Vol.2015-HPC-150 No.36
2015/8/6

すばる HSC データ処理の性能調査と Gfarm/Pwrake の適用

田中昌宏[†1†2]　建部修見[†3†1†2]　川島 英之[†3†1†2]　村田 直郁[†3†2]

すばる望遠鏡に搭載された焦点面カメラ Hyper Suprime-Cam (HSC) が出力する大量データの一次処理を行うパイプラインシステムの高速化が本研究の目的である．高速化に向けて，HSC パイプラインの処理内容，および，現状のシステムの問題点を把握するための調査を行った．パイプライン処理をワークフローシステム Pwrake で実行するために，処理フローを Rakefile で記述した．性能調査のため，Gfarm ファイルシステム上で HSC パイプラインを実行した．調査の結果，HSC パイプライン用に開発された並列処理システムを用いると 112 コアで約 40 分かかる処理において，Pwrake により 224 コアの使用が可能になり，処理時間も約 12 分に短縮できることを確認した．また，Gfarm メタデータサーバのバックエンド DB の I/O が処理性能に影響することもわかった．

1. はじめに

ハワイ島マウナケア山頂で運用されている口径 8.2m の「すばる望遠鏡」に新しく搭載された主焦点カメラ Hyper Superime-Cam (HSC)[1]は，有効視野角がそれまでの Suprime-Cam に比べて 3 倍の 1.5 度角という広視野が特徴である．この世界最高水準のサーベイ能力を発揮すべく，すばる望遠鏡の約 300 晩を HSC サーベイに割り当てる HSC 戦略枠が 2014 年 3 月から 5 年の計画で始まっている．ターゲットとするサイエンスの 1 つに，超新星の探査がある[2]．広視野である HSC は超新星観測に適しており，一晩で多くの超新星を観測できる．しかし，爆発が始まった直後の超新星の観測例は少ない．特に，超新星爆発の開始直後にはショックブレイクアウト[3]と呼ばれる現象が予想されており，このとき超新星が最も明るくなるため，遠方の超新星の発見が期待されるが，その持続時間は 1 日程度と考えられている．また，超新星爆発の物理的理解のためには，爆発直後の多波長での追観測による多角的な検証が不可欠であるが，そのためには，観測データを取得後，いかに早く超新星を発見するかが重要な鍵となる．

超新星の早期発見のために，機械学習による超新星検出の自動化など，いくつかのアプローチで研究が進められている．本研究は，CCD が出力したデータの一次処理パイプライン[4]（ここでは単に「パイプライン」と呼ぶ）の高速化が目的である．HSC パイプラインの処理は，HSC が出力した観測データに対して，装置の特性に由来する補正，および，明るさと天球上の位置のキャリブレーションを行い，整約済みのデータを生成することでる．

開発チームによって実装された HSC パイプラインのプログラムには，タスクを複数ノードで並列に実行する機能が Python 版の MPI で実装されている．しかし，比較的ファイル IO を多く行う処理であるにもかかわらず，ファイル共有については単に Lustre のような高速分散ファイルシステムによる共有を想定しており，並列処理システムについては，CCD の枚数を超えるような並列実行を行わない仕様となっており，高並列は考慮されていない．また，HSC パイプラインに対して，これまで高速化のために必要な性能プロファイル調査が行われていない．そこで，本稿では，さらなる高並列化による高速化を目的として，HSC パイプライン処理の性能プロファイルについて調査し，さらに，スケールアウト型の分散ファイルシステム Gfarm[5]およびワークフローシステム Pwrake[6]を適用し，高速化の可能性について調査する．

本稿では，第 2 節で HSC とパイプラインシステムの概要について述べる．第 3 節でパイプライン処理を Rakefile に記述する方法について述べる．第 4 節で性能調査について述べる．第 5 節で関連研究について述べ，第 6 節でまとめと今後の課題について述べる．

2. HSC について

2.1 装置の概要

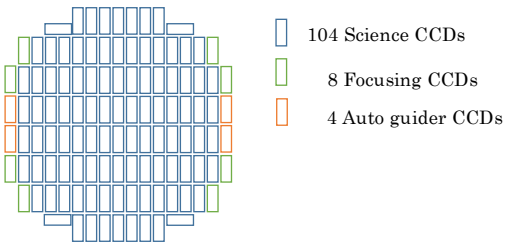

図 1　HSC 焦点面における CCD の配置

初めに，HSC の装置に関して簡単に述べる．HSC は，すばる望遠鏡の完成当初から主力の観測装置として運用されてきた主焦点カメラ Superime-Cam に代わる装置として開発された．HSC の特徴は，有効視野角が Suprime-Cam の 3 倍となる 1.5 度角とすることにより，広い天域のサーベイを可能にしたことである．HSC に用いられた CCD の枚数は，観測用 104，合焦用 8，ガイド用 4 の合計 116 である．これらの CCD は，カメラの焦点面に図 1 の様に配置され

†1 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba
†2 独立行政法人科学技術振興機構 CREST
JST CREST
†3 筑波大学システム情報系
Faculty of Engineering, Information Systems, University of Tsukuba

る．1 枚の CCD の画素数は，観測に使われないオーバースキャン領域含んで 4272×2272 である. 可視光から赤外線にかけて g, r, i, z, Y の 5 つの波長バンドフィルターを持つ.

