ワークフローシステムPwrakeによる科学データ処理

田中昌宏

筑波大学計算科学研究センター

内容

- Gfarmによる高い並列I/O性能
- 並列分散ワークフロー実行システム Pwrake
- 天文学データ処理における性能評価
- バイオインフォマティクスへの適用

天文学に必要な観測データ

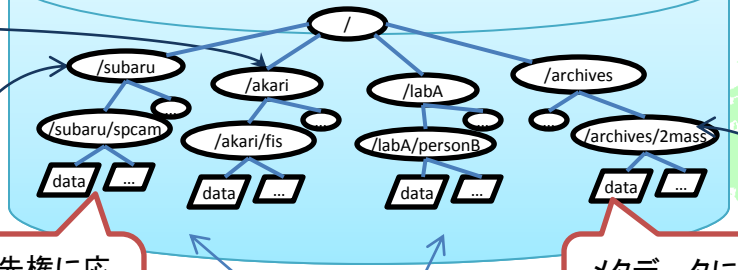
各研究機関の
望遠鏡による観測

JAXA

NAOJ

観測データの
一次処理・較正

Gfarmファイルシステム



優先権に応
じたアクセス
コントロール

メタデータによ
るデータ検索

観測データ
の解析

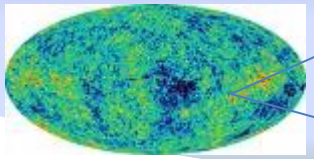
研究者

世界の天文台による
観測データの蓄積

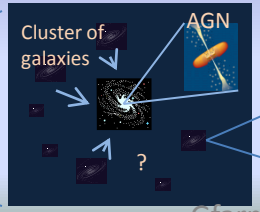
観測データ配布
の標準規格

各研究テーマの実施

