NIBB GITC「BLAST 自由自在」
September 14-15, 2017@Okazaki

大規模なBLAST検索の効率化

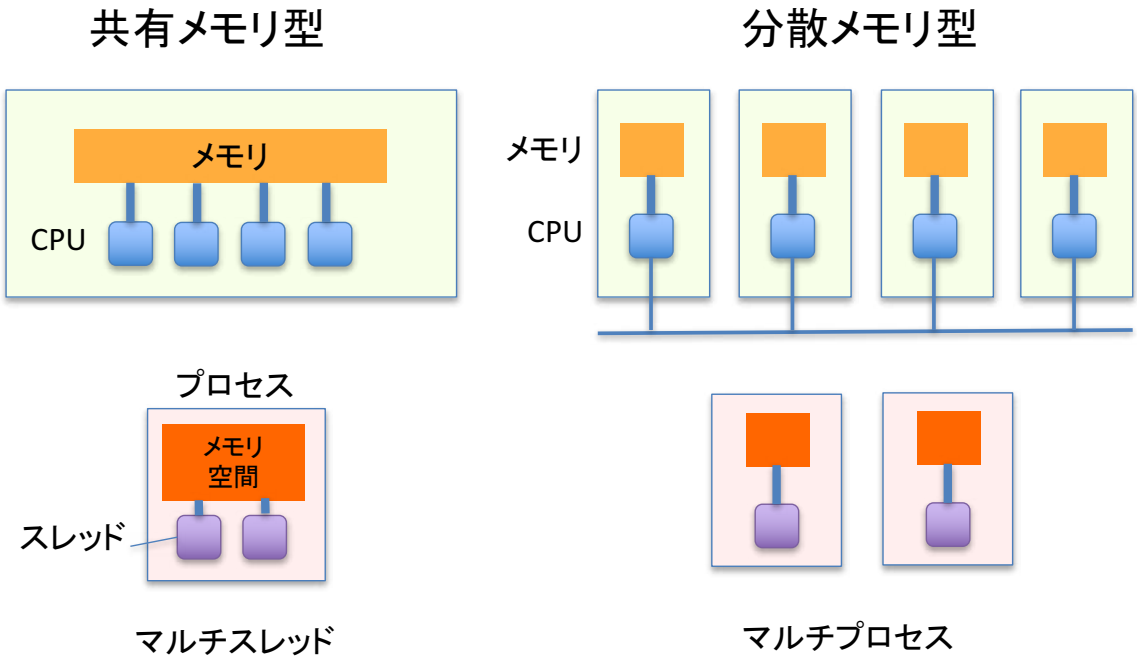
内山郁夫

大規模なBLAST検索の例

- 遺伝子予測
  - 発現遺伝子全体 x ゲノムDNA配列
  - ゲノムDNA配列 x タンパク質DB（blastx）
- 機能アノテーション
  - ゲノム内の遺伝子全体 x タンパク質DB
- 比較ゲノム解析
  - ゲノム内の遺伝子全体 x 別のゲノムの遺伝子全体