2.2 パイプラインの概要

HSC パイプラインのソフトウェア[4]は，国立天文台，Kavli-IPMU，プリンストン大学のチームで開発されている．HSC パイプラインの基本部分に，チリに建設が予定されている口径 8.4m の可視光赤外線望遠鏡 Large Synoptic Survey Telescope (LSST) のデータ処理のために開発されたソフトウェアを利用している．この LSST ソフトウェアは，CCD 画像処理における基本的なデータ操作機能および解析タスクを提供する．LSST ソフトウェアの部品を活用して，HSC パイプラインでは，HSC 固有の解析手続きを追加し，さらに Torque へのジョブ投入，Python 版 MPI による並列分散実行といった拡張が行われている．これらのソフトウェアは，C/C++の API を SWIG でラップし，Python から呼ぶという実装である．パイプラインシステムは現在も開発が続けられており，今後本稿の記述と合わなくなる可能性もある.

2.3 HSC パイプラインの処理内容

HSC パイプライン処理の大まかな流れは次のようになる．（用語はマニュアルに従う）

- Making Detrends
- Single Frame Processing
- Make a SkyMap
- Mosaic
- Coadd Processing

(1) Making Detrends

HSC パイプライン処理の前半で, CCD 画像処理に共通の画像整約（リダクション; reduction）処理を行う．この処理は，天体観測用 CCD の動作原理に由来する．CCD の仕組みを簡単に述べると，入射光がフォトダイオードに吸収されると電子-正孔対を生成し, その電荷を蓄えて電位として読み出す．例えると，雨量の測定するとき，バケツに雨水を溜めてその水位を測ることに似ている. CCD は各ピクセルの電荷は読み出し装置に転送され，そこで測定された電位が CCD からの出力データとなる．このデータから，入射光に相当する量に変換する処理がリダクションである.

リダクションでは, 各 CCD に対して, bias, dark, flat, fringe という 4 種類のデータが必要になる．Bias は，電位のゼロ点のデータである．Bias データには，露出時間 0 秒で測定したデータを用いる．Dark は，入射光がゼロのときに発生する電荷に由来する．Dark データには，シャッターを閉じて測定したデータを用いる．Flat は，CCD のピクセルによって感度にムラがあり，そのムラを補正するためのデータである．Flat の参照データとして，ドーム内部を一様な光源とみなして観測したデータを用いる．Fringe は，CCD 裏面反射によって起こる光の干渉に由来する. HSC では Y バンドのみ fringe 補正が必要とされる．Fringe データには観測データが用いられる.

リダクション処理の実行前に，Bias, dark, flat, fringe の測定データからリダクションに用いる Detrend データを作成する処理が必要である．その処理を行うコマンドラインインタフェースとして, それぞれ reduceBias.py, reduceDark.py, reduceFlat.py, reduceFringe.py が提供されている．これらのコマンドは，2.4 節で述べる並列処理システムによって，CCD 毎の並列処理をサポートする.

(2) Single Frame Processing

Detrend データ作成の次は，frame ごとのデータ処理を行う．Frame とは，観測データの最小単位であり，1 回の露出で 1 つの CCD から得られるデータが 1 frame である．Frame ごとの処理では，主に次の処理を行う：

- Bias, dark, flat, fringe の補正（リダクション）
- 天体の検出及び測定

Frame ごとの処理を行うコマンドラインインタフェースとして，reduceFrames.py と hscProcessCcd.py の 2 種類が用意されている．reduceFrames.py は，2.4 節で述べる並列処理システムにより，複数の frames に対して CCD 単位で並列処理を行う．一方，hscProcessCcd.py は，1 つの frame に対して処理を行う．HSC パイプラインの性能調査では，reduceFrames.py を用いて並列処理を行う.

(3) Making a SkyMap

1.5 度の視野を 1 枚の画像として扱うと，メモリに乗らないなど処理上の不都合が生じるため, 空の領域を分割し，その部分領域を単位として処理を行う．HSC パイプラインでは，領域の指定を，tract と patch の 2 段階のレベルで行う．Tract の大きさは HSC の視野と同じ約 1.5 度であり，Coadd を行う単位である．同じ tract の中では共通の投影軸を持つ.

