大規模なBLAST検索の効率化

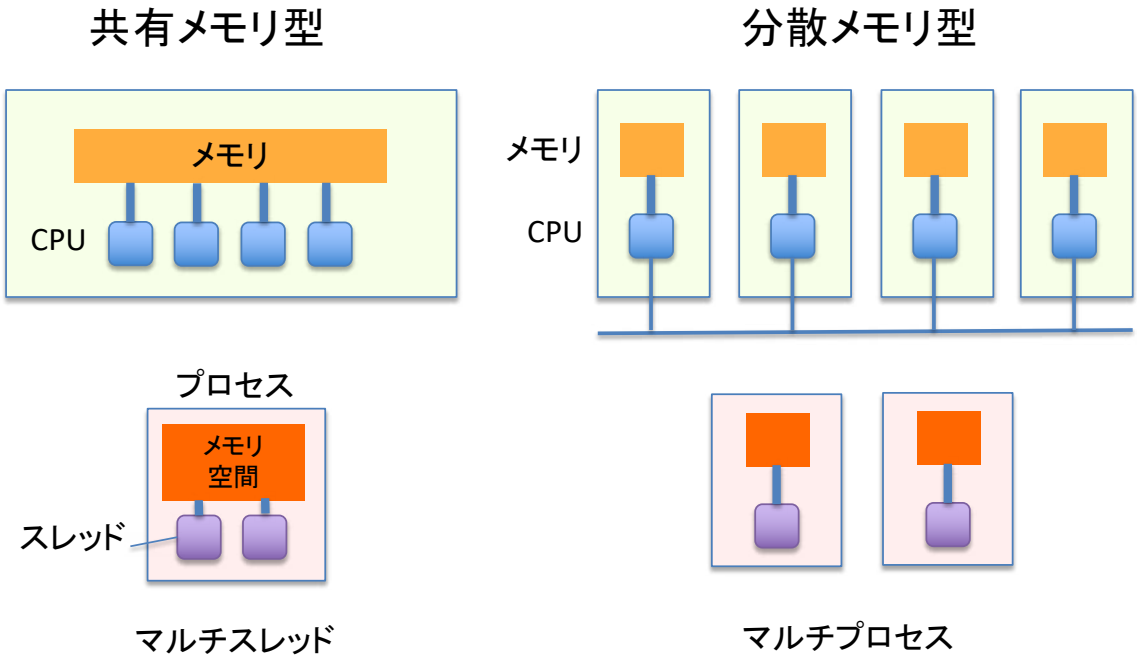
内山郁夫

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

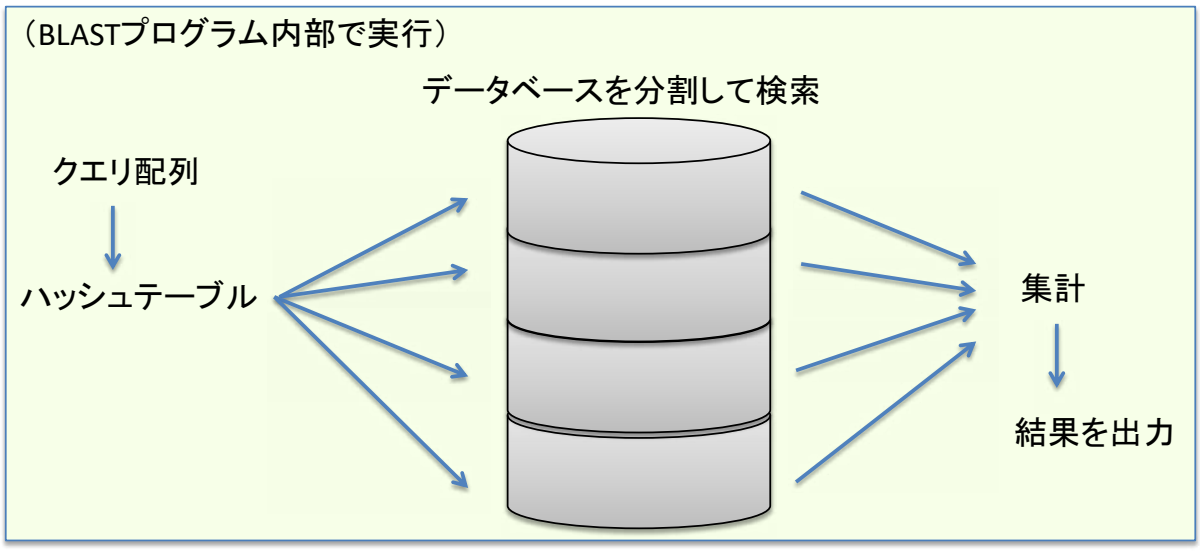


スレッド数及びメモリ空間の合計は、実CPUコア数、実メモリ数を超えて使用することができるが、一般に効率は落ちる（特にメモリが超えた場合は深刻な遅延を生じることがある）

BLASTの並列実行

マルチスレッドによる並列化

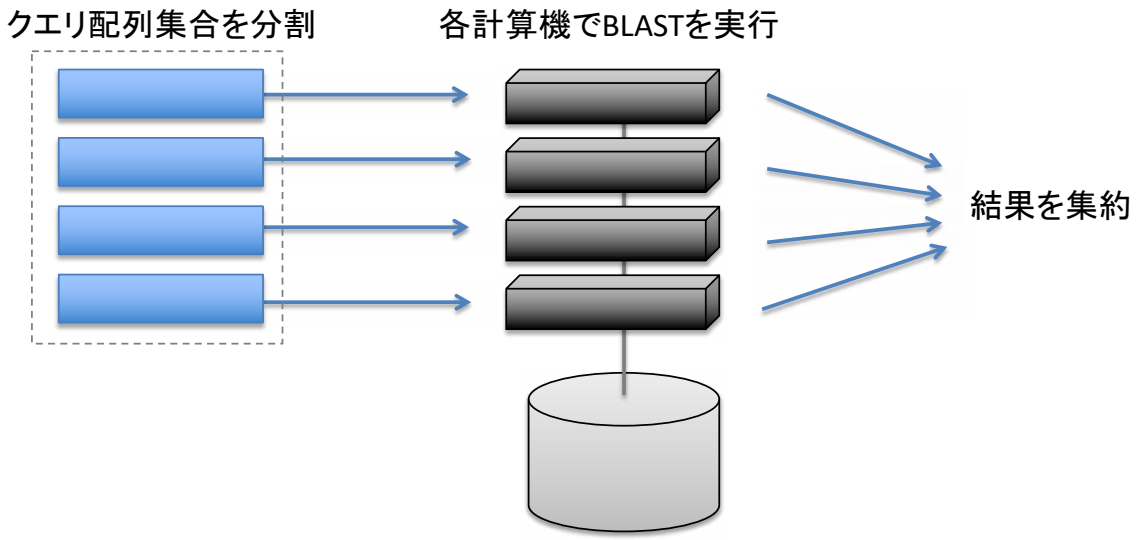
- BLASTの`-num_threads` オプションを使う
- スレッド数は、実行する計算機に搭載されたコア数を超えないようにする
- クエリひとつでも高速化が可能



