東京大学情報基盤センター
お試しアカウント付き並列プログラミング講習会

Xcryptを用いたジョブ並列処理

京都橘大学 工学部情報工学科

平石 拓

Mail: hiraishi@tachibana-u.ac.jp

2021/10/26

この講習について

- ジョブ並列スクリプト言語「Xcrypt」を使って「大量のジョブを投入する作業」を手軽に行う方法について紹介します
- まず一通り解説を行った後，実習（実際に動かしていただく）に移ります
- 実習は，指示通りにやっていけば進められるようになっています。時間内に実習が最後まで終わらなかった場合は、講習終了後にお試しください
- 質問は（講習会終了後も）Slack等で受け付けます

本講習会の資料

- このスライド（PDF）
  - 本講習会のページ
    https://www.cc.u-tokyo.ac.jp/events/lectures/168/
  - 修正・更新があった場合はSlack等でお知らせします
- 講習で使うサンプルコード
  - wget等で取得してください

```
$ wget http://ais.sys.i.kyoto-u.ac.jp/~task/xcrypt-tutorial-202110.zip
```

目次

- Xcryptとは
- Part 1：基本操作
  - スクリプトの書き方の基本的な説明
  - ジョブ投入・確認・中止
- Part 2：応用例
  - パラメータスイープ
    （実行条件を変えながらプログラムを大量実行）
  - 一部のジョブのやりなおし
  - 中断・再開
- Part 3：より高度な機能の紹介
- 補足：他のスパコンへでの利用
  - Oakforest-PACS へのインストール
  - OBCX への（最新版の）インストール

Xcryptとは

- スパコン上でジョブを（逐次 or 並列に）多数実行するような処理を記述するためのスクリプト言語
- Perlベース
  - 簡単な処理（パラメータスイープ程度）なら特にPerlを知らなくても利用可能
  - Perlの能力を駆使してより複雑な処理も記述可能
- スパコンごとのジョブ投入インターフェースの違いを気にせず，単一スクリプトを様々な環境で実行可能
- 投入したジョブの管理や一部ジョブの再実行などが容易に行える
- 様々な拡張機能
  - 同時投入数制限，宣言的なジョブ依存関係の定義など
  - 自分で拡張機能を作成することも可能

主な利用シーン

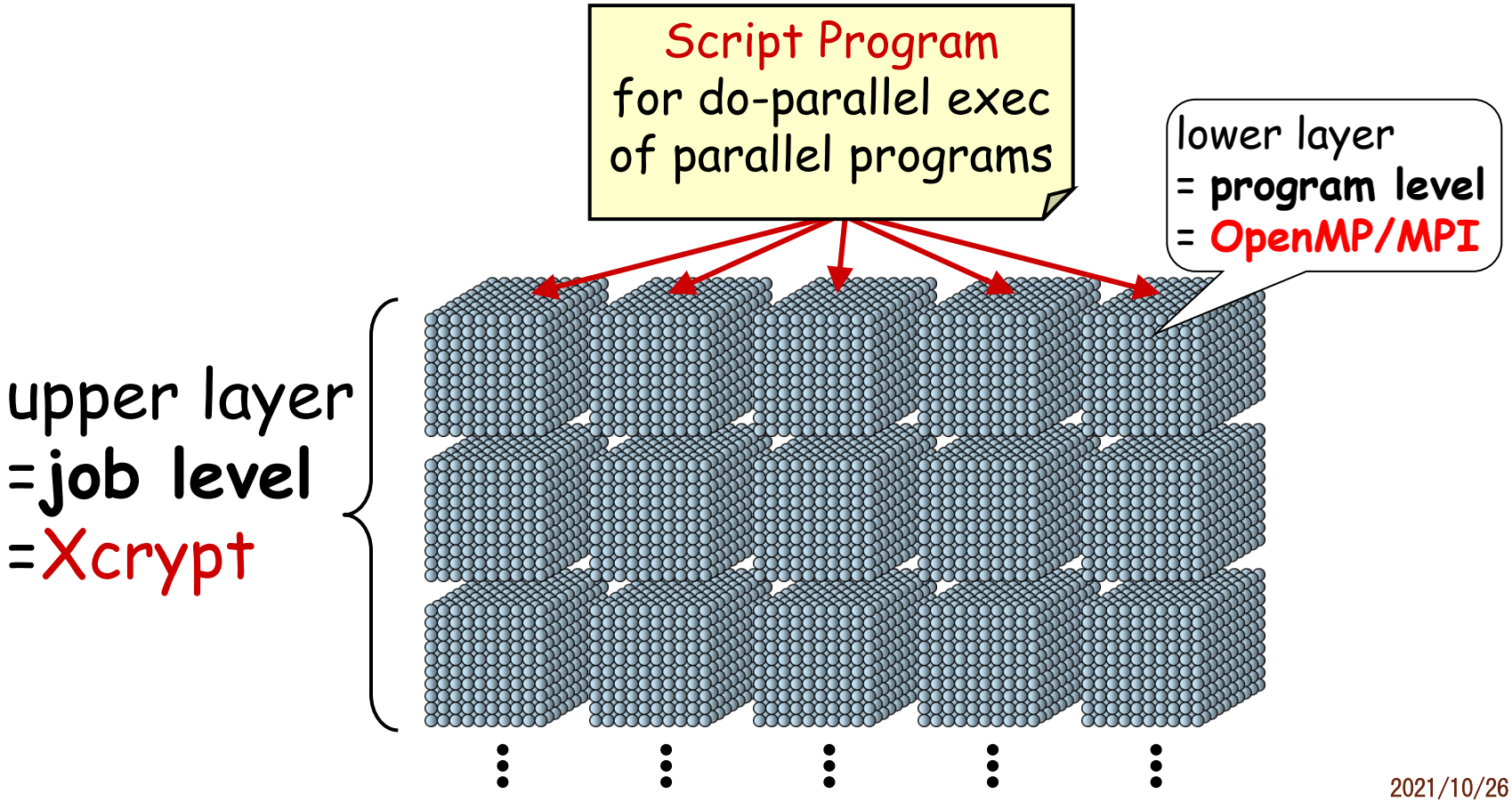
- ある単一プログラムを何度も実行したい
  - 試したい入力が何通りもある
  - 問題サイズや並列数を変えながら性能評価をとりたい（それぞれk回実行して中央値を…）
- 依存関係がなければ，空きノードを使って同時に実行
- 依存関係があれば，その順序に従って実行

