

## クラウド入門(最終回) － クラウド上でのMPI利用例 －

2012年4月28日

トップエスイー

国立情報学研究所 桑野文洋



## 本授業の位置づけ

- クラウドコンピューティング入門
- Web 三層モデル(1)–(3)
- Map-Reduce(1)–(2)
- MPIの利用例(今回)



## 本授業の内容

- MPIとは
- クラウド上でのMPI実行環境の構築実習
- MPIのサンプルプログラム実習
- モデル検査ツールDiVinE
- DiVinE Cluster(クラウド)の実習
- まとめ



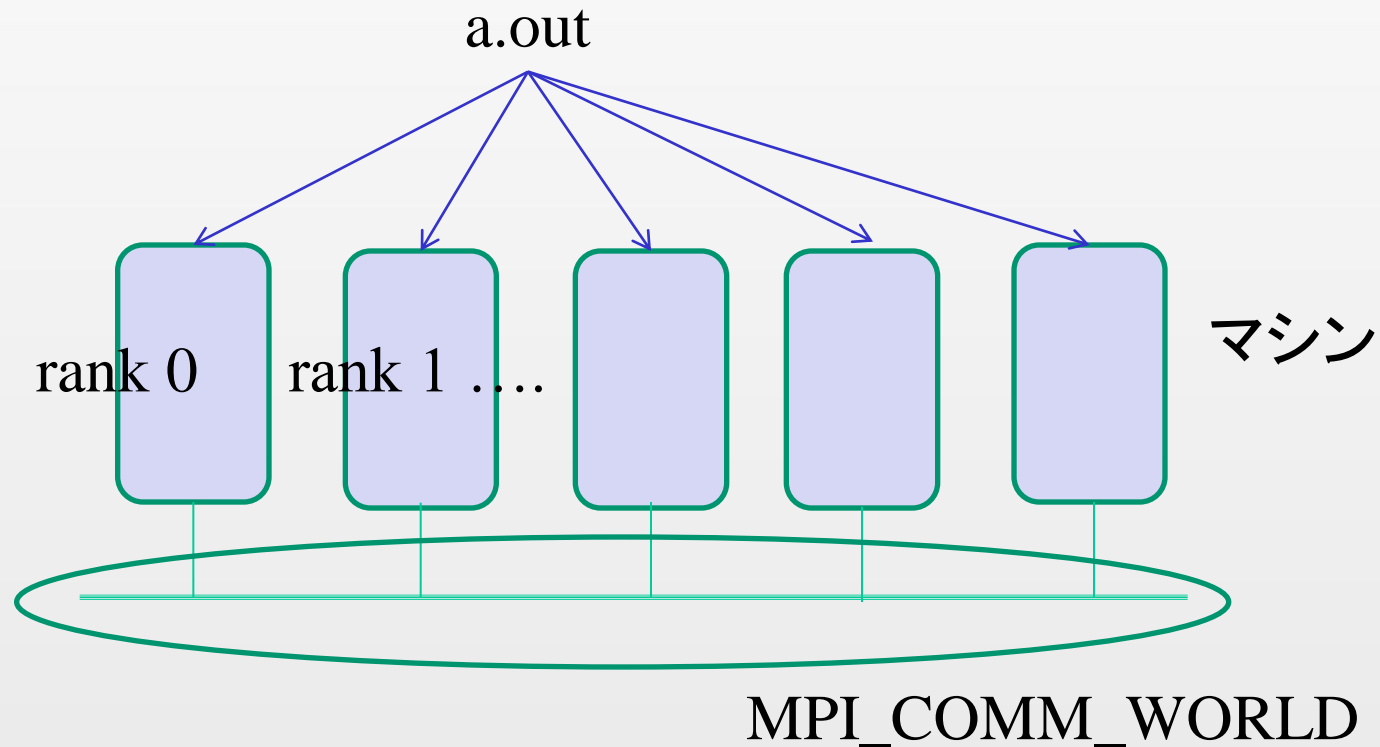
# MPI

## ■ MPI (Message Passing Interface)

- 分散メモリ型並列計算機上で並列プログラミングを行うためのAPI 仕様
- 代表的な実装 : Open MPI、MPICH2
- 各マシン上で同一のプログラムを並列実行させ、プロセス番号で分岐して処理を進めるスタイル
- 各プロセスは、メッセージを交換し合うことにより協調して処理を進める



# MPIによる並列計算





# MPIが提供する機能

## ■ 基本機能

- `MPI_Init(&argc, &argv)`
- `MPI_Comm_rank(MPI_COMM_WORLD, &my_rank)`
- `MPI_Comm_size(MPI_COMM_WORLD, &p)`
- `MPI_Finalize()`

## ■ 一対一通信(同期型)

- `MPI_Send(buffer, count, datatype, destination, tag, communicator, ierr)`
- `MPI_Recv(buffer, count, datatype, source, tag, communicator, status, ierr)`



# MPIが提供する機能

## ■ 集団通信(一部)

- MPI\_Bcast(message, count, datatype, root, comm,ierr)
- MPI\_Gather(sendbuf, sCount, sType, recvbuf, rCount, rType, root, comm,ierr)
- MPI\_Reduce(sendbuff,recvbuff,number\_of\_data,data\_type,operation,dest,communicator,ierr)

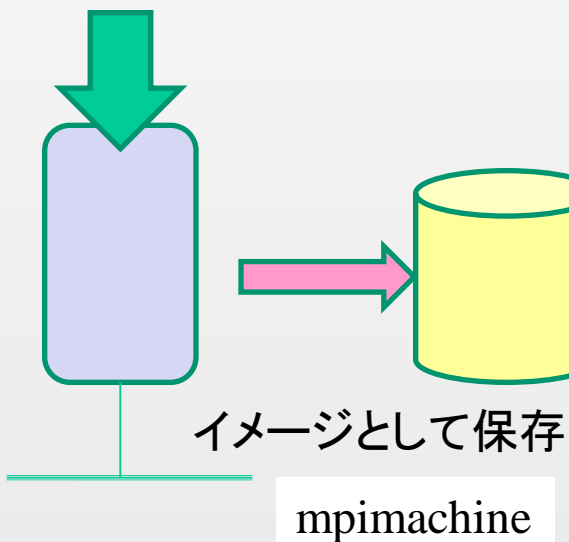
## ■ 時間計測

- MPI\_Barrier(communicator,ierr)
- MPI\_Wtime()

