

MPI.NETではじめる C#並列プログラミング演習

同志社大学大学院 知識工学専攻

博士後期課程2年 中尾昌広

2009年2月27日



内容と目的

- MPIとは
- MPIの用語・関数の説明
- MPI.NETとC#の実習
- MPI Version2についての説明

この講座の目的は、並列プログラミングの概念とそのプログラミング方法を学習して頂くことです。

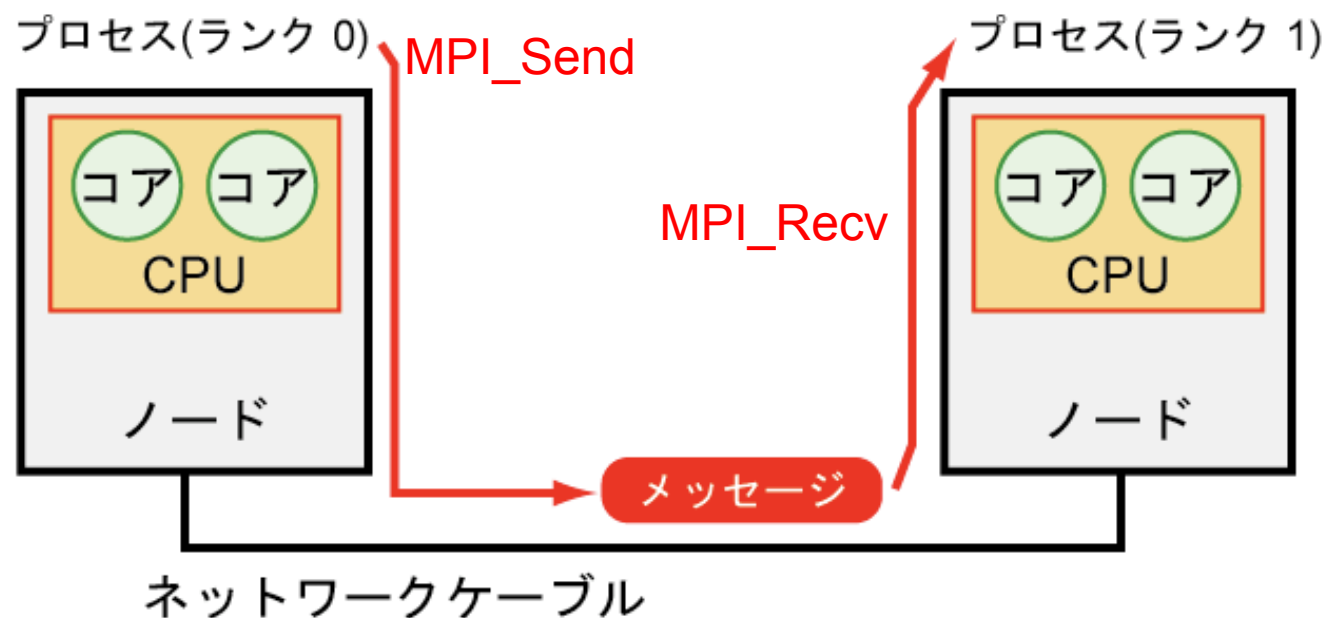


MPI (Message Passing Interface)とは

- MPIとは、分散メモリ型の並列計算機で、複数のプロセス間でデータのやりとりを行うための標準仕様
- MPI Version 1と呼ばれる仕様が1994年に策定された
- 並列計算でよく使う通信パターンを提供
 - 1対1通信
 - 1対多通信
 - 多対多通信

並列処理の概念図

- すべてのプロセスが1つのプログラムを実行
- 各プロセスは自分のID(ランク)を参照して、自分の担当処理部分を実行する
- 各プロセスが別のプロセスと通信したい時は、MPIで定義された通信関数を呼び出す



```
if(rank == 0){  
    // ランク0の処理  
    MPI_Send();  
} else if(rank == 1){  
    // ランク1の処理  
    MPI_Recv();  
} else if(rank == 2){  
    .  
    .
```



MPI Version2とは

- MPI-1が広く用いられるようになった後、いくつか仕様を拡張したい要求が出てきた
- 1997年にMPI Version2が策定
- MPI Version 1に下記の追加機能を持たせたもの
 - 並列ファイル入出力
 - 動的プロセス管理
 - リモートメモリアクセス
 - その他、細かな機能追加 (スレッドなど)

本講義の最後に、上記2つの機能について紹介します。



MPIの実装

- MPIは規格であるので、MPIに従った実装が数多く存在する

実装名	MPI.NET	MPICH	MPICH2	LAM/MPI	MS-MPI
対応Ver.	2	1	2	1	2
対応言語	.NET言語 (C#など)	C, C++, fortran			

今回はMPI.NETとC#について紹介します。



MPI.NETとは

- .NET Framework上で動作する並列計算用ライブラリ
- .NETで用いられている言語の全て(特にC#)をサポート
- ノード間の通信を簡易に行えるAPIを提供



C#とは

- Microsoft社が開発したプログラミング言語
- .NET環境の中心的言語
- javaに似たオブジェクト指向型言語であり、
プロセッサに依存しない実行ファイルを生成可能
- 他の.NET言語(Visual Basic .NETやVisual C++)と相互に
連携可能。他言語で記述されたクラスを継承することも、
その逆も可能



C#のプログラミング例

```
class Helloworld           // クラスの宣言
{
    public static void Main() // メイン関数
    {
        // コンソールにメッセージを出力
        System.Console.Write("Hello World\n");
    }
}
```

