並列分散ワークフロー処理システムの研究開発

田中昌宏（筑波大）

発表内容

- 並列分散処理の必要性
- ワークフローの記述について
- 並列分散ワークフロー実行システムPwrake
- 分散ファイルシステム Gfarm
- 天文データ処理ワークフローの記述
  - Montage
  - SDFRED1

並列処理の必要性

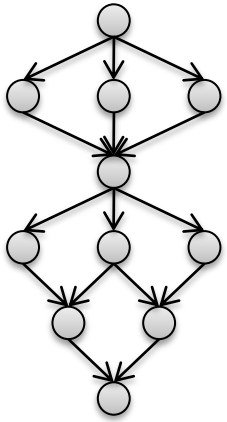
- データ量の増加
  - 観測装置の進化
  - アーカイブの蓄積
- 並列処理の必要性
  - 1コアあたりの性能が限界
  - 計算機の進化は、CPUコア数増加の方向へ
    - マルチコア
    - クラスタ

並列処理のやりかた

- 並列プログラムを書く
  - OpenMP
  - MPI
  - プログラミングが大変
- プロセスを並列に実行
  - 既存のプログラムをそのまま利用可能
  - 並列化は、ワークフロー処理系がやってくれる
  - すでに多くの処理系が存在

DAGによるワークフロー記述

- DAG (Directed Acyclic Graph)
  - 有向非循環グラフ
  - 多くのワークフロー処理系は、DAGで記述したワークフローを実行
- 問題点
  - 手書きは不可能
  - 再利用ができない
  - DAG生成ツールが必須



DAG をXMLで記述した例

```
<adag xmlns="http://www.griphyn.org/chimera/DAX"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.griphyn.org/chimera/DAX
        http://www.griphyn.org/chimera/dax-1.8.xsd"
    count="1" index="0" name="test">
<filename file="2mass-atlas-981204n-j0160056.fits"
        link="input"/>
 …
 <job id="ID000001" name="mProject" version="3.0" level="11"
        dv-name="mProject1" dv-version="1.0">
  <argument>
   <filename file="2mass-atlas-981204n-j0160056.fits"/>
   <filename file="p2mass-atlas-981204n-j0160056.fits"/>
   <filename file="templateTMP_AAAaaa01.hdr"/>
  </argument>
  <uses file="2mass-atlas-981204n-j0160056.fits" link="input"
        dontRegister="false" dontTransfer="false"/>
  <uses file="p2mass-atlas-981204n-j0160056.fits" link="output"
        dontRegister="true" dontTransfer="true"
        temporaryHint="tmp"/>
  <uses file="p2mass-atlas-981204n-j0160056_area.fits"
        link="output" dontRegister="true" dontTransfer="true"
        temporaryHint="tmp"/>
  <uses file="templateTMP_AAAaaa01.hdr" link="input"
        dontRegister="false" dontTransfer="false"/>
 </job>
 …
 <child ref="ID003006">
  <parent ref="ID000001"/>
  <parent ref="ID000006"/>
 </child>
…
</adag>
```

Makefileによるワークフロー記述

- Makeflow http://www.nd.edu/~ccl/software/makeflow/
- GXP make http://www.logos.ic.i.u-tokyo.ac.jp/gxp/
- 利点
  - タスクの依存関係から、実行順序を決定
  - 依存関係がないタスクを並列に実行可能
  - ルール定義により、共通の処理を記述
  - ファイルの有無やタイムスタンプを見て、途中から実行を再開
- 問題点
  - タスク実行中に、タスクを定義することができない。
  - ルールで記述できない依存関係もある。
  - 動的にMakefileを生成すれば、なんとか解決

Rake

- Ruby 版 make。ビルドツール
- Ruby 1.9 から標準添付
- タスク記述ファイル : Rakefile
- タスク記述文法 : Ruby
  - 内部DSL (Internal Domain Specific Language)と呼ばれる。
  - Makefile のような専用言語は、外部DSL と呼ばれる。
- Rakefile はRubyスクリプトとして実行されるので、Rubyでできることはすべて可能。