大規模検索は並列化によって高速化できる

並列計算の基礎

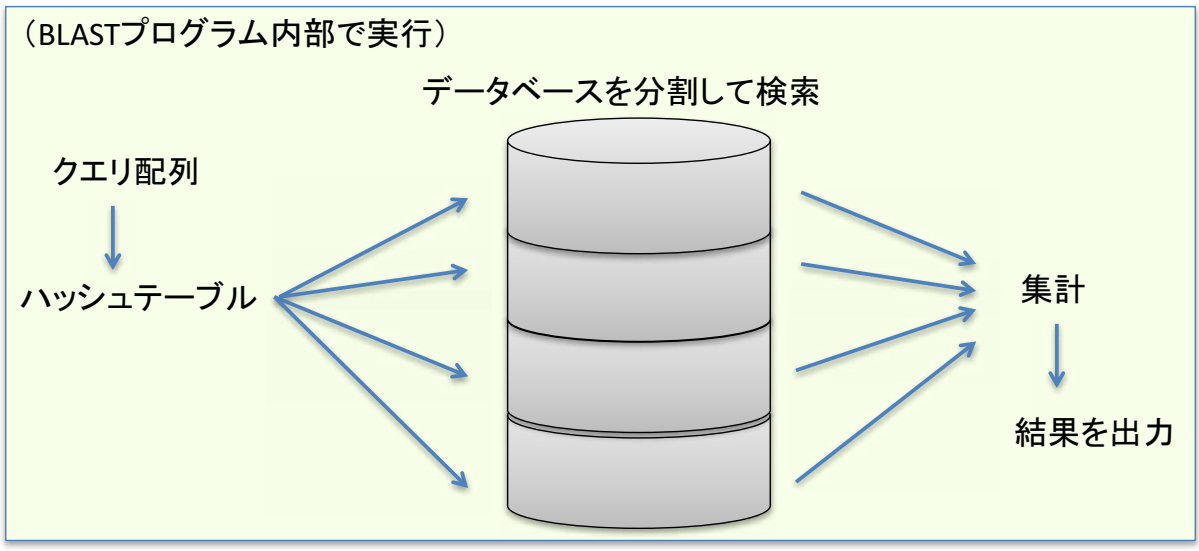


スレッド数及びメモリ空間の合計は、実CPUコア数、実メモリ数を超えて使用することができるが、一般に効率は落ちる（特にメモリが超えた場合は深刻な遅延を生じることがある）

BLASTの並列実行

マルチスレッドによる並列化

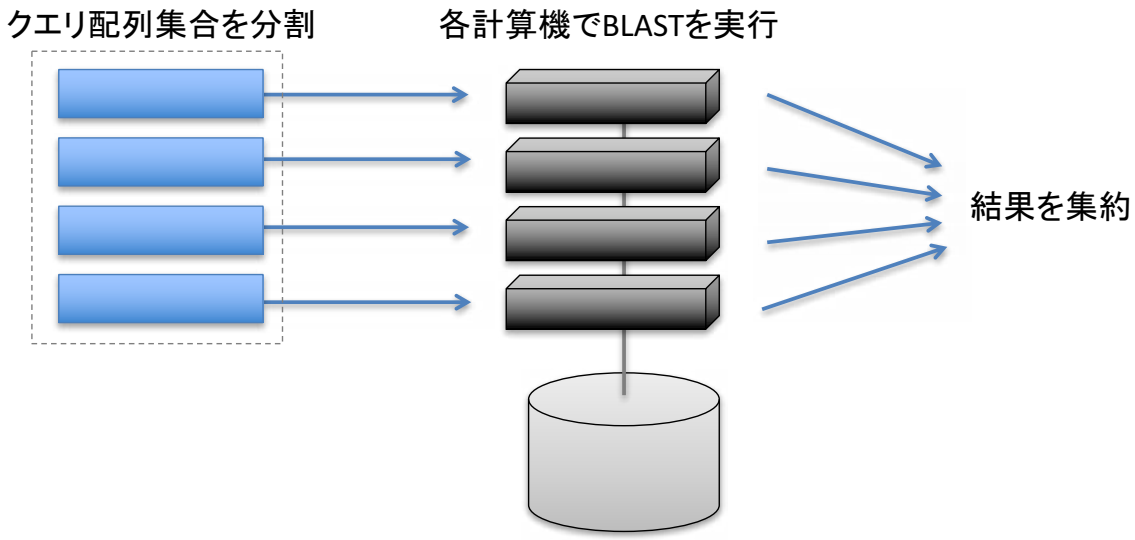
- BLASTの`-num_threads` オプションを使う
- スレッド数は、実行する計算機に搭載されたコア数を超えないようにする
- クエリひとつでも高速化が可能