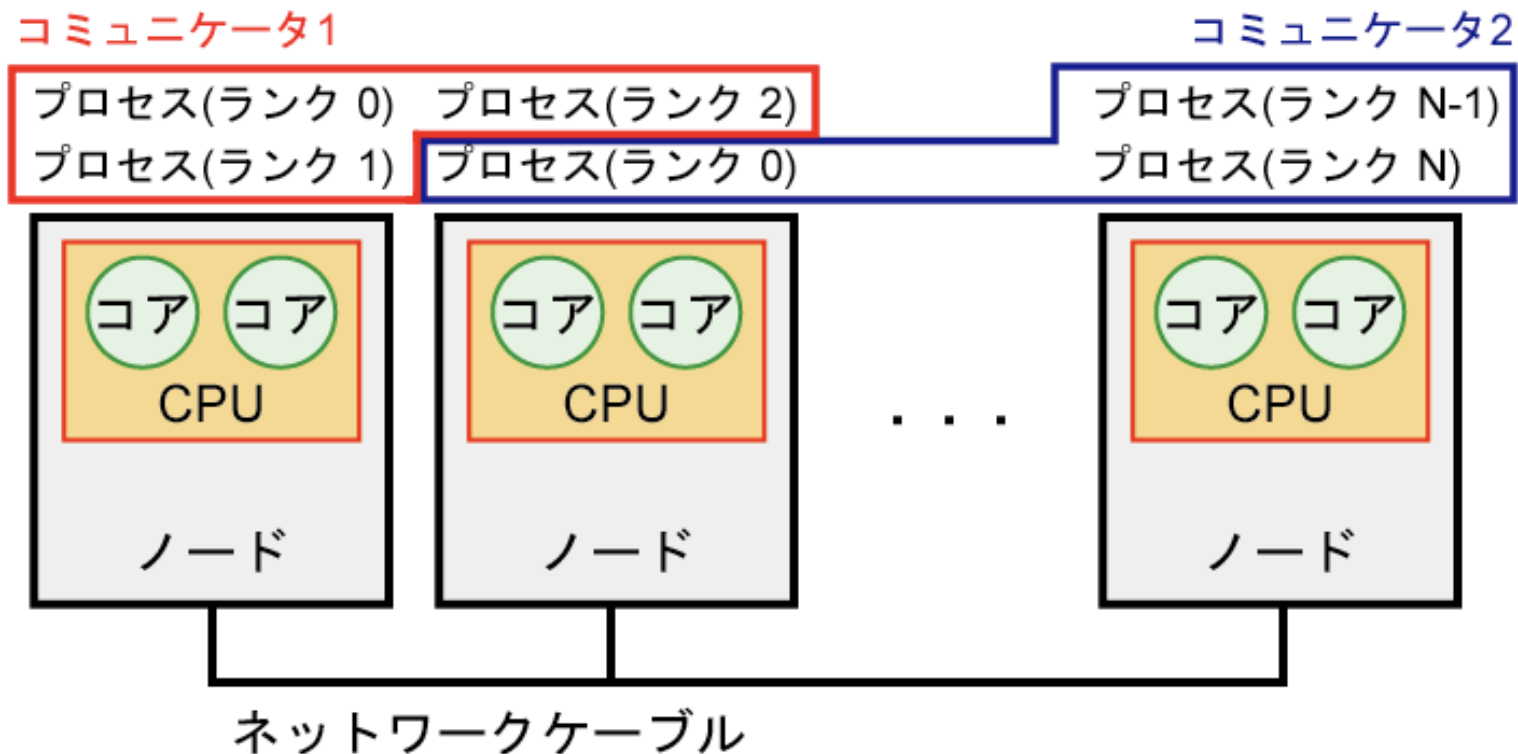
出力結果

```
C:\WINDOWS\system32\cmd.exe
Hello World
続行するには何かキーを押してください . . . . .
```



MPIプログラムの用語

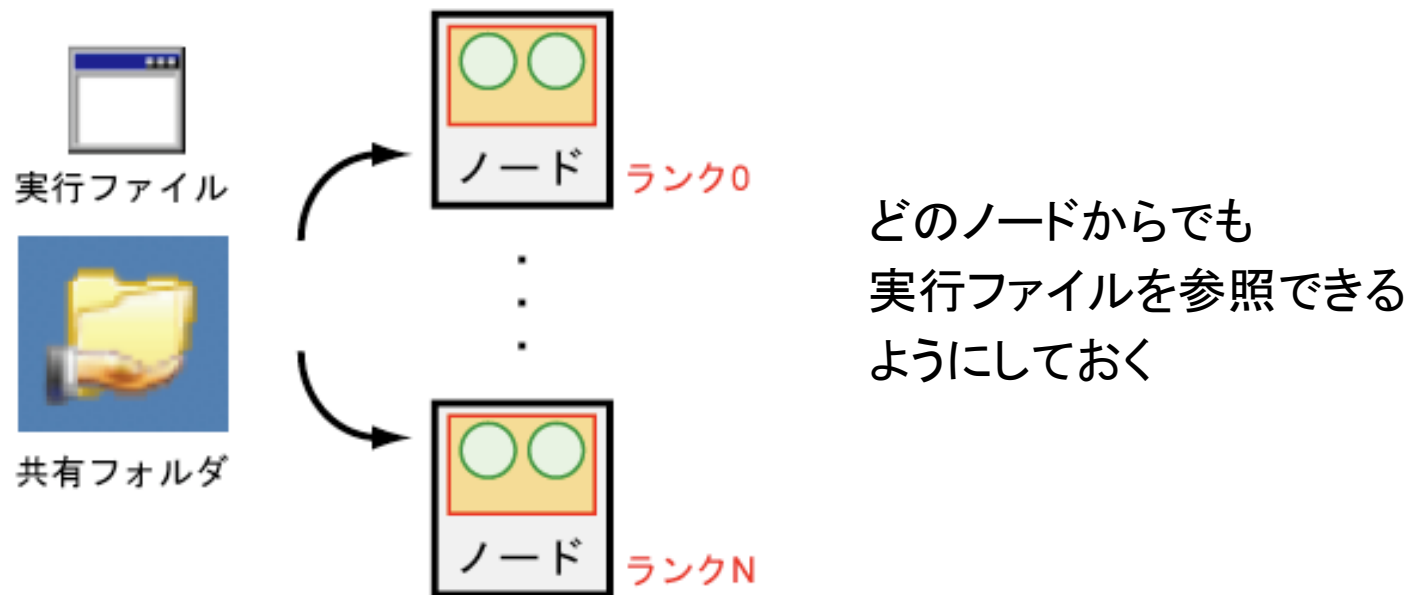
- プロセス:プログラムの実行単位。それぞれのプロセスは独立したメモリ空間を持つ
- ランク :各プロセスに付けられた名前(プロセスID)
- コミュニケータ:通信グループの単位

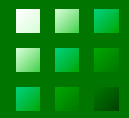




MPIプログラムの実行の流れ

1. Visual Studioなどを用いて、実行ファイルを作成
2. 実行ファイルを共有フォルダに保存
3. PowerShell もしくは MS-DOSを用いて実行



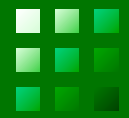


基本的な関数の説明(Send)

public void Send<T>(value, dest, tag)