ほかにも，以下の便利機能

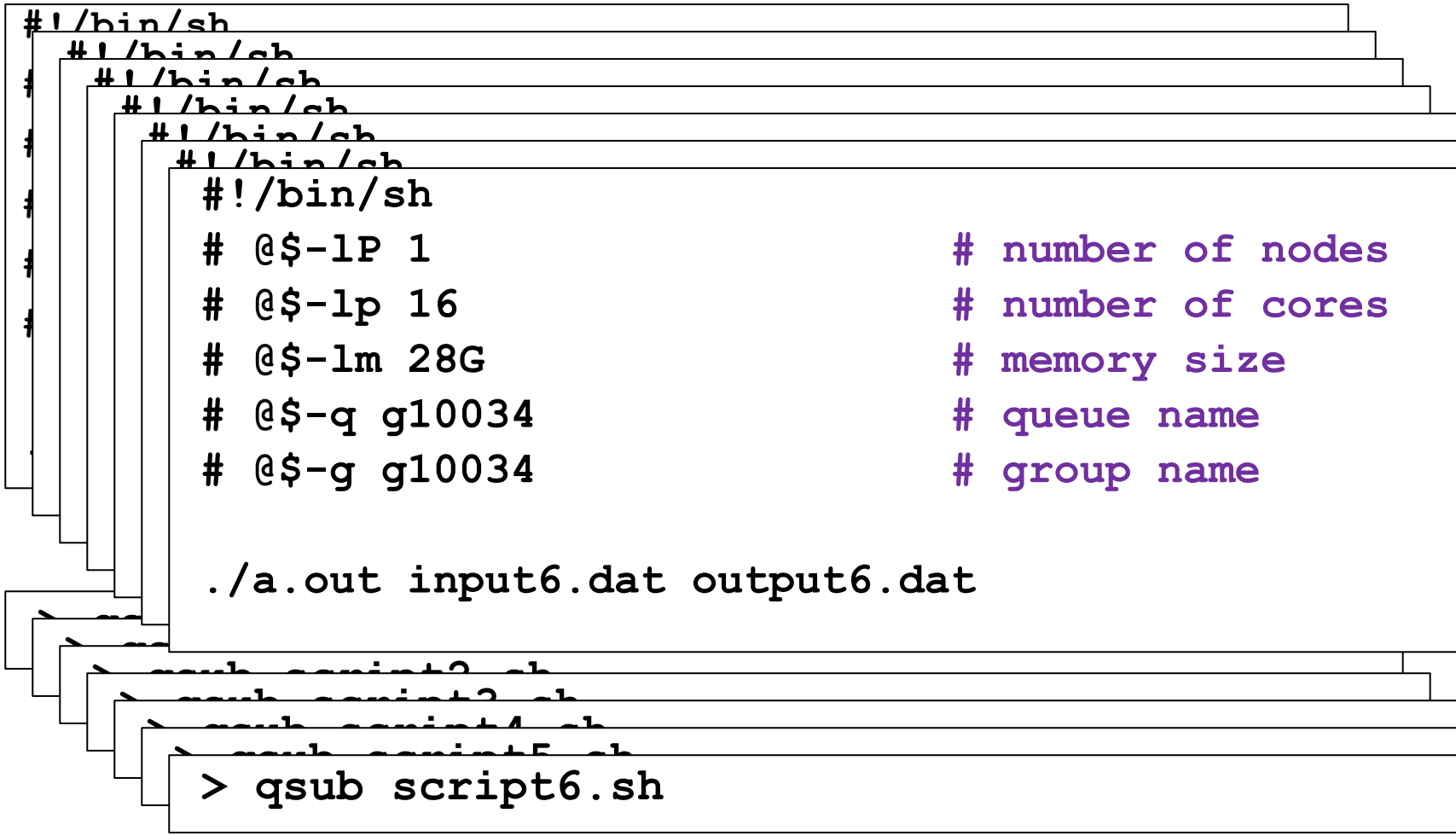
- いろんなスパコンでのジョブ実行を1つのスクリプトで
- 一部の実行が失敗したとき，手軽に把握・再実行
- 各実行ごとに結果を格納するディレクトリを勝手に作ってほしい

