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

ver 1.1

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/ix-dt
  - 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
  - Reedbush上の
    ~z30112/xcrypt-tutorial-201809.zip
    または上記スライドのURLから

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

- あるプログラムを（パラメータを変えつつ）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…）
- ↑の処理をいろんなスパコンでしたい
- ↑の処理の実行間に依存関係がある
- ↑の処理の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
- ↑の処理の一部を手軽に再実行したい（失敗，やりなおし）

などのどれかに該当すれば

Xcryptの情報源・入手先など

- 最新版ダウンロード（Bitbucket）
  https://bitbucket.org/tasuku/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，興味を持っていただいた方はこちらで

  http://super.para.media.kyoto-u.ac.jp/xcrypt/index.html
- マニュアル
  - PDFマニュアル（細かい話）
    - アーカイブ内 `doc/tutorial/xcrypt-manual.pdf`
  - --help オプション

```
$ xcrypt --help
$ xcryptstat --help
$ xcryptdel --help
```

Xcryptが利用できるシステム

- 導入済み
  - Reedbush@東大
  - 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

9 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ロード
# https://reedbush-www.cc.u-tokyo.ac.jp/
# →「ドキュメント閲覧」→「Xcrypt利用方法」に従う
# （f994ff272f56部分はバージョンにより異なる）
% 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
```

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
#  IMBの配布ページ https://github.com/intel/mpi-benchmarks
#  の下の方にある”Clone or download”→”Download ZIP”
#  のURLをコピー&ペーストして以下を実行
$ wget https://github.com/intel/mpi-benchmarks/archive/master.zip
$ unzip master.zip
$ cd mpi-benchmarks-master/src/
#  デフォルトのMakefileはmpiiccでコンパイル
$ make
```

スクリプト作成

```
# テンプレートとして利用するため，サンプルをコピー
$ which xcrypt
/lustre/app/xcrypt/tasuku-xcrypt-f994ff272f56/bin/xcrypt
$ cp /lustre/app/xcrypt/tasuku-xcrypt-f994ff272f56/sample/single.xcr imb-s.xcr
# 編集（お好きなエディタで）
$ emacs imb-s.xcr
```

色違いの部分が共通なのでコピペ

- 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よい

```
(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
%template = (
   'id' => 'job-imbs',    # 任意のジョブ名
   'exe0' => 'mpirun -np 2 ./IMB-MPI1 -npmin 2', # 実行コマンド
   'JS_node' => 2,   # MPIプロセス数
   'JS_cpu' => 1,    # コア数/MPIプロセス
   'JS_limit_time' => '00:10:00', # 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）
  # 以下は変更なし（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-imgs_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
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, 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 $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' => '00:10:00', # 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）
  # 以下は変更なし（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によって同定される)ジョブは，実行済の場合スキップされる．そのため，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，`xcerypt`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（idの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
$ vi ~/.xcryptrc
- # JS_limit_time = 3600
+ JS_limit_time = ”00:10:00”
- # JS_queue = queuename
+ JS_queue = queuename      講習会では u-tutorial
- # JS_group = groupname
+ JS_group = groupname      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

- 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）
- RANGE*n* 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） RANGE*n* 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);
use POSIX qw(floor ceil);
my $NCORE = 36;      # ノードあたりのコア数@Reedbush
%template = (
   'RANGE0' => [2,4,8,16,32],    # プロセス数の集合
   'RANGE1' => [1,2,4,8,16],   # プロセス数/物理ノードの集合
   'id@' => sub {"job-imbp-P$VALUE[0]p$VALUE[1]"},
                                 # ジョブ名の命名規則
   'exe0@' => sub {"mpirun -np $VALUE[0] ./IMB-MPI1 -npmin $VALUE[0]"},
   'JS_node@' => sub {$VALUE[0]},   # MPIプロセス数
   'JS_cpu@' => sub {floor($NCORE/$VALUE[1])},    # コア数/MPIプロセス
   'JS_phnode@' => sub { $VALUE[0]/$VALUE[1]; }, # 物理ノード数
);
# Execute the jobs
@jobs=&prepare_submit_sync (%template);
```

とりあえず実行

```
$ xcrypt imb-p2.xcr
job-imbp-P32p1 <= initialized
job-imbp-P32p1 <= prepared
job-imbp-P32p1 <= submitted
qsub: Error : Number of nodes exceeds your limit(=8).
...
```

