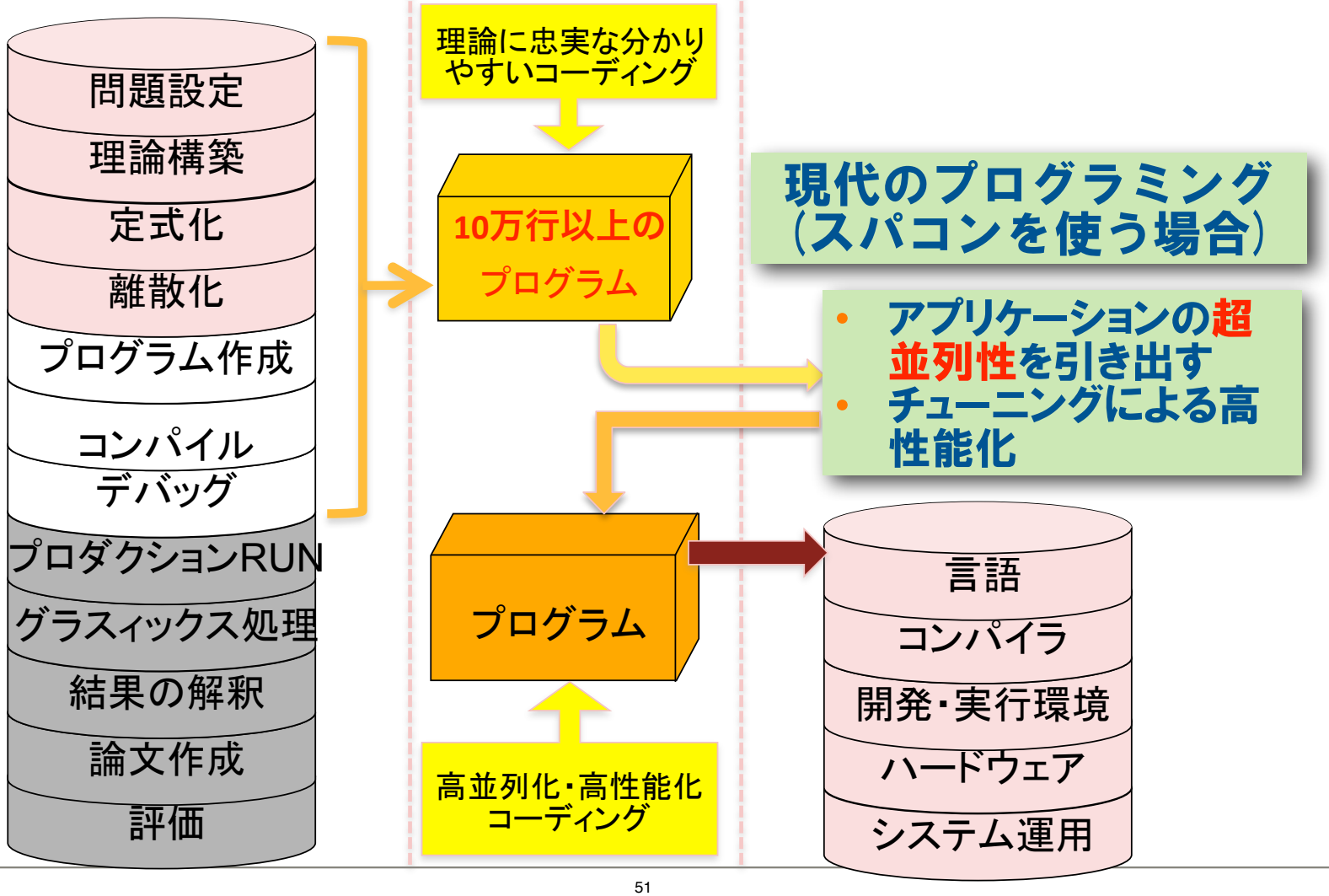
（注）ターゲット問題実行時における対京性能倍率：ブーストモードかつエコ無効モードで実行

高並列化の手順
(CPUノードを使った評価)

