Xcrypt Tutorial

平石 拓, 安部 達也, 三宅洋平
岩下 武史, 中島 浩
京都大学 学術情報メディアセンター

2010/2/18

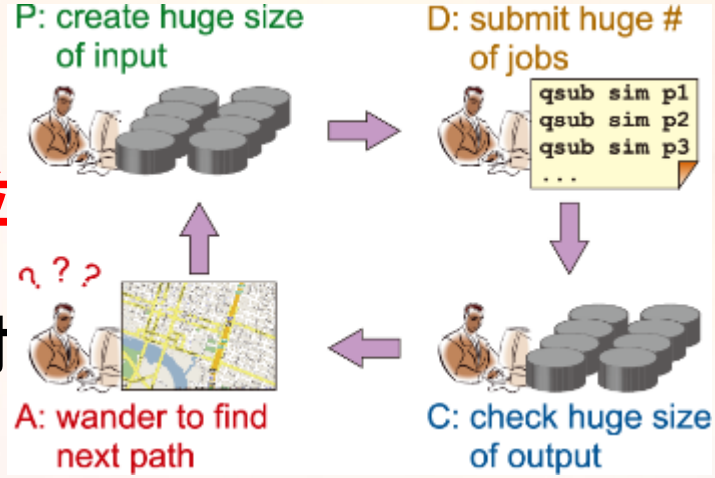
背景

- スパコンにおけるシミュレーション
  - 創薬，車体設計，天気予報
- パラメータスイープ，最適パラメータ探索

→ 同一のプログラムを多数の入力に対し繰り返し適用
=*PDCAサイクル*

  - Plan: 入力を生成
  - Do: ジョブ投入 (タスク並
  - Check: 結果を処理
  - Action: 次の計算を検討

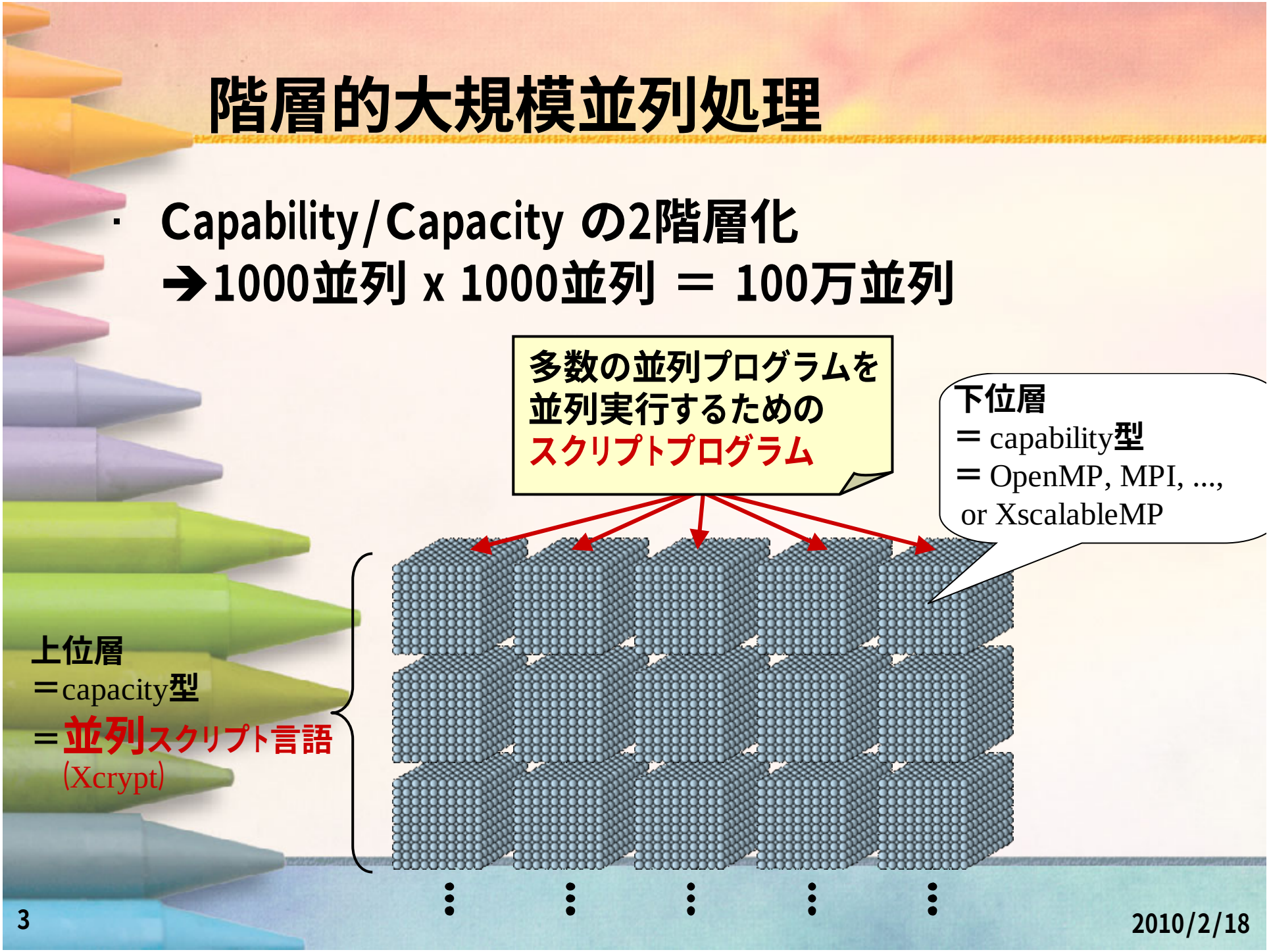
の繰り返し



- 自動化すべき
  - ワークフロー，出力から次の入力を生成

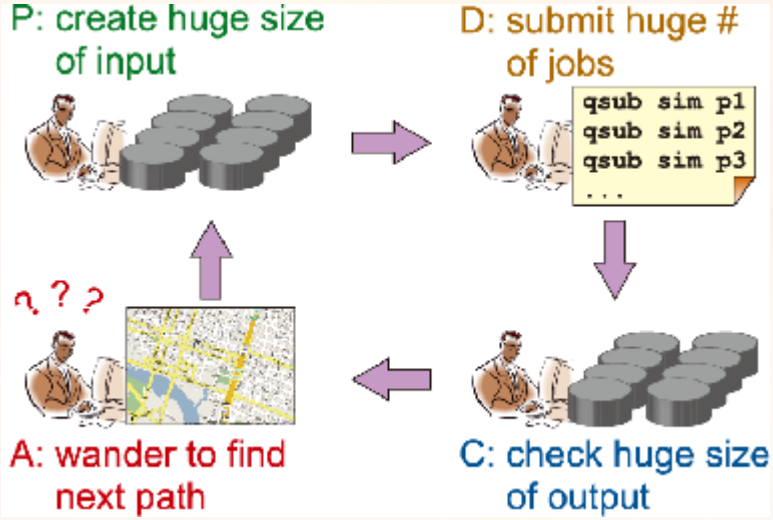
階層的大規模並列処理

- Capability/Capacity の2階層化
  ➔1000並列 x 1000並列 ＝ 100万並列



現状と目標

- 手動 (非自動化)
- ジョブの実装言語 (FortranやC) で自動化
  - 「計算科学者」に親しみのある言語
  - 生産性が低い (特に，複雑なフローへの対応)
- 汎用のスクリプト言語
  - シェルスクリプト，Perl，Ruby，．．．
  - 汎用性高，生産性も悪くない
  - ジョブ並列の記述には意外と面倒
