マルチコアプログラミングコンテスト Cellスピードチャレンジ2008 規定課題参考資料

規定課題問題作成WG

規定課題競技のポイント

- 連立一次方程式の効率的な解法として、≪LU分解法≫[1][2]が知られています。
- LU分解法において、小容量で高速なメモリであるキャッシュを利用する≪ブロック化≫[2]という手法を用いた方法（多段多列同時消去法）が知られています。
- 今回の規定課題においても、これらの手法を駆使することが重要です。
- 規定課題を解くための数値計算ライブラリとしてLAPACK[3]が知られています。
- LAPACKは密行列の連立一次方程式を高速に解くことができるライブラリです。
- 今回の規定課題では、(1)解ベクトルの本数m、および行列Aが密行列かそうでないか（帯行列など）は競技で提供される問題により異なること、および(2)LAPACKはCell用に並列化されていないこと、などから単純な既存ライブラリの利用だけでは上位入賞は難しいと思われます。

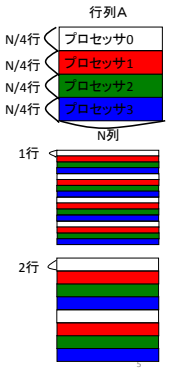
規定課題競技のポイント

- LAPACKにおいては、上記のブロック化手法を利用しています。CELLに向く実装にするため、ブロックの大きさの変更、およびライブラリ中で使われている行列-行列積ルーチン(BLASのGEMMルーチン)に対する高速化手法（アンローリングなど）を駆使してチューニングすることが重要です。
- また、解ベクトル本数（m）と行列特性（密行列、帯行列）を考慮し、適する並列アルゴリズムに変更できるようにすることも、大幅な速度向上を達成するために重要になるでしょう。
- Cell BEにおいてLINPACK[4]という数値計算ライブラリを実行した場合の性能予測[5]が公開されています。
- [5]によると8SPE利用時においてピーク性能に対し、1024×1024行列で35.7%、4096×4096行列で75.9%の効率となることが報告されています。
- 今回の規定課題についても、上位入賞のチームの性能はこれに近いものとなると予想されます。

LU分解法の参考資料

- 以下の資料は、メッセージパッシング・インタフェース（MPI）による、分散メモリ型並列計算機上でのLU分解法の並列化における参考資料です。
- Cellプログラミングでは、計算機アーキテクチャが異なることを注意してください。
- しかしCellプログラミングにおいても、並列化や高速化の際の考え方の参考となりますので、ご活用ください。

データ分散方式の説明



※列方向の分散についても同様に考える。

•（行方向） ブロック分割方式
•(Block, *) 分散方式

•(行方向) サイクリック分割方式
•(Cyclic, *) 分散方式

•(行方向)ブロック・サイクリック分割方式
•(Cyclic(2), *) 分散方式

この例の「2」: ＜ブロック幅＞とよぶ

LU分解法

- ガウス消去法のような消去処理を、行列演算として形式化した方法
- 連立一次方程式の行列表記:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_n + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

$$\downarrow$$

$$Ax = b$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix},\ x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},\ b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

LU分解法

- LU分解法では、以下の3つのステップで解を計算する
- 第1ステップ：行列AのLU分解

$$A = LU,\quad L = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix},\ U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & & \vdots \\ & & \ddots & \\ & & & u_{nn} \end{bmatrix}$$

- 第2ステップ：前進代入

$$Ax = b,\quad (LU)x = b,\quad L(Ux) = b \quad \begin{cases} Lc = b, \\ c = Ux \end{cases}$$

$Lc = b$ ：ベクトルcを求める

$$\begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- 第3ステップ：後退代入

$Ux = c$ ：解ベクトルxを求める

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & & \vdots \\ & & \ddots & \\ & & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

LU分解法の種類

■外積形式（outer-product form）ガウス法

- ガウス消去法と同等の操作でLU分解する
- 第 k列を消去したい場合、

$$\begin{aligned} & a_{11}x_1 + a_{12}x_2 + \cdots\cdots + a_{1n}x_n = b_1 \\ & a_{22}x_2 + \cdots\cdots + a_{2n}x_n = b_2 \\ & \cdots \\ & \boxed{a_{kk}x_k} + \cdots + a_{kn}x_n = b_k \\ & \cdots \\ & a_{nk}x_k + \cdots + a_{nn}x_n = b_n \end{aligned}$$

係数 a_{kk} を用いて $a_{k,k+1}, a_{k,k+2}, \cdots, a_{k,n}$を消去

外積形式ガウス法

■すなわち列の消去は、

$$a_{ik} - a_{kk}(a_{ik} / a_{kk}),\ i = k+1, k+2, \ldots, n$$

■これを行列表記にすると、行列Lを

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & l_{mk} & & 1 \end{bmatrix},\quad l_{ik} = -(a_{ik} / a_{kk}),\ i = k+1, \ldots, n$$