Rakefile 記述例: ソースコードのビルド

```
SRCS = FileList["*.c"]
OBJS = SRCS.ext("o")

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

file "prog" => OBJS do |t|
  sh "cc -o prog #{t.prerequisites.join(' ')}"
end

task :default => "prog"
```

Rakefile の記述力の例

(a b), (c d), … といったファイルのペアについて、

```
imarith a - b ab
imarith c - d cd
```

と、演算したい場合：

```
LIST = [["a","b"],["c","d"]]
for x in LIST
  file x.join => x do |t|
    sh "imarith #{t.prerequisites.join(' - ') #{t.name}"
  end
end
```

と、タスクをループで記述できる。

並列分散ワークフロー処理システム Pwrake

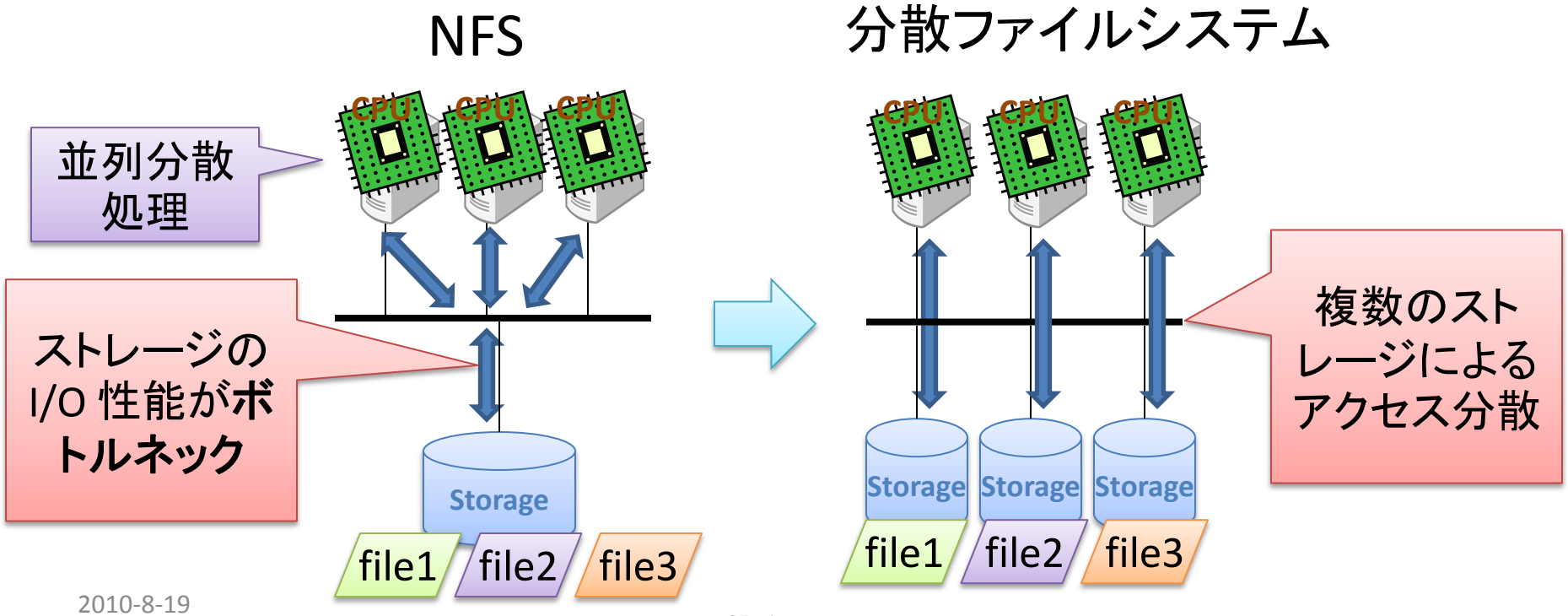
- Rakeには、並列分散処理の機能はない
- 並列分散機能を拡張：Pwrake
  - Parallel Workflow extension for Rake
    - 「プレイク」と呼んでます
    - http://github.com/masa16/pwrake
  - タスク記述は、Rakeと同じ
  - クラスタ内の各マシンにSSHで接続
  - プロセスを並列に実行