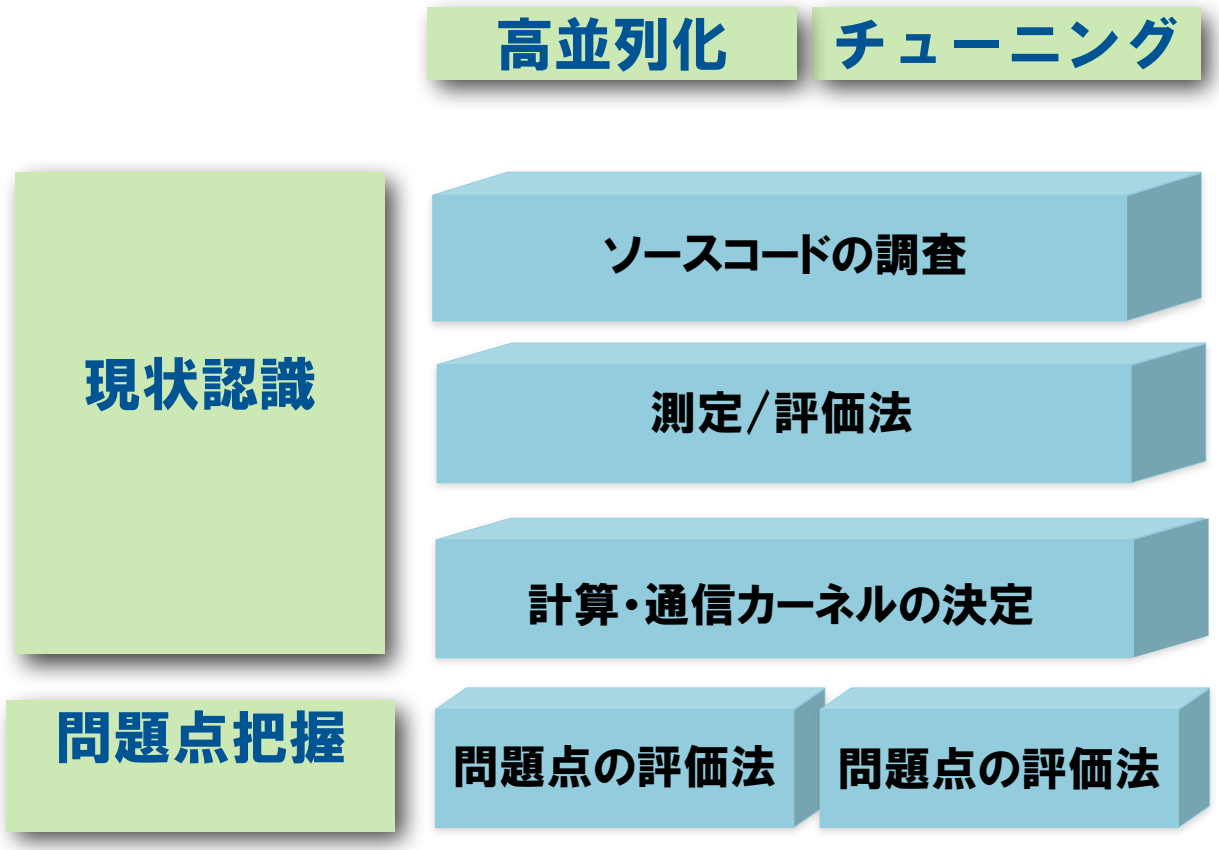
計算科学

計算機科学

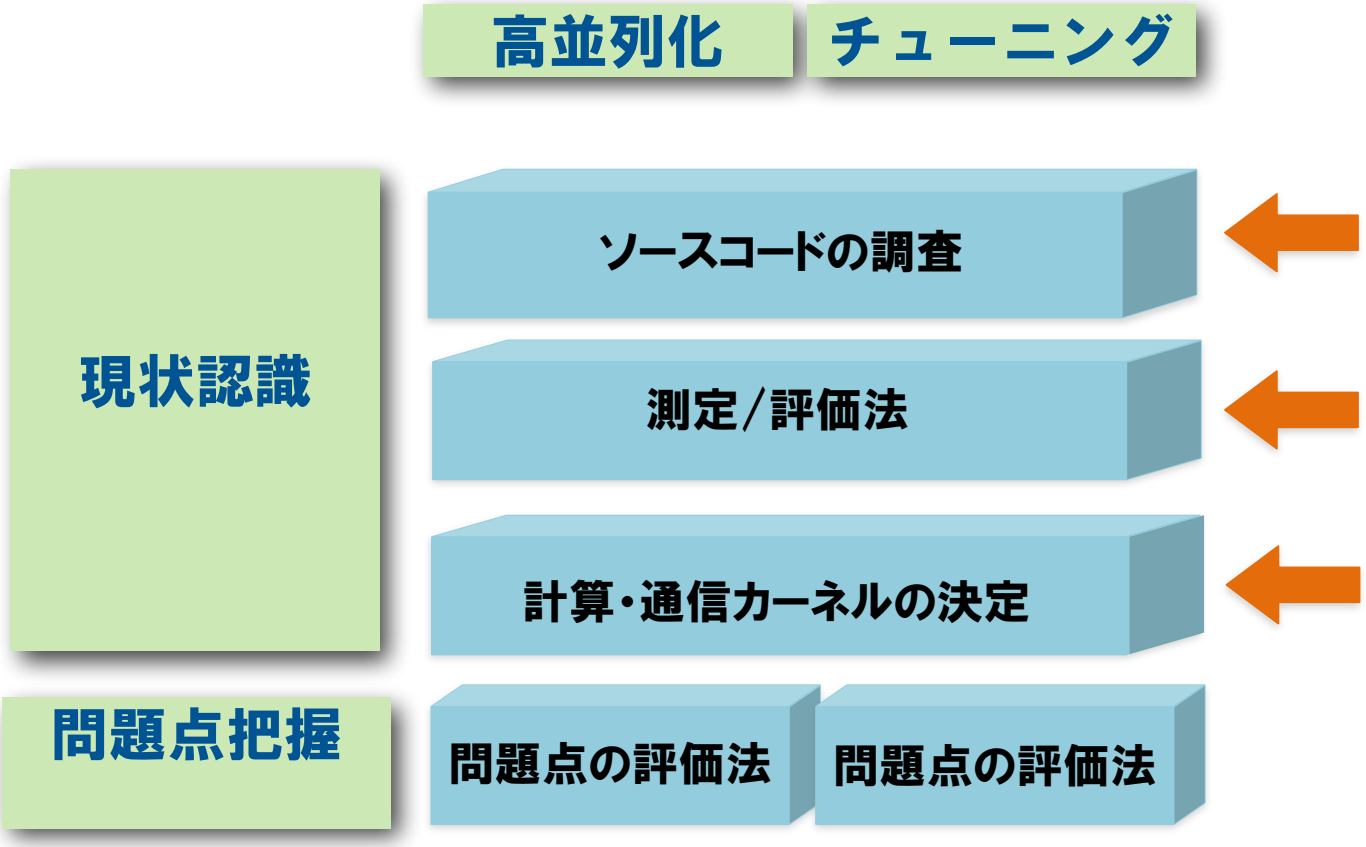
waveZ



性能最適化技術の概要

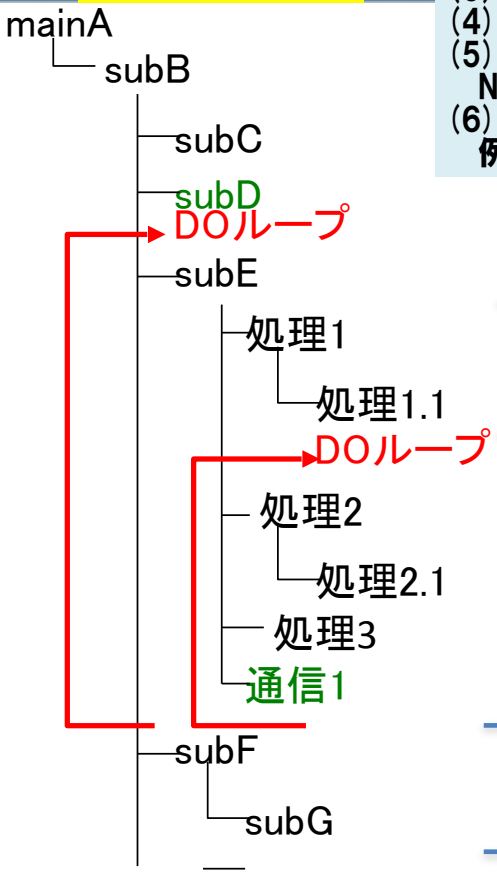


ソースコードの調査・測定/評価法・計算・通信カーネルの決定



ソースコードの調査 測定/評価法

(1) コードの構造を分析し物理に沿った処理ブロック(計算/通信)に分割
(2) コードの実行時間の実測
(3) プログラムソースコードの調査
(4) 処理ブロックの物理的処理内容を把握
(5) **計算ブロック毎の計算特性把握**(非並列/完全並列/部分並列、Nに比例/N**2に比例等)
(6) **通信ブロック毎の通信特性把握**(グローバル通信・隣接通信、隣接面に比例&隣接通信/体積に比例、等)