- value: 送信したい値
- dest : 送信先(ランクを指定)
- tag : データ識別用タグ(int型の整数)

```
if (comm.Rank == 0)
{
    string value = "Windows";
    comm.Send(value, 1, 9);
}
```



基本的な関数の説明(Receive)

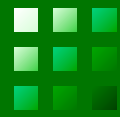
```
public T Receive<T>(source, tag)
```

- source: 送信元(ランクを指定)
- tag : データ識別用タグ(int型の整数)
返り値が受信されたデータになる

```
if (comm.Rank == 0)
{
    string value = "Windows";
    comm.Send(value, 1, 9);
}
```

```
else if(comm.Rank == 1)
{
    string msg =
    comm.Receive<string>(0, 9);

    Console.Write(msg);
}
```



Tagについて

```
public T Receive<T>(source, tag)
```

- source: 送信元 (ランクを指定)
- tag : データ識別用タグ (int型の整数)
返り値が受信されたデータになる

メッセージの種類を表す整数。

あるプロセスからあるプロセスにメッセージを送る関数が複数ある場合、プロセス同士の整合性を取る必要がある。

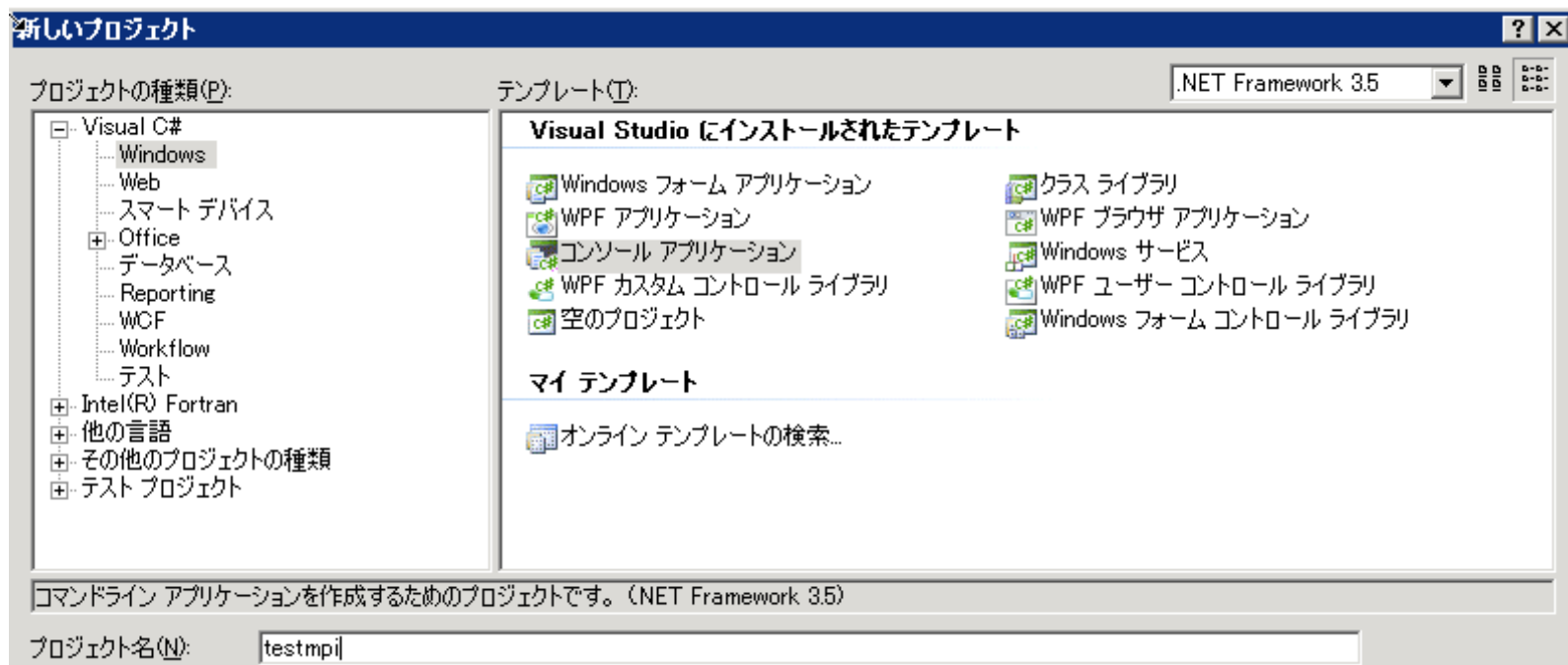


実習1: HelloWorld

- Visual Studioを使って、Hello Worldを表示させるプログラムを作成する。
- プロセス間の通信は行わずに、指定したプロセスだけ、並列にHello Worldを表示させます。

プログラムの作成準備(1/2)

- Visual Studio 2008を起動
- 「ファイル」->「新規作成」->「プロジェクト」->「コンソールアプリケーション」を選択



プロジェクト名は適当な名前を入力して下さい。

プログラムの作成準備(2/2)

- ソリューションエクスプローラの参照設定を右クリック
- 「参照の追加」で「Message Passing Interface」を選択

The screenshot shows the Visual Studio interface. On the left, the Solution Explorer displays the project 'testmpi' with a context menu open over the 'References' (参照設定) folder. The menu options are 'Add Reference...' (参照の追加(R)...), 'Add Service Reference...' (サービス参照の追加(S)...), and 'Remove Reference...' (参照の削除(R)...). The 'Add Reference...' option is selected.