BLASTの並列実行

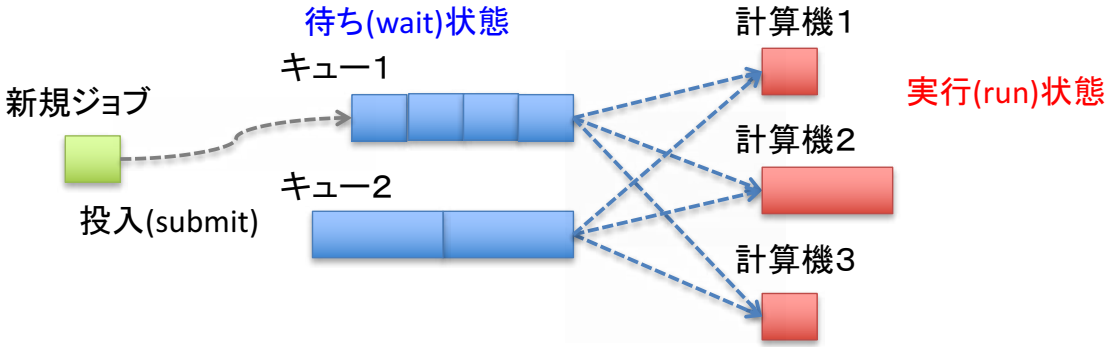
分散処理による並列化

- クエリ配列が大量にある場合、クエリ配列を複数のファイルに分割して、分散処理環境を用いて並列にBLASTを走らせるのが効果的



ジョブ管理システム

- 「キュー（待ち行列）」にたいして「ジョブ」を投入する形で、プログラムの実行をシステムに委託する。
- 「キュー」に入ったジョブは、複数の計算機に分配され並列に実行される。
- 既定の処理量を超えるジョブが投入された場合、後から投入されたジョブは先行するジョブが終了するまで「待ち状態」となる。
- 対象とするマシンや、想定されるジョブの大きさによって、複数のキューが用意されている。



Sun Grid Engineでのジョブの実行

- クラスター計算機でプログラムを走らせるには、走らせるコマンドを記述したジョブファイルを作成してから、qsub コマンドを使う

ジョブの投入 `qsub [-q キュー名] ジョブファイル`

ジョブの状態の確認（自分のジョブ） `qstat`

（全員のジョブ） `qstat -u '*'`

（キューごとのサマリ表示） `qstat –g c`

ジョブの削除（特定のジョブ） `qdel ジョブ番号`

（全部キャンセル） `qdel '*'`

NIBB 生物情報解析システムにおけるキューの構成

	分散並列処理型			共有メモリ型		
キュー名	**small**	**medium**	**large**	**smps**	**smpm**	**smpl**
ジョブの特徴	短時間 並列多	中時間	長時間	中メモリ	大メモリ	最大メモリ
利用ノード	node01 - node40			ldas-smp		
最大実行時間/job	6時間	72時間	no limit	no limit		no limit
最大メモリ/job	4.8GB	4.8GB	4.8GB	500GB	1TB	4TB
最大ジョブ数/キュー	580	200	20	8	4	1

BLASTは、クエリを細かく分割して並列度を上げてsmallキューで実行するのがよい

今回の実習では、講習会用に特別に作ったキュー blast を使う

実習：2ゲノム間の遺伝子比較

- 出芽酵母 *Saccharomyces cerevisiae*
  ファイル sce
- 分裂酵母 *Schizosaccharomyces pombe*
  ファイル spo