ジョブ並列実行

- プログラム自体が（あまり）並列化されていなくても，
  同時実行（ジョブ並列）によりスパコンの計算能力は活用可能



Example of a Job Script…



Example of a Job Script…

```
#!/bin/bash
#$QSUB -A p=1:c=16:t=16:m=28G # amount of resource
#$QSUB -q gr10034b                  # queue name
#$QSUB -ug gr10034                  # group name

./a.out input0.dat output0.dat
```

```
> qsub script0.sh
```

Xcryptの情報源・入手先など

- 最新版ダウンロード（GitHub）
  $ git clone https://github.com/xcrypt-job/xcrypt
- プロジェクトWebページ
  - 論文，講演資料など

  http://ais.sys.i.kyoto-u.ac.jp/~task/xcrypt/index.html
- マニュアル
  - --help オプション
    ```
    $ xcrypt --help
    $ xcryptstat --help
    $ xcryptdel --help
    ```
  - PDFマニュアル（細かい話）
    - アーカイブ内 doc/tutorial/xcrypt-manual.pdf

Xcryptが利用できるシステム

- 導入済み
  - OBCX / Wisteria @東大
  - Grand Chariot / Polaire @北大
- 自分でインストールすることで利用可
  - Camphor2, Laurel2 @京大
  - HOKUSAI @RIKEN R-CCS etc.
- それ以外のシステムでも，インストール＋設定ファイルの記述により利用可
  - インストール方法は後の実習で扱います

PART 1：基本

imb-s.xcr（単一ジョブ実行）

```
use base qw (core);     # とりあえず「おまじない」と思う
my $procs = 28;
%template = (
  'id' => 'job-imbs',   # 任意のジョブ名
  'exe0' => "mpiexec.hydra -n $procs ./IMB-MPI1 -npmin $procs” ,
                        # 実行コマンド
  'JS_node' => $procs,  # MPIプロセス数
  'JS_cpu' => 1,        # コア数/MPIプロセス
  'JS_limit_time' => '00:05:00',  # ジョブの制限時間
  'JS_queue' => 'queuename',      # キュー名（講習会では tutorial）
  'JS_group' => 'groupname',      # グループ名（講習会では gt00）
);
# Execute the job
@jobs=&prepare_submit_sync (%template);
```

ジョブの実行

```
# スクリプト実行
$ xcrypt imb-s.xcr
job-imbs <= initialized
job-imbs <= prepared
job-imbs <= submitted
job-imbs <= queued
job-imbs <= running
job-imbs <= done
job-imbs <= finished
```

ジョブの結果確認

```
# 生成されるファイル
$ ls -ltr
…
-rw-r--r-- … job-imbs_obcx.bat
-rw------- … job-imbs_stderr
-rw------- … job-imbs_stdout
…
# Xcryptが自動生成したジョブスクリプト
$ cat job-imbs_obcx.bat
# ジョブの標準エラー出力（ここでは0バイトのはず）
$ cat job-imgs_stderr
# ジョブの標準出力
$ cat job-imbs_stdout
```

