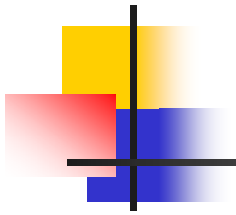


並列アルゴリズム

2005年後期　火曜　2限

青柳　睦

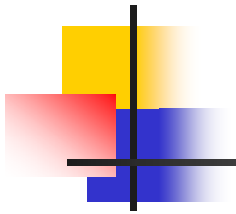
Aoyagi@cc.kyushu-u.ac.jp

http://server-500.cc.kyushu-u.ac.jp/

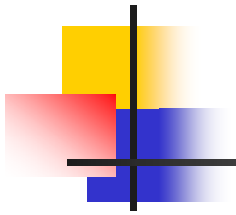
11月8日（火）

5. MPI の基礎

6. 並列処理の性能評価



もくじ



成績評価

出席点５割，レポート５割

aoyagi@cc.kyushu-u.ac.jpへメール

Subject: 並列アルゴリズム

学籍番号，氏名，専攻，

座席番号(A-1,A-2,・・C-3など)

5. MPI の基礎　　5.1　MPI send と recv

int MPI_Send(　/* 機能：送信バッファのデータを特定の受信先に送信する．*/

void*	message	/* 送信バッファの開始アドレス(IN) */,
int	count	/* データの要素数(IN) */,
MPI_Datatype	datatype	/* データタイプ(IN) */,
int	dest	/* 送信先(IN) */,
int	tag	/* メッセージ・タグ(IN) */,
MPI_Comm	comm	/* コミュニケータ(IN) */)

int MPI_Recv(　/*　機能：要求されたデータを受信バッファから取り出す．
またそれが可能になるまで待つ．　*/

void*	message	/* 受信バッファの開始アドレス(OUT) */,
int	count	/*データの要素数(IN) */,
MPI_Datatype	datatype	/*データタイプ(IN) */,
int	source	/* 送信元(IN) */,
int	tag	/* メッセージ・タグ(IN) */,
MPI_Comm	comm	/*コミュニケータ(IN) */,
MPI_Status*	status	/* ステータス(OUT) */)

定義済みMPIデータ型

MPIデータ型	Cデータ型
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

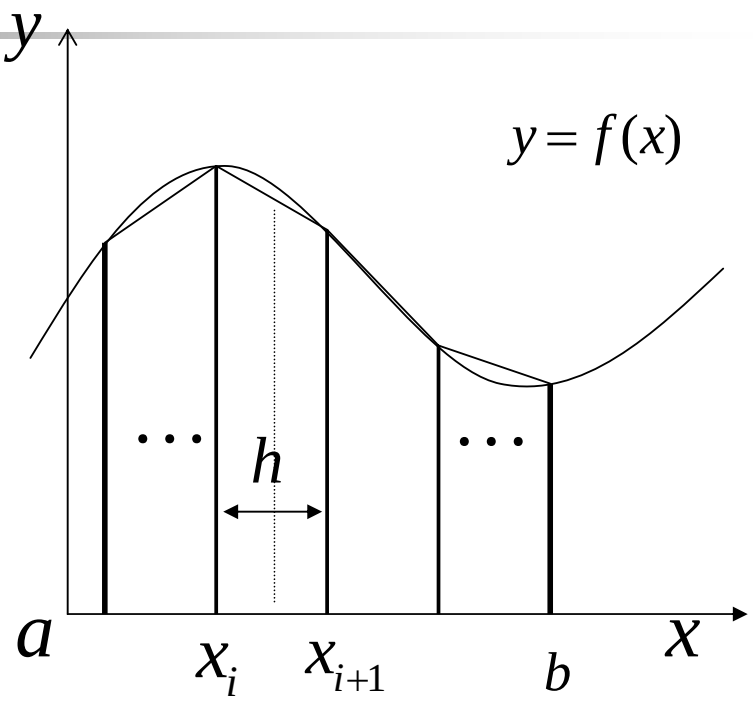
定義済みMPIデータ型：ポータビリティを高めるために，MPI によって前もって定義されたデータ型である．例えば，int型であればMPI_INT というハンドルを用いる．BYTE　とPACKED以外はC言語の型と対応している．プログラマが，新たなデータ型を定義することも可能である．

5.2 プログラム例 積分の台形公式

端点と刻幅(h)

$$a = x_0, b = x_n, h = (b-a)/n$$

$$x_i = a + ih \quad (i = 0,1,2\cdots,n)$$



（図）非負関数の定積分と台形公式

積分をn個の台形面積の和で近似

$$\int_a^b f(x)dx \simeq \sum_{i=0}^{n-1} \frac{1}{2} h[f(x_i) + f(x_{i+1})]$$

$$= \sum_{i=0}^{n-1} h f(x_i + \frac{h}{2}) \quad \text{または素直に,}$$

$$= [\frac{f(x_0)}{2} + \frac{f(x_n)}{2} + f(x_1) + f(x_2) + \cdots f(x_{n-1})]h$$