# クラウド上でのMPI実行環境構築

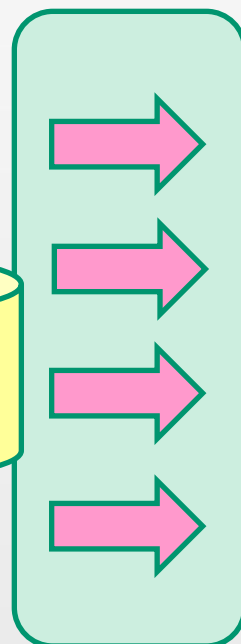
MPI本体

MPIアプリケーション  
をインストール

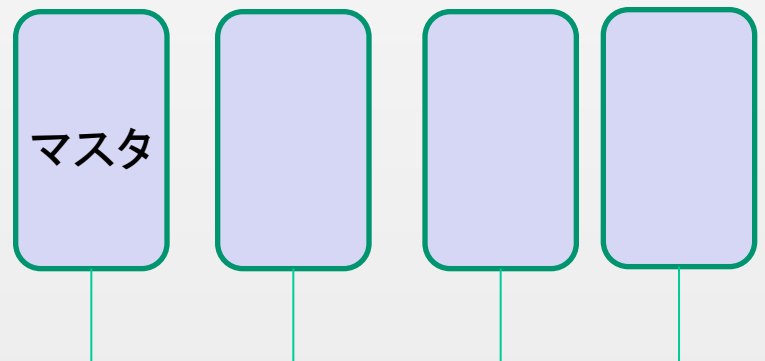


MPI実行環境起動マシン

mpiroot



イメージから必要な数  
だけMPI実行マシンを  
自動生成、自動設定







## Edubaseクラウド上のMPI実行環境構築(実習)

1. MPI実行環境起動マシンの立ち上げ
2. 環境設定
3. MPI実行マシンの生成
4. マスターマシンへのログイン
5. 実行ユーザへのログイン



# MPI実行環境起動マシンの立ち上げ

クラウドクライアント 2.0.1.201110271909

ファイル

☐ 仮想マシンイメージ一覧

名称	イメージID	所有者	アーキテ...	プラットフォ...	登録日	ルートテ...
Terracottaserver	emi-710B1697	h24-1-cld-...	x86_64	Linux	2012/04/10 1...	instance
Webserver	emi-70C4169A	h24-1-cld-...	x86_64	Linux	2012/04/10 1...	instance
CentOS-Apache-24010	emi-1E391461	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS55-Apache-topse	emi-18A21444	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS55-Tomcat-24004	emi-1D2F145C	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS-Apache-topse2	emi-1D53145E	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS-Apache-topse2	emi-1AAE1455	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS55-Apache-topse	emi-1C521454	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS55-Apache-topse	emi-1E7F1460	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS-Apache-24003	emi-1D101459	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
CentOS-Apache-topse2	emi-1FF61475	h24-1-cld-...	x86_64	Linux	2012/04/14 1...	instance
mpiroot	emi-6B12169C	h24-1-cld-...	x86_64	Linux	2012/04/20 2...	instance
mpimachine	emi-0FD4169D	h24-1-cld-...	x86_64	Linux	2012/04/20 2...	instance
mpiroot20120428	emi-1A9A1456	h24-1-cld-...	x86_64	Linux	2012/04/21 1...	instance

☐ 仮想マシン一覧

名称	バージョン情報	インスタ...	インスタンス...	キーペア	状態	パブリッ...	プライバ
misc							

# MPI実行環境起動マシンの各属性

**仮想マシン起動ダイアログ**

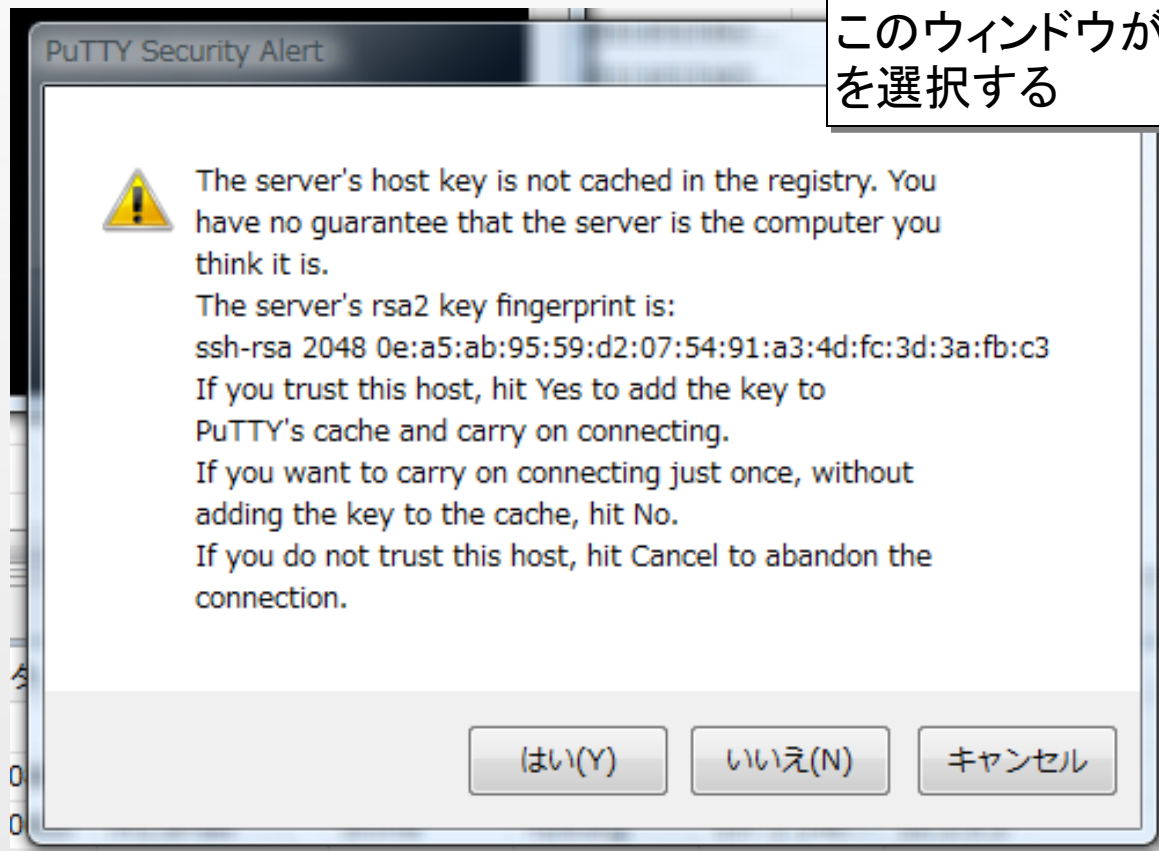
**仮想マシン起動**  
仮想マシンを起動します。

仮想マシン名	バージョン	インスタンスタイプ	キーペア	インスタンス数	セキュリティグループ
mpiroot	-	m1.small	divine	1	modelchecking

Finish Cancel

キーペアは自分のものを利用する

# SSHでログイン時の注意



このウィンドウが現れたら、いいえ (N) を選択する



## MPI実行マシンの生成

- MPI実行環境起動マシン(mpiroot\*\*\*\*)にログイン
- setupディレクトリに移動し、自分の環境に応じてenv.shを編集、保存
- 編集にはviを利用
- viの使い方(たとえば以下)  
<http://www.envinfo.uee.kyoto-u.ac.jp/user/susaki/command/vi.html>
- 以下のコマンドを実行し、MPI実行マシンの生成を行う
- ./Setup.sh -f ./env.sh



## env.shの編集(1)

```
root@localhost:~/setup
#例 : CLC=172.30.75.1
CLC_IP=vclc0006.ecloud.nii.ac.jp
#CLC_IP=AWS

#####
#2. ミニクラウド認証情報
# 仮想計算機システム上のミニクラウドに接続するために必要な認証情報です。
# クラウドクライアントの情報を参考に記載して下さい

#Access Key ID
#例:ACCESS_KEY='p3Ar1C2uRBwa2FBXnCwWKy3Cw'
ACCESS_KEY='Go76gzMWRZYSIg5dSt0VA'

#Secret Access Key
#例:SECRET_KEY='VrTHSnIBs46DwF9RAAM8KrvY90aPf1IsXfIyLA'
SECRET_KEY='zmLaK3Rlut9ZM3Xn8yU1QZ1hKz6Os6AxGP0PQ'

####
#3. AWS認証情報
# AWS上で起動する際に必要な認証情報です。

#PRIVATE_KEY(PRIVATE_KEYのファイルパスを記入して下さい)
例:PRIVATE_KEY=/root/ec2/pk-G5CZ35MWRQZ.pem
```



## env.shの編集(2)

```
root@localhost:~/setup  
#####  
#4. 仮想マシン設定情報  
# 起動する仮想マシンの情報を記載して下さい。  
#Image_ID  
IMAGE_ID=emi-6FD4169D  
  
#keypair  
KEYPAIR=divine  
  
#RSA秘密鍵  
#RSASEC=/root/.ssh/id_rsa-pub  
#RSASEC=/root/.ssh/test.ppk  
#RSASEC=/root/.ssh/divine.ppk  
RSASEC=/root/.ssh/id_rsa  
  
#仮想マシン台数  
#NUM_VM=64  
NUM_VM=4  
  
# The instance type:  
#INSTANCE_TYPE="m1.small"
```



## env.shの編集(3)

```
root@localhost:~/setup  
#仮想マシン台数  
#NUM_VM=64  
NUM_VM=4  
  
# The instance type:  
#INSTANCE_TYPE="m1.small"  
#INSTANCE_TYPE="m1.large"  
#INSTANCE_TYPE="m1.xlarge"  
INSTANCE_TYPE="c1.medium"  
#INSTANCE_TYPE="c1.xlarge"  
  
#Security Group  
#何も指定しなければデフォルトのグループになります。  
SECGROUP=divine  
  
#####  
#5.仮想マシン停止情報  
# 停止する仮想マシンの情報を記載して下さい。  
  
#MASTER仮想マシンのIP  
#MASTERIP=157.1.145.135
```





# MPI実行マシンの生成

```
root@localhost:~/setup
```

```
Using username "root".  
Authenticating with public key "imported-openssh:  
Last login: Thu Oct 20 20:32:17 2011 from 138.18  
[root@localhost ~]# cd setup  
[root@localhost setup]# ./Setup.sh -f ./env.sh  
仮想マシンの起動を指示しました  
起動確認中です・・・  
仮想マシンがRUNNING状態になりました  
MPIの設定を開始します  
MPIの起動に成功しました  
masterの仮想マシン:  
instance: i-38A90710 ip: 10.3.4.3 hostname: r-381206E61  
[root@localhost setup]#
```

- MPI実行環境起動マシンにログイン後、setupディレクトリに移動し、自分の環境に応じてenv.shを編集する
- env.shを保存後、MPI実行環境マシンの生成を行う。



# MPI実行マシンの生成

MPI実行環境マシンの生成後の画面

クラウドクライアント

ファイル

```

root@localhost: ~/setup
Using username "root".
Authenticating with public key "imported-openssh-key"
Last login: Thu Oct 20 20:32:17 2011 from 136.187.36.110
[root@localhost ~]# cd setup
[root@localhost setup]# ./Setup.sh -f ./env.sh
仮想マシンの起動を指示しました
起動確認中です...
仮想マシンがRUNNING状態になりました
MPIの設定を開始します
MPIの起動に成功しました
masterの仮想マシン:
instance: i-36A90710 ip: 10.3.4.3 hostname: r-381206E61
[root@localhost setup]#
  
```

参照先 コメント

セキュリティグループ ポリ:

Region:

名前	説明
axsh	axsh gr...
default	default ...
mapred	第5-6回
modelchecking	クラウド...
ogawa_test	sample
test	test用
topse23002	セキュリ...
topse23006	sample
topse23007	topse23...
topse23012	sample
topse23013	sample
tuc10050	sample
ystak_test	test

仮想マシン一覧

名称	バージョン情報	インスタンスID	インスタンスタイプ	キーペア	状態	PublicDNS	PrivateDNS	セキュリティ...
axsh								
misc								
Webserver		i-44B508D7	m1.small	topse23003	running	157.1.145.138	10.35.3	default
mpiroot		i-402E07D9	m1.small	divine	running	157.1.145.135	10.3.4.2	modelchecki...
mpimachine		i-36A90710	c1.medium	divine	running	157.1.145.136	10.3.4.3	modelchecki...
mpimachine		i-4096087E	c1.medium	divine	running	157.1.145.141	10.3.4.6	modelchecki...
mpimachine		i-41EB0761	c1.medium	divine	running	157.1.145.140	10.3.4.5	modelchecki...
mpimachine		i-5EF90B0F	c1.medium	divine	running	157.1.145.139	10.3.4.4	modelchecki...
Appserver		i-2C7F04A4	m1.small	topse23003	running	157.1.145.137	10.35.2	default
public								

## マスターマシンへのログイン

```
root@r-381206E61:~  
Using username "root".  
Authenticating with public key "imported-openssh-  
Last login: Thu Oct 20 20:32:17 2011 from 136.187.  
[root@localhost ~]# cd setup  
[root@localhost setup]# ./Setup.sh -f ./env.sh  
仮想マシンの起動を指示しました  
起動確認中です.....  
仮想マシンがRUNNING状態になりました  
MPIの設定を開始します  
MPIの起動に成功しました  
masterの仮想マシン:  
instance: i-36A90710 ip: 10.3.4.3 hostname: r-381206E61  
[root@localhost setup]# ./divrun.sh  
[root@localhost setup]# ssh -i /root/.ssh/id_rsa 10.3.4.3  
Last login: Wed Oct 19 00:48:50 2011 from 136.187.36.110  
[root@r-381206E61 ~]#
```

MPIの設定スクリプト(divrun.sh)を実行後、クラスタのマスターマシンにsshでログインする。マスターマシンのIPアドレスは起動時に提示されている。



# 実行ユーザへのログイン

クラスターのマスターマシンにログイン後、DiVinEユーザとなる。これでMPIアプリケーション実行の準備は完了。

```
divine@r-381206E61:~$  
Using username "root".  
Authenticating with public key "imported-openssh-  
Last login: Thu Oct 20 20:32:17 2011 from 136.187.36.110  
[root@localhost ~]# cd setup  
[root@localhost setup]# ./Setup.sh -f ./env.sh  
仮想マシンの起動を指示しました  
起動確認中です・・・  
仮想マシンがRUNNING状態になりました  
MPIの設定を開始します  
MPIの起動に成功しました  
masterの仮想マシン:  
instance: i-36A90710 ip: 10.3.4.3 hostname:r-381206E61  
[root@localhost setup]# ./divrun.sh  
[root@localhost setup]# ssh -i /root/.ssh/id_rsa 10.3.4.3  
Last login: Wed Oct 19 00:48:50 2011 from 136.187.36.110  
[root@r-381206E61 ~]# su - divine  
[divine@r-381206E61 ~]$
```