パラメータの意味

パラメータ名	意味
id	Xcrypt（およびユーザ）がジョブを同定するための識別子
exe*n*	ジョブ内で実行されるコマンド文字列 *n* は0以上の整数で，この数字が小さいものから順に実行される．
arg*n*_*m*	*m*は0以上の整数．この数字が小さいもの順に，指定された文字列が，空白区切りでexe*n*の後ろに連結される
JS_node, JS_cpu, JS_memory（※1）, JS_thread	ジョブが使用する資源量：プロセス数，コア数/プロセス，メモリサイズ/プロセス，スレッド数/プロセス
JS_phnode（※2）	ジョブが使用する物理ノード数を直接指定
JS_queue, JS_group	投入するジョブキュー名，グループ名
before, after	ジョブの投入直前，完了直後に実行するPerl手続（imb-s.xcrのコメントアウト部分が記述例）

（※1）一部システムでは不可（OBCXも不可）

（※2）一部システムでは不可（OBCXは可）
省略時，JS_nodeとJS_cpuから自動算出

資源量の指定について

- 使用するシステムによらず
  - JS_node はジョブで使用する全プロセス数
  - JS_cpu はプロセスあたりのコア数
  - JS_memory はプロセスあたりのメモリ量
  - JS_thread はプロセスあたりのスレッド数
- JS_threadは省略時，JS_cpuと同じ値
- これらの設定値に基づいて，ジョブスクリプトの資源量(e.g., -L node=…)の記述が自動生成される，という仕組み

プロセス数変更

```
use base qw (core);                    # とりあえず「おまじない」と思う
my $procs = 56;
%template = (
  'id' => 'job-imbs',                  # 任意のジョブ名
  'exe0' => ” mpiexec.hydra -n $procs ./IMB-MPI1 -npmin $procs” ,
                                       # 実行コマンド
  'JS_node' => $procs,                 # MPIプロセス数
  'JS_cpu' => 1,                       # コア数/MPIプロセス
  'JS_limit_time' => '00:05:00', # ジョブの制限時間
  'JS_queue' => 'queuename',     # キュー名（講習会では tutorial）
  'JS_group' => 'groupname',     # グループ名（講習会では gt00）
);
# Execute the job
@jobs=&prepare_submit_sync (%template);
```

修正版imb-s.xcrを実行

```
$ xcrypt imb-s.xcr
job-imbs <= initialized
job-imbs <= prepared
job-imbs <= finished
```

- そのままでは再実行されない
- idが同一のジョブは，実行済の場合スキップされる
  → キャンセルまたは別のidをつける必要がある

ジョブのキャンセル

```
# ジョブ名を指定してキャンセル
$ xcryptdel --cancel job-imbs
job-imbs is signaled to be uninitialized.
# 全ての実行履歴を忘れたい場合は以下
$ xcryptdel --clean

# 再実行＆実行結果の確認
$ xcrypt imb-s.xcr
$ ls -ltr
→ job-imbs_obcx.batや実行結果が更新されていることを確認
$ cat job-imbs_obcx.bat
→ --mpi proc=以降が意図通りになっていることを確認
```

ジョブの実行履歴

- ジョブ履歴は，xcryptを実行したディレクトリのinv_watchというディレクトリに保管されている
  - そのため，このディレクトリをrm -rで削除したり，xcryptを別のディレクトリで実行した場合は，履歴は反映されない
- xcrypdel --cancelで指定されたジョブは，実行中であれば(qdel等で)キャンセルし，inv_watch履歴中の当該ジョブの状態を未実行に戻す
  - --cancelなしでジョブ名を指定すると，実行中のジョブのみキャンセルし，完了したジョブはキャンセルされない
- xcryptdel --cleanは，実行中の全てジョブを(qdel等で)キャンセルし，全ての実行履歴を忘れる
  - この履歴は，inv-watch.2等の別名のディレクトリに退避される．このディレクトリは削除しても差し支えない

別のidをつける

```
# スクリプトを編集して，自分で別のidにする
$ vi imb-s.xcr
- 'id' => 'job-imbs',
+ 'id' => 'job-imbs-p56',
% xcrypt imb-s.xcr
…
job-imbs-p56 <= finished
# xcrypt実行時に--genidオプションをつけると，重複しない適当なidが
振られる（同一idによりスキップされる仕様が鬱陶しい人向け）
% xcrypt --genid imb-s.xcr
…
job-imbs_002 <= finished
```

パラメータのデフォルト値の定義

- JS_queue, JS_group, JS_limit_timeに設定すべき値は
  - システム内ではほぼ固定，かつ，システム毎に異なる
- スクリプトごとにこれらの設定を書く／スクリプトを別のシステムで動かすとき書き換えるのは面倒