【例題】

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

を用いて，πを数値計算で求める

【証明】

$\int_0^1 \frac{4}{1+x^2} dx$　変数変換 $x = \tan\theta$により，

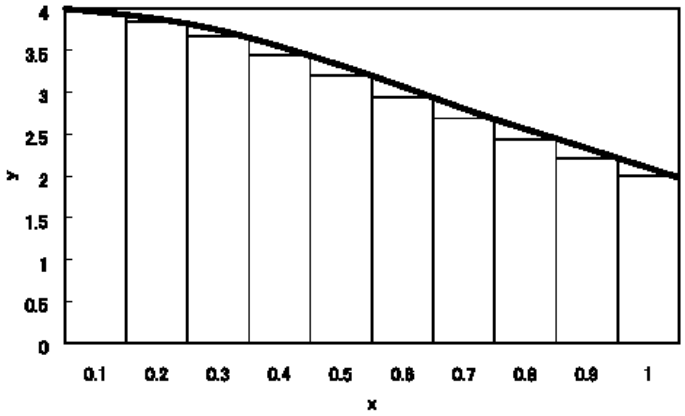
$dx = \frac{d\theta}{\cos^2\theta}, [0,1] \to [0, \pi/4]$, だから

$$\int_0^{\pi/4} \frac{4}{1+\tan^2\theta} \frac{d\theta}{\cos^2\theta} = 4\int_0^{\pi/4} d\theta = \pi$$

台形公式を用いたπの計算　逐次プログラム（例）

```
#include <stdio.h>
int main(int argc, char **argv)
{
        int i,loop;
        double width,x,pai=0.0;
        loop = atoi(argv[1]);
        width = 1.0 / loop;

        for(i=0; i<loop; i++){
                x = (i + 0.5) * width;
                pai += 4.0 / (1.0 + x * x);
        }
        pai = pai * width;
        printf("PAI = %f¥n",pai);
        return 0;
}
```

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$



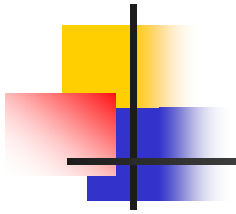
5.3 MPIを用いた並列化　台形公式を用いたπの計算

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>

double PI25DT =
3.14159265358979323846264 3;

int main( int argc, char *argv[])
{
    int i, loop;
    int numprocs, my_rank;
    double pai, width, sum, x;
    int source, dest , tag = 0;
    int local_loop;

    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,
        &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,
        &my_rank);
```

```
    if (my_rank == 0) {
        loop=atoi(argv[1]);
        for (dest = 1; dest < numprocs; dest++) {
            MPI_Send(&loop, 1, MPI_INT, dest, tag,
                MPI_COMM_WORLD);
        }
    } else {
            source=0;
            MPI_Recv(&loop, 1, MPI_INT, source, tag,
                MPI_COMM_WORLD, &status);
    }

    width = 1.0 / loop;
    local_loop = loop / numprocs;

    sum = 0.0;
    for (i = my_rank*local_loop;
        i < (my_rank+1)*local_loop; i++) {
        x = (i + 0.5) * width;
        sum += 4.0 / (1.0 + x*x);
    }
```

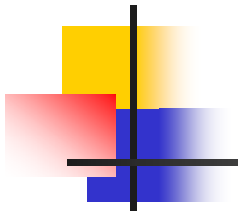
MPIを用いた並列化（続き）　台形公式を用いたπの計算

```
    tag=1;
    if (my_rank == 0) {
        pai=sum;
        for (source = 1; source < numprocs; source++) {
            MPI_Recv(&sum, 1, MPI_DOUBLE, source,
                tag, MPI_COMM_WORLD, &status);
            pai=pai+sum;
        }
    } else {
        dest=0;
        MPI_Send(&sum, 1, MPI_DOUBLE, dest, tag,
            MPI_COMM_WORLD);
    }

    pai = pai * width;

    if (my_rank == 0) {
        printf("PAI =%.16f¥n",pai);
        printf("Error = %.16f¥n", fabs(pai - PI25DT));
    }