NCBIで配布されたファイルのエントリ名に生物種名のコードを付加

```
% sed 's/^>/>sce:/' GCF_000146045.2_R64_protein.faa > sce
% sed 's/^>/>spo:/' GCF_000002945.1_ASM294v2_protein.faa > spo
```

```
>sce:NP_001018029.1 hypothetical protein YBR230W-A
MRNELYQLWCVASAARGVAKSSFVRANSAMCEYVRTSNVLSRWTRDRQWE
>sce:NP_001018030.1 L-serine/L-threonine ammonia-l
MSIVYNKTPLLRQFFPGKASAQFFLKYECLQPSGSFKSRGIGNLIMKSAI
LSLPCTVVVPTATKKRMVDKIRNTGAQVIVSGAYWKEADTFLKTNVMNKI
QDLKSQHISVNKVKGIVCSVGGGGLYNGIIQGLERYGLADRIPIVGVETN
VISNQTFEYARKYNTRSVVIEDKDVIETCLKYTHQFNMVIEPACGAALHL
NTIKDLEEALDSMRKKDTPVIEVADNFIFPEKNIVNLKSA
>sce:NP_001018031.2 Adf1p [Saccharomyces cerevisia
MGKCSMKKKGVGKNVGVGKKVQKKRSISTAERKRTKLQVEKLNKSSETMI
EKDSKVREQIRTEKSKTNDSMLKQIEMISGFSL
```

```
>spo:NP_001018179.1 hydroxymethylbilane synthase (
MPSCTSFPIGTRKSKLAVIQSEIIREELEKHYPHLEFPIISRDTIGDEIL
ILVHSLKDLPSEMPDGMVIACIPKRSCPLDAIVFKAGSHYKTVADLPPGS
TRLAKLDAPDSQFDCLVLAAAGLFRLGLKDRIAQMLTAPFVYYAVGQGAL
RALMKRLQGGCAIPIGVQTDVLAISNSSYRISLLGTVLSADGLRAAFGNA
EEHQRSSDSEESLKNY
>spo:NP_001018181.1 poly(A) polymerase Cid14 [Schi
MGKKSVSFNRNNYKKRKNERTEPLPRRIFKNDKPSKFKSKRKEKDKNSDA
NDSEGIRDKGGVEISNKNDPYIQFGKADPLEPLEKPDLPEEAIKRGEPTI
WNSDEDEDSVSNDKSKNNESLKKSSKNEIPGFMRQRGRFFHEANEKSDSN
FHQDILHFIDYITPTPEEHAVRKTLVSRINQAVLQKWPDVSLYVFGSFET
AHHLKKLKLASEVQVITTANVPIIKFVDPLTKVHVDISFNQPGGLKTCLV
```

生物情報解析システム bias4.nibb.ac.jpにログインする。

```
% ssh bias4.nibb.ac.jp
```

実習用ディレクトリに移動

```
% makeblastdb –in spo –dbtype prot –parse_seqids
```

配列集合を分割する

```
# FASTA形式の配列を、長さの和がBLOCK_SIZEを超えるごとに分割

BLOCK_SIZE=100000

foreach line in Lines do
    if (line が '>' で始まる) then
        # FASTA形式のタイトル行
        if (savedLen >= BLOCK_SIZE) then
            新しいファイルをオープンし、そこに
                SavedLines を出力する
            SavedLines を空にする
            savedLen = 0
        fi
    else
        # 配列行
        savedLen += length(line)
    fi
    SavedLines に line を加える
done
# まだ出力されていない行
新しいファイルをオープンし、そこに SavedLines を出力する
```

配列を分割する

- スクリプト split_seq.pl

```
split_seq.pl [–BLOCK_SIZE=block_size]
          [-QUERY_OUT_DIR=query_dir]
          [-QSUB_SCRIPT_FILE=[script_file]] query_file
```

FASTA形式のファイル *query_file* 中の配列を長さの和がblock_sizeを超えるごとに別のファイルに分割して、ディレクトリ *query_dir* 以下に *query_file.fileNo* (*fileNo*は1から*split_num*までの数字)という名前で格納する。合わせて、qsub_blast.shというqsub 用のスクリプトを作成する。

```
% ls
sce  spo
% makeblastdb –in spo –dbtype prot –parse_seqids
% split_seq.pl sce
% ls
qsub_blast.sh  query_sce sce  spo  spo.phr  spo.pin  spo.psq
% ls query_sce
spo.1  spo.2  spo.3  ...
```