    - 非同期に実行されるジョブの管理とか
    - 特に「計算科学」研究者にとって
- 専用スクリプト言語 (DSL)：Xcrypt
  - タスク並列処理の記述を楽にする言語機能・ライブラリを提供
  - 手軽な記述と汎用性を両立
- ワークフローツール
  - 手軽な記述 (GUI-based)
  - 複雑なフローの記述は困難



What is Xcrypt?

- バッチスケジューラ (NQS, SGE, Torque, ...) 上のジョブ並列処理を記述するためのスクリプト言語
- (大量の) ジョブの投入, 待ち合わせ, 結果解析などが簡単に書ける
- Perlベースの手続き型記述
  - たとえば
    入力準備→ジョブの投入→結果解析
    の繰り返しがwhileループで書ける

設計思想

- 簡便性
  - 想定ユーザ ＝ 計算科学研究者 (≠計算機科学研究者)
  - 典型的な処理を手軽に書ける
    - 簡単なパラメータスイープを10行前後
  - 既存のプログラムに手を加えなくてもよい
  - スクリプトに手を加えなくても様々な環境で動作
    (多様なバッチスケジューラに対応)
- 柔軟性，拡張性
  - 複雑なことも書こうと思えば書ける
  - *Perl "wizards"* can add various helpful *spells* as *modules*
- 可搬性
  - 既存のプログラムが改変なしに動く
  - 様々な環境 (バッチスケジューラ，ファイルシステム) で動く

