

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

京都大学 学術情報メディアセンター

平石 拓

Mail: `tasuku@media.kyoto-u.ac.jp`

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「たくさんのジョブを投げる作業」をなるべく手軽にやる方法について実習します
- 基本的にこのスライドは書いてある通りにコマンド実行すれば実習を進められるようになっていきますので、予定していた実習が最後まで終わらなかった場合は帰ってからお試しください。

本講習会の資料

- このスライド (PDF)

- <https://u.kyoto-u.jp/toapr>

- 本講習会のページにも掲載されるはず

- 講習で使うサンプルコード (コードの手入力が面倒な人向け)

- Reedbush上の

- `/lustre/gt00/z30112/xcrypt-tutorial-201909.zip`

- または以下のように `wget` で取得

- `% wget http://super.para.media.kyoto-u.ac.jp/~tasuku/xcrypt-tutorial-201909.zip`

目次

- Xcryptとは
- 実習1: 基本操作 (ジョブ投入・確認・中止)
 - Reedbushで動作確認
 - スクリプトの書き方の基本的な説明
- 実習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
- 実習3: 他のスパコンへでの利用
 - Oakforest-PACS へのインストール
 - Reedbush への (最新版の) インストール
- より高度な機能の紹介

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
- 特徴
 - 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
 - その他,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

こんなときに便利です

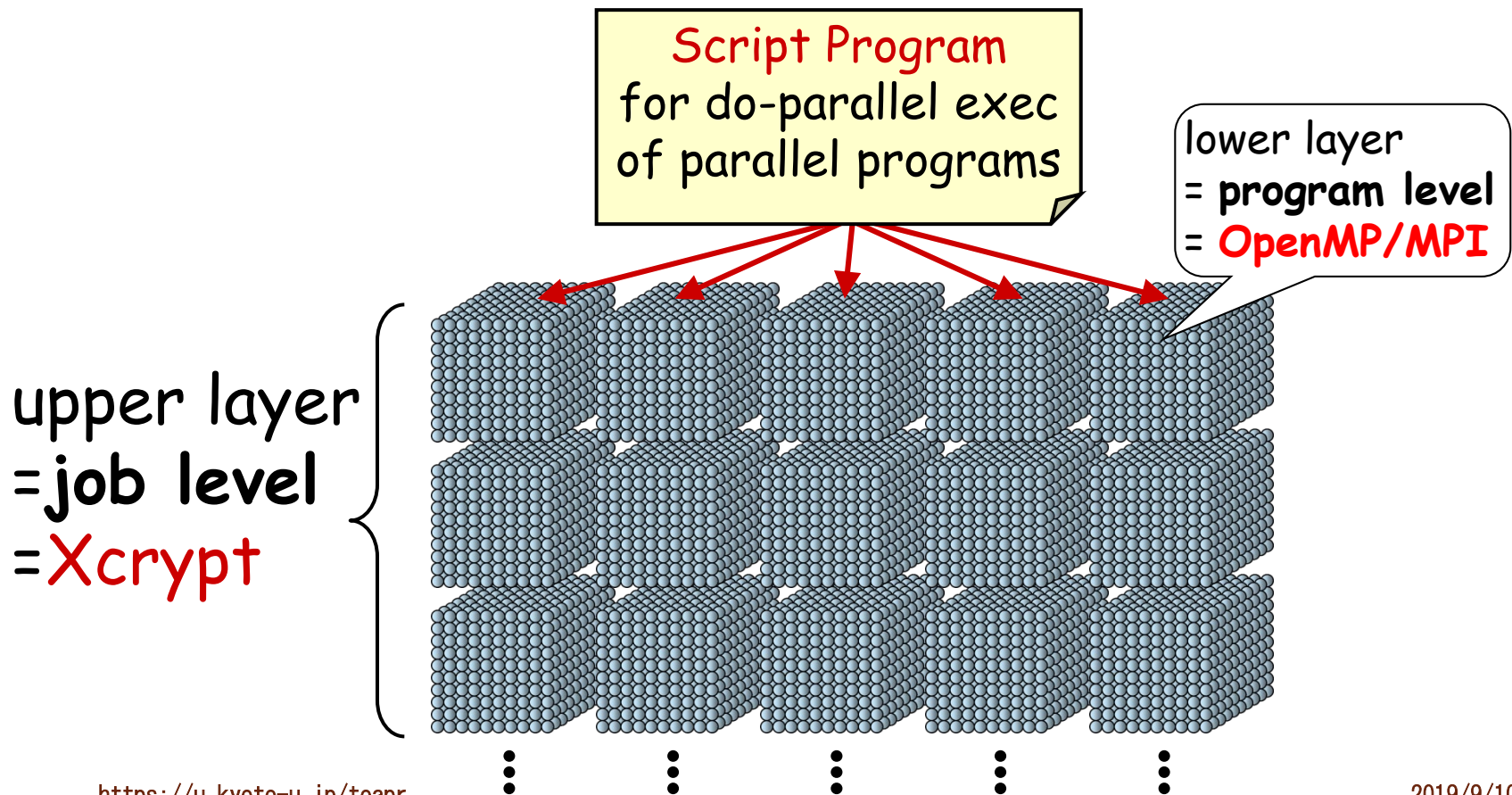
- ある単一プログラムを何度も実行したい
 - 試したい入力が何通りもある
 - 問題サイズや並列数を変えながら性能評価をとりたい
(それぞれk回実行して中央値を...)
- 依存関係がなければ, 大量のノードを使って同時に
- 依存関係があれば, それに従って