- 要求ノード数が多すぎ＆1未満のものが

不適切なジョブの除外

imb-p2.xcr の
@jobs=&prepare_submit_sync (%template);
の部分を以下のように修正

```
# テンプレートの展開のみを行う
@jobs=&prepare(%template);
print "jobs¥n";
foreach $j (@jobs){print "n=$j->{JS_node}: p=$j->{JS_phnode}¥n";}
# ジョブ集合@jobsから，物理ノード数が4を超えるものと，
# 総プロセス数がノードあたりプロセス数未満のものを取り除く
@jobs2=grep{$_->{JS_phnode}>=1 && $_->{JS_phnode}<=4} @jobs;
print "jobs2¥n";
foreach $j(@jobs2){print "n=$j->{JS_node}: p=$j->{JS_phnode}¥n";}
# 上記フィルターで残ったジョブのみ投入
submit_sync(@jobs2);
```

まだ怒られる(ことがある？)

```
$ xcryptdel --clean
$ xcrypt imb-p2.xcr
…
qsub: Maximum number of jobs
for group group already in
complex
…
```

- 一度に投入するジョブが多すぎ？

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

```
use base qw (limit core);
use POSIX qw (floor ceil);
# 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-P32p8 <= finished
job-imbp-P32p16 <= done
job-imbp-P32p16 <= 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);
use POSIX qw(floor ceil);
limit::initialize (3);
my $NCORE = 36;      # ノードあたりのコア数@Reedbush
%template = (
  'RANGE0' => [2,4,8,16,32],    # プロセス数の集合
  'RANGE1' => [1,2,4,8,16],  # プロセス数/物理ノードの集合
  'id@' => sub {"job-imbp-P$VALUE[0]p$VALUE[1]"},
                                # ジョブ名の命名規則
  'exe0@' => sub {"mpirun -np $VALUE[0] ../IMB-MPI1 -npmin
$VALUE[0]"},
  'JS_node@' => sub {$VALUE[0]},  # MPIプロセス数
  'JS_cpu@' => sub {floor($NCORE/$VALUE[1])},    # コア数/MPIプロセス
  'JS_phnode@' => sub { $VALUE[0]/$VALUE[1]; }, # 物理ノード数
);
```

… 以下，imb-p2.xcrと同じ …

実行

```
$ xcryptdel --clean
$ xcrypt imb-p3.xcr
…
job-imbp-P32p8 <= finished
job-imbp-P32p16 <= done
job-imbp-P32p16 <= 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-P32p8
# 出力ファイルやジョブスクリプトがあるはず
```

imb-p3b.xcr（RANGE*n*の値として文字列を指定）

```
use base qw (sandbox limit core);
use POSIX qw(floor ceil);
limit::initialize (3);
my $NCORE = 36;      # ノードあたりのコア数@Reedbush
%template = (
  'RANGE0' => [2,4,8,16,32], # プロセス数の集合
  'RANGE1' => [1,2,4,8,16],  # プロセス数/物理ノードの集合
  'RANGE2' => ['SendRecv', 'Allreduce', 'Alltoall', 'Bcast'], # ベンチマーク名
  'id@' => sub {"job-imbp-P$VALUE[0]p$VALUE[1]_$VALUE[2]"},
                                  # ジョブ名の命名規則
  'exe0@' => sub {"mpirun -np $VALUE[0] ../IMB-MPI1 -npmin $VALUE[0] $VALUE[2]"},
  'JS_node@' => sub {$VALUE[0]},  # MPIプロセス数
  'JS_cpu@' => sub {floor($NCORE/$VALUE[1])},   # コア数/MPIプロセス
  'JS_phnode@' => sub { $VALUE[0]/$VALUE[1]; }, # 物理ノード数
);
```

… 以下, imb-p2.xcrと同じ …

結果確認

```
# 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
```

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)` ≡ `prepare(submit(sync(%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をダウンロード

# https://bitbucket.org/tasuku/xcrypt →「ダウンロード」→「リポジトリをダウンロードする」 の

# URLをコピー＆ペースト（8bc…の記号列はバージョンにより異なる）

% wget https://bitbucket.org/tasuku/xcrypt/get/8bc2679debcc.zip

# 展開＆適当にディレクトリ名変更

% unzip 8bc2679debcc.zip

% mv tasuku-xcrypt-8bc2679debcc 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をダウンロード
# https://bitbucket.org/tasuku/xcrypt →「ダウンロード」→「リポジトリをダウンロードする」 の
# URLをコピー＆ペースト（8bc…の記号列はバージョンにより異なる）
% wget https://bitbucket.org/.../8bc2679debcc.zip
# 展開＆適当にディレクトリ名変更
% unzip 8bc2679debcc.zip
% mv tasuku-xcrypt-8bc2679debcc 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（利用マニュアルを送ってください）