    MPI_Finalize();
    return 0;
}
```



5.4 並列化の効果

- 何故並列化するか？
  - 高速に実行したい！
    - メモリをたくさん使いたいという理由で並列化する場合もある。
- 本当に速くなったか？
  - 計測してみないとわからない．⇒　６．性能評価



5.5 MPIプログラムの実行

MPIプログラムの翻訳方法、実行方法は計算機によって異なる。

MPICH の場合:

MPIプログラム(C言語) の翻訳コマンド　mpicc

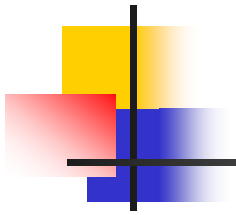
例)

% mpicc test.c –o test

MPIプログラムの実行コマンド　mpirun

例)

% mpirun –np 8 ./test



並列プログラムの実行時間

プログラムの評価に用いる時間は二通り

- CPU使用時間： CPUが働いた時間.
- 経過時間： 計算機の動作にかかわらず，消費した時間.

計算が主体のプログラムでは，

CPU使用時間 ≒ 経過時間

だが，CPU以外の装置（ディスク，ネットワーク等）を使用している時間が長いプログラムでは，その間CPUは待機するので

CPU使用時間 ＜ 経過時間

となる.

並列プログラムは通信時間による影響も評価の対象となるので，通常は経過時間を用いる.

実行環境　PCクラスタ

プロセッサ	Intel Itanium2 900MHz
メモリ	512MB
OS	RedHat Linux Advanced Server
コンパイラ	Intel C compiler 7.1
ノード数	16
ネットワーク	Myrinet2000 PCI64B (GM-1.6.4)
ライブラリ	MPICH-GM 1.2.5.10

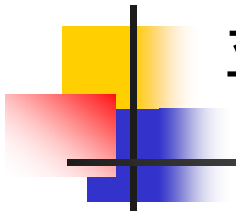
MPIプログラムの実行時間 （台形公式プログラム）

Loop = 1000000000

使用プロセッサ数	経過時間（秒）
1	11.41
2	5.94
4	3.26
8	2.20
16	1.95

Loop = 1000

使用プロセッサ数	経過時間（秒）
1	0.28
2	0.37
4	0.47
8	0.78
16	1.30

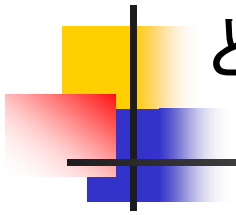


並列化すればよいというものではない

並列プログラムの作成は非並列プログラムより難しい.

- 問題解決のアルゴリズム以外に少なくとも処理の分割方法を考慮する必要がある.
- MPIの場合，さらにデータの分割方法や通信のタイミングも重要である.

苦労に見合うだけの結果が得られるかどうかを事前に検討する.



どんなプログラムでも並列化可能というわけではない

並列化とは，複数の処理を同時に進行させることであるので，実行の順序が非並列の場合と異なる．そのため，実行順序によって値が変わる処理は並列化できない．

並列化できないループの例：

```
for (i = 0; i < N; i++){
      a[i] = f(a[i-1]);
}
```

再帰参照

6. 並列処理の性能指標 （1）

◆ 所要時間（Turn Around Time）

最初に実行を開始したプロセスの開始時刻から、最後に実行を終了したプロセスの終了時刻までを計測し，所要時間とする.

```
MPI_Barrier(MPI_COMM_WORLD);
  t_start = MPI_Wtime();
    /*この間に所要時間を測定する処理を記述*/
  MPI_Barrier(MPI_COMM_WORLD);
  t_stop = MPI_Wtime();