アレイジョブ

- 同じコマンドを、入力（及び出力）ファイルを変えて複数回並行して実行させる。
- qsub のオプションに –t 開始番号–終了番号を指定し、スクリプトファイルに変数 ${SGE_TASK_ID}を埋め込むと ${SGE_TASK_ID} を開始番号から終了番号まで順次置き換えたジョブとしてサブミットされる。
- サブミットするジョブ数が大きいときは、最大の同時実行ジョブ数を-tcオプションで指定する。

```
#!/bin/sh
#$ -t 1-200
#$ -tc 50
#$ -cwd
command input.${SGE_TASK_ID} > output.${SGE_TASK_ID}
```

input.1 からinput.200までのファイルを入力とし、結果を対応するoutput.1からoutput.200までのファイルに出力するジョブ200個を生成し、最大同時実行数50で実行する

qsub による実行

- スクリプト qsub_blast.sh を編集

```
% emacs qsub_blast.sh
```

```
#!/bin/sh
#$ -cwd
#$ -t 1-30
#$ -tc 32
#$ -N blastjob
#$ -S /bin/sh
DB=spo
SEQDIR=query_sce
OUTPUTDIR=blastout_sce

OPTIONS=(-outfmt 6 -evalue 0.001)

if [ ! -d $OUTPUTDIR ]; then
        mkdir $OUTPUTDIR
fi
blastp -db $DB -query query_sce/sce.$SGE_TASK_ID -out $OUTPUTDIR/sce.blast.$SGE_TASK_ID "${OPTIONS[@]}"
```

検索対象DB名を変更

必要に応じてオプションやコマンド行を変更

カーソルキーで移動
デリートキーで文字を消去して書き換え
保存するには Control-X Control-S を順に押す
終了するには Control-X Control-C を順に押す

qsub による実行

- ジョブをサブミット（blast キューを使う）

```
% qsub –q blast qsub_blast.sh
```

- 実行状況の確認

```
% qstat
```

（何も表示されなくなったら終了）

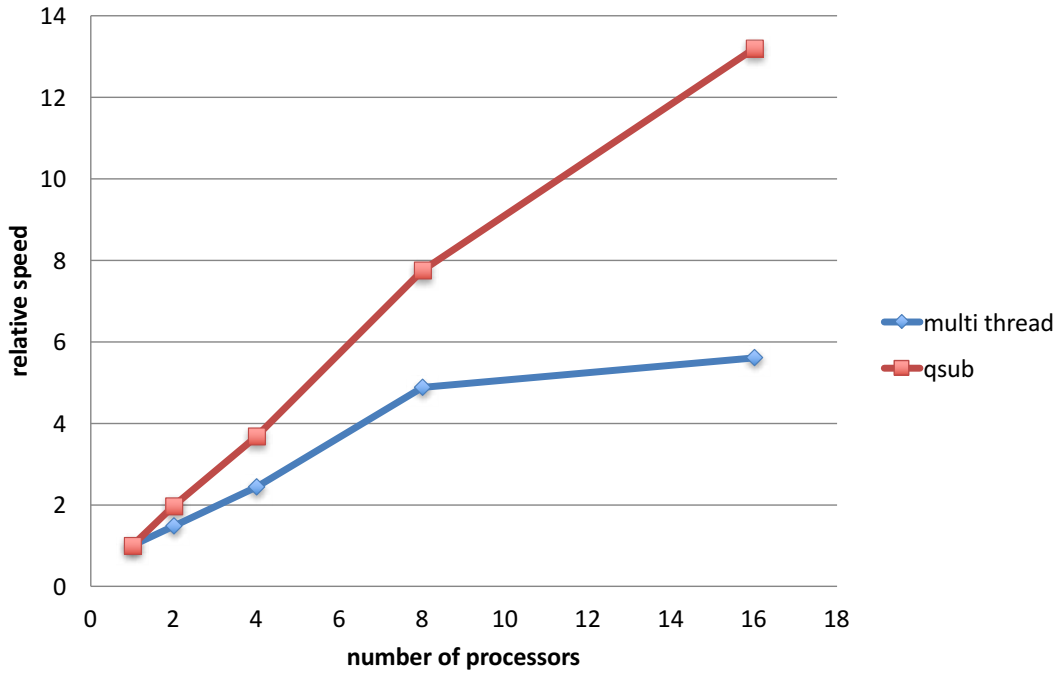
- 結果を一つにまとめる

```
% cat blastout_sce/sce.blast.* > sce-spo.blast
```