BLASTの並列実行

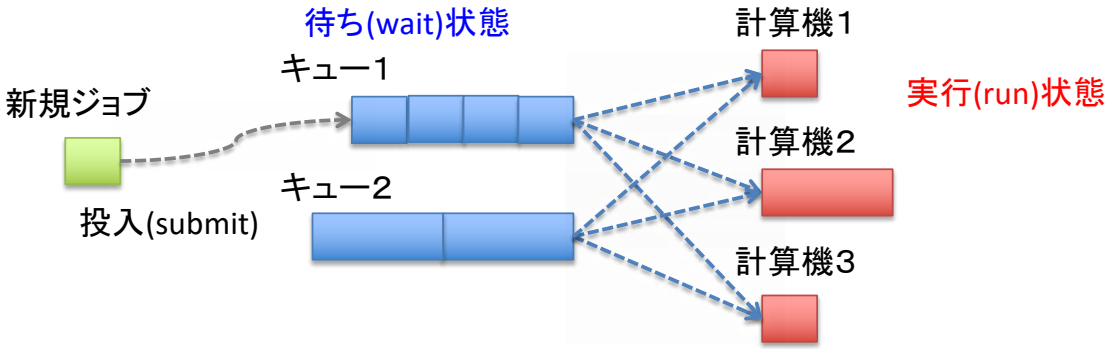
分散処理による並列化

- 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。



PBS 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 -u ユーザ名`

（全員のジョブ） `qstat`

（キューごとのサマリ表示） `qstat –Q`

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 `qselect -u ユーザ名``

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	bias5-node01 – bisa5-node20			bias5-smp		
最大実行時間/job	6時間	72時間	no limit	no limit	no limit	no limit
最大メモリ/job	96GB	96GB	96GB	500GB	1TB	3TB
最大cpu数/キュー	580	200	20	48	48	36

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.fas`
- 分裂酵母 *Schizosaccharomyces pombe*
  ファイル `spo.fas`

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.fas
% sed 's/^>/>spo:/' GCF_000002945.1_ASM294v2_protein.faa > spo.fas
```

```
>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 bias5.nibb.ac.jpにログインする。

```
% ssh bias5.nibb.ac.jp
```

`~/data/IU`: データ
`~/scripts/IU`: スクリプト （パスが通っている）

`data` ディレクトリに移動

```
% cd data/IU
```

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.fas  spo.fas
% makeblastdb -in spo.fas -out spo -dbtype prot –parse_seqids
% split_seq.pl sce.fas
% ls
qsub_blast.sh  query_sce sce.fas  spo.fas  spo.phr  spo.pin
spo.pog spo.psd spo.psi 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 –J 開始番号–終了番号を指定し、スクリプトファイルに変数 ${PBS_ARRAY_INDEX}を埋め込むと ${PBS_ARRAY_INDEX} 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。

```
#!/bin/sh
#PBS -J 1-200
#PBS -cwd
command input.${PBS_ARRAY_INDEX} > output.${PBS_ARRAY_INDEX}
```

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
```

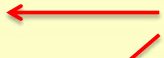
```
#!/bin/sh
#PBS -J 1-30
#PBS -N blastjob
#PBS -S /bin/sh

DB=spo
SEQDIR=query_sce
OUTPUTDIR=blastout_sce

OPTIONS=(-outfmt 6 -evalue 0.001)

cd $PBS_O_WORKDIR

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



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 -u（ユーザ名）
（何も表示されなくなったら終了）
```

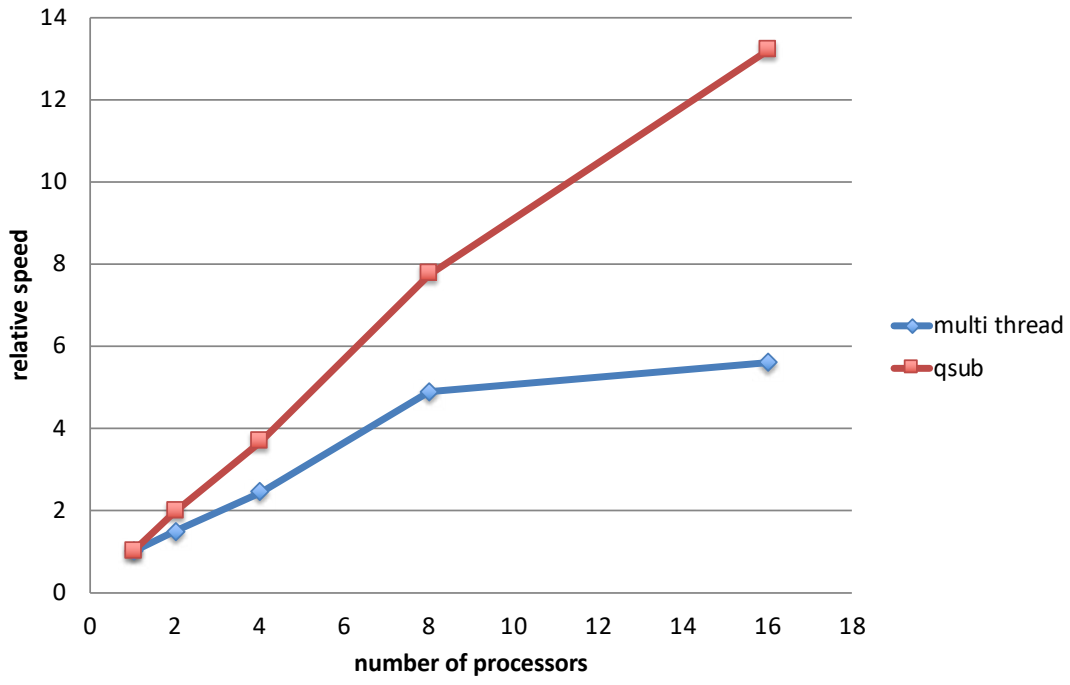
- 結果を一つにまとめる