→ システムごとのデフォルト値を定義しておくとよい

```
# システムデフォルトのユーザ設定ファイルをホームディレクトリにコピー
$ cp /work/opt/local/apps/xcrypt/1e8a958966aa/etc/xcryptrc ~/.xcryptrc
# ~/.xcryptrc が存在すると，こちらが優先的に読み込まれる．このファイルを編集
$ emacs ~/.xcryptrc
- # JS_limit_time = 3600
+ JS_limit_time = ” 00:05:00”
- # JS_queue = queuename
+ JS_queue = queuename                講習会では tutorial
- # JS_group = groupname
+ JS_group = groupname                講習会では gt00
```

→ .xcrスクリプトのJS_limit_time/JS_queue/JS_groupを消しても動作することを確認

- これ以外でも，任意のパラメータのデフォルト値を設定できる
- スクリプトと~/.xcryptrcの両方に定義があった場合，スクリプトの定義が優先

PART 2：複数ジョブ実行

imb-p1.xcr（複数ジョブ実行）

```
use base qw (core);                          # とりあえず「おまじない」と思う
%template = (
  'RANGE0' => [2,4,8,16,32],                 # プロセス数の集合
  'id@' => sub {"job-imbp-P$VALUE[0]"},      # ジョブ名の命名規則
  'exe0@' => sub {
    "mpiexec.hydra -n $VALUE[0] ./IMB-MPI1 -npmin $VALUE[0]"},
                                             # 実行コマンド
  'JS_node@' => sub {$VALUE[0]},             # MPIプロセス数
  'JS_cpu' => 1,                             # コア数/MPIプロセス
);
# Execute the jobs
@jobs=&prepare_submit_sync (%template);
```

imb-p1.xcrの実行

```
$ xcrypt imb-p1.xcr
job-imbp-P2 <= initialized
job-imbp-P2 <= prepared
…
job-imbp-P2 <= finished
job-imbp-P4 <= finished
…
job-imbp-P32 <= finished
# 生成されたファイルを確認
$ ls -ltr
```

imb-s→imb-p1 の変更箇所

- RANGE*n* により，実行したいパラメータの集合を定義（*n*は0以上の整数）
- ジョブごとに設定したい値が異なるようなパラメータ *param*は，*param*@として定義し，=>の右辺は値をどのように設定するかを定義する関数とする
  - この関数の中で，集合RANGE*n* の要素（ジョブごとに異なる値になる）を$VALUE[*n*]で参照できる
  - 今回の例では文字列にパラメータ値を埋め込む単純な例だが，もちろんもっと複雑なPerl関数も書ける
- 集合RANGE*n*を複数定義した場合，それらの集合の直積の要素に対応したジョブが生成・実行される

imb-p2.xcr（複数パラメータによるジョブ集合定義）

```
use base qw (core);

%template = (
  'RANGE0' => [2,4,8,16,32],
  'RANGE1' => ['Sendrecv','Reduce','Bcast','Alltoall'],  # ベンチマークの種類
  'id@' => sub {"job-imbp-P$VALUE[0]p-$VALUE[1]"},       # ジョブ名の命名規則
  'exe0@' => sub {
    "mpiexec.hydra -n $VALUE[0] ./IMB-MPI1 -npmin $VALUE[0] $VALUE[1]"},
                                                         # 実行コマンド
  'JS_node@' => sub {$VALUE[0]},                         # MPIプロセス数
  'JS_cpu' => 1,                                         # コア数/MPIプロセス
);
# Execute the job
@jobs=&prepare_submit_sync (%template);
```

実行

```
$ xcrypt imb-p2.xcr
…
qsub: would exceed group ○○○'s limit
on resource u_count in complex
…
```

- 一度に投入するジョブが多すぎる

→環境によってはエラーになる

→エラーにならないとしても，一度に数百以上のジョブを一度に投入してしまうのは，システムに負荷がかかり好ましくない

  - それくらいジョブスケジューラ側でなんとかすべき，という本音はさておき

一度に投入するジョブ数の制限

imb-p2.xcr の冒頭を以下のように修正・追加

```
use base qw (limit core);
# 一度に投入するジョブ数を最大3に制限
limit::initialize (3);
```

再実行

```
$ xcryptdel --clean
$ xcrypt imb-p2.xcr
…
job-imbp-P32p-Reduce <= running
job-imbp-P32p-Reduce <= done
job-imbp-P32p-Reduce <= finished
```