分散ファイルシステム

- Pwrake では、分散ファイルシステムの利用が前提
  - ファイル転送の記述が不要
  - ファイルのタイムスタンプにより、タスクが実行済かどうかを判断
    - 複数マシン間のタイムスタンプの一貫性が必要
  - 並列I/O性能

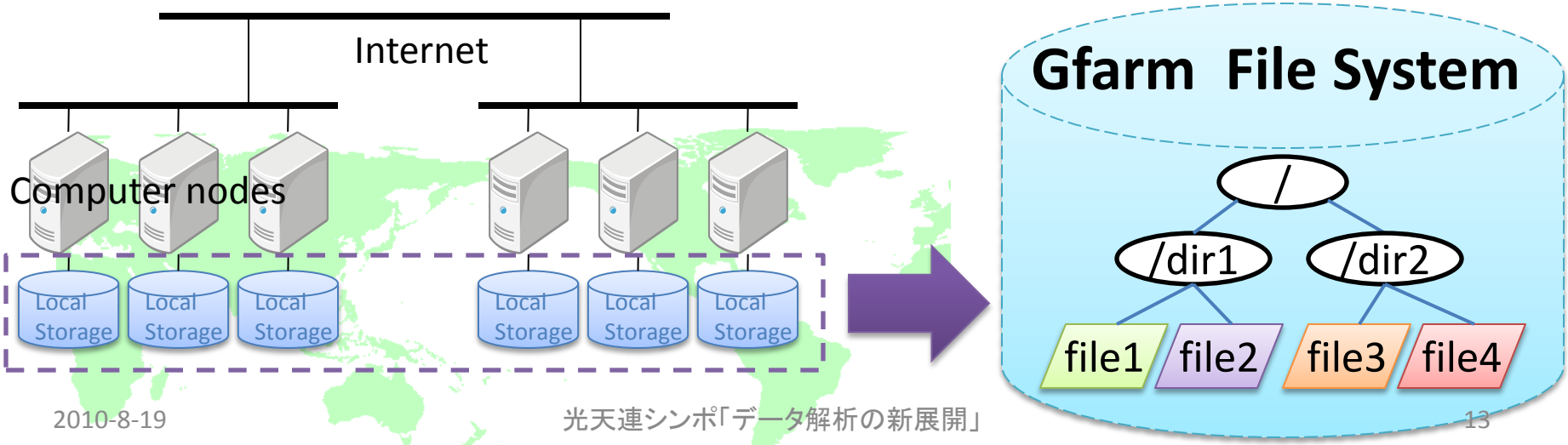
分散ファイルシステムによる並列I/O

- 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



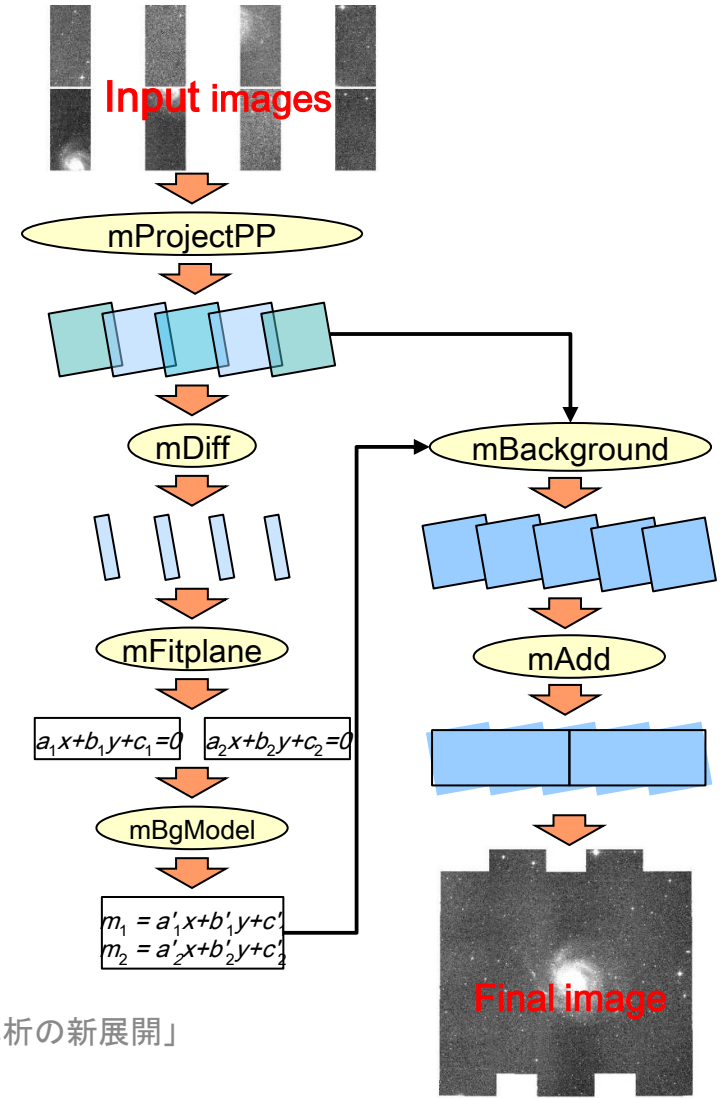
Gfarm

- 広域分散ファイルシステム
- 各計算機のストレージを統合
- 計算機グリッドにおいて使用実績
  - 普通の計算機クラスターにも適用可能
- http://datafarm.apgrid.org/



Montage Workflow

- Montage : FITS画像モザイキングソフトウェア
- http://montage.ipac.caltech.edu/
- ワークフローをRakefileとして記述