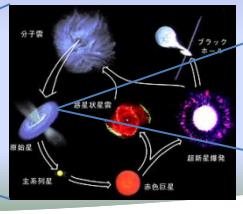
ビッグバンと
宇宙の晴れ上がり



銀河の形成・進化



星の形成・進化と
物質循環



系外惑星の探索

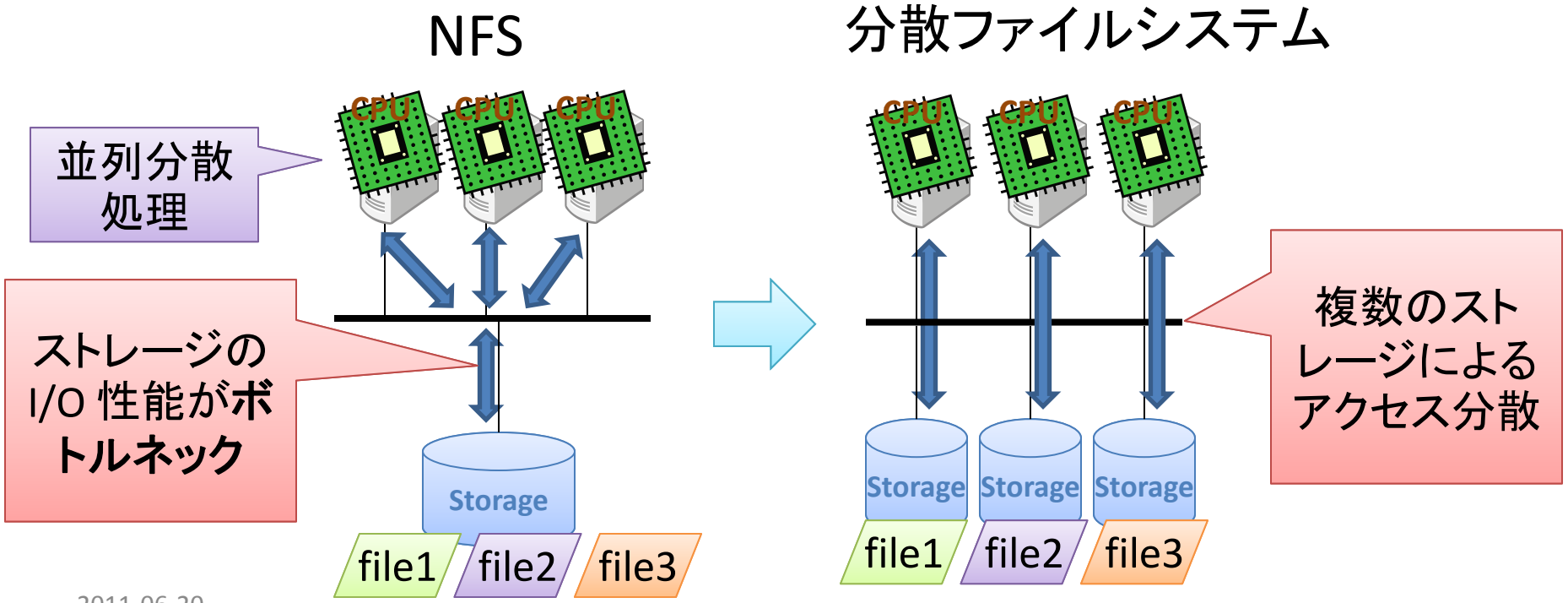
膨大な天文観測データ

- 広域で共有したい
- **高速に処理したい**

- 分散ファイルシステム **Gfarm**

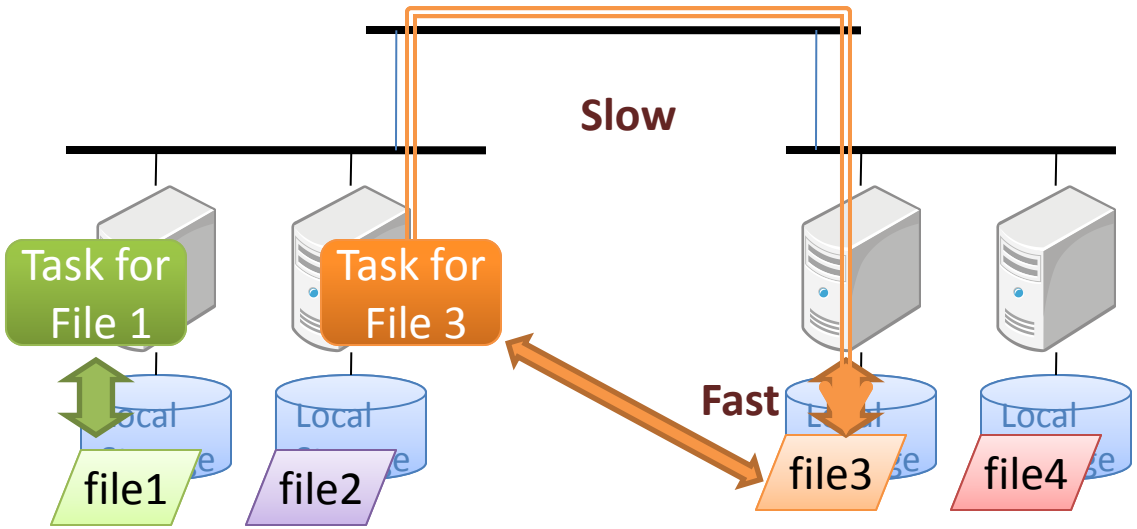
高性能な並列I/O

- 集中型ファイルシステムでは、ストレージI/Oがボトルネック
- 分散ファイルシステムにより、スケーラブルな並列I/O性能が実現



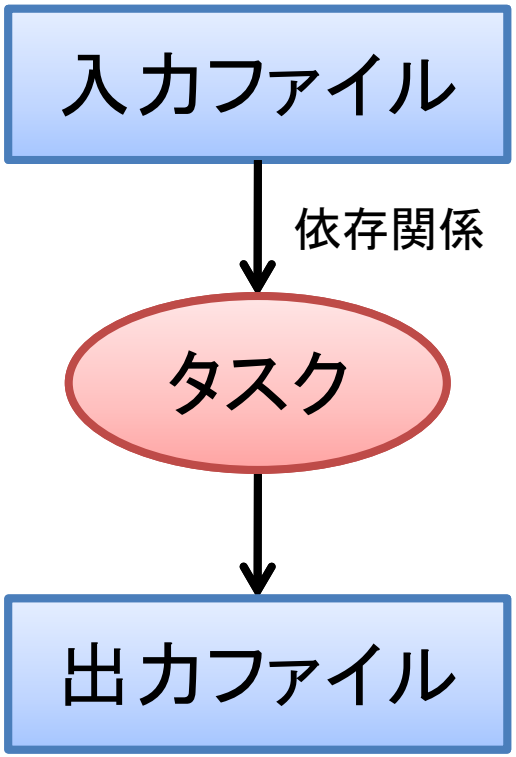
ローカリティの活用による性能向上

- Gfarmは、計算ノードとストレージノードを兼ねることができる
- ファイルが存在するノードで処理すれば、性能が向上
- タスク実行をコントロールする、ワークフローシステムが必要