printf("Turn around time = %.16f¥n",
  t_stop - t_start );
```

MPI_Barrier
「バリア同期」と呼ばれ、すべてのプロセスがこれを呼び出すまで各プロセスが待ち合わせる機能を持つ，MPI標準の関数.

並列処理の性能指標 (2)

◆ 高速化率(Speed up ratio)

ある計算を p 台の演算装置で並列処理した場合の所要時間を T(p) とすると，
S(p) = T(1) / T(p) を高速化率と呼ぶ.

理想的には p 台で実行すれば 1 台の p 倍の速さになるはずであるから，S(p) = p となり，このときの高速化を ideal speedup と呼ぶ.

【補足】

原理的には S(p) ≦ p となるはずであるが，キャッシュの実質的な容量増加等が原因で S(p) > p となることがある（superlinear speedupと呼ぶ）．ただし，逐次プログラムのキャッシュチューニングが足りないことを意味している場合もある.

共有メモリ構成の場合には、キャッシュとメモリの間のスループットが実質的に減少するので，スーパーリニアー効果は得にくい。

◆ 高速化率(Speed up ratio)のp依存性

並列処理の効果を示すとき，最もよく用いられるのは高速化率のグラフである．横軸にプロセッサ数 p を取り，縦軸に高速化率 S(p) を，それぞれリニアスケールで取り，さらに，実際の高速化率とともに ideal speedup を表す直線を示すのが通例．

【例】

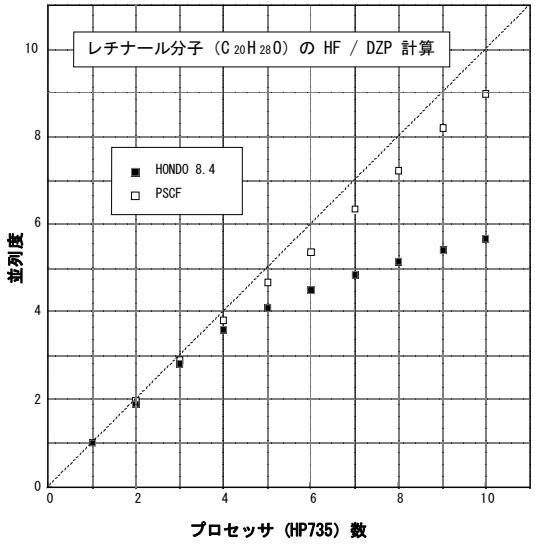
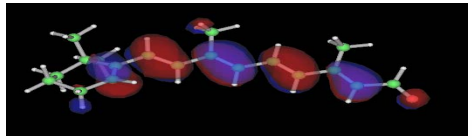
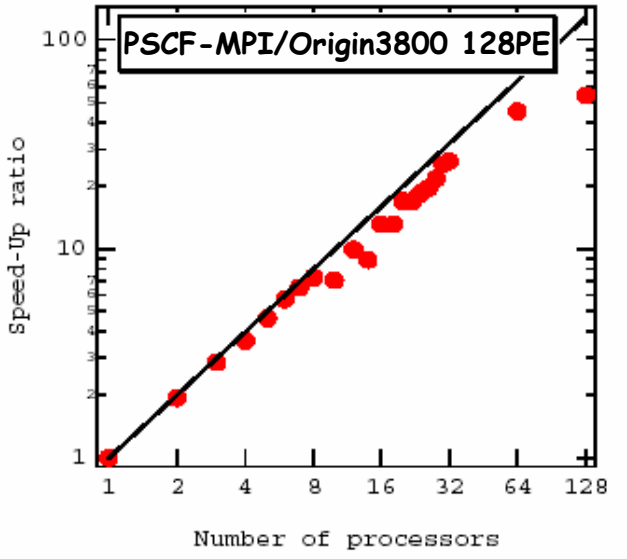
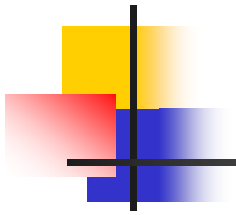


図-4　並列分子軌道計算プログラムの比較　並列度







並列処理の性能指標 (3)

◆ 並列化効率

並列化効率 E(p) は，E(p) = S(p) / pで定義され，Ideal speedup の場合には E(p) = 1，superlinear speedup の場合には E(p) > 1 となる.

並列処理の性能指標 (4)

◆ オーバーヘッド時間

ideal speedup の場合の所要時間 T(1)/p に比べて実際の所要時間 T(p) がどれだけ余計にかかっているかを示す**指標として，O(p)=T(p)−T(1)/p (オーバーヘッド時間)を用いることがある.**

O(p)には逐次部分の所要時間，負荷の不均衡，通信オーバーヘッドなどが重なって取り込まれている .

オーバーヘッド時間をpに対してプロットすると，プロセッサ数がさらに大きくなった場合に性能がどのように変化するかを，ある程度予測できる.

並列処理の性能指標 (5)

◆ 問題サイズ(N)との関係

これまでは，演算装置の台数p の関数として，所要時間 T(p) ，高速化率S(p)， 並列化効率 E(p)，オーバーヘッド時間O(p) を定義してきたが，一般にこれらの指標は計算の問題サイズ(N)(ひとつのパラメタとは限らないが)にも依存する.

問題とする計算内容によっては逐次処理のｽｹｰﾗﾋﾞﾘﾃｨｰT(1,N)を基本に，T(p,N) ， S(p,N)などを2(多)次元的に考察する必要が生じる場合もある.

レポート　課題1（11月8日出題）

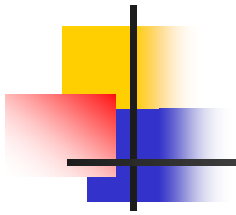
使用CPU数を1から16と変え，並列版台形公式プログラムを実行したところ，以下の結果を得た.

Loop = 1000000000

使用プロセッサ数	経過時間（秒）
1	11.41
2	5.94
4	3.26
8	2.20
16	1.95

所要時間，高速化率，並列化効率，オーバーヘッド時間についてグラフを用いて解析せよ.

提出〆切　11月22（火），メール
Subject: 並列アルゴリズム課題1
添付書類，書式：ワード書類またはExcelまたはPS，PDF.



休講のお知らせ

11月15日（火）休講
国際会議Supercomputing2005@シアトル出張の為休講
（補講については追って連絡致します）

11月22日（火）休講
理学部休講の為