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

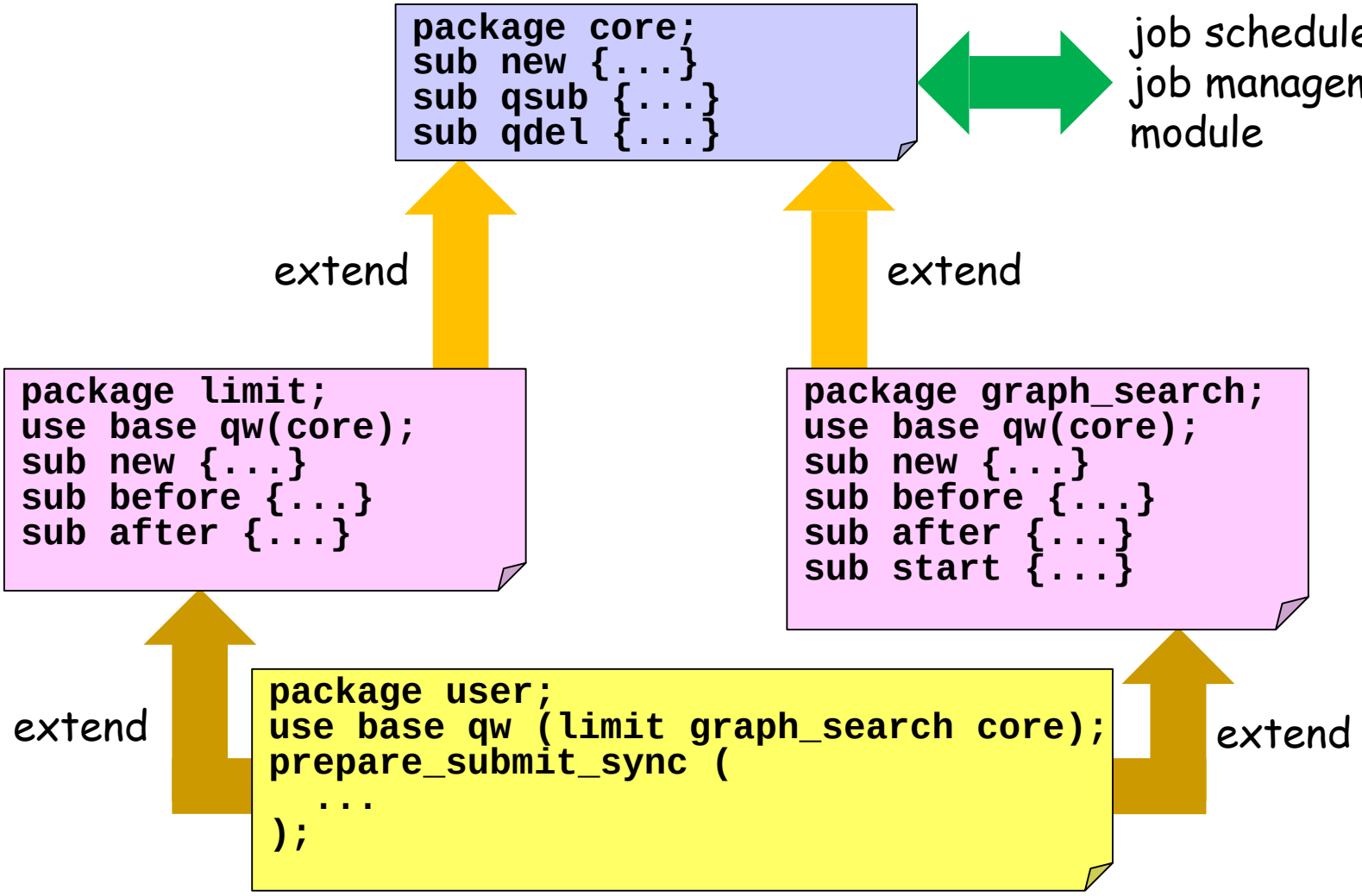
“limit” モジュールの実装

```
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で本格的なコードを書くことにより，さらに様々な処理も可能になりますが，今回紹介した簡単なパラメータスイープのみでも，人によっては有用性が高いと思います

- 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