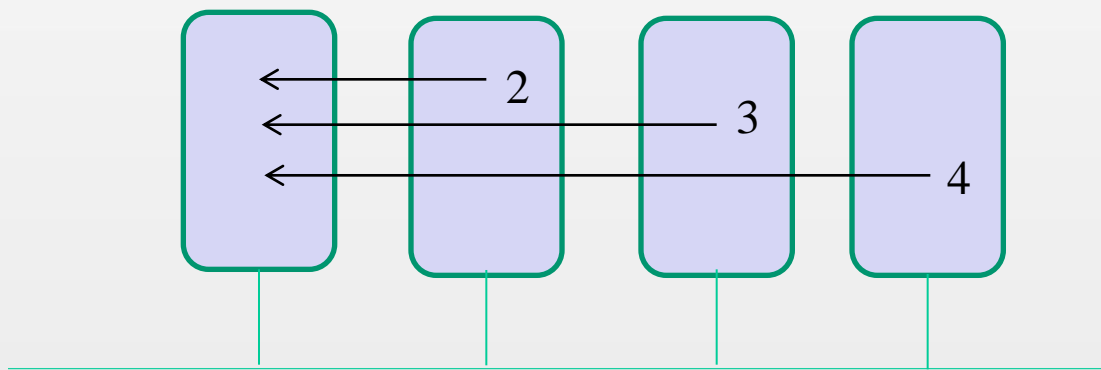


# MPIのサンプルプログラム

## ■ sample1.c

size=4

rank=0    rank=1    rank=2    rank=3



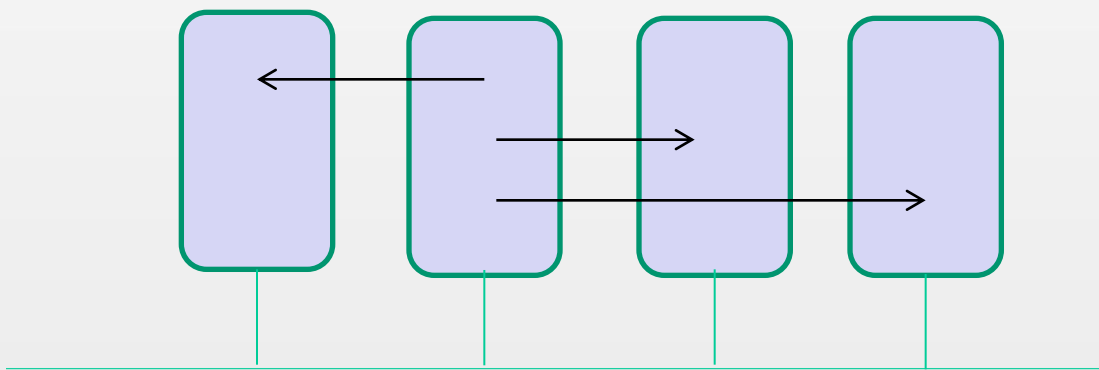


# MPIのサンプルプログラム

## ■ sample2.c

size=4

rank=0    rank=1    rank=2    rank=3



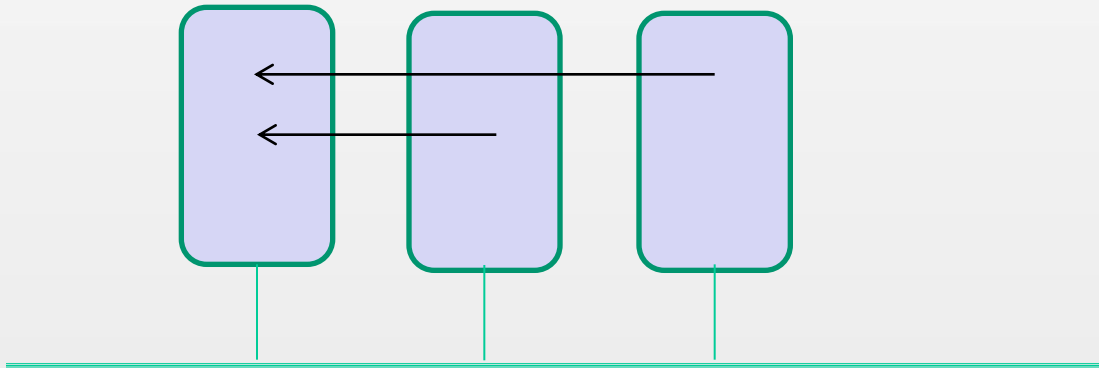


# MPIのサンプルプログラム

## ■ sample3.c

size=3

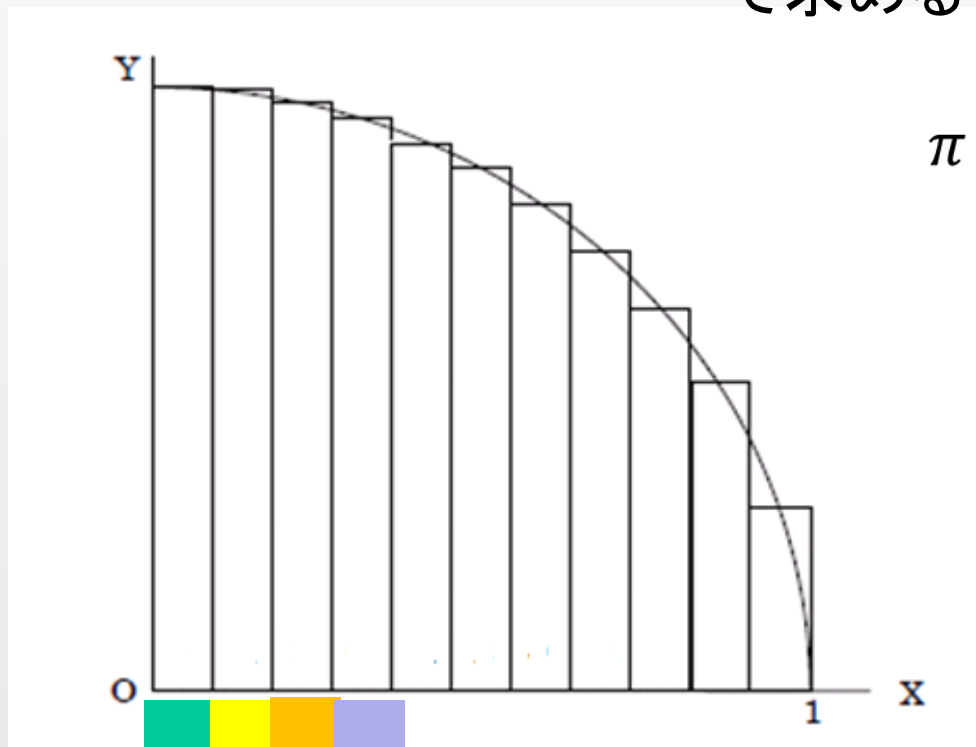
rank=0    rank=1    rank=2



# MPIのサンプルプログラム

## ■ sample4.c

円周率を区分求積法の積分計算で求める



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

-  rank 0
-  rank 1
-  rank 2
-  rank 3





# MPIサンプルプログラムの実行 (実習)

## ■ シェルからの入力コマンド

`mpirun -n ノード数 sample/sample{1,2,3,4}`

## ■ 実行と出力結果の確認

■ sample1

■ sample2

■ sample3

## ■ 時間計測

■ sample4(1台、2台、4台)

■ 間隔が粗い場合、細かい場合で台数効果を比較してみる



## sample4の実行例

```
divine@r-41B308561:~  
/root/setup  
[root@localhost setup]# vi ./env.sh  
[root@localhost setup]# pwd  
/root/setup  
[root@localhost setup]# ./Setup.sh -f ./env.sh  
仮想マシンの起動を指示しました  
起動確認中です . . . . .  
仮想マシンがRUNNING状態になりました  
MPIの設定を開始します  
MPIの起動に成功しました  
masterの仮想マシン:  
instance: i-4F320886 ip: 10.3.4.130 hostname:r-41B308561  
[root@localhost setup]# ssh -i /root/.ssh/id_rsa 10.3.4.130  
Last login: Wed Oct 19 00:48:50 2011 from 136.187.36.110  
[root@r-41B308561 ~]# su - divine  
[divine@r-41B308561 ~]$ ls  
data error mpd.hosts sample test  
[divine@r-41B308561 ~]$ mpirun -n 4 sample/sample4  
Enter the number of intervals  
1000  
n=1000  
pi is approximately: 3.1415927369231267 Error is:0.0000000833333336  
execution time = 0.0014 [sec.]  
[divine@r-41B308561 ~]$
```

# モデル検査ツールDiVinE



## モデル検査技術

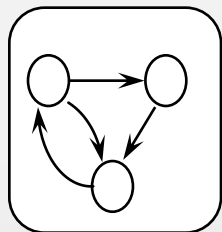
- システムをモデル化した状態遷移システムの状態空間を網羅的に探索することにより、与えられた性質が正しいかどうかを判定する検証方法
- 特長
  - 完全な自動化が可能
  - 必要な専門知識が比較的少ない
  - 部分的な仕様でも検証可能、様々な仕様記述
  - 反例の提示
- 問題点
  - 探索空間のサイズ (state explosion problem)
  - 取り扱える状態数:  $10^4$  から  $10^5$  (初期)、 $10^{120}$  (現在)

# モデル検査ツール

対象システムが満たすべき性質

時相論理の論理式等で記述する

$p$



有限状態モデル

対象システムの振る舞いを有限状態モデルで表現する

モデル検査  
ツール

有限状態モデルの検証

True:  $P$ が成り立つ  
False:  $P$ を満たさない反例の  
状態遷移系列を提示

有限状態モデルの振る舞  
いを提示

有限状態モデルのシミュレーション



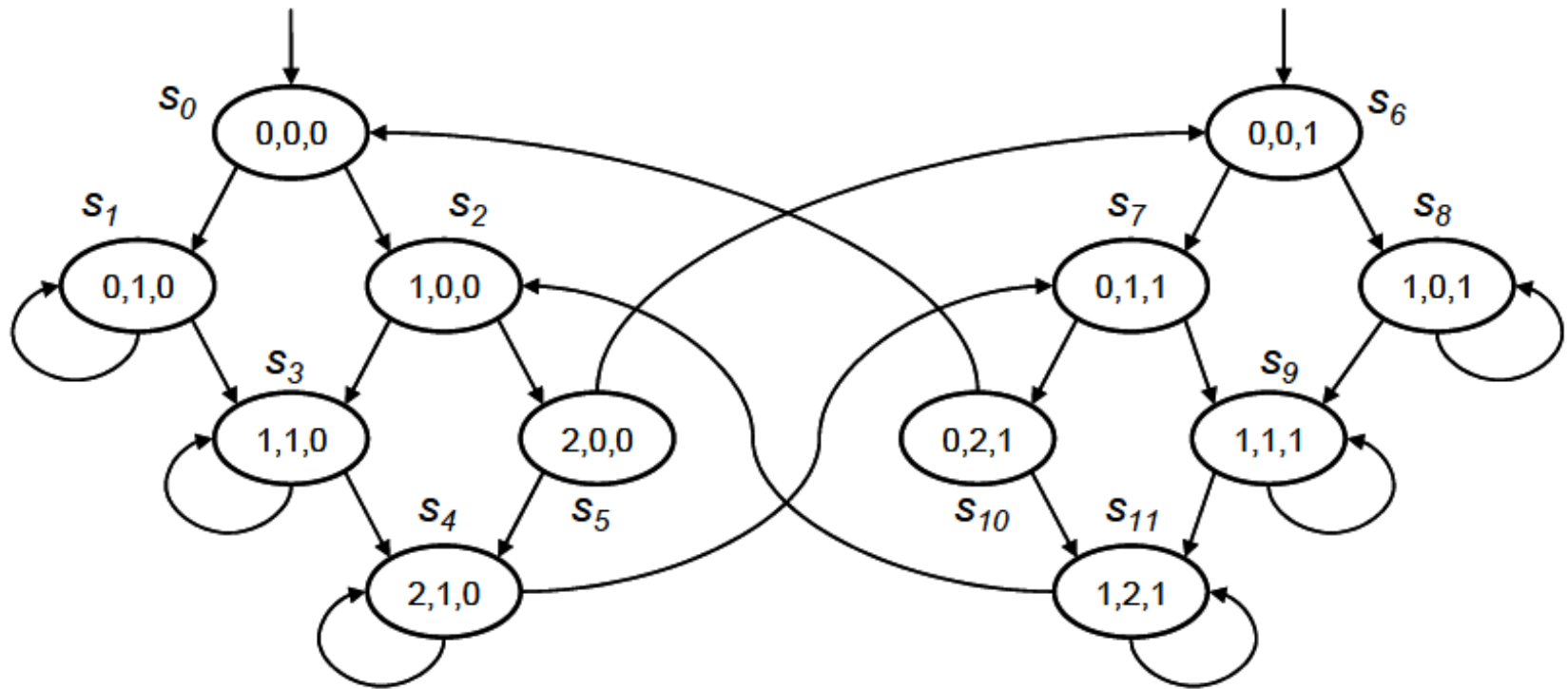
## 相互排除プログラム

```
P0    0: while True {  
        1:      wait(t == 0);  
        2:      t := 1; }
```

```
P1    0: while True {  
        1:      wait(t == 1);  
        2:      t := 0; }
```