Xcryptの設計・実装

- Perlのライブラリ群として実装
  - 既存言語ベース→開発・習得が用意
  - 柔軟性が高い
    - eval関数, オブジェクトへの動的なメンバ追加
  - Ruby, Pythonよりは馴染みやすい？
- Perlのオブジェクト指向機能を利用
  - ジョブ ＝ オブジェクト
  - 拡張モジュール ＝ クラス拡張
    - 多重継承により複数のモジュールを同時適用
  - エンドユーザは「オブジェクト指向」は気にしなくてよい

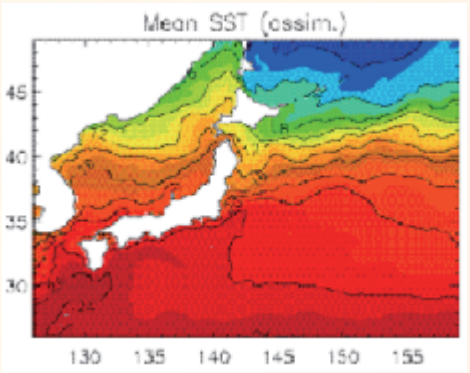
想定する用途

- パラメータスイープ
  - 異なる多数の入力ファイルに対して同一のプログラムを適用する
- 最適パラメータ探索 (二分法 etc.)
  1. 入力ファイルを生成
  2. ジョブを投入
  3. 出力ファイルを解析
  4. 結果が収束していなければ 1. に戻る

  の繰り返し
- 単に，システムごとにジョブスクリプトを用意するのが面倒なのをどうにかしたい

例：アンサンブルシミュレーション

- 異なる多数の初期パラメータに対して同一の予測計算
  (1アンサンブル=1ジョブ)
- 最後に結果をまとめる



動作要件

- Perl 5.8以上 (5.10以上を推奨)
- バッチスケジューラが導入されたシステム
  - qsub, qstat, qdel相当のコマンドが動作
- ログインノードと計算ノードが次のいずれかの方法で通信可能
  - TCP/IP
    - ポート番号は任意, デフォルトは9999
  - NFSによるファイル経由
    - ログインノードと計算ノードが同一絶対パス指定できる, ユーザ権限で読み書きできるディレクトリが存在すること

動作要件（補足）

- スクリプトの動作を確認してみたいだけなら，バッチスジューラがない環境（手元のノートPCなど）でも，一応動かせる
  - OSスケジューラモード

バッチスケジューラ	OSスケジューラ
qsub	sh
qstat	ps
qdel	kill
request ID	process ID

- 現状Cygwinでは動きません
  （利用しているCPANパッケージが原因）

インストール (SCore環境)

- RPMパッケージが用意されている
  - score7-src/eScience/xcrypt
  - bininstallコマンドでインストール
- SCoreはデフォルトのバッチスケジューラが用意されていないので，OSスケジューラモードでインストールされる
  - 自分でバッチスケジューラをインストールして，それを使うようにすることは可能 (後述)

インストール (SCore以外)

1. 以下から .tar.gz をダウンロード，解凍
   (SCore版より新しいです)
   http://super.para.media.kyoto-u.ac.jp/~tasuku/xcrypt/
2. xcrypt-*x.x*/source-me.sh の環境変数を編集

環境変数	設定値
XCRYPT	xcrypt-*x.x*/ の絶対パス
XCRJOBSCHED	利用するバッチスケジューラ

   - デフォルトでサポートするXCRJOBSCHEDの値
     - TSUKUBA (T2K筑波，SGE)，TOKYO (T2K東大，NQS)，KYOTO (T2K京大，NQS)
     - SGE (普通にインストールしたSGE)
     - sh (OSスケジューラモード)：設定ファイルの編集が必要？
   - これ以外の環境で動かすためには自分で設定ファイルを追加する必要がある (後述)

インストール (SCore以外・続き)