ワークフロー

- 1ステップ:
  - **入力ファイル** を読み、
  - **タスク** が処理し、結果を
  - **出力ファイル** に書き出す。
- 複数ステップ → **ワークフロー**
- **DAG**
  - **D**irected **A**cyclic **G**raph
    （非循環有向グラフ）
- Makefile もワークフローの1種



並列分散ワークフロー処理システム
Pwrake

- Rake がベース （Ruby版のMake）

- 並列分散機能を拡張 : **Pwrake （プレイク）**
  - Parallel Workflow extension for Rake
    - http://github.com/masa16/pwrake

Rakefile の例: ソースコードのビルド

```
rule "*.o" => "*.c" do |t|
  sh "cc -o #{t.name} #{t.prerequisites[0]}"
end

file "prog" => ["foo.o","bar.c"] do |t|
  sh "cc -o prog #{t.prerequisites.join(' ')}"
end

task :default => "prog"
```

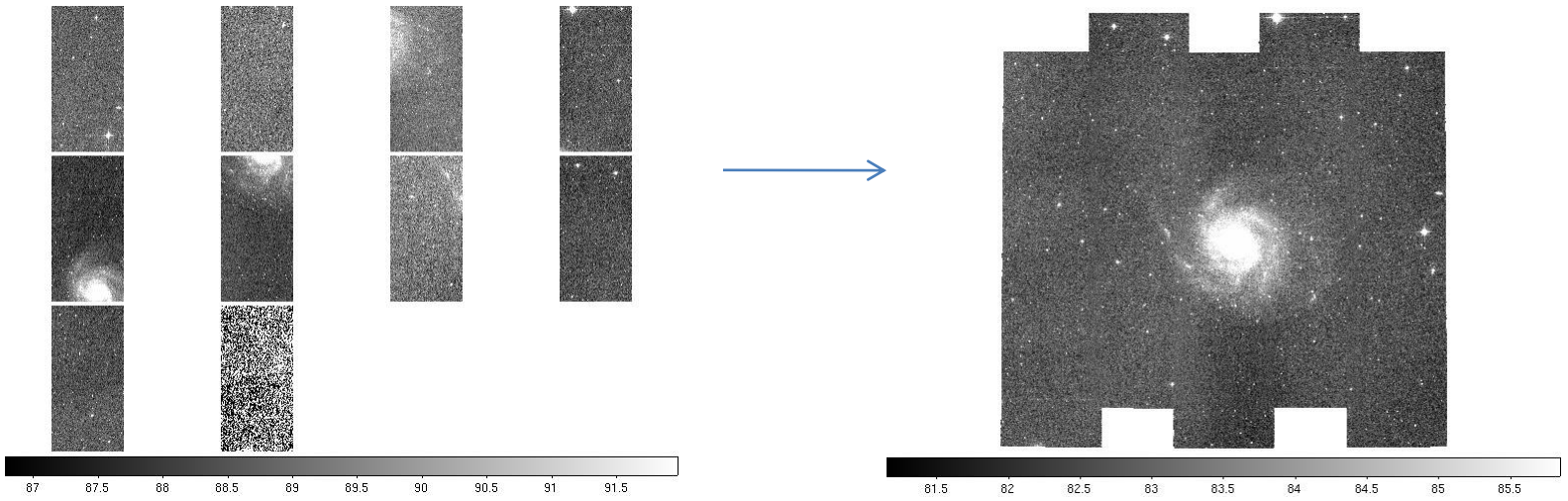
Rakeの利点

- 複雑なワークフローを記述可能
  - Rubyの内部DSL (Domain Specific Language)
  - 複雑な科学データ処理が可能に
- 機能拡張が容易
  - 依存関係がないタスクの並列実行
  - SSHによる分散実行
  - ローカリティに基づくタスクの配置