- プログラム $P_i$  のプログラムカウンタ $pci$  ( $i = 0, 1$ )
- クリティカル・セクション  $pci = 2$  ( $i = 0, 1$ )
- 共有変数  $t$  (初期値は不定だが、ここでは0 か1とする )
- 状態 各プログラムのプログラムカウンタと共有変数の対 ( $pc0, pc1, t$ )

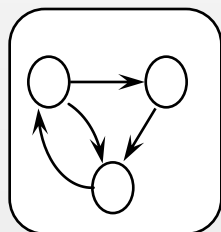
# 状態遷移図



# モデル検査の実装

オートマトン理論に基づくモデル検査  
有限状態モデルと性質をBüchiオートマトンに変換  
Büchiオートマトンの合成および空検査

有限状態モデル



満たしてほしい性質

P

Büchiオートマトン

モデルに基づくオートマトン

not Pで受理状態となるオートマトン

受理状態ループに至る遷移系列が見つければ、Pは成り立たない。見つからなければPは成り立つ





## 分散モデル検査とは

- 課題：状態爆発による検証能力の限界
- 解決アプローチ
  - 抽象化
  - 実装アルゴリズムの改良
  - 分散・並列化



## DiVinE Project

- チェコ Masaryk University ParaDise Laboratory  
で開発
- DIVINE
  - DIVINE Cluster
  - DIVINE Multi-Core
  - PROB-DIVINE
  - DivSPIN、NIPS
- DIVINE CUDA



## DiVinE で採用しているモデル検査実装

- 並列処理向けの様々なモデル検査実装を開発
  - Algorithm based on dependency structure
  - Algorithm based on negative cycles
  - Property Driven Nested DFS
  - SCC-based algorithm (OWCTY)
  - Algorithm based on back-level edges
  - Algorithm based on maximal accepting predecessors (MAP)

# スケーラビリティ(DiVinE Cluster)

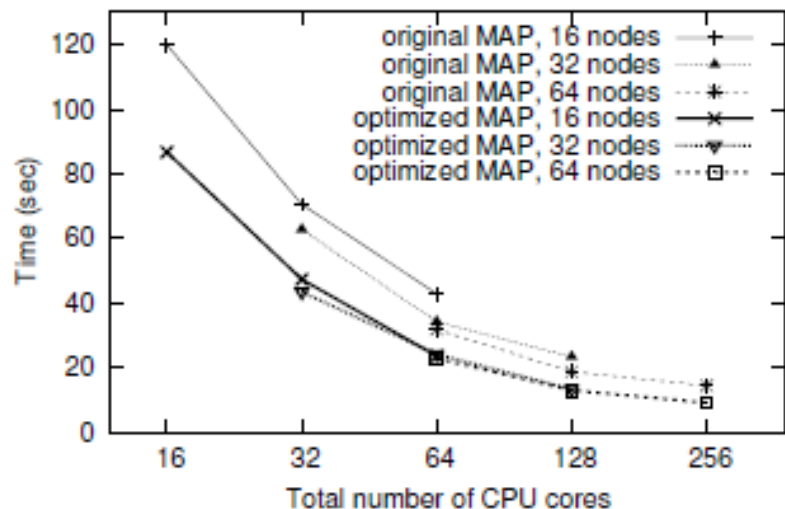


Figure 3. Optimization effects for MAP

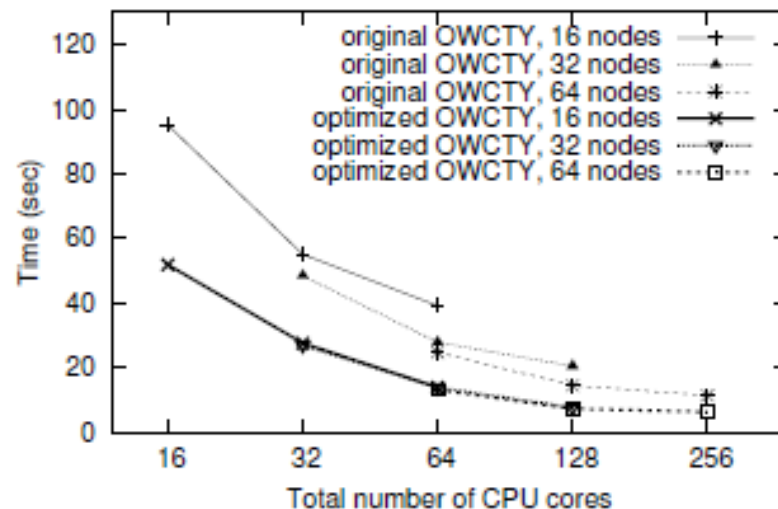


Figure 4. Optimizations effects for OWCTY



## BEEM: BEnchmarks for Explicit Model checkers

- DiVinE Clusterの性能ベンチマーク
- <http://anna.fi.muni.cz/models/index.html>
- 非常に多くの問題がとりあげられている



## DVE Specification Language

- 例題：人間と自動販売機
  - 人間は働き、金を稼ぐ
  - 稼いだ金を持って自動販売機のある場所に行く
  - 自動販売機に金を入れる
  - 飲み物を選び、出てくるのを待つ
  - 選んだ物が出てきたら幸せな状態になる
  - 違う物が出てきたら悲しい状態になる
  - 幸せな状態になったら、再び働く状態に戻る



# DVE Specification Language

- 例題：人間と自動販売機（続き）
  - 自動販売機の制御部は金が入るのを待っている
  - 金が入り、飲み物が選ばれると機械に飲み物を用意するように要求する
  - 機械は制御部からの要求を待っている
  - 要求をもらうと要求された飲み物を用意する。
  - 飲み物を取り出されると、もとの待ち状態に戻る

# DVE Specification Language

## ■ 記述例：人間

```
process man {  
  byte what, want, money;  
  state working, give_money, wait, got, happy, sad;  
  init working;  
  trans  
    working ->give_money { effect money = 1; },  
    give_money->give_money { guard money>0; sync in!;  
    effect money = 0; },  
    give_money->wait { sync req!0; effect want = 0; },  
    give_money->wait { sync req!1; effect want = 1; },  
    wait ->got { sync take?what;},  
    got ->happy { guard what == want; },  
    got ->sad { guard what != want; },  
    happy ->working { };  
}
```





# DVE Specification Language

## ■ 記述例：自動販売機（制御部）

```
process control_unit {  
  byte money, choice;  
  state ready, request;  
  init ready;  
  trans  
  ready -> ready { sync in?; effect money = 1; },  
  ready -> request { sync req?choice; },  
  request->ready { guard choice==0 and money;  
    sync make!0; effect money=0;},  
  request->ready { guard choice==1 and money;  
    sync make!1; effect money=0;},  
  request->ready { guard not money; };  
}
```