3. 環境変数の設定を適用

% cd xcrypt-*x.x*

% source source-me.sh

- 今後のログイン用にこのファイルの内容を .bashrc などに追加しておくとよい

4. CPANパッケージのインストール (xcryptのディレクトリ以下にインストールされる)

% cd CPAN

% ./do-install.sh

ドキュメント

- xcrypt-*x.x*/doc/Xcrypt_manual.pdf
  - インストール, 起動方法
  - 言語の機能
  - 実装内部の一部
  - 拡張モジュールの書き方

  などが一通り書いてあります

起動方法

% **xcrypt** *myscript*.**xcr** *options*

- 主なコマンドラインオプション

オプション	意味	デフォルト
--port *num*	Xcryptプロセスとジョブが TCP/IPで通信するポート番号 (0：NFS経由)	9999
--localhost *name*	port>0 のとき，計算ノードからログインノードにアクセスするホスト名	hostnameコマンドの出力結果
--inventory-path *pathname*	各ジョブの状態をファイルとして保存する場所	./inv_watch
--verbose *n*	verbosity	0

How to write an Xcrypt script

1. Preamble
   - 利用したいモジュールの指定
   - 必要であればグローバル変数の定義 など
2. ジョブテンプレートの定義
   - ジョブに関する情報を宣言的に記述
   - 1つのテンプレートからジョブ「列」を生成可能
3. ジョブオブジェクトの生成・投入・終了待ち処理を手続き的に書く
   - prepare()：ジョブオブジェクト（列）生成
   - submit()：ジョブ投入
   - sync()：ジョブ終了待ち

例1：1つだけジョブを投入する

```
use base qw (core);

%template = (
    ‘id’ => ‘example’,
    'exe' => './a.out',
    ‘arg0’ => ’input’,
    ‘arg1’ => ’output’,
    'copiedfile0' => 'a.out',
    'copiedfile1' => 'input',
    ‘queue’ => ‘gh*xxxxx*’,
    ‘group’ => ’gh*xxxxx*’,
)
@jobs = prepare (%template);
submit(@jobs);
sync(@jobs);
```

ジョブには任意のジョブIDをつける
実行ファイル
コマンドライン引数（ここでは入出力ファイル名）
実行に必要なファイル（copy対象）
ジョブを投入するキュー
（T2K京大ではグループ名も必要）
作業ディレクトリ作成
ジョブ投入
ジョブ終了待ち

実行

```
% xcrypt single.xcr
example <= active
example <= prepared
example <= submitted
example id <= 8011
example <= queued
example <= running
example <= done
example <= finished
check_and_write_aborted:
%
```

ジョブオブジェクト生成

作業ディレクトリ準備終了

ジョブ投入実行（成否はまだ不明）

ジョブのリクエストID

ジョブ投入成功

プログラム実行開始

プログラム実行完了

ジョブ実行完了

実行結果

% ls

a.out example/ input inv_watch/ single.xcr*

% cd example　作業ディレクトリ

% ls

KYOTO.sh a.out* input invwrite-sock.log*

output request_id stderr stdout

ワーキングディレクトリの中身

- KYOTO.sh
  - Xcryptが生成したジョブスクリプト
- invwrite-sock.log
  - ジョブプロセスとxcryptプロセスの通信ログ
- request_id
  - qsubしたときのリクエストIDの番号
- stdout, stderr
  - バッチスケジューラが生成した標準出力，標準エラー出力ファイル

キュー指定に関する注意

'queue' => 'gh*xxxxx*',

'group' => 'gh*xxxxx*',

は，T2K京大でのキューの指定方法

- 京大は，キュー名とグループ名の両方必須（利用手続きの形態による）
- 東大は，キュー名のみ必須
- 筑波は，グループ名のみ必須（ただし，ノード数および制限時間も要指定）

例2a：複数の独立なジョブを投入

```
use base qw (core);

%template = (
    ‘id’ => ‘example’,
    ’RANGE0’ => [1..5],
    'exe' => './a.out',
    ‘arg0@’ => ’”input$R0”’,
    ‘arg1@’ => ’”output$R0”’,
    'copiedfile0' => 'a.out',
    'copiedfile1@' => ‘”input$R0”’,
    ‘queue’ => ‘gh*xxxxx*’,
    ‘group’ => ’gh*xxxxx*’,
)
@jobs = prepare (%template);
submit(@jobs);
sync(@jobs);
```