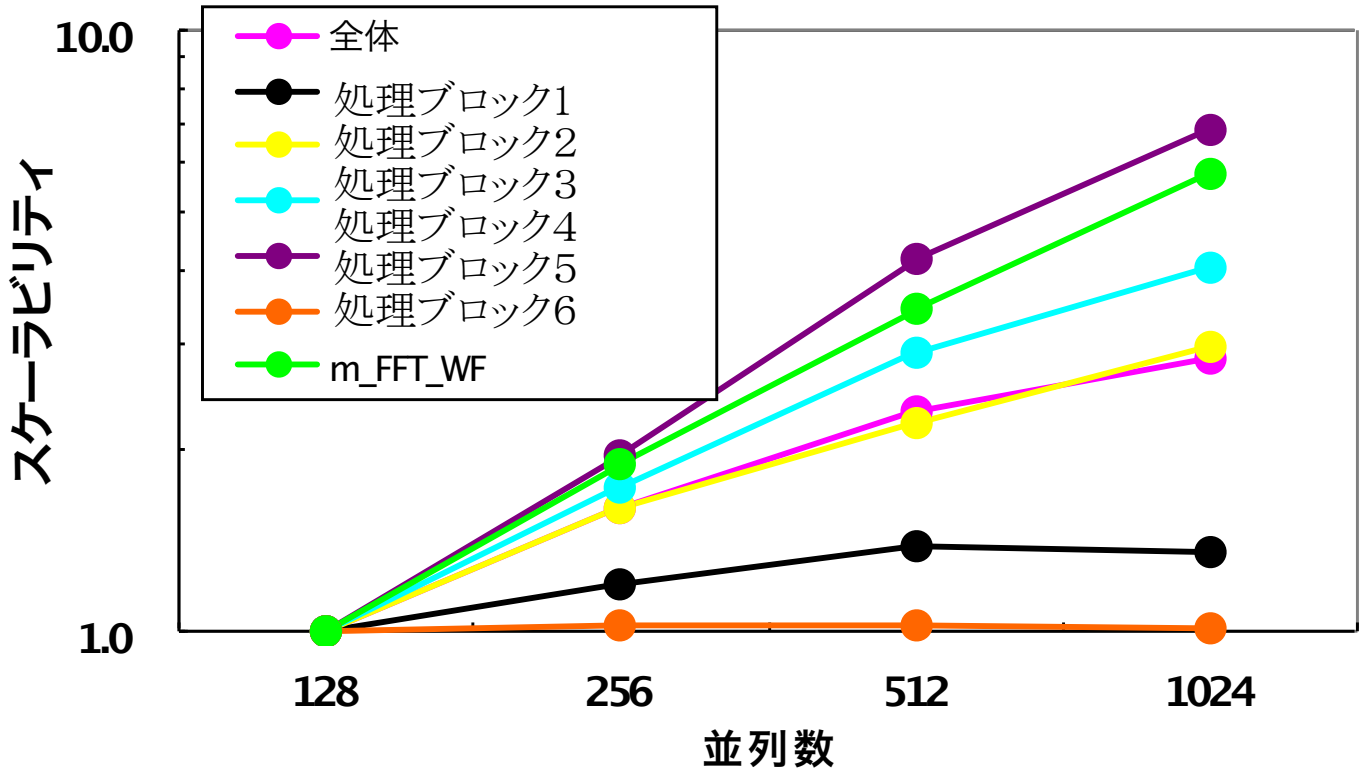


	実行時間・スケーラビリティ	物理的処理内容	演算・通信特性	演算・通信見積り	カーネル
ブロック1(計算)			部分並列	Nに比例	
ブロック2(計算)			完全並列	N**3に比例	○
(通信)			隣接通信	隣接面に比例	○
ブロック3					

計算・通信カーネルの決定

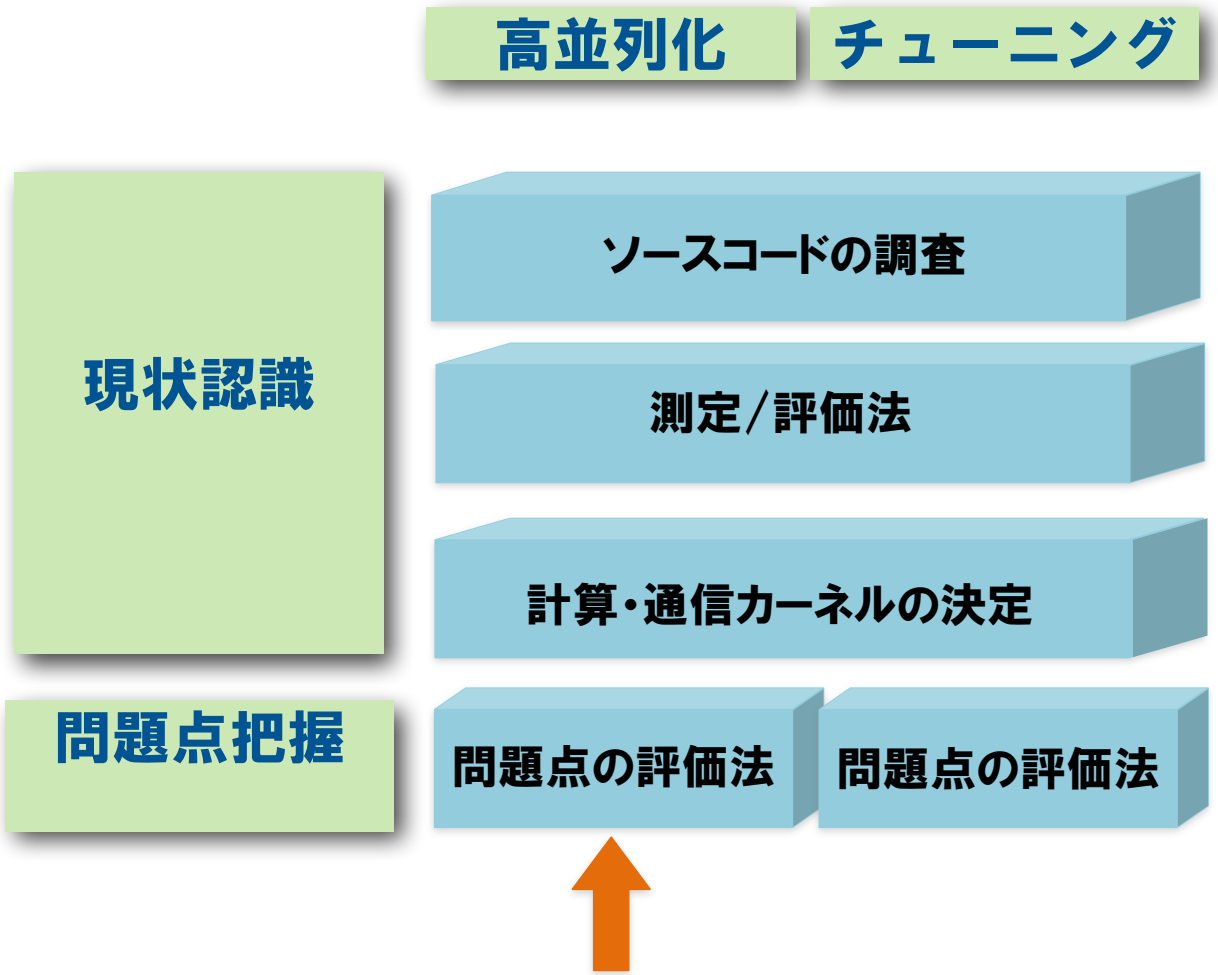
(1) 計算・通信ブロックについて物理的処理内容・コーディングの評価を行い同種の計算・通信ブロックを評価し異なる種類の計算・通信ブロックをカーネルの候補として洗い出す
(2) 並列特性分析の結果から得た問題規模に対する依存性の情報を元にターゲット問題実行時に、また高並列実行時にカーネルとなる計算・通信カーネルを洗い出す