とすると、この消去は

$$L_k A_k = U_{k+1}$$

外積形式ガウス法

■一般的に

$$L_{n-1}L_{n-2}\cdots L_2 L_1 A = U$$

■したがってLU分解は

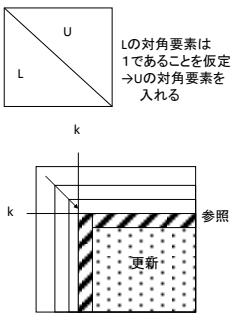
$$\begin{aligned} A &= (L_{n-1}L_{n-2}\cdots L_2 L_1)^{-1}U \\ &= (L_1^{-1}L_2^{-1}\cdots L_{n-2}^{-1}L_{n-1}^{-1})U \\ &= LU \end{aligned}$$

■ここで、L_k^{-1} は L_k の要素の符号を反転させたものであり、容易に得られる

- ●消去作業が終われば行列Lが得られる

外積形式ガウス法

```
for (k=0; k<n; k++) {
  dtmp = 1.0d0 / A[k][k];
  for (i=k+1; i<n; i++) {
    A[i][k] = A[i][k]*dtmp;
  }
  for (j=k+1; j<n; j++) {
    dakj = A[k][j];
    for (i=k+1; i<n; i++) {
      A[i][j]=A[i][j]–A[i][k]*dakj;
    }
  }
```



外積形式ガウス法

■外積形式ガウス法では分解列の右側の領域が更新される

- ■right-lookingアルゴリズムと呼ばれる

■外積形式ガウス法は並列化に向く

- ■処理の中心の更新領域が多い
  - ●負荷バランスよくデータ分散できる
- ■更新処理が、分解列と分解列という少ないデータを所有するだけで、要素ごとに独立して行える

ブロック形式ガウス法

- 行列Aを小行列に分解し、その小行列単位でLU分解する方法。LU分解と行列-行列積で実現できる。
- 具体的には（各小行列を各プロセッサが所有）

$$\begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} & \tilde{A}_{13} \\ \tilde{A}_{21} & \tilde{A}_{22} & \tilde{A}_{23} \\ \tilde{A}_{31} & \tilde{A}_{32} & \tilde{A}_{33} \end{bmatrix} = \begin{bmatrix} \tilde{L}_{11} & & 0 \\ \tilde{L}_{21} & \tilde{L}_{22} & \\ \tilde{L}_{31} & \tilde{L}_{32} & \tilde{L}_{33} \end{bmatrix} \begin{bmatrix} \tilde{U}_{11} & \tilde{U}_{12} & \tilde{U}_{13} \\ & \tilde{U}_{22} & \tilde{U}_{23} \\ 0 & & \tilde{U}_{33} \end{bmatrix}$$

とすると、右辺は

$$\tilde{A}_{11} = \tilde{L}_{11}\tilde{U}_{11},\ \tilde{A}_{12} = \tilde{L}_{11}\tilde{U}_{12} \quad ,\tilde{A}_{13} = \tilde{L}_{11}\tilde{U}_{13},$$
$$\tilde{A}_{21} = \tilde{L}_{21}\tilde{U}_{11},\tilde{A}_{22} = \tilde{L}_{21}\tilde{U}_{12} + \tilde{L}_{22}\tilde{U}_{22},\tilde{A}_{23} = \tilde{L}_{21}\tilde{U}_{13} + \tilde{L}_{22}\tilde{U}_{23},$$
$$\tilde{A}_{31} = \tilde{L}_{31}\tilde{U}_{11},\tilde{A}_{32} = \tilde{L}_{31}\tilde{U}_{12} + \tilde{L}_{32}\tilde{U}_{22},\tilde{A}_{33} = \tilde{L}_{31}\tilde{U}_{13} + \tilde{L}_{32}\tilde{U}_{23} + \tilde{L}_{33}\tilde{U}_{33}$$

ブロック形式ガウス法

- 第1ステップ　LU分解

$$\tilde{A}_{11} = \tilde{L}_{11}\tilde{U}_{11},\ \tilde{A}_{12} = \tilde{L}_{11}\tilde{U}_{12} \quad ,\tilde{A}_{13} = \tilde{L}_{11}\tilde{U}_{13},$$
$$\tilde{A}_{21} = \tilde{L}_{21}\tilde{U}_{11},\tilde{A}_{22} = \tilde{L}_{21}\tilde{U}_{12} + \tilde{L}_{22}\tilde{U}_{22},\tilde{A}_{23} = \tilde{L}_{21}\tilde{U}_{13} + \tilde{L}_{22}\tilde{U}_{23},$$
$$\tilde{A}_{31} = \tilde{L}_{31}\tilde{U}_{11},\tilde{A}_{32} = \tilde{L}_{31}\tilde{U}_{12} + \tilde{L}_{32}\tilde{U}_{22},\tilde{A}_{33} = \tilde{L}_{31}\tilde{U}_{13} + \tilde{L}_{32}\tilde{U}_{23} + \tilde{L}_{33}\tilde{U}_{33}$$