The 'Add Reference' dialog box is open, showing a list of .NET components. The 'Message Passing Interface' component is selected. The dialog box has tabs for '.NET', 'COM', 'プロジェクト', '参照', and '最近使用したファイル'. The table below shows the list of components:

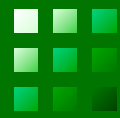
コンポーネント名	バージョン	ランタイム	パス
IEExecRemote	2.0.0.0	v2.0.50727	c:#WINDOWS#Microsoft.NE...
IEHost	2.0.0.0	v2.0.50727	c:#WINDOWS#Microsoft.NE...
IEHost	2.0.0.0	v2.0.50727	c:#WINDOWS#Microsoft.NE...
InfoPath.DomainControl	11.0.0.0	v1.1.4322	C:#Program Files#Microsoft...
InfoPath.DomainControl	11.0.0.0	v1.1.4322	C:#Program Files#Microsoft...
ISymWrapper	2.0.0.0	v2.0.50727	c:#WINDOWS#Microsoft.NE...
Message Passing Interface	1.0.0.0	v2.0.50727	C:#Program Files#MPINET#...
Microsoft SQL Mobile	9.0.242.0	v2.0.50727	C:#Program Files#Microsoft...
Microsoft.AnalysisServices...	9.0.242.0	v2.0.50727	c:#Program Files#Microsoft...
Microsoft.Build.Conversion	2.0.0.0	v2.0.50727	c:#WINDOWS#Microsoft.NE...
Microsoft.Build.Conversion.v...	3.5.0.0	v2.0.50727	C:#Program Files#Referenc...
Microsoft.Build.Engine	2.0.0.0	v2.0.50727	c:#WINDOWS#Microsoft.NE...
Microsoft.Build.Engine	3.5.0.0	v2.0.50727	C:#Program Files#Referenc...
Microsoft.Build.Framework	2.0.0.0	v2.0.50727	c:#WINDOWS#Microsoft.NE...
Microsoft.Build.Framework	3.5.0.0	v2.0.50727	C:#Program Files#Referenc...



HelloWorld

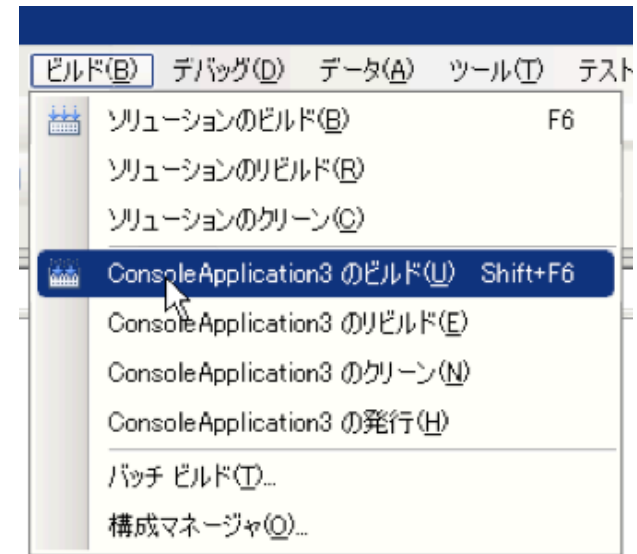
```
using System;
using MPI;

class MPIHello
{
    static void Main(string[] args)
    {
        // MPI.Environmentのインスタンス作成 ( MPIの初期化処理 )
        using (new MPI.Environment(ref args))
        {
            Console.WriteLine("Hello World from rank " + Communicator.world.Rank
                + " (running on " + MPI.Environment.ProcessorName + ")");
        }
    }
}
```



コンパイル + 実行

5. 「ビルド」->
(プロジェクト名のビルド)



6. できあがった実行ファイルを共有フォルダにコピーする

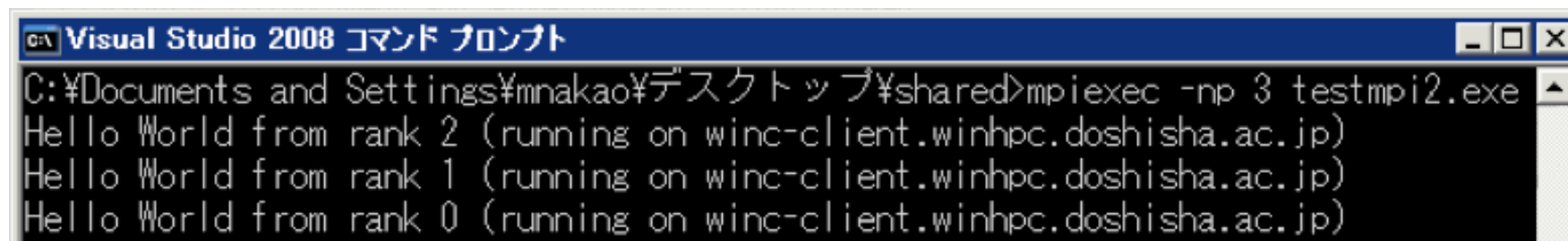
実行ファイルの場所は、

C:\Documents and Settings\<(アカウント名)\My Documents
\Visual Studio 2008\Projects\<(プロジェクト名)\(プロジェクト名)
\bin\Debug

コンパイル + 実行

7-1. 実行方法 (MS-DOSの場合)

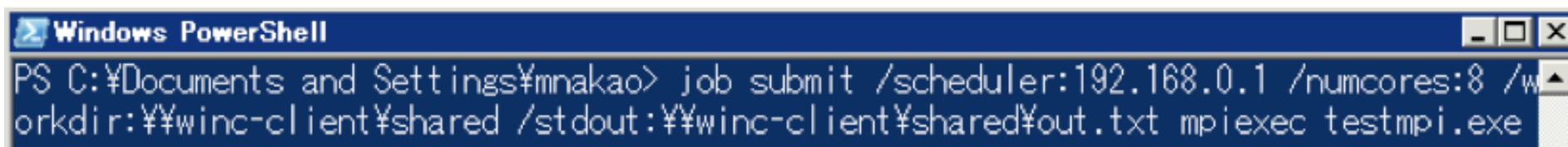
> mpiexec -np (プロセス数)(実行ファイル)



```
Visual Studio 2008 コマンド プロンプト
C:\Documents and Settings\mnakao\Desktop\shared>mpiexec -np 3 testmpi2.exe
Hello World from rank 2 (running on winc-client.winhpc.doshisha.ac.jp)
Hello World from rank 1 (running on winc-client.winhpc.doshisha.ac.jp)
Hello World from rank 0 (running on winc-client.winhpc.doshisha.ac.jp)
```

7-2. 実行方法 (PowerShellの場合)

> job submit /scheduler:(ジョブスケジューラ) /numcores:(プロセス数)
/workdir:(実行ファイルの場所) /stdout:(出力ファイルの場所)
mpiexec (実行ファイル)



```
Windows PowerShell
PS C:\Documents and Settings\mnakao> job submit /scheduler:192.168.0.1 /numcores:8 /workdir:¥¥winc-client¥shared /stdout:¥¥winc-client¥shared¥out.txt mpiexec testmpi.exe
```