測定/評価法
ブロック毎の実行時間とスケーラビリティ評価（例）
→従来の評価はサブルーチン毎・関数毎等の評価が多い
→サブルーチン・関数は色々な場所で呼ばれるため正しい評価ができない

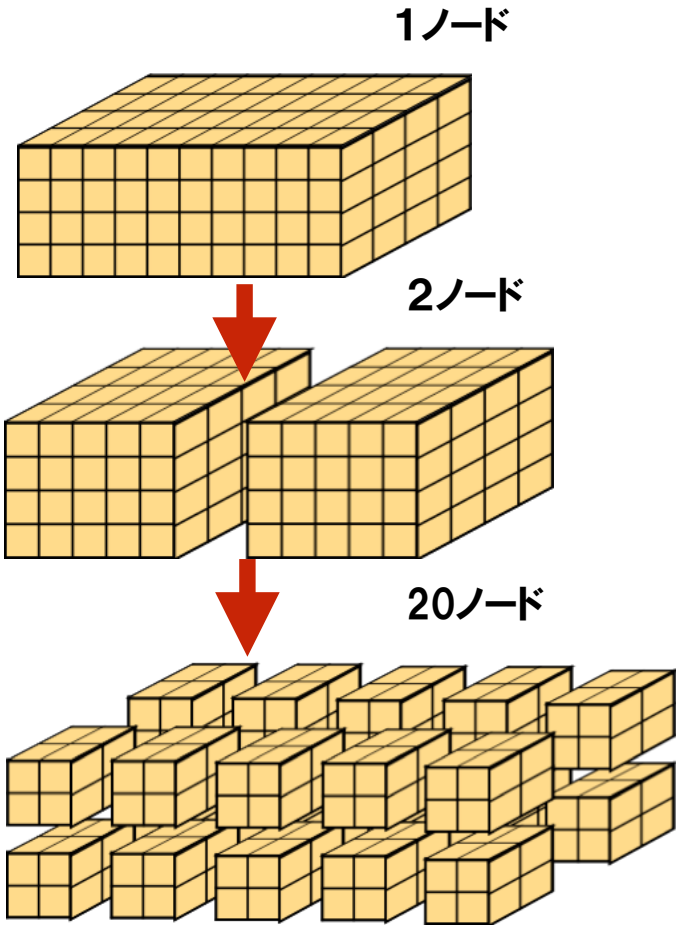


高並列における問題点の評価法

waveZ

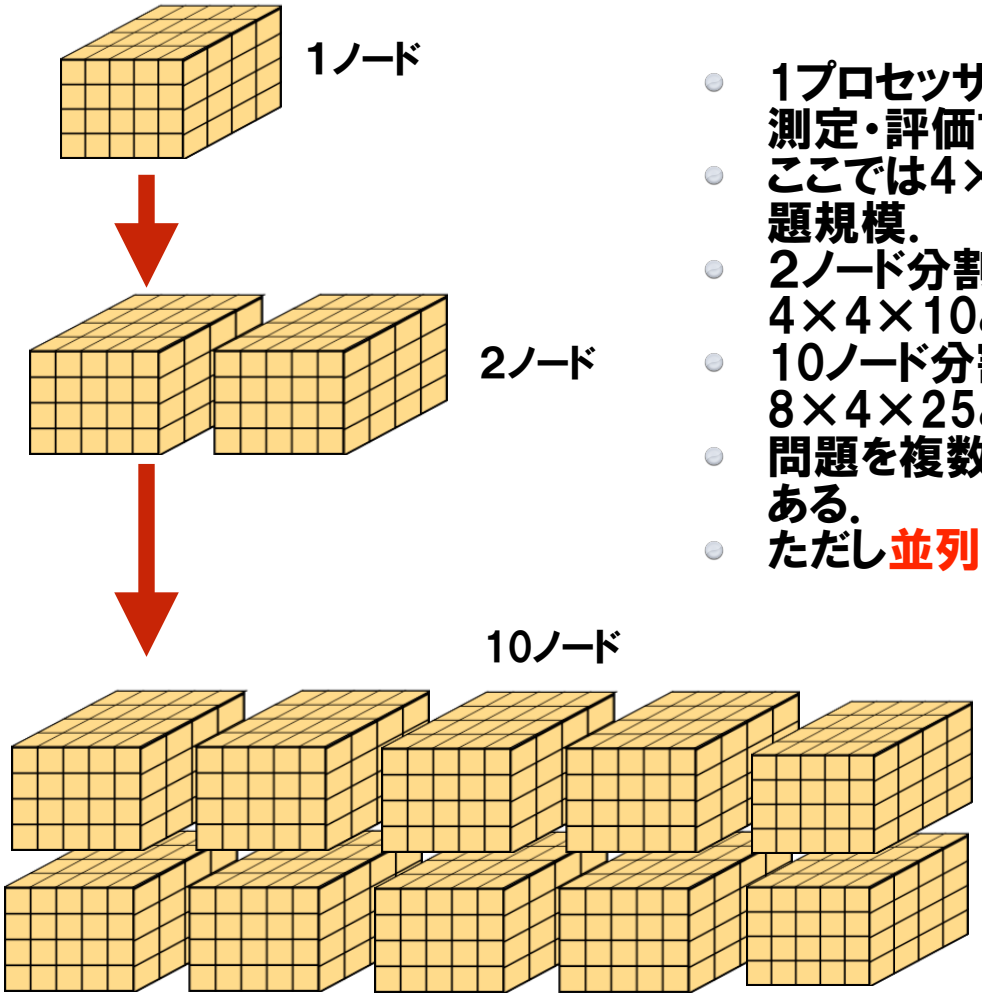


評価法　ストロングスケールによる評価



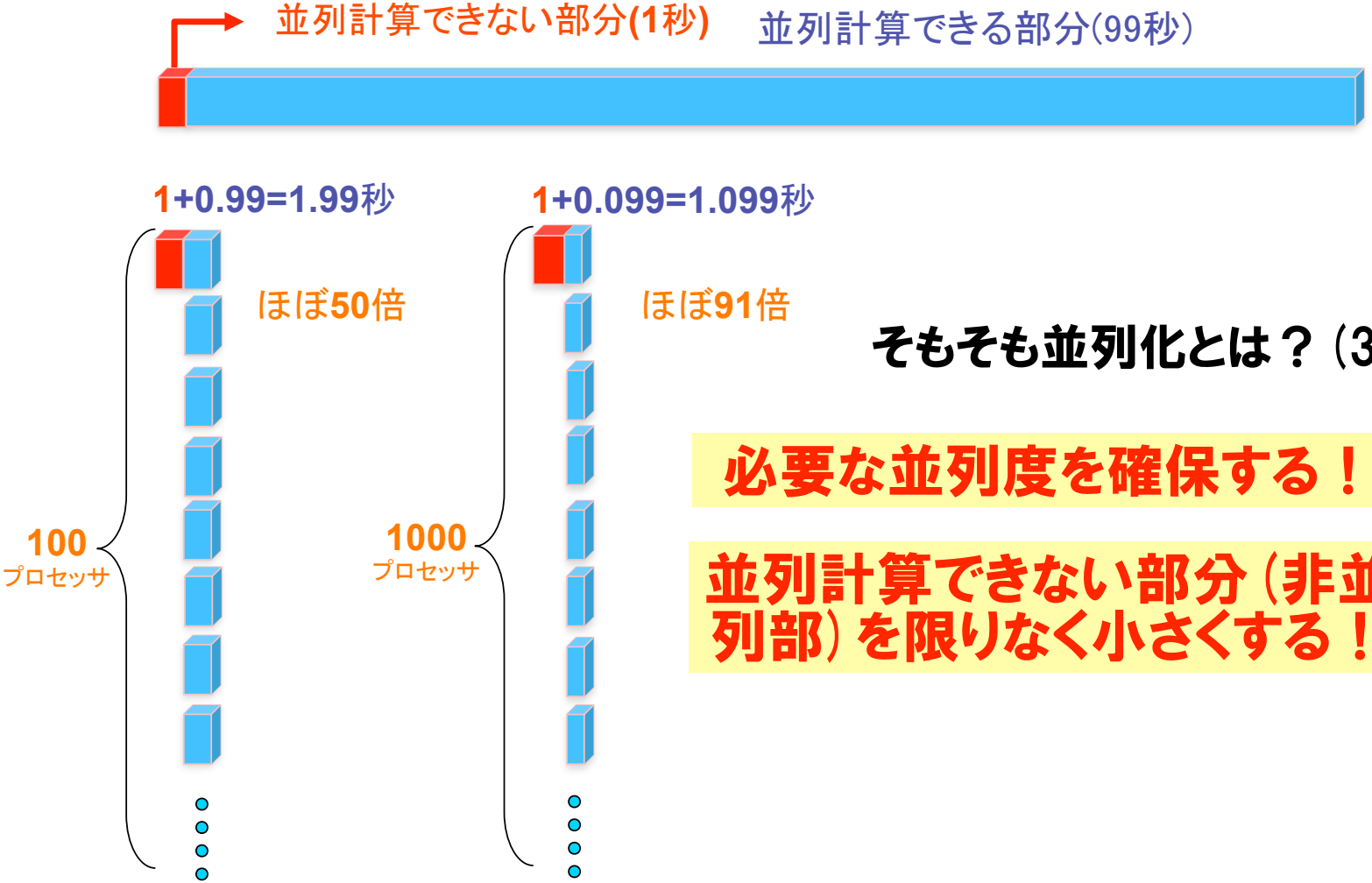
- 全体の問題規模を一定にして測定・評価する方法.
- ここでは4×4×10が全体の問題規模.
- 2ノード分割では1ノードあたりの問題規模は4×4×5となる.
- 20ノード分割では1ノードあたりの問題規模は2×2×2となる.
- 問題を1種類作れば良いので測定は楽である.
- 並列時の挙動は見えにくい.

評価法　ウィークスケーリングによる評価



- 1プロセッサあたりの問題規模を一定にして測定・評価する方法.
- ここでは4×4×5が1プロセッサあたりの問題規模.
- 2ノード分割では全体の問題規模は4×4×10となる.
- 10ノード分割では全体の問題規模は8×4×25となる.
- 問題を複数作る必要があり測定は煩雑である.
- ただし**並列時の挙動が見え易い**.