- 同時に投入されるジョブ数が3に抑えられる
- ジョブが完了して投入中のジョブが3未満になると，自動的に新たにジョブが投入される

imb-p3.xcr
（ジョブごと作ったディレクトリ内でプログラム実行）

```
use base qw (sandbox limit core);
# 一度に投入するジョブ数を最大3に制限
limit::initialize (3);
%template = (
  'RANGE0' => [2,4,8,16,32],
  'RANGE1' => ['Sendrecv','Reduce','Bcast','Alltoall'],  # ベンチマークの種類
  'id@' => sub {"job-imbp-P$VALUE[0]-$VALUE[1]"},        # ジョブ名の命名規則
  'exe0@' => sub {
    "mpiexec.hydra -n $VALUE[0] ../IMB-MPI1 -npmin $VALUE[0] $VALUE[1]"},
                                                         # 実行コマンド
  'JS_node@' => sub {$VALUE[0]},                         # MPIプロセス数
  'JS_cpu' => 1,                                         # コア数/MPIプロセス
);
# Execute the job
@jobs=&prepare_submit_sync (%template);
```

実行

```
$ xcryptdel --clean
$ xcrypt imb-p3.xcr
...
job-imbp-P32-Reduce <= running
job-imbp-P32-Reduce <= done
job-imbp-P32-Reduce <= finished
# 実行終了後，結果確認
$ ls -ltr
# 各ジョブ名に対応するディレクトリがあるはず
# どれかのディレクトリの中身を確認
$ ls job-imbp-P8-Bcast
# 出力ファイルやジョブスクリプトがあるはず
```

xcryptstat: 結果・状況確認コマンド

```
# 状況確認
$ xcryptstat
# 全てのジョブの標準出力を出力
$ xcryptstat --cat stdout
# ファイル名も合わせて出力
$ xcryptstat --ls stdout --cat stdout
# csv形式でいろいろ出力（Excel等で閲覧しやすい）
$ xcryptstat --ls stdout --ls batch --cat stdout --cat
batch --csv > result.csv
# 名前が正規表現にマッチするジョブのみ対象
$ xcryptstat --name ".*P16.*" --ls stdout --ls batch -
-cat stdout --cat batch --csv > result.csv
```

xcryptdel: ジョブの中断・やり直し

```
# すべてやり直し
$ xcryptdel --clean
→ プログラムやスクリプトに必要な修正
$ xcrypt imb-p3.xcr

# 1つのジョブだけキャンセル
$ xcryptdel --cancel job-imbp-P2-Bcast
→ プログラムやスクリプトに必要な修正
# 同じスクリプトを再実行すると，キャンセルしたジョブのみ実行される
$ xcrypt imb-p3.xcr

# 正規表現で複数のジョブを指定してキャンセル
$ xcryptdel --cancel ".*P2.*"
→ プログラムやスクリプトに必要な修正
$ xcrypt imb-p3.xcr
```

Tips

- xcrypt実行中にCtrl+Cを押したり，スパコンとのSSH接続が切断されると，スクリプトの実行が止まってしまう
- ただし，投入ずみのジョブはそのまま実行されたままになる
- 再び，同じディレクトリで同じスクリプトを実行すると，（多くの場合）中断時の実行状態を回復できる
  - 試しに，先ほどのimb-p3.xcrの実行をCtrl+Cで止めて再実行してみてください
- スクリプトを止めずに，SSH接続を切って席を離れたい場合は，screenコマンド等の利用がおすすめ
  - 使い方は Google("GNU screen")等で

ジョブの状態表示の意味

- initialized, prepared: templateの展開により，ジョブオブジェクトが生成された
- submitted: ジョブの投入処理を行った
- queued: 投入したジョブがジョブキューに追加されたことをXcryptが認識した
- running: 投入したジョブが実行状態になったことをXcryptが認識した
- done: 投入したジョブが完了したことをXcryptが認識した
- finished: after手続き等のXcryptの後処理を含め，ジョブが完全に終了した
- aborted: ジョブが異常終了，またはユーザによりキャンセルされた

prepare/submit/sync

- prepare_submit_sync (%template)
  ≡ sync(submit(prepare(%template)))
  - prepare: テンプレートを展開し，ジョブオブジェクトのリストを生成する
  - submit: ジョブオブジェクトのリストを受け取り，ジョブを投入する
  - sync: submitにより投入したジョブの完了を待ち合わせる
- 通常はprepare_submit_syncでよい
- submitの前にジョブオブジェクトを編集したり，submitしたジョブの完了を待っている間，別のPerl処理をやっておきたい場合には分けて呼び出すとよい