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)並列化されていない場合、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...

```
#!/bin/sh
#!/bin/sh
#!/bin/sh
#!/bin/sh
#!/bin/sh
#!/bin/sh
# @%-lP 1 # number of nodes
# @%-lp 16 # number of cores
# @%-lm 28G # memory size
# @%-q g10034 # queue name
# @%-g g10034 # group name

./a.out input6.dat output6.dat

> qsub script6.sh
```


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)
<https://github.com/xcrypt-job/xcrypt>
- プロジェクトWebページ
 - 過去の論文, 講演資料などがある
 - このチュートリアルでは, 設計理念などの話にはあまり触れないので, 興味を持っていた方はこちらで
<http://super.para.media.kyoto-u.ac.jp/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
 - Reedbush @東大
 - Grand Chariot / Polaire @北大
 - ~~京 @RIKEN R-CCS~~
<http://www.aics.riken.jp/jp/k/aics-software/>
- 自分でインストールすることで利用可
 - Oakforest-PACS@東大
 - Camphor2, Laurel2@京大
 - HOKUSAI@RIKEN R-CCS etc.
- それ以外のシステムでは、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



実習1: 動作確認

動作環境確認@Reedbush

Reedbushにログイン

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

module名の確認

```
% module -s avail xcrypt
```

```
--- /lustre/app/modulefiles/tools ---
```

```
xcrypt/f994ff272f56
```

```
% module load xcrypt/f994ff272f56
```

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

(ログインシェルがbashの場合)

```
% vi ~/.bashrc
```

```
+ module load xcrypt/f994ff272f56
```

f99...は将来の更新で変更の可能性アリ

実行プログラムの準備

本実習では, Intel MPI Benchmark (IMB)を使う

/lustre/グループ名/ユーザ名 に移動(Reedbushの場合必須)

\$ cdw

Google等で「Intel IMB github」くらいで検索すると見つかる

<https://github.com/intel/mpi-benchmarks>

\$ git clone <https://github.com/intel/mpi-benchmarks>

\$ cd mpi-benchmarks/src_c/

デフォルトのMakefileはmpiccでコンパイル

\$ make

スクリプト作成

サンプルスクリプトをコピー(最後の". "を忘れずに)

```
$ cp /lustre/gt00/z30112/xcrypt-tutorial-201909/imb-s.xcr .
```

内容確認・編集(好きなエディタで)

```
$ emacs imb-s.xcr
```

■ Tips

- 拡張子は `.xcr` でなければならない
- Emacsを使う場合は, `.emacs`に以下の設定を足しておくとい

```
(add-to-list 'auto-mode-alist (cons "¥¥.xcr" 'perl-mode))
```

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 = 36;
%template = (
    'id' => 'job-imbs',    # 任意のジョブ名
    'exe0' => "mpirun -np 2 ./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 u-tutorial)
    'JS_group' => 'groupname',      # グループ名(講習会では gt00)
    # ジョブ実行前後のメッセージ出力
    'before' => sub { print "Submitting $self->{id}¥n"; },
    'after' => sub { print "$self->{id} finished¥n"; },
);
# 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
Submitting job-imbs
job-imbs <= submitted
job-imbs <= queued
job-imbs <= running
job-imbs <= done
job-imbs finished
job-imbs <= finished
```

ジョブの結果確認

できたファイルの確認

```
$ ls -ltr
```

...