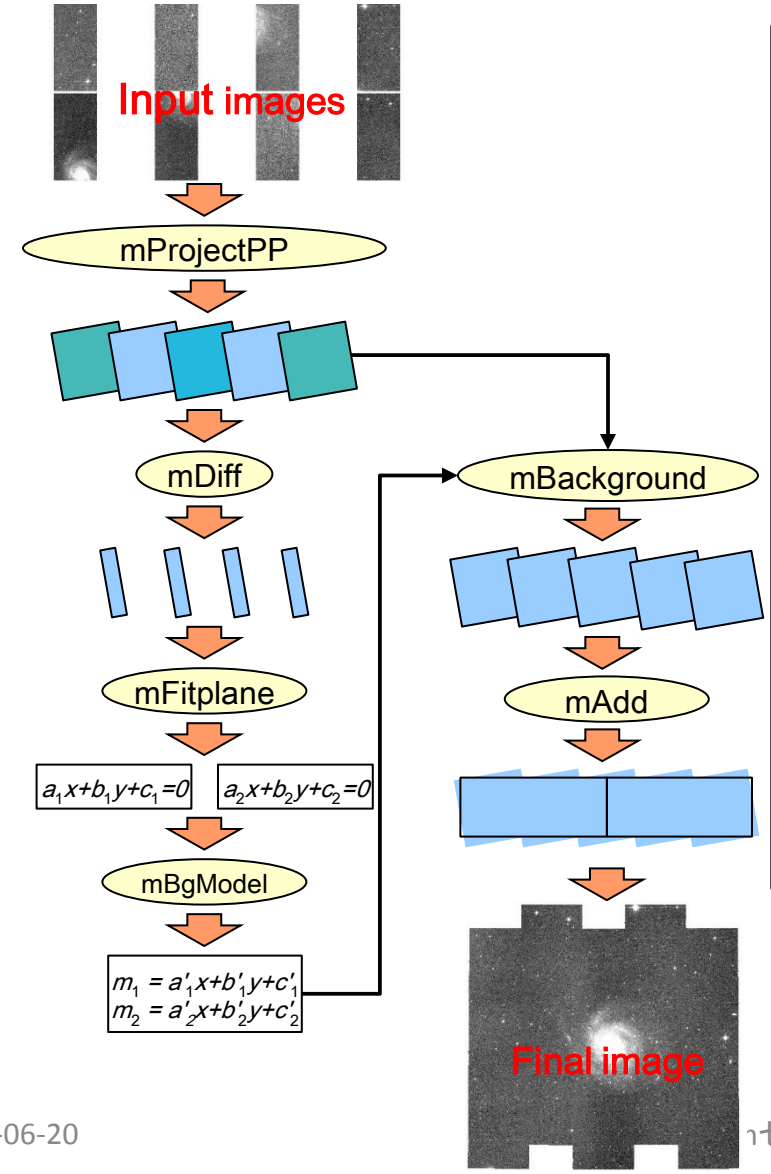
  → **Pwrake**

天文データ処理の例

- Montage
  - 複数ショットの画像を、1つの画像に結合
  - http://montage.ipac.caltech.edu/



Montage Workflow



- 処理内容:
  - 座標変換
  - 明るさ補正
  - 足し合わせ
- 処理ごとにプログラムが分かれている
- 1 image : 1 process

Montage ワークフローの Rakefile

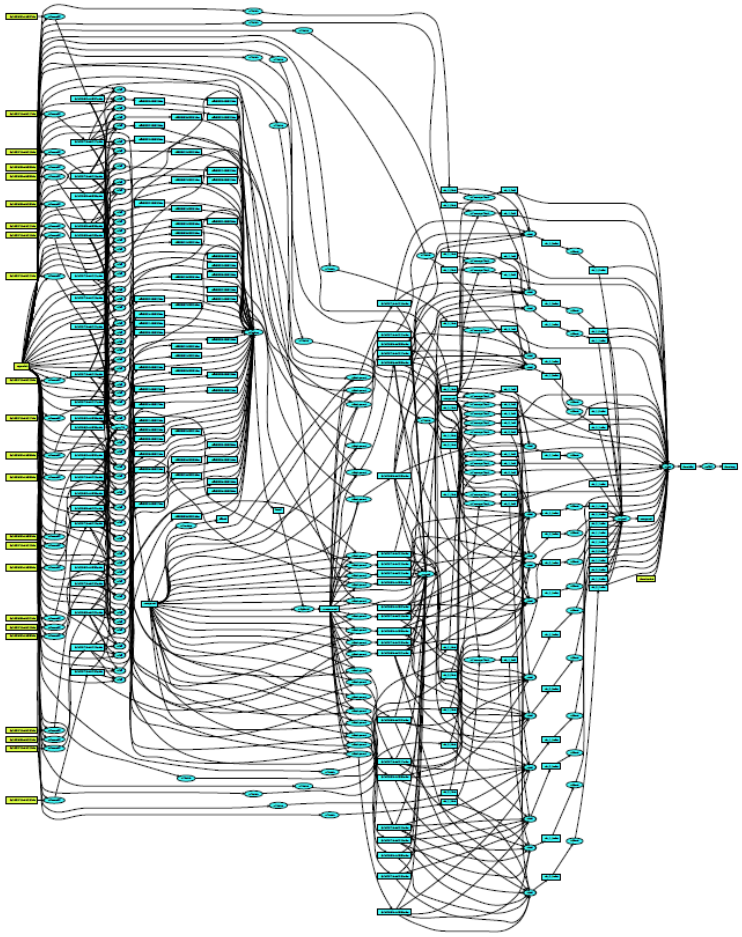
```
require 'montage_tools'
require 'rake/clean'

task( :default => "mosaic.jpg" )

dir=Dir.glob(Dir.pwd+"/Montage_*/bin")
ENV['PATH'] = "#{dir.last}:"+ENV['PATH']

INPUT_DIR = ENV["INPUT_DIR"] || "m101/rawdir"
REGION_HDR = "m101/template.hdr"

### Projection
SRC_FITS = FileList["#{INPUT_DIR}/*.fits"]

P_IMGTBL = []
PRJ_FITS=[]
SRC_FITS.each do |src|
  desc prj = src.sub( %r|^(.*?)([^/]+).fits|, 'p/¥2.p.fits' )
  file( prj => [src,REGION_HDR] ) do |t|
    sh "mProjectPP #{src} #{prj} #{REGION_HDR}" do |*x| end
    Montage.collect_imgtbl( t, P_IMGTBL )
  end
  PRJ_FITS << prj
end

file( "pimages.tbl" => PRJ_FITS ) do
  Montage.put_imgtbl( P_IMGTBL, "p", "pimages.tbl" )
end

### dif & fit
file( "diffs.tbl" => "pimages.tbl" ) do
  sh "mOverlaps pimages.tbl diffs.tbl"
end

file( "fitfits.tbl" => "diffs.tbl" ) do
  DIFF_FITS=[]
  FIT_TXT=[]
  FIT_TBL=[]
  diffs = Montage.read_overlap_tbl("diffs.tbl")
  diffs.each do |c|
    p1 = "p/"+c[2]
    p2 = "p/"+c[3]
    DIFF_FITS << dif_fit = "d/"+c[4]
    file( dif_fit => [c[2],c[3],REGION_HDR,"pimages.tbl"] ) do |t|
      x1,x2,rh = t.prerequisites
      sh "mDiff #{x1} #{x2} #{t.name} #{REGION_HDR}"
      r = `mFitplane #{t.name}`
      puts "sh 'mFitplane #{t.name}' => #{r}"