パラメータの範囲

ジョブごとに異なる値
→ メンバ名の末尾に@
→ $R0 が 1〜5 に置き換わる

作業ディレクトリ作成

ジョブ投入

ジョブ終了待ち

例2b：複数の独立なジョブを投入（関数版）

```
use base qw (core);

%template = (
    ‘id’ => ‘example’,
    ’RANGE0’ => [1..5],
    'exe' => './a.out',
    ‘arg0@’ => sub {”input$_[0]”},
    ‘arg1@’ => sub {”output$_[0]”},
    'copiedfile0' => 'a.out',
    'copiedfile1@' => sub {”input$_[0]”},
    ‘queue’ => ‘gh*xxxxx*’,
    ‘group’ => ’gh*xxxxx*’,
)
@jobs = prepare (%template);
submit(@jobs);
sync(@jobs);
```

パラメータの範囲

ジョブごとに異なる値
→ メンバ名の末尾に@
→ $_[0] が 1〜5 に置き換わる

作業ディレクトリ作成

ジョブ投入

ジョブ終了待ち

実行

```
% ls
a.out*  input1 input2 input3 input4 input5
psweep.xcr
% xcrypt psweep.xcr
example_1 <= active
example_2 <= active
example_3 <= active
example_4 <= active
example_5 <= active
example_1 <= prepared
example_2 <= prepared
...
example_5 <= finished
%
```

実行結果

```
% ls
example_1/ example_2/ … example_5/ …
% cd example_2
% ls
KYOTO.sh a.out input2 output2 …
%
```

追加モジュールを使う

```
use base qw (limit core);
limit::initialize(1);
%template = (
    'id' => 'example',
    'RANGE0' => [1..5],
    'exe' => './a.out',
    'arg0@' => '"input$R0"',
    'arg1@' => '"output$R0"',
    'copiedfile0' => 'a.out',
    'copiedfile1@' => '"input$R0"',
    'queue' => 'gh_____',
    'group' => 'gh_____',
)
@jobs = prepare (%template);
submit(@jobs);
sync(@jobs);
```