- 第2ステップ　L_{11} を転送、U_{1*}を計算

$$\tilde{A}_{11} = \tilde{L}_{11}\tilde{U}_{11},\ \tilde{A}_{12} = \tilde{L}_{11}\tilde{U}_{12} \quad ,\tilde{A}_{13} = \tilde{L}_{11}\tilde{U}_{13},$$
$$\tilde{A}_{21} = \tilde{L}_{21}\tilde{U}_{11},\tilde{A}_{22} = \tilde{L}_{21}\tilde{U}_{12} + \tilde{L}_{22}\tilde{U}_{22},\tilde{A}_{23} = \tilde{L}_{21}\tilde{U}_{13} + L_{22}U_{23},$$
$$\tilde{A}_{31} = \tilde{L}_{31}\tilde{U}_{11},\quad U_{11}\text{を転送、}L_{*1}\text{を計算}\quad {}_3 = \tilde{L}_{31}\tilde{U}_{13} + \tilde{L}_{32}\tilde{U}_{23} + \tilde{L}_{33}\tilde{U}_{33}$$

- 第3ステップ　LU分解　L_{21}を転送　U_{12}を転送　U_{13}を転送　L_{31}を転送

$$\tilde{A}_{11} = \tilde{L}_{11}\tilde{U}_{11},\ \tilde{A}_{12} = \tilde{L}_{11}\tilde{U}_{12},\ \tilde{A}_{13} = \tilde{L}_{11}\tilde{U}_{13},$$
$$\tilde{A}_{21} = \tilde{L}_{21}\tilde{U}_{11},\ \tilde{A}_{22} = \tilde{L}_{21}\tilde{U}_{12} + \tilde{L}_{22}\tilde{U}_{22},\ \tilde{A}_{23} = \tilde{L}_{21}\tilde{U}_{13} + \tilde{L}_{22}\tilde{U}_{23},$$
$$\tilde{A}_{31} = \tilde{L}_{31}\tilde{U}_{11},\ \tilde{A}_{32} = \tilde{L}_{31}\tilde{U}_{12} + \tilde{L}_{32}\tilde{U}_{22},\ \tilde{A}_{33} = \tilde{L}_{31}\tilde{U}_{13} + \tilde{L}_{32}\tilde{U}_{23} + \tilde{L}_{33}\tilde{U}_{33}$$

ブロック形式ガウス法

- 対角要素がLU分解して、行方向、列方向に部分的なLU分解を転送する。
- ブロック形式ガウス法の実現法は二通りある
  1. 実際に小行列L、Uの逆行列を求める方法
     例） $L_{21} = A_{21}\, U_{11}^{-1}$
  2. 逆行列を求めず、LU分解を用いる方法
     例） $A_{21} = L_{21}\, U_{11}$
- 1の実装の場合、行列-行列積が主演算となる
  - 高効率で実装可能

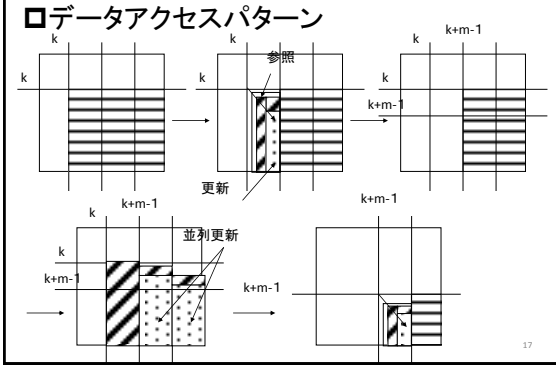
縦ブロックガウス法

- 縦ブロックガウス法は、列方向のみデータを分割する方法
- 並列化した場合、プロセッサ内に列データを全て所有しているため、ピボッティング処理が実装しやすい
  - ブロック形式ガウス法は実装が難しい
- 外積形式ガウス法の並列化に比べ
  1. 通信回数の削減
  2. ループアンローリングによる性能向上

  が期待できる

縦ブロックガウス法

- データアクセスパターン



縦ブロックガウス法

- 縦ブロックガウス法は、ある幅ごとにLU分解を行う
  - この幅のことを≪ブロック幅≫とよぶ
  - ブロック幅を用いて設計されたアルゴリズムを一般的に≪ブロック化アルゴリズム≫とよぶ
- ブロック化をすることで、演算カーネルが、2重ループ（レベル2BLAS）から、3重ループ（レベル3BLAS演算）になる
  - 実装による性能向上が得られやすい。
  - 後述の≪ループアンローリング≫による高速化がしやすい。