```
-rw-r--r-- ... job-imbs_reedbush.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_reedbush.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-imbs_stderr
```

ジョブの標準出力

```
$ cat job-imbs_stdout
```

パラメータの意味

(※)物理ノード数の直接指定に対応しているシステムのみ(Reedbush,は可). 省略時, JS_nodeとJS_cpuから自動算出

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
exen	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.
argn_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exenの後ろに連結される
JS_node, JS_cpu, JS_memory, JS_thread ※ReedbushではJS_memoryは無視される	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	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

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 select=...`) の記述が自動生成される

プロセス数変更 (imb-s.xcrを編集)

```
use base qw (core);      # とりあえず「おまじない」と思う
my $procs = 72;
%template = (
    'id' => 'job-imbs',    # 任意のジョブ名
    'exe0' => "mpirun -np 2 ./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 u-tutorial)
    'JS_group' => 'groupname',    # グループ名(講習会では gt00)
    # ジョブ実行前後のメッセージ出力
    'before' => sub { print "Submitting $self->{id}¥n"; },
    'after' => sub { print "$self->{id} finished¥n"; },
);
# 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_reedbush.bat` や実行結果が更新されていることを確認

```
% cat job-imbs_reedbush.bat
```

→ `-l select=` 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
- `xcryptdel --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消去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-p72',
```

```
% xcrypt imb-s.xcr
```

```
...
```

```
job-imbs-p72 <= 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毎に異なる
- スクリプトを別のシステムで動かすときにいちいち書き換えるのは面倒なので、システムごとのデフォルト値を定義しておく

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 /lustre/app/xcrypt/tasuku-xcrypt-f994ff272f56/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 = queue_name
```

```
+ JS_queue = queue_name 講習会では u-tutorial
```

```
- # JS_group = group_name
```

```
+ JS_group = group_name 講習会では gt00
```

→ .xcr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のでお好みで
- 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



実習2: 複数ジョブ実行と 性能評価への応用

複数ジョブの実行

imb-s.xcr をベースにするのでコピー

```
% cp imb-s.xcr imb-p.xcr
```

このファイルを編集 (好きなエディタで)

```
% emacs imb-p.xcr
```

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 {"mpirun -np $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
```

修正のポイント

- `RANGEn` 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)
- `RANGEn` で定義した各値によって, 設定したい値が異なるパラメータ `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, (ジョブごとに異なる) `RANGEn` 中の各値を `$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 `RANGEn` 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 {"mpirun -np $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  
pz0112's limit on resource  
u_count in complex
```

...

- 一度に投入するジョブが多すぎる

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);
```

再挑戦

```
$ xcryptdel --clean
```

```
$ xcrypt imb-p2.xcr
```

```
...
```

```
job-imbp-P32p-Reduce <= running
```

```
job-imbp-P32p-Reduce <= done
```

```
job-imbp-P32p-Reduce <= finished
```

- これで成功するはず

imb-p3.xcr (ジョブごとにディレクトリを作り, そこでプログラム実行)

```
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 {"mpirun -np $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: ジョブのやりなおし

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-p2.xcr
```

1つのジョブだけキャンセル

```
$ xcryptdel --cancel job-imbp-P64p8
```

→ 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-p2.xcr
```

正規表現で複数のジョブを指定してキャンセル

```
$ xcryptdel --cancel ".*P16.*"
```

→ プログラムやスクリプトに必要な修正

```
$ xcrypt imb-p2.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-p1.xcrの実行をCtrl+Cで止めて再実行してみるとよい
- 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

ジョブの状態表示について

- 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手続きが完了し、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



実習3: 他のシステムへの セットアップ

Oakforest-PACSでのセットアップ(1/2)

Oakforest-PACSにログイン

```
mypc$ ssh myID@ofp.jcahpc.jp
```

working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/groupname/$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

Oakforest-PACSでのセットアップ(2/2)

```
# (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/groupname/$USER/xcrypt/bin:$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/groupname/$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 = ofp
# 以下はreedbushの設定と同様
+ JS_limit_time = "15:0" # 時間指定は[[hour:]min:]sec
+ JS_queue = queuename # OFPではリソースグループ(debug-flatなど)
+ JS_group = groupname # OFPではプロジェクトコード
```

Reedbushでのセットアップ(1/2)

Reedbushにログイン

```
mypc$ ssh myID@reedbush-u.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)

```
$ cdw
```

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

Reedbushでのセットアップ(2/2)

```
# (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=cdwで表示されるディレクトリ名/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 cdwで表示されるディレクトリ名/xcrypt/etc/xcryptrc ~/.xcryptrc
% vi ~/.xcryptrc
# ~/xcrypt/lib/config にある設定ファイルを指定
- sched = sh
+ sched = reedbush
# 以下はOakleafの設定と同様(ただし、時間の指定方法が異なる)
+ 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
 - 利用マニュアル(できればログイン権限)をください