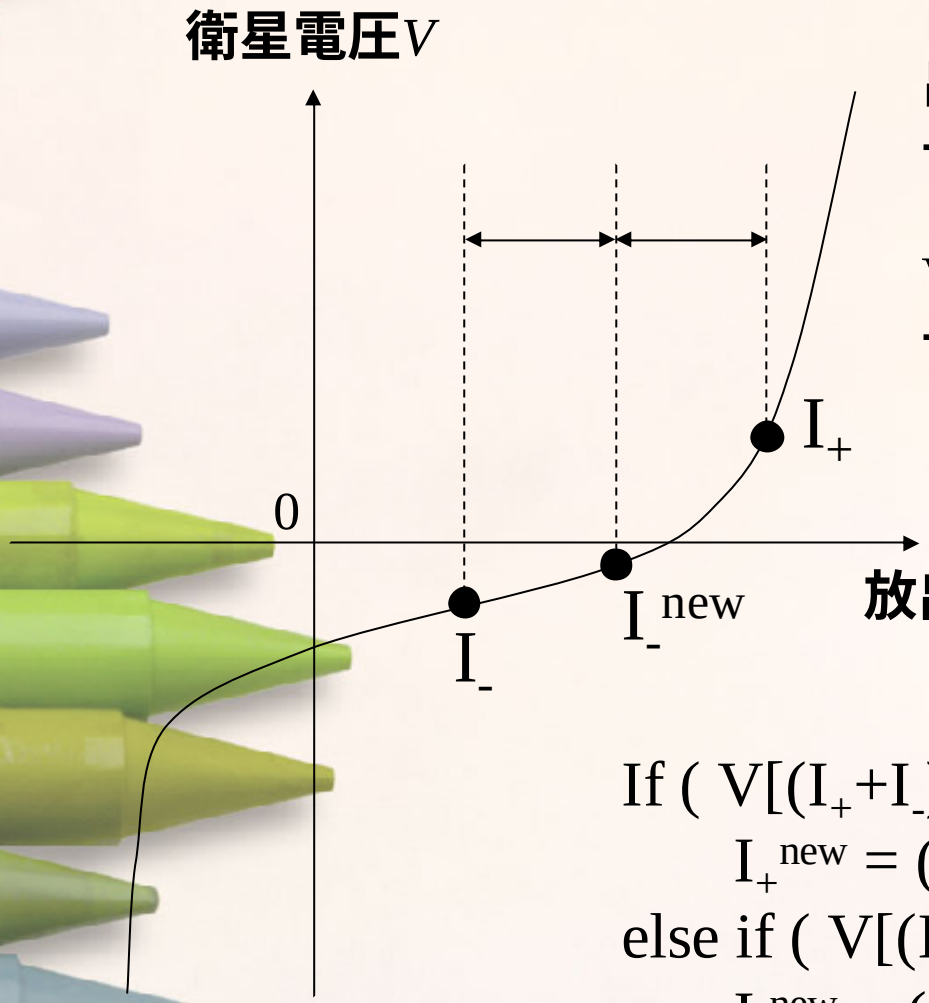
limitモジュール使用
同時ジョブ投入数を1に制限

作業ディレクトリ作成
ジョブ投入
ジョブ終了待ち

実行

```
% xcrypt psweep-limit.xcr
example_1 <= active
...
example_5 <= active
example_1 <= prepared
...
example_5 <= prepared
example_1 <= submitted
example_1 <= running
example_1 <= done
example_1 <= finished
example_2 <= submitted
...
example_5 <= finished
```

例3：二分法による最適パラメータ探索



目的：衛星電圧Vを0とする最適放出電子電流値Iを求める
→二分法を利用した最適値探索

Vの計算1回 ＝ 1ジョブ実行
→ 収束するまで繰り返し投入

If (V[$(I_+ + I_-) / 2 > 0$]) then
　　$I_+^{new} = (I_+ + I_-) / 2$
else if (V[$(I_+ + I_-) / 2 < 0$]) then
　　$I_-^{new} = (I_+ + I_-) / 2$
end if

テンプレート定義

「計算科学者」にマニュアルだけを与えて書いてもらったコード

```
%template = (
    'id' => 'job00',
    'exe' => 'mpiexec',
    'cpu' => '1',    'proc' => '64',
    'arg0' => '-n',    'arg1' => '${QSUB_VNODES}',
    'arg2' => './mpikempo3D',
    'arg3' => '4';,    'arg4' => '4',    'arg5' => '4',    'arg6' => '< plasma.inp',
    'linkedfile0' => 'mpikempo3D',
    'stdofile' => 'stdout',
    'stdefile' => 'stderr',
    'queue' => 'gh xxxxx',
    'group' => 'gh xxxxx'
);
```

ジョブ投入の反復実行

```
my $wpe1 = 1.0;   my $wpe2 = 1.5;  my $wpem = 1.0;  my $phi = 1;
until (abs($phi) < 0.1) {  # 結果が誤差範囲内になるまで反復
    $wpem = ($wpe1+$wpe2)/2;
    $template{‘id’} = $template{‘id’} . ‘+’;   # ジョブ名が重複しないようにIDを変更
    my @job = prepare(%template);
    foreach (@job) {   # 入力ファイルの作成（ネームリストplasma.inpのwp(3)の値のみ$wpemに変更）
        my $generate = CF("$ENV{'PWD'}/plasma.inp","$ENV{'PWD'}/"."$_->{id}");
        $generate -> KR("wp(3)", "$wpem");
        $generate -> do(); }
    @results = submit_sync(@job), "\n";
     foreach (@results) {  # 出力ファイルの抽出（最終行3列目の値を$phiに代入）
        my $phiext = EF("file:$ENV{PWD}/$_->{id}/pbody");
        $phiext->ED('L/E');   $phiext->ED('C/3');
        my @phis = $phiext->ER();
        $phi = $phis[0];  }
    print "Potential: ", $phi, "\n";
    # $phiの正負に応じて左右どちらかから範囲を狭める
    if ($phi < 0) { $wpe1 = $wpem; } else{ $wpe2 = $wpem; }
}
```

T2K以外の環境で動かすには

- xcrypt-*x.x*/lib/config/*mysched*.pm
  を自分で定義して，環境変数
  XCRJOBSCHED を *mysched* に設定する
- 厳密な仕様は近日中に決定･公開予定
- Perlに少し慣れている必要
  - システム管理者が書くことを想定
- 既存の設定ファイルをベースにすればそれほど難しくはない？
- 「動くレベル」でよいなら世の中の多くのスケジューラに対応できると思われる