      FIT_TBL << [c[0..1],r]
    end
  end

  task( :dif_fit_exec => DIFF_FITS ) do
    Montage.write_fitfits_tbl(FIT_TBL, "fitfits.tbl")
  end.invoke
end

### background-model
file( "corrections.tbl" => ["fitfits.tbl", "pimages.tbl"] ) do
  sh "mBgModel pimages.tbl fitfits.tbl corrections.tbl"
end

### background correction
C_IMGTBL=[]

file( "cimages.tbl" => ["corrections.tbl","pimages.tbl"] ) do
  pfiles = FileList["p/*.p.fits"]
  cfiles = pfiles.map do |s|
    src = s.sub(%r{p/(.*)¥.p¥.fits}, '¥1.p.fits')
    desc dst = src.sub(%r{(.*)¥.p¥.fits}, 'c/¥1.c.fits')
    file( dst => ["p/#{src}","corrections.tbl","pimages.tbl"] ) do |t|
      sh "(cd p; mBackground -t
              #{src} ../#{dst} ../pimages.tbl ../corrections.tbl)"
      Montage.collect_imgtbl( t, C_IMGTBL )
    end
    dst
  end

  task( :cimages_tbl_exec => cfiles ) do
    Montage.put_imgtbl( C_IMGTBL, "c", "cimages.tbl" )
  end.invoke
end

file( "mosaic.fits" => ["cimages.tbl", REGION_HDR] ) {|t|
  sh "mAdd -p c #{t.prerequisites.join(' ')} #{t.name}"
}

file( "mosaic.jpg" => "mosaic.fits" ) {|t|
  sh "mJPEG -ct 0 -gray #{t.prerequisites[0]} -1.5s 60s gaussian -out
          #{t.name}"
}

mkdir_p "p"
mkdir_p "d"
mkdir_p "c"

CLEAN.include %w[ p d c  ]
CLEAN.include %w[ mosaic.fits mosaic_area.fits mosaic.jpg ]
CLEAN.include %w[ fittxt.tbl fitfits.tbl ]
CLEAN.include %w[ rimages_all.tbl rimages.tbl ]
CLEAN.include %w[ pimages.tbl cimages.tbl simages.tbl ]
CLEAN.include %w[ diffs.tbl corrections.tbl ]
```