確認すること

実行結果が

```
Hello World from rank 番号(running on マシン名)
・
・
```

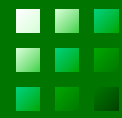
となり、指定したプロセス数だけ出力されること。



実習2:1対1通信

- 2プロセス間で文字列の送受信を行う
- 各プロセスで動作を変える
- 通信関数には SendとReceiveを用いる

もう一度、「ファイル」->「新規作成」で、別名の
コンソールプロジェクトを作成して下さい。

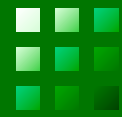


Point-to-Point Communication(1/2)

HelloWorldという文字列をランク0がランク1に送る

```
using System;
using MPI;           // MPI環境のための名前空間

class PtoPHello
{
    static void Main(string[] args)
    { // MPIインスタンスの作成(環境の初期化)
        using (new MPI.Environment(ref args))
        {
            (次ページの内容をここに書く)
        }
    }
}
```



Point-to-Point Communication(2/2)

// コミュニケーターの宣言(ランクの取得など)

```
Intracommunicator comm = Communicator.world;
```

```
int tag = 9; // 今回は適当な数字を代入
```

```
if (comm.Rank == 0)
```

```
{
```

```
    comm.Send("Hello World", 1, tag); // ランク1に送信
```

```
}
```

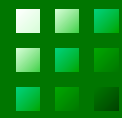
```
else if(comm.Rank == 1)
```

```
{
```

```
    string msg = comm.Receive<string>(0, tag); // ランク0から受信
```

```
    Console.WriteLine(msg + " from rank " + comm.Rank);
```

```
}
```

確認すること

2プロセスで実行し、その結果が

```
Hello World from rank 1
```

となること。



集合通信

例えばあるプロセスが複数のプロセスにデータを送りたい場合、

```
for(i=0; i<num; i++){  
    comm.send("Hello World", i, tag);  
}
```

この書き方だとコードが複雑化する。

MPIではone-to-all, all-to-one, all-to-allの関数(集合通信)が定義されている。

一般的に集合通信を用いた方がパフォーマンスも高い。

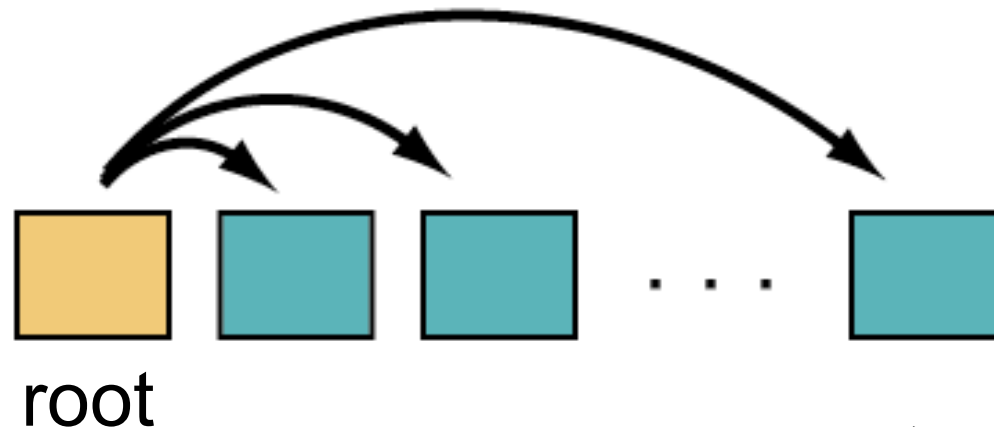
(例えば二分木で送信が可能)

集合通信の例：one to all

```
public void Broadcast<T>(ref T value, root)
```

1つのプロセスから全てのプロセスにデータを
送信するための関数

- ref T value: 送信する値
- root: 送信するランク



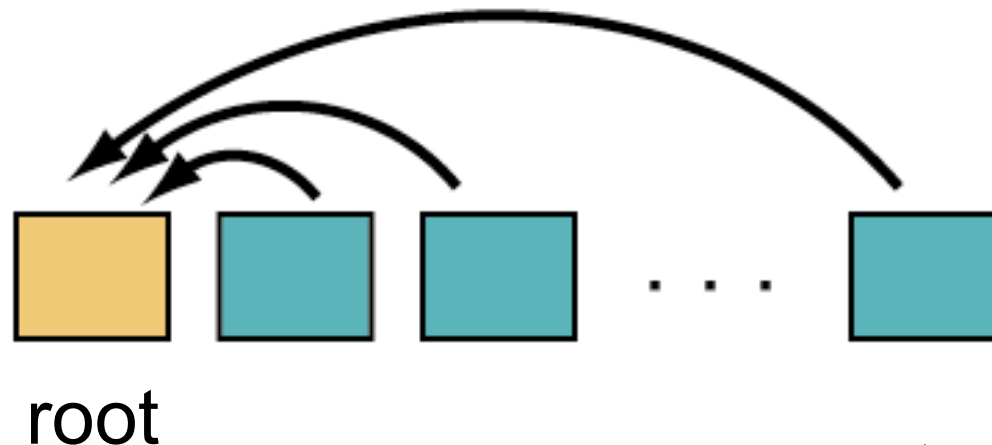


集合通信の例：all to one

```
public T[] Gather<T>(value, root)
```

あるプロセスに全てのプロセスからデータを受信するための関数

- value: 集めるデータ
- root : 受信するランク
- 戻り値: 集めた値の配列



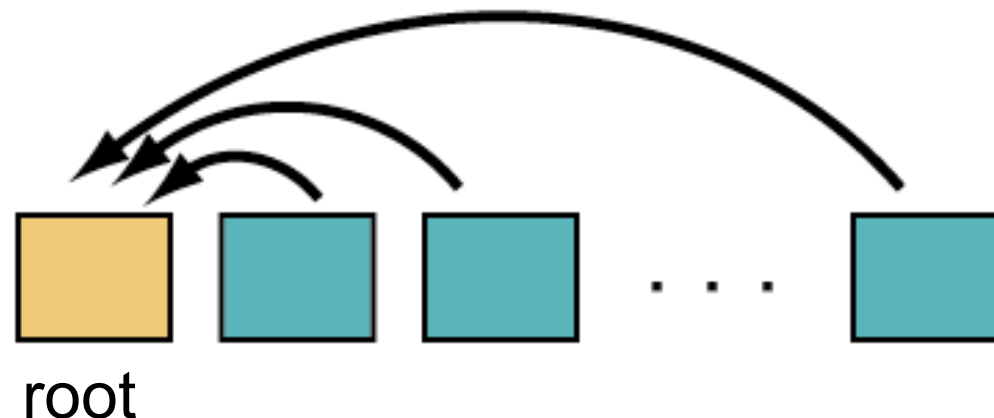
集合通信の例：Reduction(集約)

```
public T Reduce<T>(value, Operation, root)
```