# DVE Specification Language

## ■ 記述例：自動販売機（機械）

```
process mechanic_parts {  
  byte product;  
  state ready, produce,  
  error_st;  
  init ready;  
  trans  
  ready -> produce  
  { sync make?product; },  
  produce -> error_st  
  { sync make?product; },  
  error_st -> error_st  
  { sync make?product; },  
  produce -> ready  
  { sync take!product; };  
}
```



# DVE Specification Language

## □ 構文

### Processes

```
process プロセス名  
    { プロセスのコード }
```

### Variables

グローバル変数とローカル変数がある。以下は変数定義の例である。

```
byte A[9];  
int i,j;
```

### Constants

以下は定数定義の例である。

```
const byte k = 3;
```

なお、配列定義のパラメータに利用できない。以下はエラーである。

```
byte field[k];
```



# DVE Specification Language

## □ 構文

### Process states

state変数でプロセスの状態を表現する。この定義の後、どの変数がプロセスの初期状態かを指定する必要がある。

またアトミックアクションを定義するためのstate変数を定義することもできる。

### Transitions

プロセスの状態遷移はstate変数を使い、以下のように定義する。  
変数名 -> 変数名 { 遷移が起きる条件(ガード部分) }

# DVE Specification Language

## □ 構文

### Expressions

ガード部や同期変数で使える算術式の構成要素は以下のとおりである。

- ・定数(数, true, false)
- ・カッコ、カンマ
- ・変数識別子
- ・以下の論理演算子  
-, ~, not, imply, or, and, |, ^, &, ==, !=, <, <=, >, >=, <<,  
>>, -, +, /, \*, %
- ・プロセスの状態に関する問い合わせ  
たとえばプロセスSenderがready状態であれば、問い合わせ  
Sender.ready は1に等しい。それ以外の状態なら0に等しい。



# DVE Specification Language

## □ 構文

### Channels

プロセス間の通信チャンネルはグローバル変数で定義する。定義例を以下に示す。

```
channel send, receive, toK, fromK, toL, fromL; // 型なしのチャンネル
```

```
channel {byte} b_send[0], b_receive[0]; // バッファなしのチャンネル
```

```
channel {byte,int} bi_send[0], bi_receive[0]; //2つの値を同時に送信  
するチャンネル
```

```
channel {byte,int} buf_bi_send[4], buf_bi_receive[1]; //型つき、  
バッファ付きチャンネル
```

### Type of a system

同期的動作する並行システムか非同期並行に動作するシステムかを指定できる。全体定義の一番最後に記述する。



# DVE Specification Language

## □ 構文

### Assertions

プロセスが指定の場所でのべき状態の条件 (State変数に関する条件式) をAssertionで定義できる。以下は定義の例である。

```
process My_process
{
    byte hello = 1;
    state one, two, three;
    init one;
    assert one: hello >= 1,
           two: hello < 1,
           one: hello < 6;
    trans
        ...
    ;
}
```



## 別の記述例(哲学者の食事の問題、4人の場合)

```
byte fork[4];

process phil_0 {
  state think, one, eat, finish;
  init think;
  trans
    think -> one {guard fork[0] == 0; effect fork[0] = 1;},
    one -> eat {guard fork[1] == 0; effect fork[1] = 1;},
    eat -> finish {effect fork[0] = 0; },
    finish -> think {effect fork[1] = 0; };
}

process phil_1 {
  state think, one, eat, finish;
  init think;
  trans
    think -> one {guard fork[1] == 0; effect fork[1] = 1;},
    one -> eat {guard fork[2] == 0; effect fork[2] = 1;},
    eat -> finish {effect fork[1] = 0; },
    finish -> think {effect fork[2] = 0; };
}
```





## 別の記述例(続き)

```
process phil_2 {  
  state think, one, eat, finish;  
  init think;  
  trans  
    think -> one {guard fork[2] == 0; effect fork[2] = 1;},  
    one -> eat {guard fork[3] == 0; effect fork[3] = 1;},  
    eat -> finish {effect fork[2] = 0; },  
    finish -> think {effect fork[3] = 0; };  
}
```

```
process phil_3 {  
  state think, one, eat, finish;  
  init think;  
  trans  
    think -> one {guard fork[3] == 0; effect fork[3] = 1;},  
    one -> eat {guard fork[0] == 0; effect fork[0] = 1;},  
    eat -> finish {effect fork[3] = 0; },  
    finish -> think {effect fork[0] = 0; };  
}
```

```
system async;
```



## 性質の記述

- 検証したい性質をDVE言語で記述(process LTLproperty)
- LTLの検証式からDVE記述の変換ツールも一応サポート
- VDE言語によるシステム記述と性質の記述をひとつのファイルにする。



## 性質の記述例

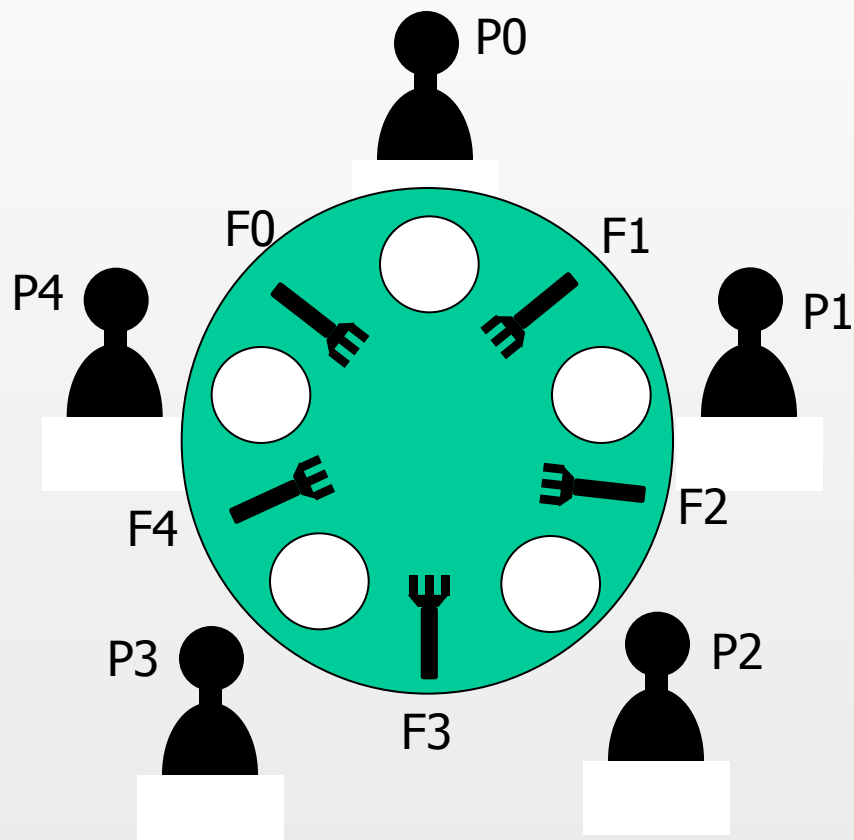
- 人間と自動販売機の例
- 検証したい性質：人間は決して悲しい状態にならない。
  - process manにおいて、man.sadは真にならない。
  - man.sadが真になると受理状態ループになるオートマトンをDVE記述に追加する。
- 具体例：ファイル man\_and\_machine.prop.dve



## Edubaseクラウドを使ったDiVinE Cluster実習

1. 例題説明
2. DiVinE Clusterによる並列実行
3. 台数効果の検証

## 例題：食事をする哲学者の問題



食事を食べ続けられる各哲学者プロセスを開発しなさい



## 哲学者が行える動作

- フォークをひとつずつ握ることができる
- 握っているフォークをひとつずつ離すことができる
- 二つのフォークを握ることで食事を食べることができる
- 隣の哲学者がフォークを握っていないときにのみフォークを握ることができる。