100行で記述

Montageワークフローのグラフ

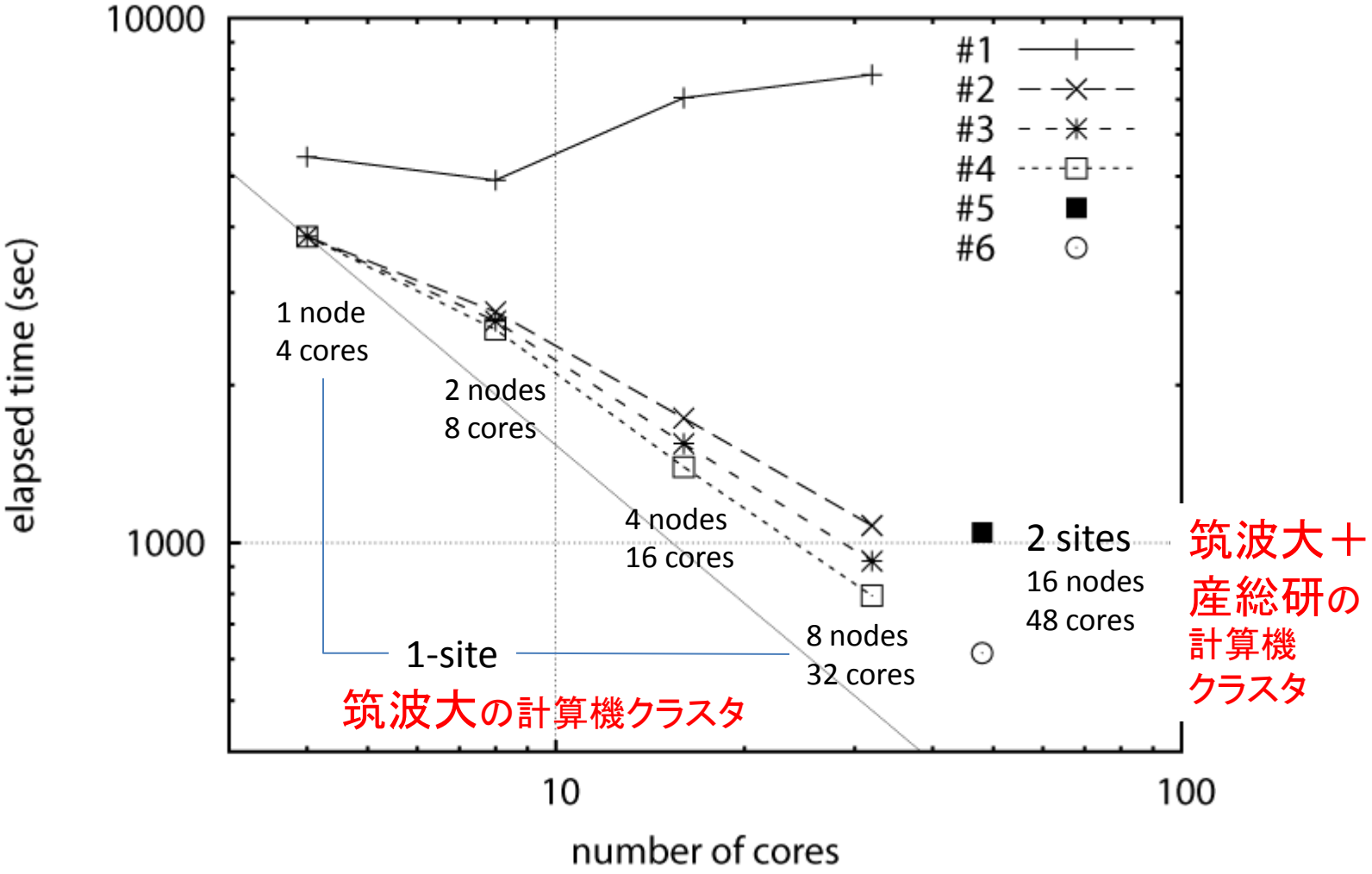
処理の流れ

入力ファイル
（23個）



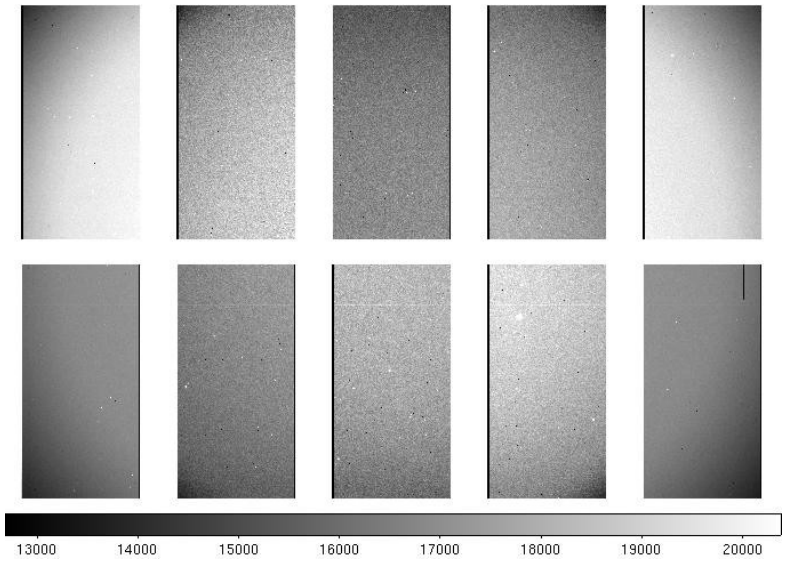
出力ファイル

Montage Workflow の実行時間

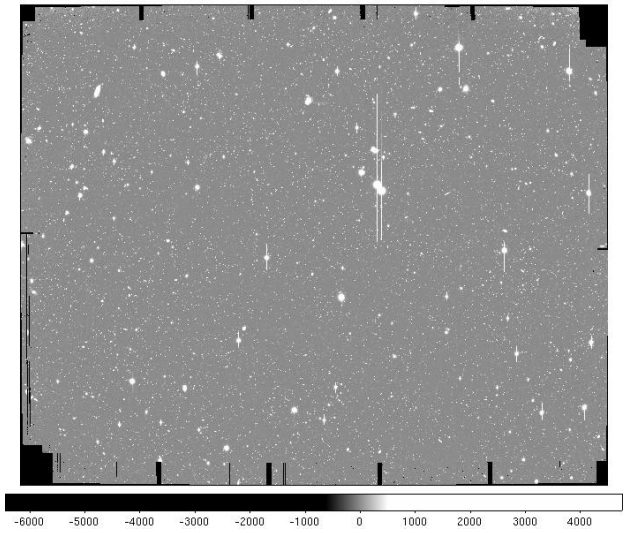


すばる望遠鏡主焦点カメラ SprimeCam

- SprimeCam：すばるの主焦点に 4096x2048 画素の CCD を 5x2 枚並べたカメラ。満月とほぼ同じ大きさの広い視野が特徴。
- SDFRED : SprimeCamが撮影したデータから、位置補正、感度補正などの処理をおこない、1枚の画像に合成するプログラム群
- http://subarutelescope.org/Observing/DataReduction/