大規模並列のウィークスケーリング評価



そもそも並列化とは？（3）

必要な並列度を確保する！

並列計算できない部分（非並列部）を限りなく小さくする！

大規模並列のウィークスケーリング評価

waveZ

非並列部の評価

逐次実行時の実行時間を T_s

並列化率を α

非並列化率は $1-\alpha$

n 並列での実行時間 T_n　　$T_n = T_s(\frac{\alpha}{n} + (1-\alpha))$

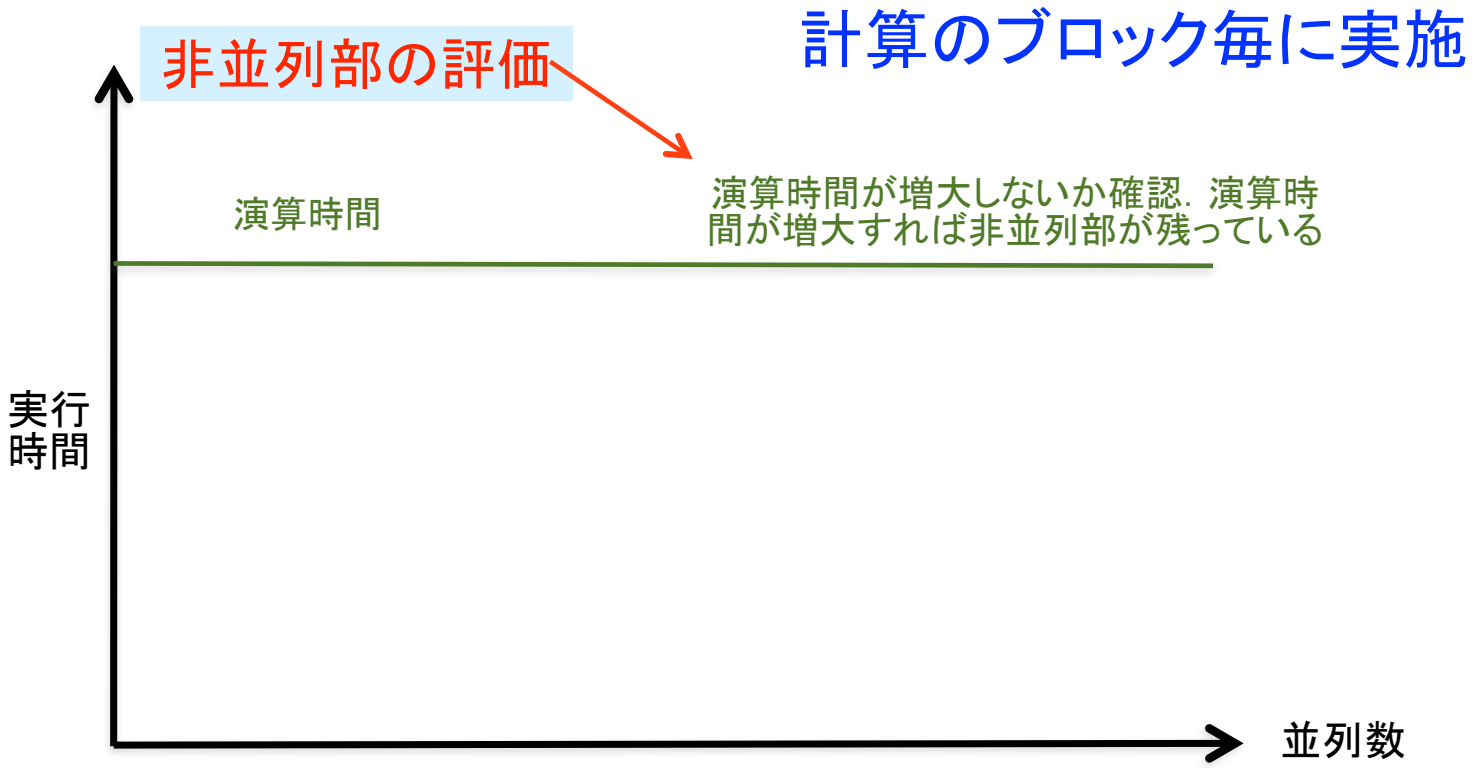
$$T_n = T_s\frac{\alpha}{n} + T_s(1-\alpha)$$

並列度を上げると，ここがどんどん大きくなる

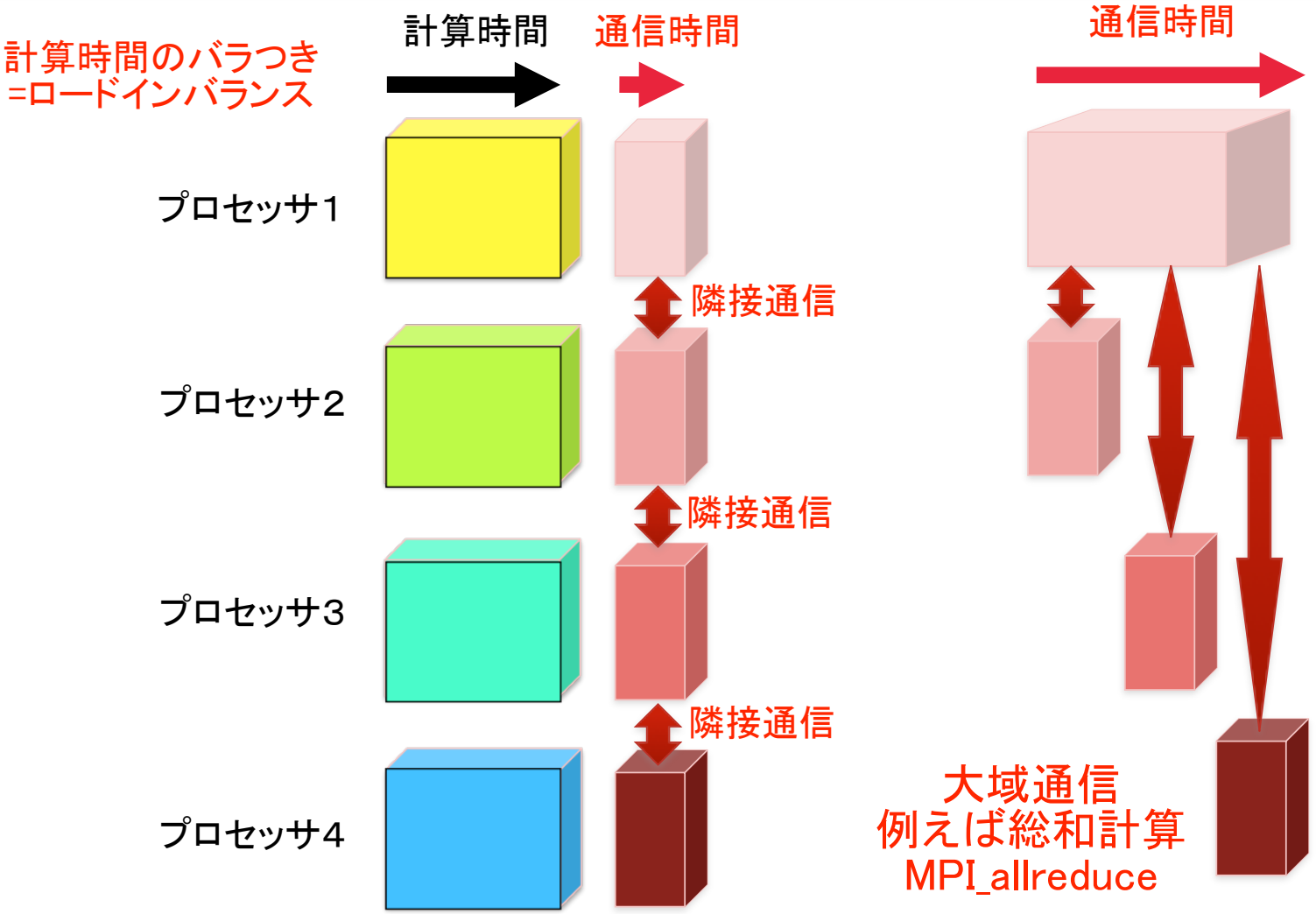
$$T_{2n} = 2T_s\frac{\alpha}{2n} + 2T_s(1-\alpha) = T_s\frac{\alpha}{n} + 2T_s(1-\alpha)$$