PART 3：より高度な機能

拡張モジュール

- Xcryptでは(エキスパートユーザが)様々な機能を拡張できるようにするための*module interface*の仕組みを提供
  - 同時実行ジョブ数の制限(本資料のimb-p2.xcrおよび sample/limit.xcr)
  - ジョブごとにサンドボックスディレクトリを作成(本資料のimb-p3.xcr)
  - ジョブ間の依存関係を宣言的に定義できるようにする(sample/dependency.xcr)

dependencyモジュール

- ジョブ間の依存関係を宣言的に定義できるようにするモジュール
  - `$j1->{depend_on} = [$j2, $j3];`

  → ジョブ`$j2`と`$j3`が終わるまで`$j1`は実行できない

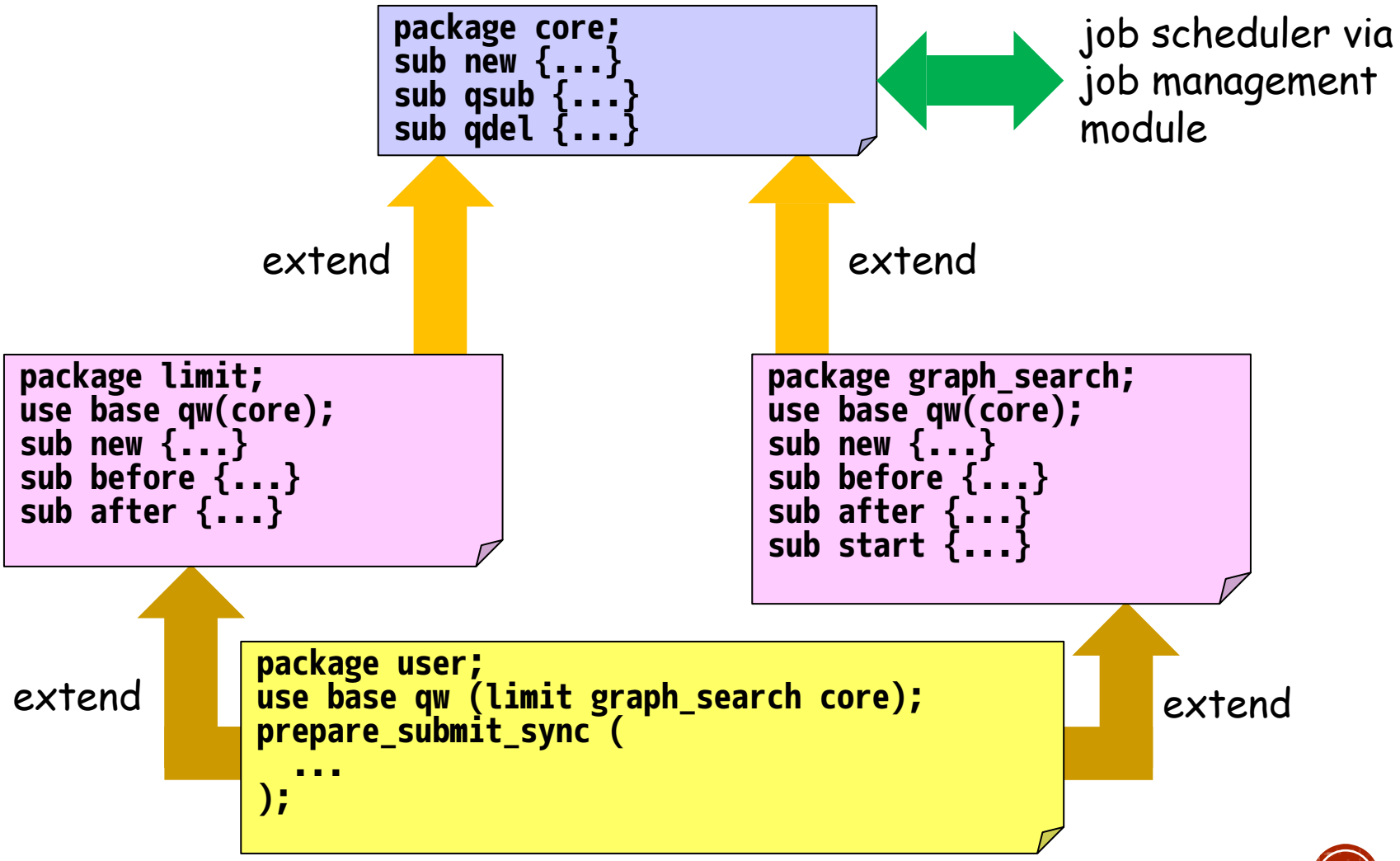
limitモジュールの実装

```
package limit;
use NEXT;
use Coro::Semaphore;

my $smph;
sub initialize {
  $smph = Coro::Semaphore->new($_);
}

sub before {
  $smph->down;
}
sub after {
  $smph->up;
}
```