```
% 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



超高速タンパク質ホモロジー検索

- UBLAST (Edgar 2010)
- RAPSearch (Zhao et al 2012)
- GHOSTX (Suzuki et al. 2014)
- DIAMOND (Buchfink et al. 2015)
- MMSeq (Hauser et al. 2016)

BLASTの数十倍程度の検索速度で、パソコンレベルでも大規模な配列群間の総当たり検索が可能

超高速タンパク質ホモロジー検索
高速化のための工夫

高い類似性を持つトップヒットを取りこぼさないようにしつつ、初期検索によるフィルタリングをより強力にして高速化を図る

- Double indexing
  - クエリ、データベースともに索引付けする（BLASTはクエリのみ）
    →索引の比較で候補領域を絞り込み
- より強力なseedの利用
  - 長いワード、multiple spaced (discontiguous) seeds
- Reduced alphabet
  - Seed検索の際、類似アミノ酸をまとめて文字を減らしたアルファベットを使用
    [KREDQN][C][G][H][ILV][M][F][Y][W][P][STA]
- ハードウェアの特性に合わせた最適化

DIAMONDの実行

- データベースインデックスの作成
  `% diamond makedb --in` *dbfile* `--db` *dbname*
- 検索の実行
  `% diamond blastp --query` *queryfile* `--db` *dbname*

例）spo.fasをデータベース、sce.fasをクエリとして検索する

```
% diamond makedb --in spo.fas --db spo
% diamond blastp --query sce.fas --db spo --out sce-spo.diamond
          --threads 4
```

検索のオプション:

`--threads` 使用するコア数
`--evalue` E-valueの閾値
`--max_target_seqs` クエリ当たりの最大アライメント出力数
`--top` トップのスコアから(100-N)%のスコアまでを出力
`--sensitive` より高感度だが低速な検索

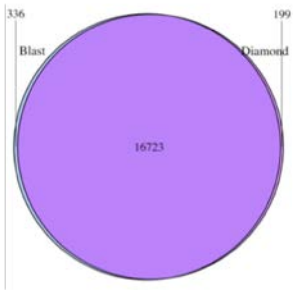
BLASTとDIAMOND比較

Human(20213 genes) vs Mouse (22089 genes)

	計算時間（秒）	ヒット数	ヒット数 (BBH)
BLAST	11099	3671618	17059
DIAMOND	207	155590	16921

(BBH 平均一致度 84.4%)

BLAST vs DIAMOND (BBH)



S.cerivisiae (5910 genes) vs *S. pombe* (5133 genes)

	計算時間（秒）	ヒット数	ヒット数 (BBH)
BLAST	510	32049	3074
DIAMOND	14	8238	2294
DIAMOND (sensitive)	42	14111	2843

(BBH 平均一致度 42.3 %)

BLAST vs DIAMOND (BBH)

Default (fast)　Sensitive