大規模並列のウィークスケーリング評価

- 現状使用可能な実行環境を使用し100程度/1000程度/数千程度と段階を追ってできるだけ高い並列度で並列性能を確認する(ウィークスケーリング測定).
- ウィークスケーリングが難しいものもあるが出来るだけ測定したい. 難しい場合は, 演算時間をモデル化して実測とモデルとの一致を評価する.



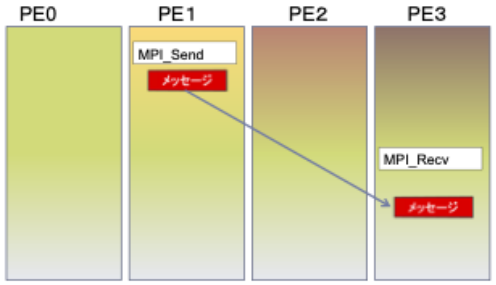
MPI並列化に使用する通信



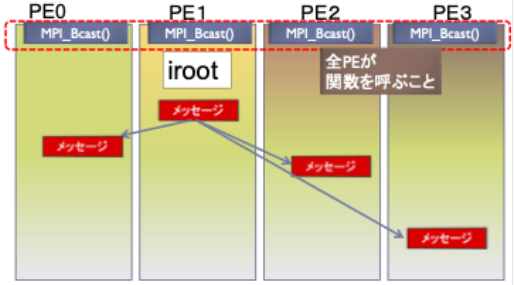
MPIの概要

waveZ

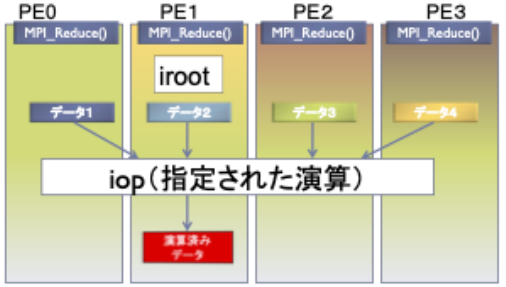
Send－Recvの概念（1対1通信）



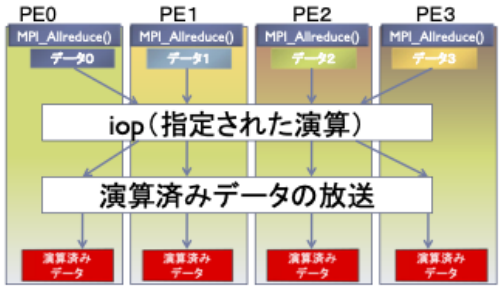
MPI_Bcastの概念（集団通信）



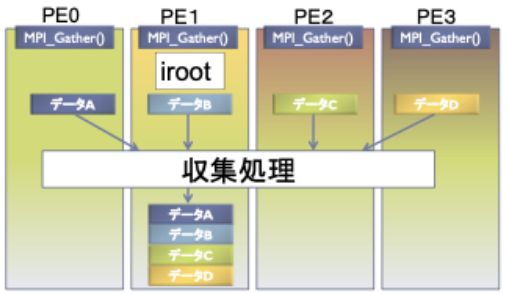
MPI_Reduceの概念（集団通信）



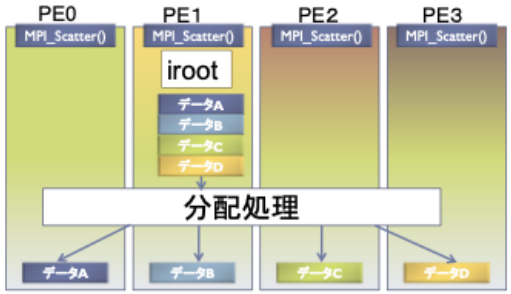
MPI_Allreduceの概念（集団通信）



MPI_Gatherの概念（集団通信）



MPI_Scatterの概念（集団通信）



2017年度　計算科学技術特論A

名古屋大学 NAGOYA UNIVERSITY

超高並列を目指した場合の評価点
−ブロック毎に以下を評価する−

waveZ

- 非並列部が残っていないか？残っている場合に問題ないか？
- 隣接通信時間が超高並列時にどれくらいの割合を占めるか？
- 大域通信時間が超高並列時にどれくらい増大するか？
- ロードインバランスが超高並列時に悪化しないか？