縦ブロックガウス法

- ブロック幅mぶんだけ、先行してLU分解を行う
  - 普通のLU分解は、
    - 第kステップ時、k行の1行分の必要情報をもとに、k×kの領域を消去する。
    - 次は、(k+1)×(k+1)の領域の消去に移る。
  - 縦ブロックガウス法では、
    - 第kステップ時、k行からk+m行のm行分の必要情報を作るために、k行からk+m行の領域について、先行的にLU分解する。
    - 上記で作成したm行分の必要情報をもとに、k×kの領域を消去する。
    - 次は、(k+m)×(k+m)の領域の消去に移る。
- mステップづつ処理が進む
  - つまり、必要な情報を得るための同期回数を1/m回に削減できるので、高速化につながる。

19

縦ブロックガウス法

- 実際のカーネル部分

```
for (jm=k;  jm<k+m;  jm++)  {
  for (j=k+m;  j<n;  j++)  {
    dakj = A[jm][j];
    for (i=jm+1;   i<n;   i++)  {
      A[i][j]=A[i][j] - A[i][jm]*dakj;
    }
  }
}
```

- ループ jm, j, iについてループの展開（ループアンローリング）可能

20

縦ブロックガウス法

- jmについて2段のアンローリング

```
for (jm=k;  jm<k+m; km+=2) {
  for (j=k+m;  j<n;  j++) {
    dakj0 = A[jm    ][j];
    dakj1 = A[jm+1][j];
    for (i=jm+1; i<n;  i++) {
      A[i][j]=A[i][j] - A[i][jm   ]*dakj0
                      - A[i][jm+1]*dakj1;
    }
  }
}
```

21

縦ブロックガウス法

- さらにjについても、2段のアンローリング

```
for (jm=k;  jm<k+m;  km+=2) {
  for (j=k+m;  j<n;  j+=2) {
    dakj00 = A[jm   ][j  ];
    dakj10 = A[jm+1][j  ];
    dakj01 = A[jm   ][j+1];
    dakj11 = A[jm+1][j+1];
    for (i=jm+1;  i<n;  i++) {
      A[i][j   ]=A[i][j   ] -A[i][jm   ]*dakj00
                            - A[i][jm+1]*dakj10;
      A[i][j+1]=A[i][j+1] -A[i][jm   ]*dakj01
                            - A[i][jm+1]*dakj11;
    } } }
```

- この処理は、ループ内で2段2列分の消去を同時にしているとみなせる（*多段多列同時消去法*）

22

縦ブロックガウス法

- ブロック化するとできる通信隠蔽
- 縦ブロックガウス法において、データを列方向ブロックサイクリック分散（＊，Cyclic(m)）するだけで実現可能

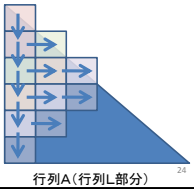
- LU分解が必要なブロックを所有するPE
  1. 優先してLU分解を行い結果を転送
  2. その他の行列更新を行う
- そのほかのPE
  1. LU分解データ受信待ち
  2. 行列更新

通信と計算のオーバーラップ→通信時間隠蔽

23

前進代入・後退代入

- 右辺ベクトルbが1つの場合、前進代入、後退代入を並列化すると、場合により高速化できる
  - 右辺ベクトルbが複数の場合は、右辺ベクトルの本数分の並列性があるのは自明
- 行列Aが列方向分散方式（＊，Block）の場合
  - ブロック単位で処理を進める
  - ウエーブフロント処理（右図）で並列化可能
  - 後退代入は、前進代入の処理に対して、方向を逆にすれば、同様に実現可能



行列A（行列L部分）

24

その他の高速化手段について

- 2べきの値で配列を確保し、一定の間隔で配列をアクセスすると、メモリやキャッシュの物理的なアクセス単位である≪バンク≫の一部にアクセスが集中し、実行性能が特定の問題サイズで劇的に低下することがあります。≪バンクコンフリクト≫
  - 2べきでない値で配列を確保して利用するだけでも、性能改善が望める可能性があります。
- 帯行列や3重対角行列の特徴（行列の形状）を利用して、LU分解法の演算量を削減できます。

参考文献

[1] 奥村晴彦著、
「C言語による最新アルゴリズム辞典」、技術評論社

[2] 小国力編著、
「行列計算ソフトウエアーWS、スーパーコン、
並列計算機」、丸善株式会社

[3] LAPACKについて、
http://www.netlib.org/lapack/

[4] LINPACKについて、
http://www.netlib.org/linpack/

[5] Cell上でのLINPACK性能について、
http://www.ibm.com/developerworks/power/library/pa-cellperf/