あるプロセスに全てのプロセスからデータを受信するための関数。受信時に算術演算を行える。並列計算においてよく使う計算を提供している。

- value: 送信する値
- Operation: 算術演算(自分でも定義できる)
- root: 受信するランク

受信するデータ群の
最大値、合計値などを
自動的に計算

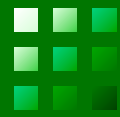




実習3: 集合通信

- Broadcastで各プロセスに異なる値を送信する
- 各プロセスの値をReduceし、合計値・最大値・最小値を出力する。

もう一度、「ファイル」->「新規作成」で、別名のコンソールプロジェクトを作成して下さい。



Collective Communication(1/2)

```
using System;
using MPI;           // MPI環境のための名前空間

class Collective
{
    // 合計値を求める関数を定義する
    public static int AddInts(int x, int y) { return x + y; }

    static void Main(string[] args)
    { // MPIインスタンスの作成(環境の初期化)
      using (new MPI.Environment(ref args))
      {
          (次ページの内容をここに書く)
      }
    }
}
```



Collective Communication(2/2)

```
Intracommunicator comm = Communicator.world;
int msg = 0;

if (comm.Rank == 0)
    msg = 5;

comm.Broadcast(ref msg, 0); // ランク0がmsgを全プロセスに送信する

msg = comm.Rank * msg; // 各プロセスでランク * 5 の計算をする

Console.WriteLine(msg + " from rank " + comm.Rank); // 現時点の情報を出力

int sum = comm.Reduce(msg, AddInts, 0); // 合計値をランク0に送信
int sum2 = comm.Reduce(msg, Operation<int>.Add, 0); // 合計値をランク0に送信
int max = comm.Reduce(msg, Operation<int>.Max, 0); // 最大値をランク0に送信
int min = comm.Reduce(msg, Operation<int>.Min, 0); // 最小値をランク0に送信

if (comm.Rank == 0)
    Console.WriteLine("Sum : " + sum + " Sum : " + sum2 +
        " Max : " + max + " Min : " + min); // Reduceされた情報を出力
```




確認すること

適当なプロセスで実行し、その結果が正しいこと。

実行例(5プロセスの場合)

```
Visual Studio 2008 コマンド プロンプト
C:\Documents and Settings\mnakao\Desktop\shared>mpiexec -np 5 testmpi2.exe
0 from rank 0
20 from rank 4
10 from rank 2
15 from rank 3
5 from rank 1
Sum : 50 Sum : 50 Max : 20 Min : 0
```



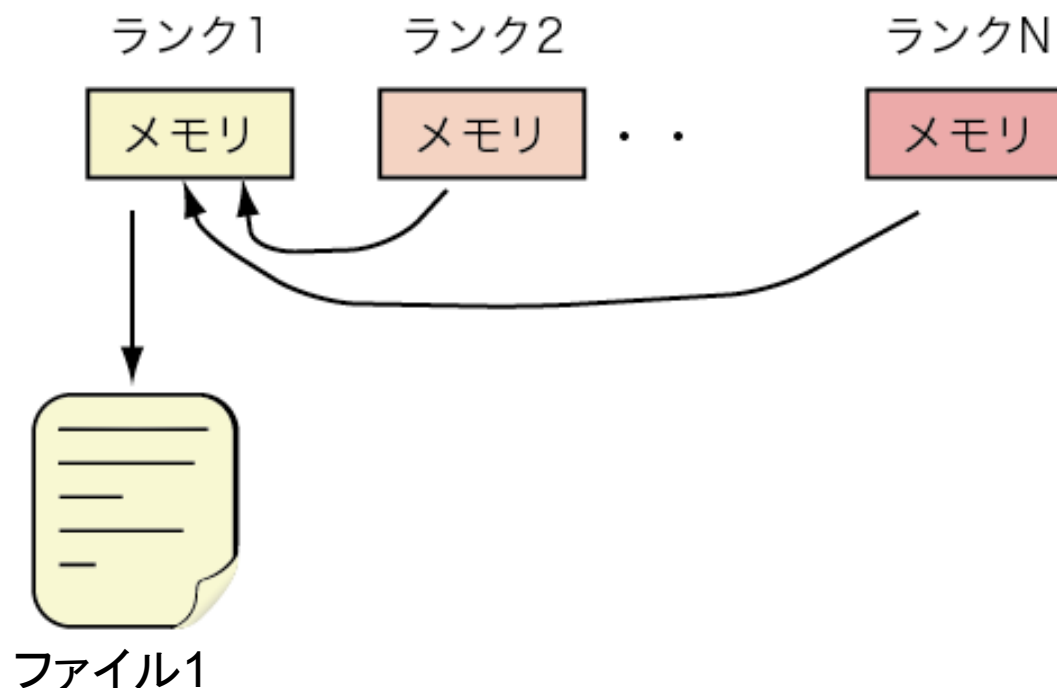
MPI Version 2の説明

- 並列ファイル入出力
- 動的プロセス管理
- リモートメモリアクセス
- その他(スレッドなど)