より高度な機能の紹介

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, $j3];`
 - `$j2` and `$j3` cannot be in executed until `$j1` is finished
- ジョブ `$j1` が終わるまで `$j2` と `$j3` は実行できない

“limit” モジュールの実装

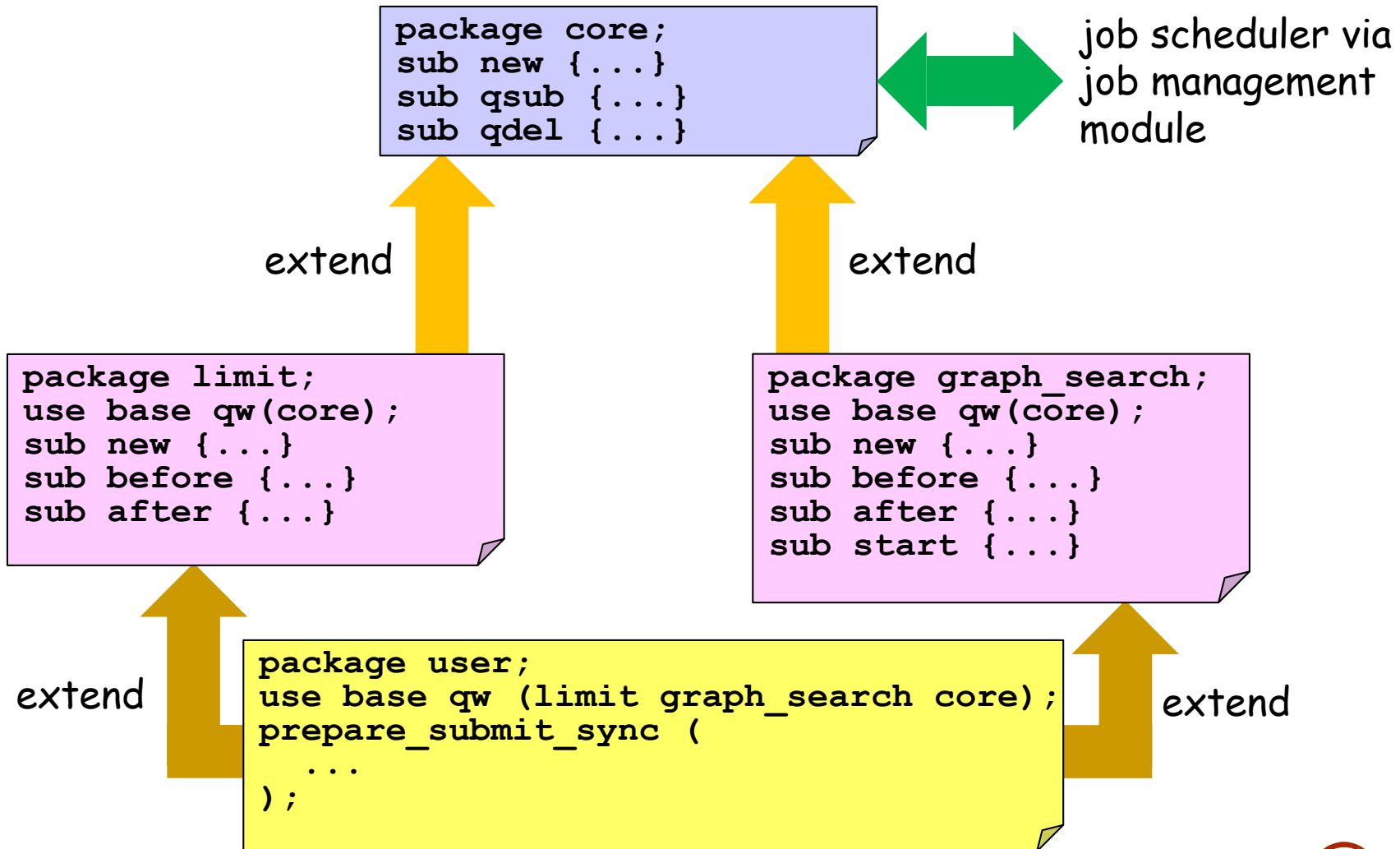
```
package limit;
use NEXT;
use Coro::Semaphore;

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;
```

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
- ただ、今回の簡単なパラメータスイープのみでも、有用性は高いと思います
- 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