（参考） 元の順番通りにつなげたい場合、bashだと以下のようにして行える

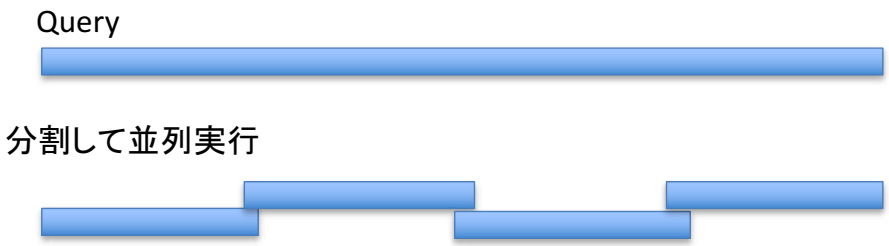
```
for ((i=1; i<=30; i++)); do
  cat blastout_sce/sce.blast.$i >>sce-spo.blast
done
```

並列度と検索速度の関係



クエリ配列の分割

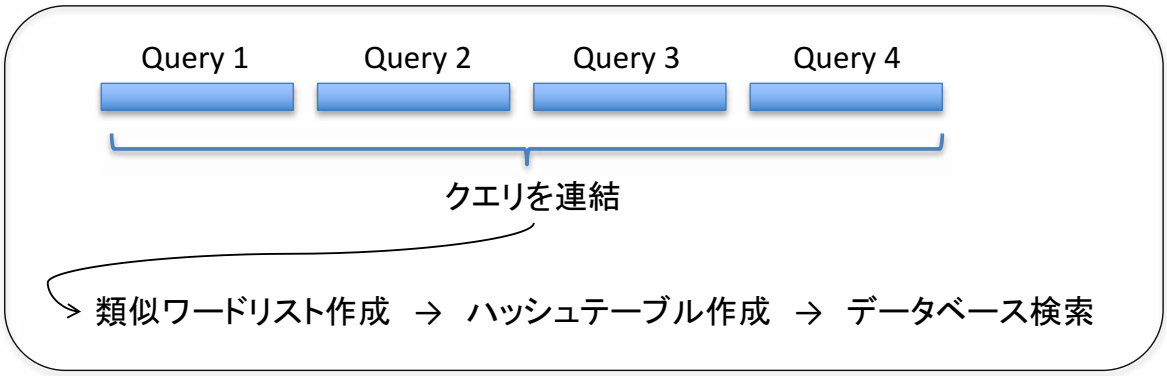
長いクエリ配列が一つある場合は、配列を分割することにより並列に実行できる



`-query_loc` 開始位置-終了位置
クエリ配列の開始位置から終了位置までをクエリとして使う

クエリ配列の連結

BLASTでは、多数のクエリを検索する際は、クエリを連結することによって検索回数を少なくしている。従って、クエリを分割しすぎるとかえって効率が悪くなる。



一方、クエリの連結によってメモリの使用量は増大する

連結する際の長さは、環境変数 BATCH_SIZE で調節できる