MPI.NETは現時点では並列ファイル入出力、動的プロセス管理の機能には対応していません。

■ 並列ファイル入出力

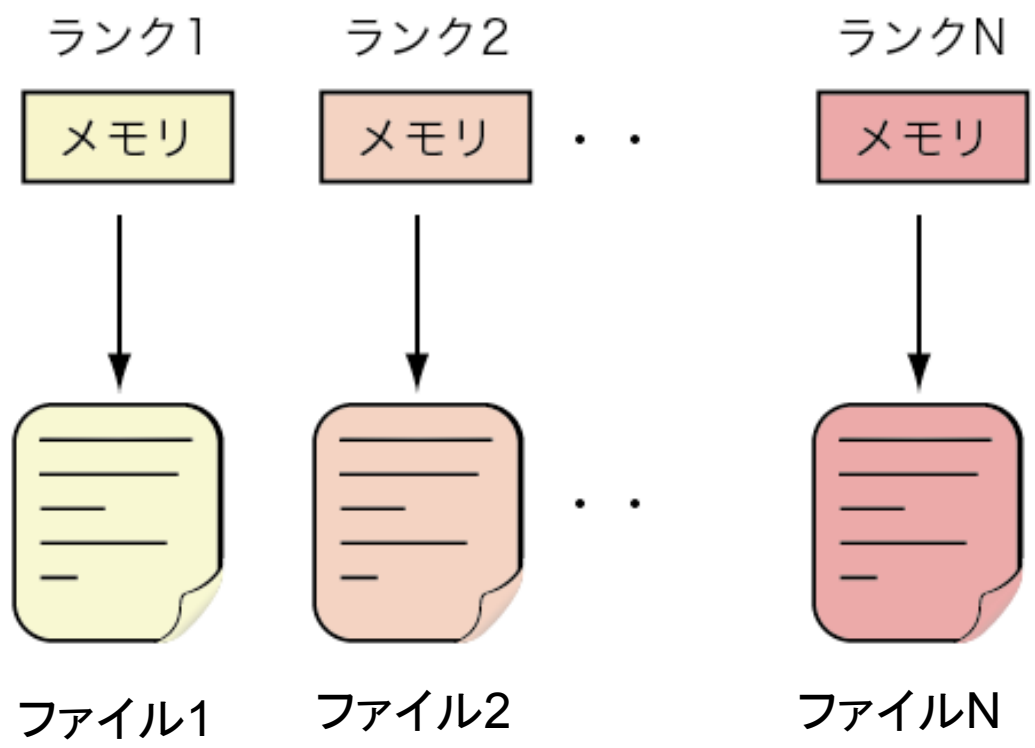
- MPI Version 1でファイルのI/Oをする場合



MPI_Gatherなどを用いて、
全プロセスのファイルに書き込みたい内容を1つのプロセスの
メモリに格納し、そのプロセスのみがファイルに書き込む

並列ファイル入出力

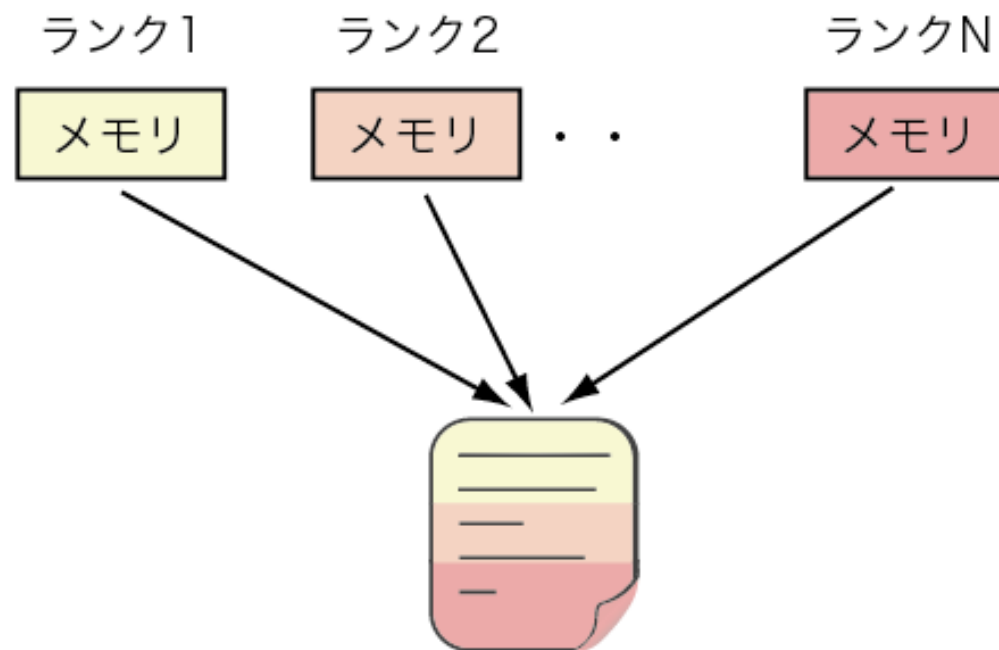
- MPI Version 1でファイルのI/Oをする場合



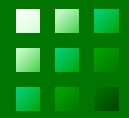
もちろん、各プロセスがファイルの保存場所を変えて、個別にI/Oすることもできるが、管理が大変

並列ファイル入出力

- MPI Version 2では並列にファイルのI/Oを行う機能を提供している



ファイルの操作機能 (open、close、seek、read、write) に対応するMPI関数がそれぞれ存在する。



並列ファイル入出力の流れ

1. MPI_FILE型の変数を宣言する
2. MPI_FILE_openでファイルをオープンする
3. MPI_FILE_set_viewでファイルビューの設定を行う
(各プロセスがファイルのどの部分に書き込むかの指定)
4. MPI_FILE_read/MPI_FILE_writeでファイルの読み書きを行う
5. MPI_FILE_closeでファイルを閉じる

動的プロセス管理

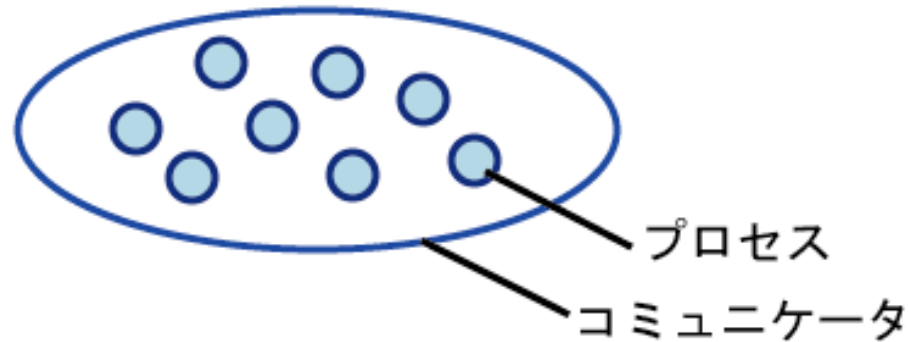
MPI Version 1では、-npなどのオプションで指定するプロセス数は、プログラム終了まで不変