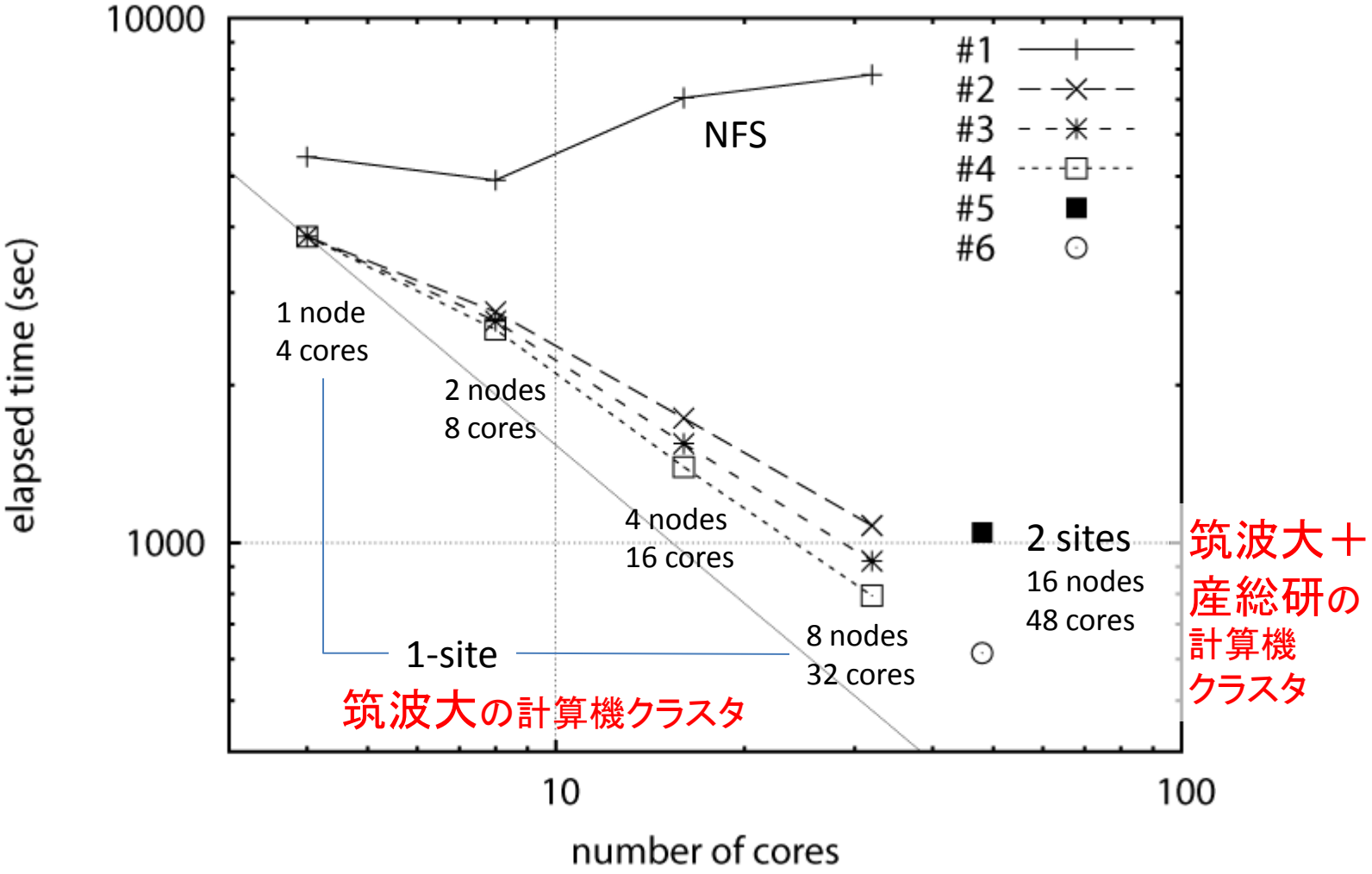


Montage Workflow (基本バージョン)

```
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 ]
```