## DiVinE Clusterを使った演習

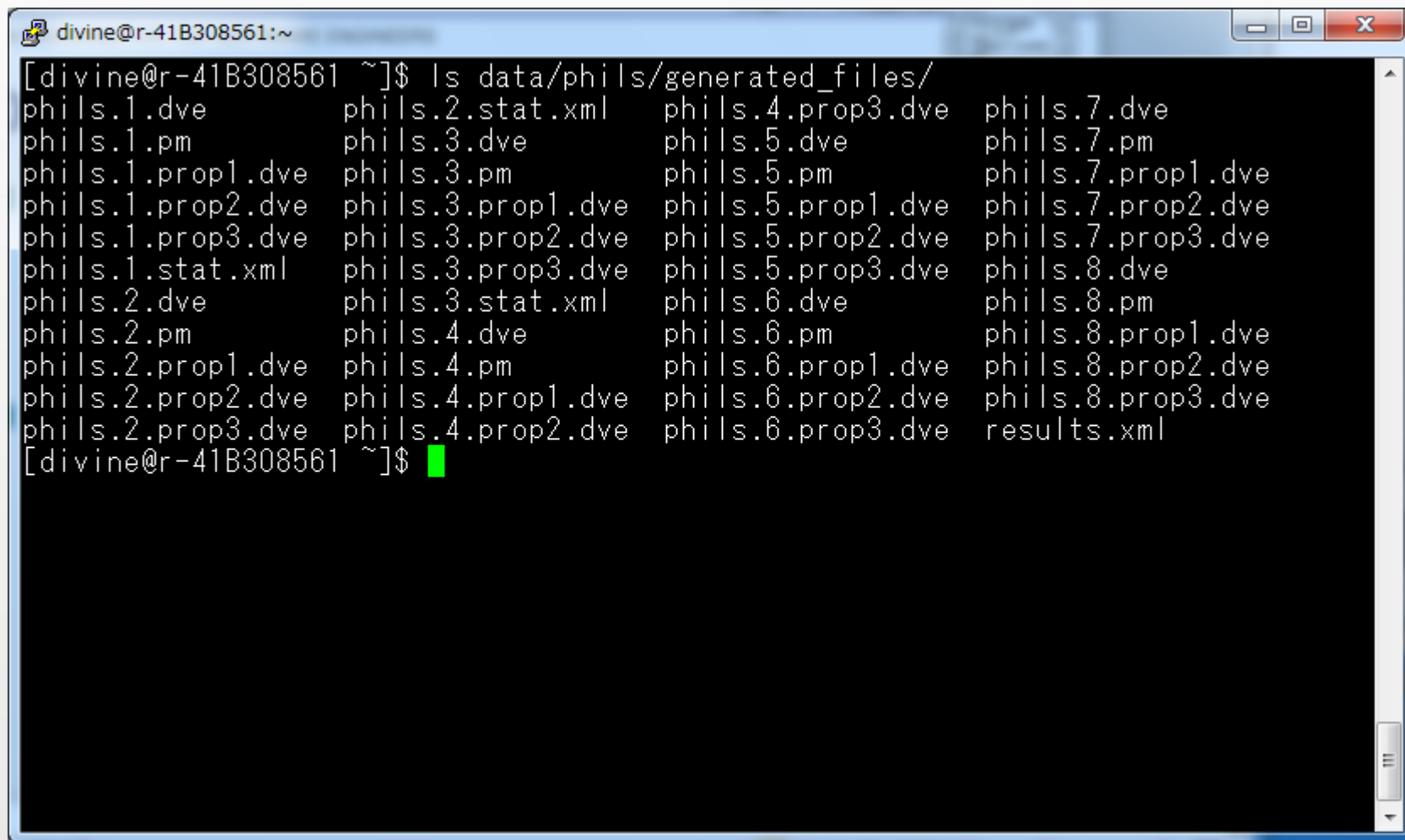
### ■ 哲学者の食事の問題

- phils.1.dve: 哲学者は4名。動作は同じ
- phils.5.dve: 哲学者は12名。動作は同じ
- phils.6.dve: 哲学者は15名。動作は同じ

### ■ 検証したい性質

1. 哲学者0は食事であり続けることが永続される  
(phils.\*.prop1.dve)
2. 哲学者0は最初のフォークを掴んだら、いつかは食事であり続ける(phils.\*.prop2.dve)
3. 哲学者の誰でも食事にあるつけることが永続される  
(phils.\*.prop3.dve)

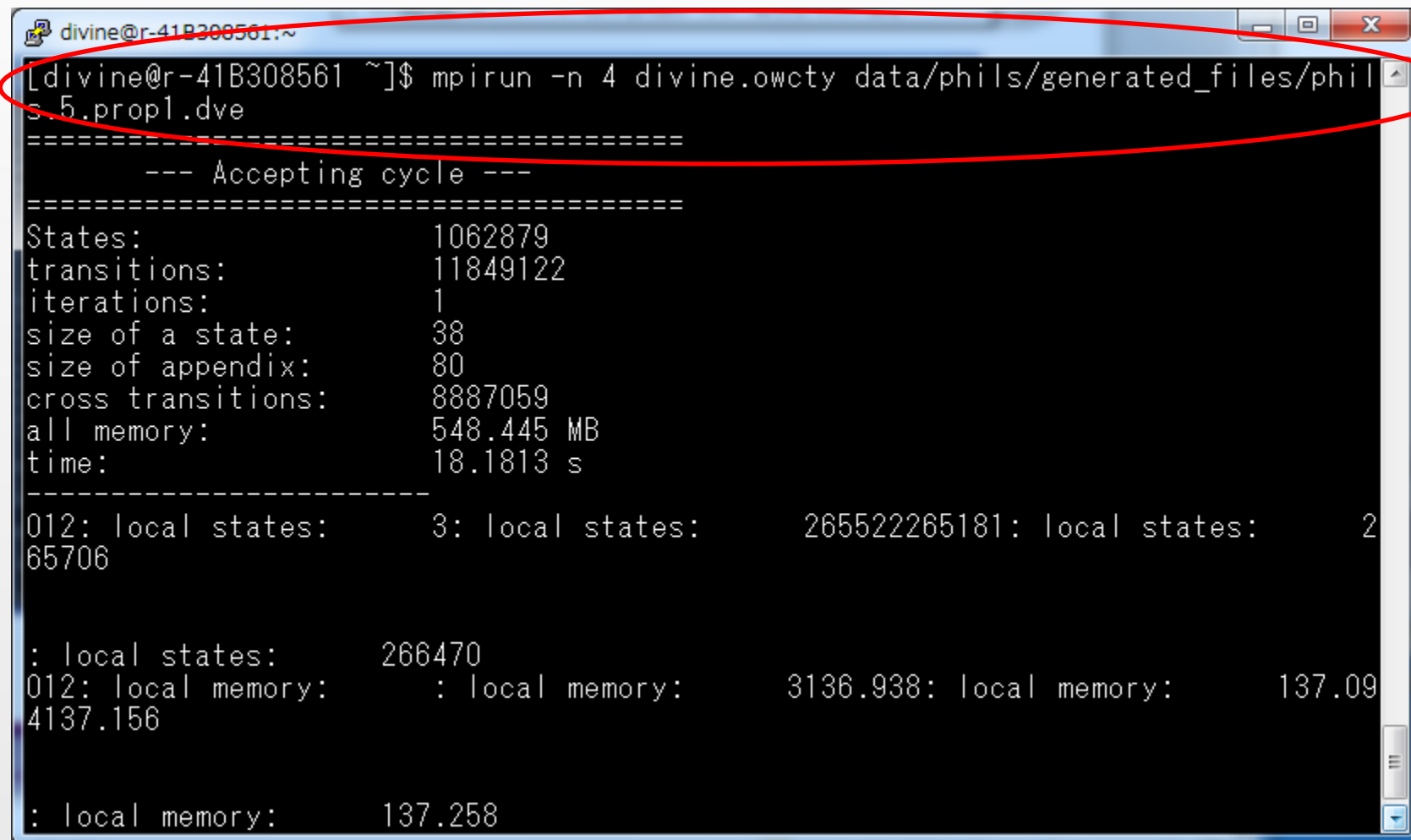
## 問題ファイル

A screenshot of a terminal window with a blue title bar. The title bar text is "divine@r-41B308561:~". The terminal content shows a command to list files in a directory, followed by a multi-column list of files. The files are organized into four columns. The first column lists files from phil.s.1.dve to phil.s.2.prop3.dve. The second column lists files from phil.s.2.stat.xml to phil.s.4.prop2.dve. The third column lists files from phil.s.4.prop3.dve to phil.s.6.prop3.dve. The fourth column lists files from phil.s.7.dve to results.xml. The prompt "[divine@r-41B308561 ~]\$ " is followed by a green cursor.

```
divine@r-41B308561:~  
[divine@r-41B308561 ~]$ ls data/phils/generated_files/  
phils.1.dve      phils.2.stat.xml  phils.4.prop3.dve  phils.7.dve  
phils.1.pm       phils.3.dve      phils.5.dve        phils.7.pm  
phils.1.prop1.dve phils.3.pm       phils.5.pm         phils.7.prop1.dve  
phils.1.prop2.dve phils.3.prop1.dve phils.5.prop1.dve  phils.7.prop2.dve  
phils.1.prop3.dve phils.3.prop2.dve phils.5.prop2.dve  phils.7.prop3.dve  
phils.1.stat.xml  phils.3.prop3.dve phils.5.prop3.dve  phils.8.dve  
phils.2.dve       phils.3.stat.xml  phils.6.dve        phils.8.pm  
phils.2.pm        phils.4.dve      phils.6.pm         phils.8.prop1.dve  
phils.2.prop1.dve phils.4.pm       phils.6.prop1.dve  phils.8.prop2.dve  
phils.2.prop2.dve phils.4.prop1.dve phils.6.prop2.dve  phils.8.prop3.dve  
phils.2.prop3.dve phils.4.prop2.dve phils.6.prop3.dve  results.xml  
[divine@r-41B308561 ~]$
```