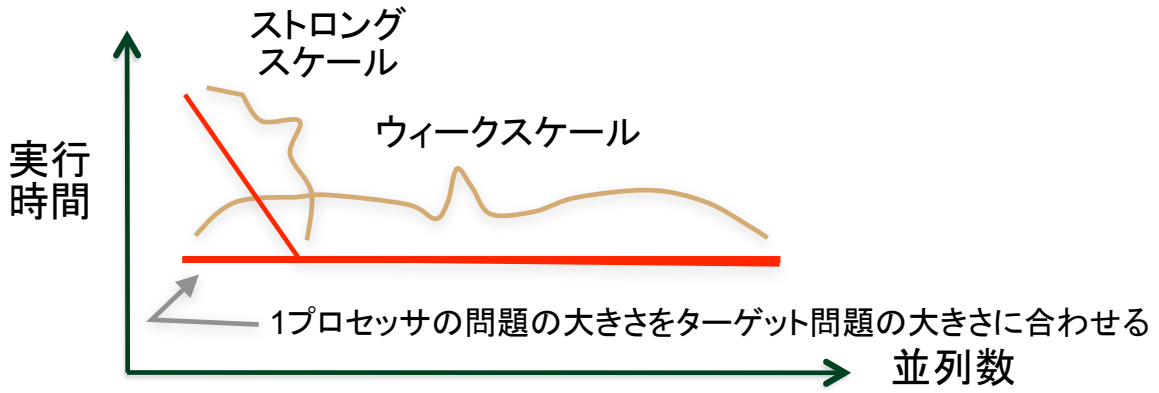
→ これらの評価が重要

そのために

ストロングスケールとウィークスケーリング 測定を使う評価

waveZ

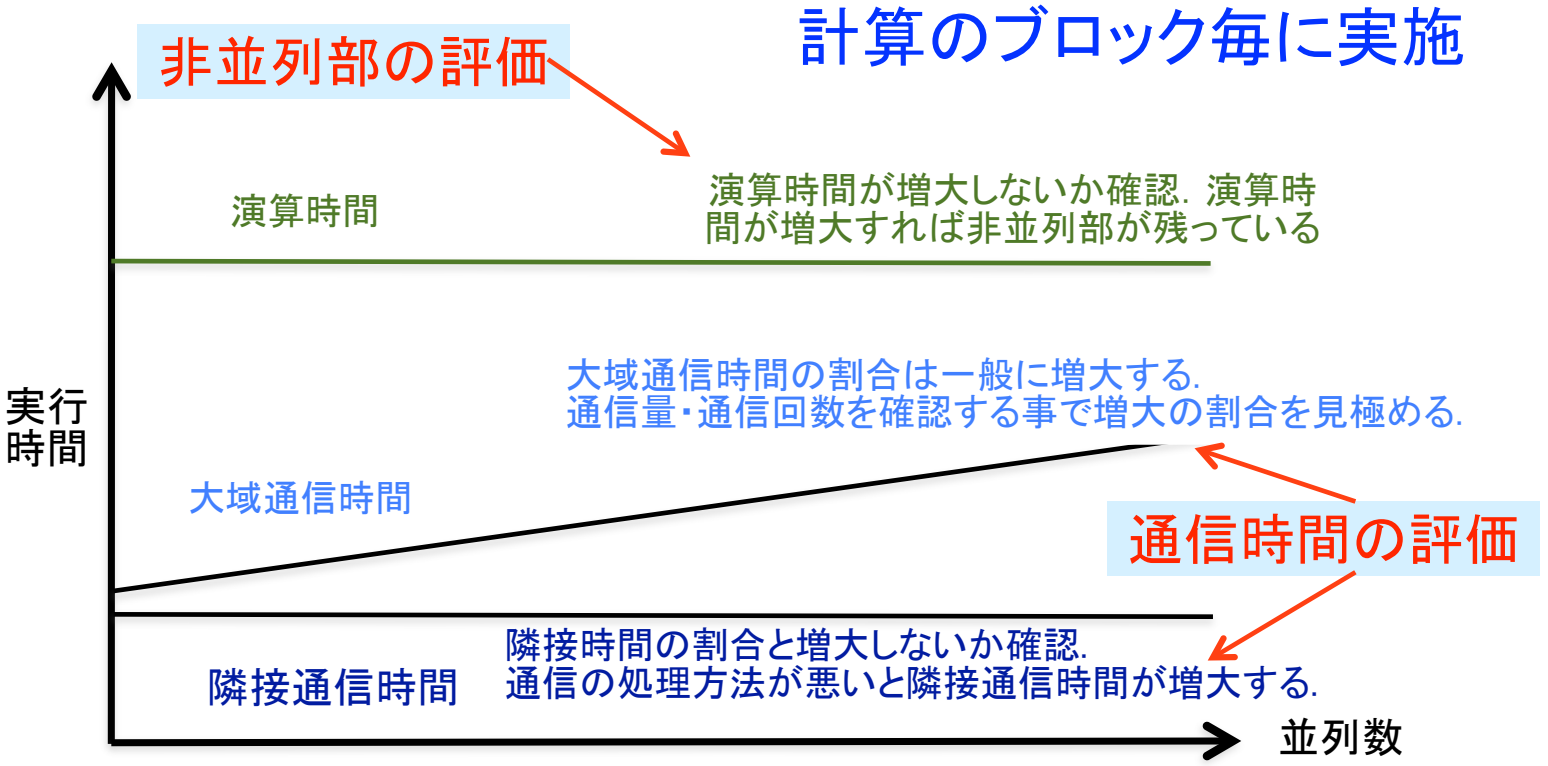
- ターゲット問題を決める→どれくらいの問題を解きたいか？そのためにどれくらいのノード数が必要か？
- 1プロセッサの問題規模がターゲット問題と同程度で，できればウィークスケーリングで実行時間・ロードインバランス・隣接通信時間・大域通信時間を測定・評価する.
- 難しければ100ノード並列程度までストロングスケールで測定しても良い.
  - その程度の並列度でスケールしていなければ根本的な問題があるので解決する
- 問題なければ並列度を上げてウィークスケーリングで大規模並列の挙動を測定する.



大規模並列のウィークスケーリング評価

waveZ

- 現状使用可能な実行環境を使用し100程度/1000程度/数千程度と段階を追ってできるだけ高い並列度で並列性能を確認する(ウィークスケーリング測定).
- ウィークスケーリングが難しいものもあるが出来るだけ測定したい. 難しい場合は, 演算時間をモデル化して実測とモデルとの一致を評価する.



まとめ

waveZ

- 反復法(CPU/GPU)
  - 並列化できるアルゴリズムとして反復法について説明しました.
- アプリケーション並列度の拡張(CPU/GPU)
  - RSDFTを例にアプリケーション並列度拡張の説明をしました.
- 高並列化手法の手順
  - CPUノードを使用した高並列化手法の手順について説明しました