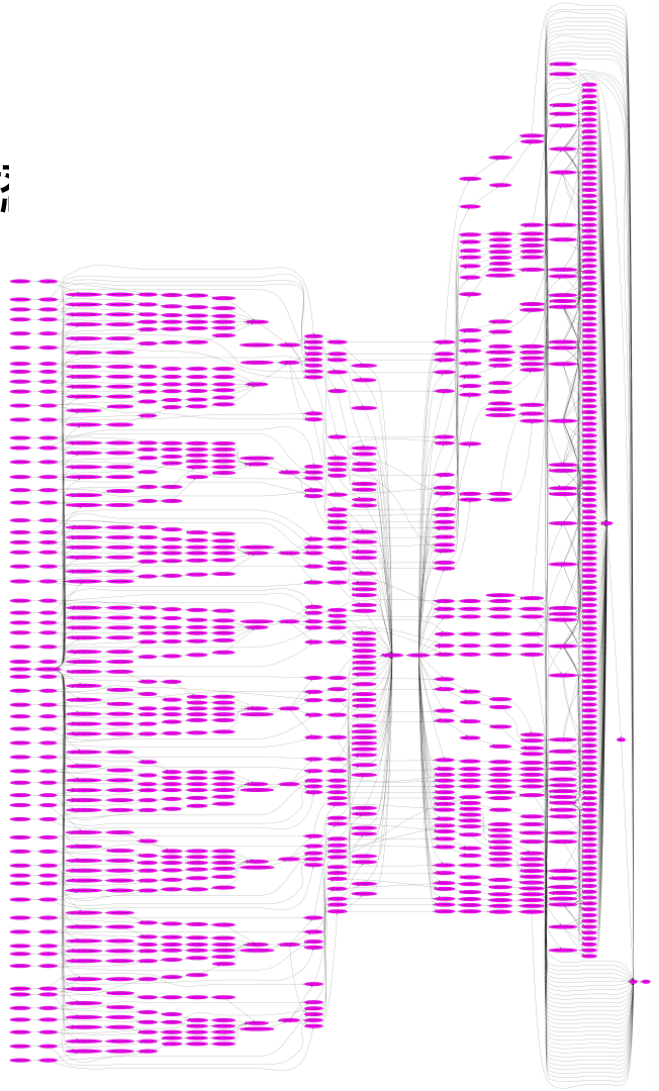
複数ショット
合成



すばる画像処理ワークフロー

処理の内容

1. 画像ファイル名の変換および画像の確認
2. bias引きおよびoverscanの切り取り
3. flat 作り
4. 感度補正 (flat fielding)
5. 歪補正および微分大気差補正
6. PSF 測定
7. PSF 合わせ
8. sky の差し引き
9. AG probeの影を自動でマスク
10. 画像を目で見て、悪い部分をマスク
11. 組み合わせ規則作り(matching)
12. 組み合わせ(mosaicing)



クロスマッチ

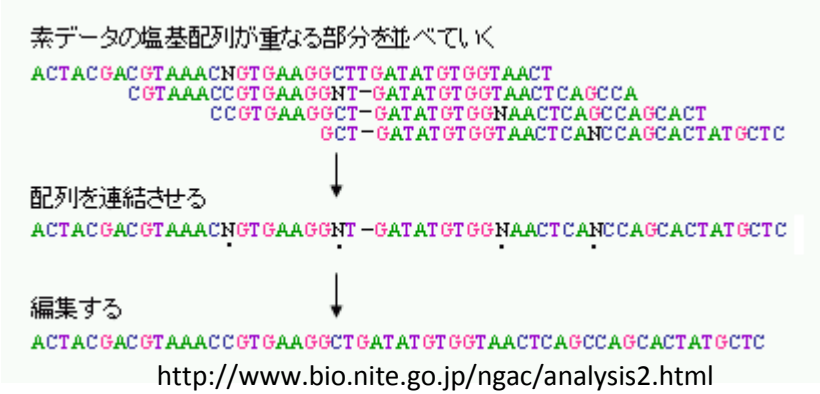
- 別の観測によって得られた星のデータから、同一天体のデータをマッチさせる処理
- 国立天文台（三鷹市）と宇宙科学研究所（相模原市）の9ノード、60コアで並列処理
  - 入力データ: 200億レコード、1ノードのみに配置
- Gfarm+Pwrake :　12.6時間

- （比較）国立天文台で処理
- HDFS+Hadoop:　10.2時間

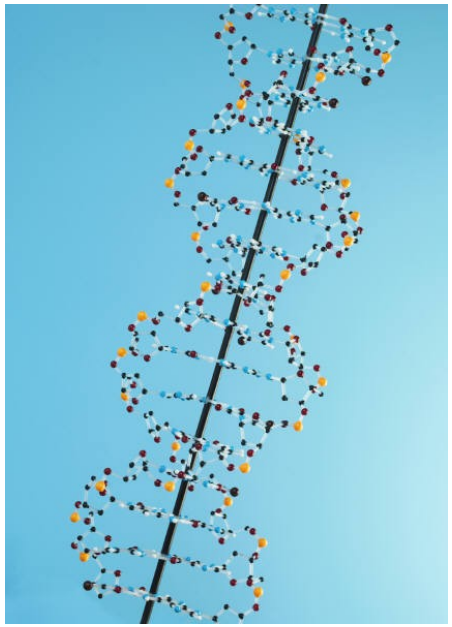
ゲノム解析

- ヒトの進化、家系、疾患を理解
- 膨大な塩基配列データ
- 膨大な解析計算処理

●塩基配列データの処理の例



http://www.bio.nite.go.jp/ngac/analysis2.html



2種類のゲノム解析ツールについて ワークフロー作成、Pwrakeによる実行

- 三嶋氏（長崎大）による
- https://github.com/misshie/Workflows

GATK

Dindel

Pwrake　今後の開発

- 性能・機能向上
  - 100万並列実行が目標
  - 自律分散実行
  - データ移動最小化
  - 障害対策
- マニュアル整備

まとめ

- 大量データの迅速な解析には、並列処理が必要
- ワークフロー記述言語として、Rakeを採用
- 並列分散ワークフロー処理システムPwarkeを開発
- 分散ファイルシステムGfarmを用いることにより、スケーラブルな性能向上を達成