しかし、プログラムの途中で、プロセス数を動的に変化させたいという要求がある

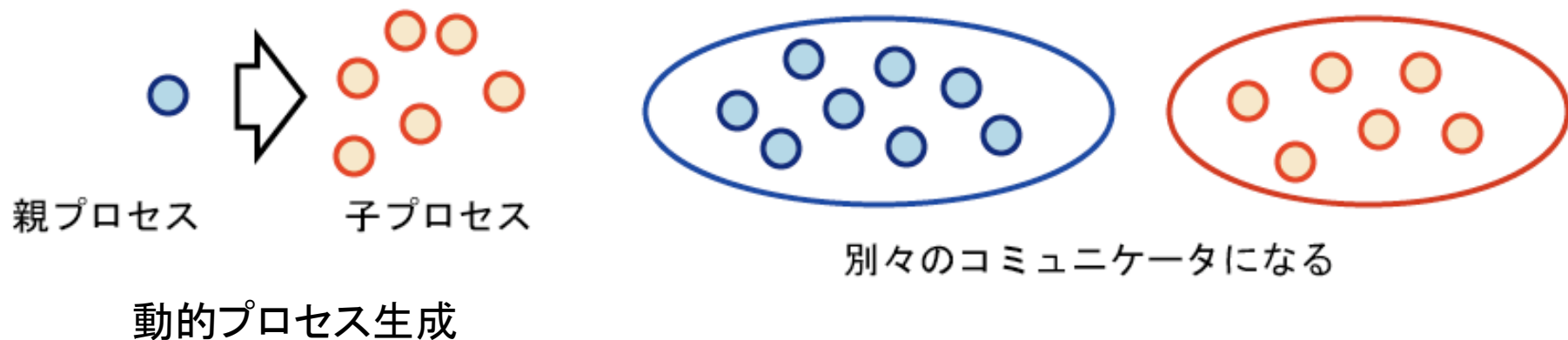
そこで、MPI Version 2からは、各プロセスが別プロセスを自由に生成/削除できるようになった。



コミュニケータとプロセス



あるプロセスがBroadcastする場合、そのコミュニケータ内のすべてのプロセスにデータを送る事ができる。



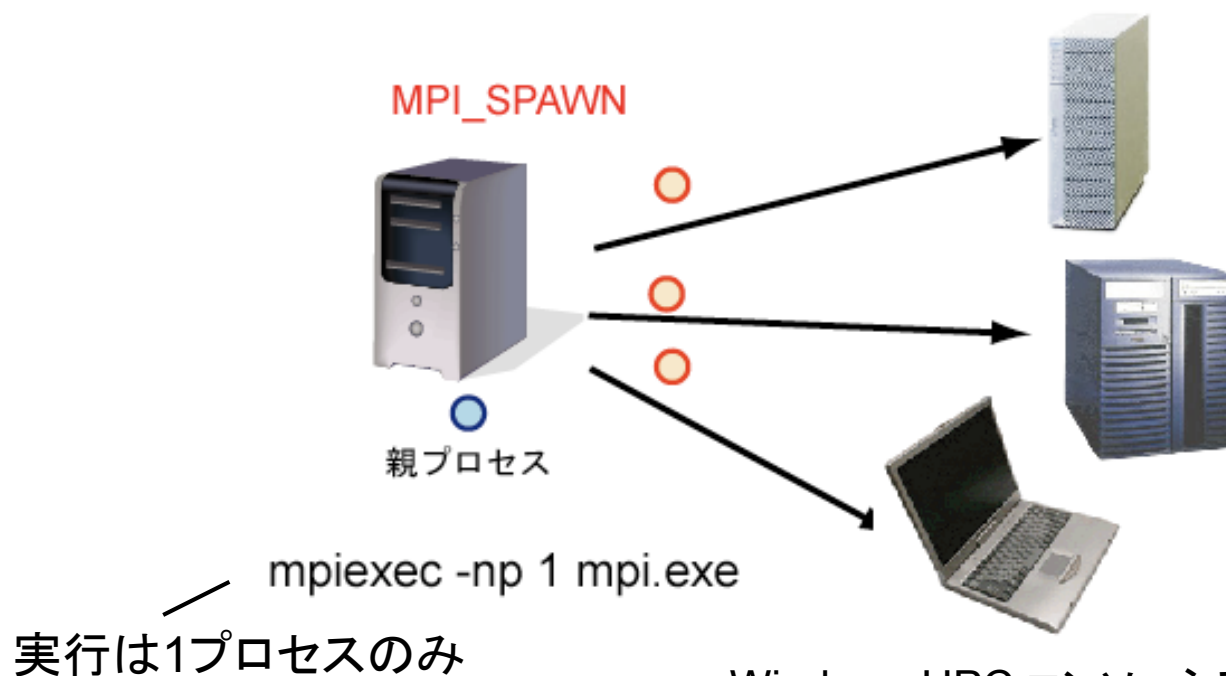


動的プロセス管理の例(1/2)

- ・マスタ・ワーカによる負荷分散プログラムを考える。
- ・スペックの異なる複数のマシンで、大規模計算を行いたい。
- ・しかし、それらのマシンに均等に仕事を割り振ると、遅いマシンに処理が引っ張られることになり、効率が悪い

動的プロセス管理の例(2/2)

- ・常にN個(下図では3個)の子プロセスを保持し、それを計算ノードに投入する
- ・計算ノードから処理した内容が返ってきたら、親プロセスは新しい子プロセスを作成し、再度計算ノードに投入する
- ・これを繰り返すことで、計算ノードのスペックを考慮に入れた、スケジューリングが可能になる





まとめ

- MPIの概要
- C#とMPI.NETのプログラミング例
- MPI version2の話題



参考文献

- MPI.NET(本家)
<http://www.osl.iu.edu/research/mpi.net/>
インストール方法、プログラム例が豊富
- 「実践MPI-2」, Gropp, Lusk, Thakur (訳: 畑崎隆雄)
ピアソン・エデュケーション, 2002
- MPI Version 2
<http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html>
- 東工大・計算数理実践 <http://compview.titech.ac.jp/Members/endot/adv-app-hpc/>
第7回目の講義資料にリモートメモリアクセスの話があります