Xcryptの拡張モジュール機構



spawn-syncスタイル記述

```
use base qw(core);

sub analyze {
    analyze output file (application dependent)
}

foreach $i (0..999) {
  spawn {              # executed in a concurrent job
    system ( “./a.out input$i.dat output$i.dat");
    analyze("output$i.dat"); # time-consuming post processing
  } (JS_node=> 1, JS_cpu => 16);
}
sync;
```

補足：
手動および他のシステムでのセットアップ

最新版手動インストール(1/2)（OBCXの例）

```
# OBCXにログイン

mypc$ ssh myID@obcx.cc.u-tokyo.ac.jp

# Lustreディレクトリに移動（計算ノードから見えるディレクトリにインストールする必要がある）

$ cd /work/gt00/$USER

# XcryptをGithubからダウンロード

% git clone https://github.com/xcrypt-job/xcrypt

# インストールスクリプトを実行

% cd xcrypt

% ./do-install
```

→ 質問には全て何も押さずにEnterキーでよい

最新版手動インストール(2/2)（OBCXの例）

```
# （完了時のメッセージに従って）xcryptコマンドにパスを通す
# 以下はbashの場合
% vi ~/.bashrc
+ export PATH=/work/gt00/$USER/xcrypt/bin:$PATH
→ 再ログインにより設定を反映
# ユーザ設定ファイルを設置・編集
% cp /work/gt00/$USER/xcrypt/etc/xcryptrc ~/.xcryptrc
% vi ~/.xcryptrc
# xcrypt/lib/config にある設定ファイルを指定
- sched = sh
+ sched = obcx
# デフォルトのジョブ制限時間，キュー名，グループ名を指定
+ JS_limit_time = "0:15:0" # 時間指定はhour:min:sec
+ JS_queue = queuename
+ JS_group = groupname
```

動作確認

```
$ cd ~/xcrypt/sample
# サンプルが使用する実行ファイルをmake
$ cd bin
$ make
# サンプルを実行
$ cd ..
$ xcrypt single.xcr
```

→ 正常実行できていることを確認

他のシステムへのセットアップ

- lib/config に設定ファイルが存在するシステムであれば，前述と同様の方法で利用可能
- 設定ファイルが未作成なシステムでは，自分で設定ファイルを書く必要がある
- 既存のファイルやマニュアルを参考に
  - qsub/qdel/qstat のコマンドの定義
  - それらの出力の解釈方法
  - JS_node/cpu/queue/group/limit_time などの設定値からジョブスクリプトの当該部分を生成する処理

  などを書く必要がある
- 可能な限り作成依頼も承ります
  - ジョブの投入方法などが記載された利用マニュアルがあれば可能です（ログイン権限もあると捗ります）

実習

準備1：Xcrypt動作確認

```
# OBCXにログイン
mypc$ ssh myID@obcx.cc.u-tokyo.ac.jp
# Xcryptモジュールをロード
% module load xcrypt
# 起動確認（ヘルプ表示）
% xcrypt --help
# 次回以降，自動的にmodule loadされるようにする
# （ログインシェルがbashの場合）
% nano ~/.bashrc
+ module load xcrypt
```

準備2：実行プログラム＆サンプルスクリプト

```
# 並列ファイルシステムのディレクトリに移動
$ cd /work/gt00/$USER
# Intel MPI Benchmark (IMB)をダウンロード
# Xcryptとは関係ないが，何か動かすプログラム例として
$ git clone https://github.com/intel/mpi-benchmarks
$ cd mpi-benchmarks/src_c/
$ make
# サンプルスクリプトのダウンロード，展開，コピー
$ wget http://super.para.media.kyoto-u.ac.jp/~tasuku/xcrypt-tutorial-202010.zip
$ unzip xcrypt-tutorial-202010.zip
$ cp xcrypt-tutorial-202010/*.xcr .
```

実習

- Part 1, Part 2のスライドの内容に沿って，コマンド実行やプログラム・設定ファイルの編集を進めていきましょう
- 準備を含めた詳細な手順は，サンプルスクリプトzip（xcrypt-tutorial-202110.zip）内のtutorial.txt に記載しています
  - 基本的にこれに書いている手順を上からやっていけばOKです

まとめ

- Xcryptの基本機能な利用方法を実習により体験していただきました
- Perlの本格的なコードを組み合わせることで，さらに複雑な処理も可能になります
  - より複雑な依存関係をもつジョブの実行
  - 性能評価グラフの自動生成，など
- 今回の簡単なパラメータスイープのみの利用でも，有用性は高いと思います
- なお，Wisteriaにもインストール済ですので，
  `$ module xcrypt`
  を実行したのち，OBCXと同様に利用可能です

- 新しいシステムへの対応，その他機能要望，質問，バグ報告等気軽にご連絡ください