例：T2K京大用の設定ファイル

- xcrypt-*x.x*/lib/config/KYOTO.pm

```
$jobsched::jobsched_config{“KYOTO"} = {
    qsub_command => "/thin/local/bin/qsub",
    qdel_command => "/usr/bin/qdel -K",
    qstat_command => "/thin/local/bin/qstat",
    jobscript_queue => '# @$-q ',
    jobscript_stdout => '# @$-o ',
    jobscript_stderr => '# @$-e ',
    jobscript_proc => '# @$-lP ',
    jobscript_cpu => '# @$-lp ',
    jobscript_memory => '# @$-lm ',
    jobscript_verbose => '# @$-oi',
    jobscript_verbose_node => '# @$-Ol',
    jobscript_workdir => '$QSUB_WORKDIR‘,
    extract_req_id_from_qsub_output => sub {
        my (@lines) = @_;
        if ($lines[0] =~ /([0-9]*)\.nqs/) {
            return $1;
        } else {
            return -1;
        }
    },
    extract_req_ids_from_qstat_output => sub {
        my (@lines) = @_;
        my @ids = ();
        foreach (@lines) {
            if ($_ =~ /([0-9]+)\.nqs/) {
                push (@ids, $1);
            }
        }
        return @ids;
    },
};
```

最低限必要な設定

- qsub/qdel/qstat-command
  - qsub, qstat, qdelコマンドへのパス
- extract_req_id_from_qsub_output
  - qsubの出力を受け取り，投入したジョブのrequest ID を返すPerl関数
- extract_req_ids_from_qstat_output
  - qstatの出力を受け取り，キューに存在するrequest IDのリスト を返すPerl関数
- jobscript_workdir
  - ジョブスクリプトにおいて，作業ディレクトリのパスを獲得する手段
  - 計算ノードとログインノードで，作業ディレクトリを 同一の絶対パスで指せない環境では必須

自分で追加モジュールを作成する

- 追加モジュール定義でできること
  - 独自のジョブのパラメータ（属性）を追加できる
  - prepare関数で作業ディレクトリを作るときの挙動を拡張，変更できる ← newメソッド
  - ジョブ投入直前の「フック」を追加できる ← beforeメソッド
    - 依存するジョブの完了を待ち合わせるなど
  - ジョブ投入処理の挙動を拡張，変更できる ← startメソッド
    - ある条件を満たすジョブはqsubをスキップするなど
  - ジョブ完了直後の「フック」を追加できる ← afterメソッド
    - 次のジョブを生成，投入する
    - 所望の結果が得られたら残りのジョブを殺す など
  - （予定）ジョブスクリプトの生成方法に手を加えられる
- Perlのオブジェクト指向プログラミングにある程度慣れている人向け

例：limitモジュールの定義

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

my $smph;
# モジュールの利用者が最初に呼び出す関数（同時ジョブ投入数の上限設定）
sub initialize {  $smph = Coro::Semaphore->new($_); }

sub new {
    my $class = shift;
    my $self = $class->NEXT::new(@_);
    return bless $self, $class;
}
# ジョブ投入前のフック：セマフォ獲得
sub before {$smph->down;}
# ジョブ完了後のフック：セマフォ解放
sub after  {$smph->up;}
```

Xcryptの拡張モジュールの提供機構

- ジョブオブジェクトはcoreクラスのインスタンス
- 拡張モジュールの開発者は，
  - メンバ（つエンドユーザに提供するパラメータ）の追加
  - メソッドの拡張・追加

  によりcoreクラスを拡張できる
- 基本的にはオブジェクト指向Perlのクラス拡張のスタイル
- 以下の名前のメソッドは特別な意味を持つ
  - new：コンストラクタ（prepare()時に実行される）
  - before：ジョブ投入直前の処理
  - start：ジョブ投入処理
  - after：ジョブ投入後の処理

limit以外の拡張モジュール

xcrypt-*x.x*/lib/

以下のディレクトリにいくつか

- dry.pm
  - 全てのジョブの実行を飛ばす（ドライ実行）
- convergence.pm
  - 今回と前回とのジョブの結果の差が設定値以下に収束するまで実行を繰り返す
- n_section_method.pm：n分法
  - successor.pm: ジョブの依存関係を宣言的に書けるようにする

最後に

- Xcryptで，通常のマルチスレッドプログラミングに近い感覚でジョブ並列処理が書ける
- 基本的な機能は実装済み
- 今後の機能追加予定
  - チェックポインティング
  - 手元のPCからのリモートジョブ投入
- スパコン「利用者」側の視点でコメントをいただければ幸いです
  - 書き易さ，使い勝手
  - 機能追加要望
  - バグ報告