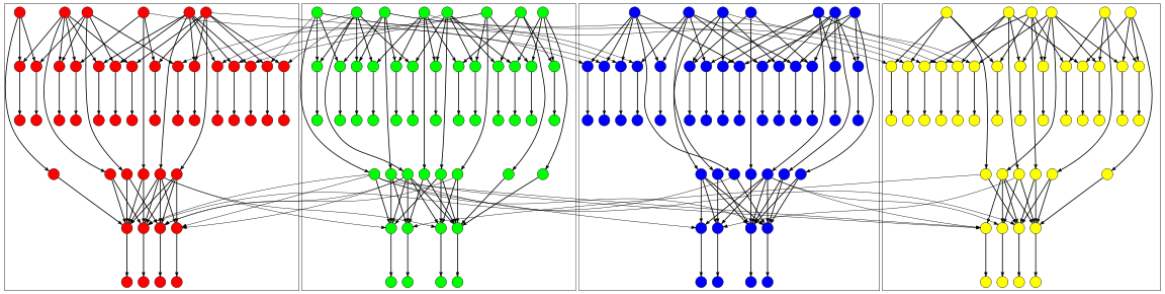
100行で記述

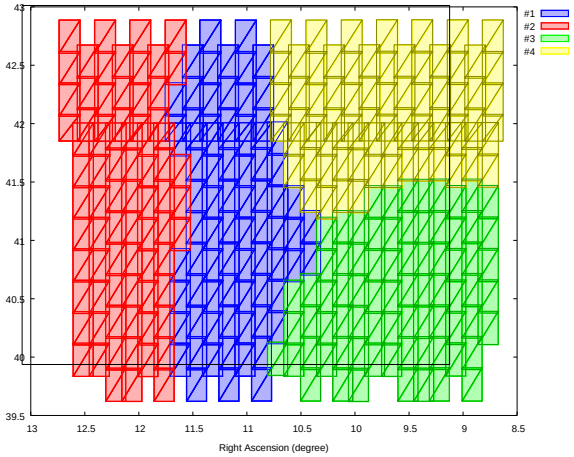
Montage Workflow の実行時間



タスク配置による性能向上

- 複数拠点のクラスタを用いたワークフロー実行
- 適切なタスク配置により、拠点間のファイル転送を少なくする
- 「エッジカット最小」となるグラフ分割問題を解くことにより、タスクをグループ化
- 座標によるグループ化と、同等の結果
- 座標情報がなくても、ワークフローを記述するだけで適切に配置

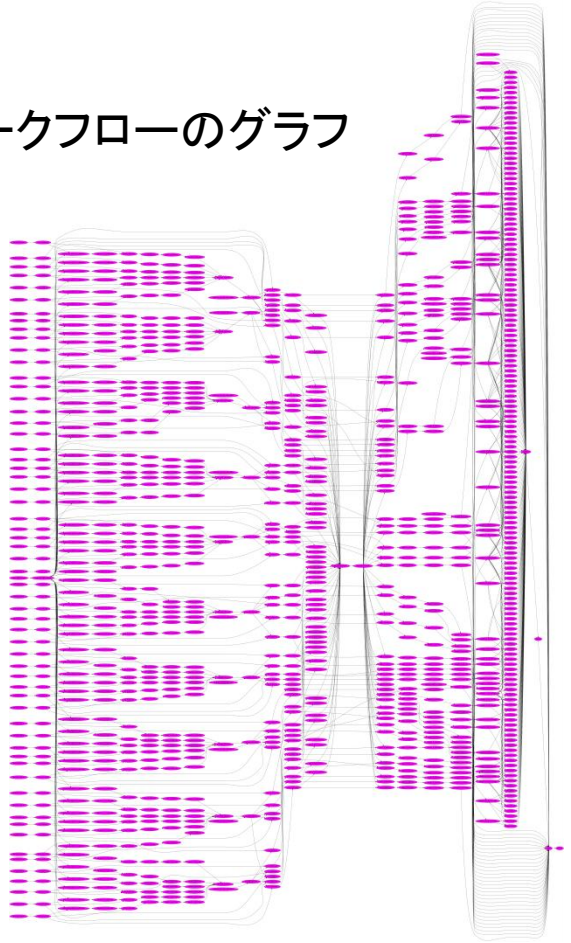




SuprimeCam Data Reduction SDFRED1

- 下記スクリプトをRakefileで記述
  - overscansub.csh
  - mask_mkflat_HA.csh
  - blank.csh
  - ffield.csh
  - distcorr.csh
  - fwhmpsf_batch.csh
  - fwhmpsf.csh
  - psfmatch_batch.csh
  - psfmatch.csh
  - skysb.csh
  - mask_AGX.csh
  - blank.csh
  - makemos.csh
- トレーニング画像の自動処理に成功

ワークフローのグラフ

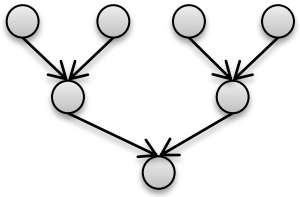


Pwrake開発の今後の予定

- 障害対策
  - 障害情報の取得
  - タスクの再試行
  - 故障ノードの判別
- 性能向上
  - タスク配置アルゴリズムの向上
  - ファイル複製の自動配置

並列分散を意識した解析ソフトの設計

- ネットワーク遅延を考慮した設計
  - open 時にメタデータサーバへアクセス
  - 不必要な open – close を避ける
  - Montage では、CFITSIO と Montage側の双方にこの問題がある
- 並列処理の結果を集める処理の効率化
  - ファイルの結合処理など
    - SDFRED1全体処理時間の半分が imcio2a
  - 段階的な結合処理が必要



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に着目
- 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