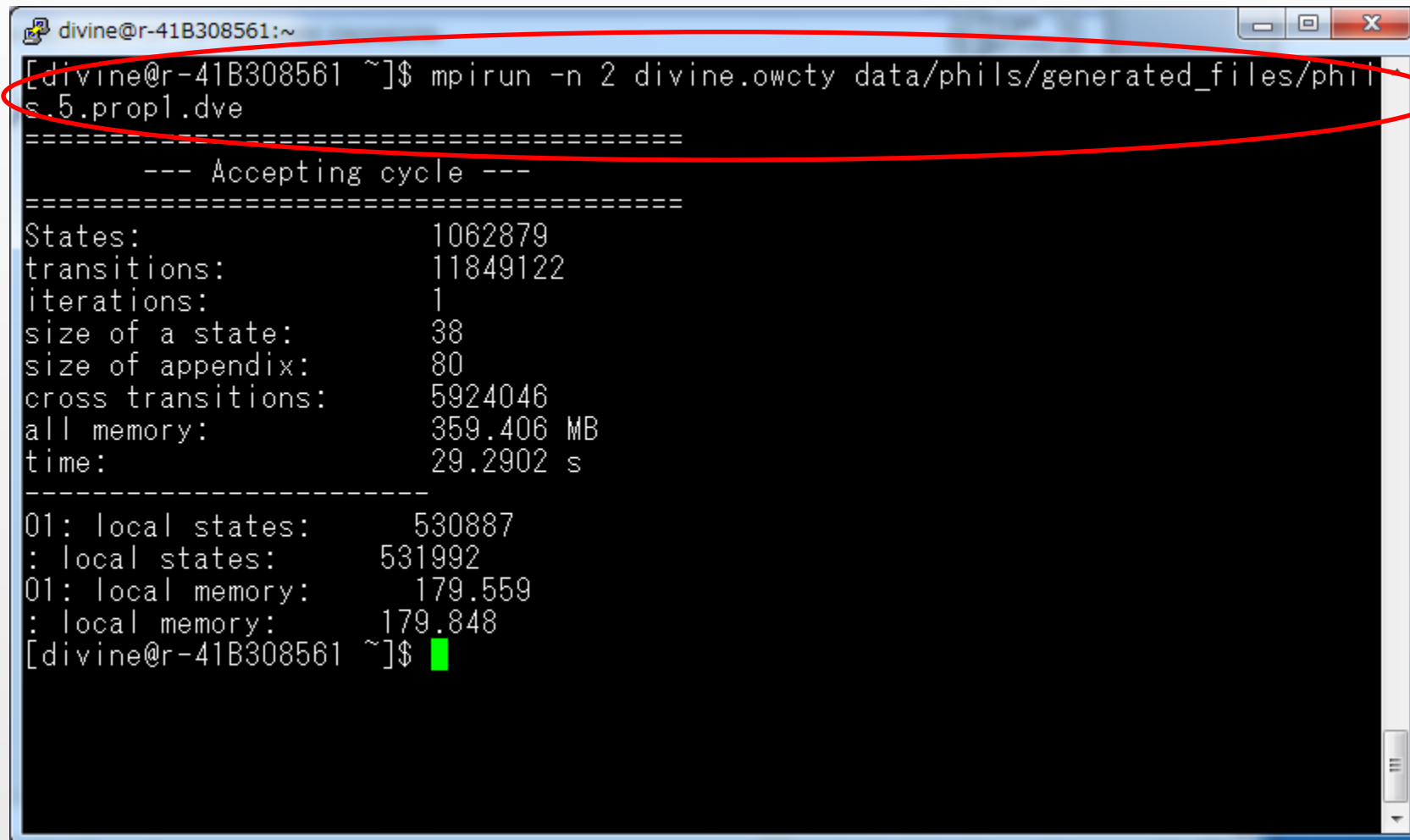


DiVinEの並列実行はmpirunを使って実行する。以下の画面は4ノードを利用している例である。



```
divine@r-41B308561:~$ mpirun -n 4 divine.owcty data/phils/generated_files/phils_5.prop1.dve
-----
--- Accepting cycle ---
-----
States:                1062879
transitions:           11849122
iterations:             1
size of a state:        38
size of appendix:       80
cross transitions:      8887059
all memory:             548.445 MB
time:                   18.1813 s
-----
012: local states:      3: local states:      265522265181: local states:      2
65706
: local states:         266470
012: local memory:      : local memory:      3136.938: local memory:      137.09
4137.156
: local memory:         137.258
```

以下の画面は2ノードを利用している例である。

A terminal window titled "divine@r-41B308561:~" showing the execution of a command. The command and its output are circled in red. The output shows statistics for a model checking run using 2 nodes.

```
divine@r-41B308561:~$ mpirun -n 2 divine.owcty data/phils/generated_files/phils.5.prop1.dve
=====
--- Accepting cycle ---
=====
States:                1062879
transitions:           11849122
iterations:             1
size of a state:        38
size of appendix:       80
cross transitions:      5924046
all memory:             359.406 MB
time:                   29.2902 s
-----
01: local states:      530887
: local states:        531992
01: local memory:      179.559
: local memory:        179.848
[divine@r-41B308561 ~]$
```

Divineユーザのホームディレクトリ下にdataというディレクトリがある。ここに様々なサンプルが格納されている。



```
divine@r-41B308561:~/data
adding          elevator_planning  leader_filters   public_subscribe
anderson        exit                lifts            reader_writer
at              extinction       lloyd           resistance
bakery          firewire_link      lup             rether
blocks          firewire_tree      mcs             rushhour
bopdp           fischer            msmie           schedule_world
bridge          frogs              needham         sokoban
brp             gear               peg_solitaire   sorter
brp2            hanoi              peterson        synapse
cambridge       iprotocol          pgm_protocol    szymanski
collision       krebs              phil            telephony
cyclic_scheduler lamport            plc             train-gate
driving_phils   lamport_nonatomic pouring          production_cell
elevator        lann               protocols
elevator2       leader_election
```



## DiVinE Clusterを使った演習

- 問題ファイルのパス
  - data/phils/generated\_files/phils.\*.prop\*.dve
- クラウド上における分散モデル検査の台数効果は？
  - ノードが1つの場合
  - ノードが2つの場合
  - ノードが4つの場合

# シャットダウンの方法

DivineユーザおよびMPI実行マシンをログアウトし、MPI実行環境起動マシンに戻る。

MPI実行環境起動マシンのsetupディレクトリからシャットダウン用のスクリプトを実行する

```
root@localhost:~/setup
Using username "root".
Authenticating with public key "imported-openssh-key".
Last login: Thu Oct 20 19:47:11 2011 from 136.171.1.100
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# cd setup/
[root@localhost setup]# ls
Setup.sh  Setuphelp  Shutdown.sh  Shutdownhelp  divrun.sh  env.sh
[root@localhost setup]# ./Shutdown.sh -f ./env.sh -m 10.3.4.3
masterIPで指定されたマシンより、停止させる仮想マシンの一覧を取得します
仮想マシンに停止を指示しました
停止した仮想マシン:
10.3.4.3
10.3.4.4
10.3.4.5
10.3.4.6
[root@localhost setup]#
```



## 本講義のまとめ

- クラウドによるMPIの利用事例
- MPI実行環境の構築
- MPIのサンプルプログラム、アプリケーションの実習
- クラウドによる並列計算の効果