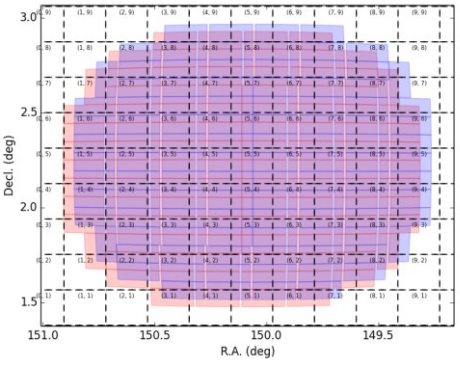

図 2 露出 2 回分の frame（赤と青の四角）と patch（破線のグリッド）の位置を赤道座標系で表示

1 つの tract は, 図 2 の破線のグリッドで表したように，約 10×10 個の patch に分割する．Patch は，画像作成および天体検出の処理を行う単位である．本稿の調査では，

SkyMap 作成に makeDiscreteSkyMap.py というコマンドを用いた．これは，入力データの座標に適した tract 領域を入力画像に基づいて自動的に設定し，その tract ID は 0 となる．このタスクはシングルコアで動作し，実行時間は 30 秒程度であるから，並列実行性能の調査対象とはしない．

(4) **Mosaic**

次に，星の位置と明るさに基づいたキャリブレーション（較正）を行う．このタスクを実行するコマンドは mosaic.py である．入力ファイルは，FITS ファイルに格納された星のカタログ約 7 GB と，frame ごとの処理で検出された星のリストである．mosaic.py で行う主な計算は，線形一次方程式の求解であり，数万次元と大きな疎行列を解く場合もある．現在の実装では自動的に Intel MKL または OpenBLAS の dgesv 関数を探し，見つからなければ Eigen の solve 関数を使う．本調査では，OpenBLAS の dgesv 関数を用いて，1 ノード内のマルチコアで実行する．

(5) **Coadd Processing**

パイプライン後半の処理は，patch 領域ごとの処理である．処理内容は，

1. Warp
2. Assemble the coadd
3. Process the coadd images

の 3 種類である．1 番目の Warp は，CCD 座標の画像を tract で決められた天球座標系に投影する処理である．CCD の座標として mosaic.py において星の位置から決定された情報が使われる．視野の周辺では歪みの補正が大きくなる．2 番目の Assemble the coadd では，複数回の露出の画像を足し合わせる．足し合わせによってノイズを減らし，CCD の間の隙間を埋めることができる．3 番目の Process the coadd images では，足し合わされた画像に対して再び星の検出と測定を行い，最終的な天体リストを出力する．

Coadd 処理についても，並列処理版と patch 領域ごと版の 2 種類のコマンドラインインタフェースがある．並列処理版は stack.py というコマンドである．stack.py では，並列処理の単位は patch ごとであり，上記 1.から 3.までの処理をすべて実行する．

2.4 HSC パイプライン並列処理システム

HSC パイプラインでは，並列処理のため，Torque へジョブを投入する機能および MPI によりタスクを分配する機能を独自に実装している．HSC パイプラインでは，処理すべきデータセットをコマンドラインから指定する仕様が統一されている．並列処理を行う場合，コマンドラインを解析した後，自動的にパラメータを設定して Torque にジョブを投入する．投入されたジョブは，Python 版の MPI を用いて各計算ノードでワーカープロセスを起動する．各ワーカーが処理するデータの ID（visit 番号，CCD 番号，Patch 番号など．詳細は 2.6 節）は，Python の multiprocessing.Pool を MPI で拡張したモジュールを用いてワーカーに分配される．ワーカー側では，Pool 経由でデータ ID を受け取り，そのデータに対して処理を実行する．

2.5 調査対象の処理

本稿において調査対象とする HSC パイプライン処理は，次のコマンドを順に発行して起動する．用いたデータが後述のように I バンドの観測データであるため，fringe 処理は行わない．

- reduceBias.py
- reduceDark.py
- reduceFlat.py
- reduceFrames.py
- makeDiscreteSkyMap.py
- mosaic.py
- stack.py

これらのうち，前半の reduce*.py と最後の stack.py については，2.4 節で述べた並列処理システムを用いて並列処理を行う．

2.6 リポジトリ

HSC パイプラインで扱うファイルの多くは，天文学における標準フォーマットである FITS 形式である．HSC の観測装置が出力したデータも FITS ファイルとして渡され，それがパイプラインの入力データとなる．HSC パイプラインにおいて，入出力ファイルを格納するディレクトリを，データリポジトリと呼んでいる．このリポジトリの構造は，HSC パイプラインシステムで決められており，レジストリデータベースによってファイルを素早く検索できるようにしている．また，直接パスを指定してファイルにアクセスすることもできる．

HSC パイプラインを実行する前に，入力データをリポジトリに登録する必要がある．これを行うために hscIngestImages.py というコマンドが用意されている．入力ファイルのレジストリ内のファイルのパスは，FITS のヘッダ部分に記録されたメタ情報に基づいて決められる．入出力ファイルのパス名のルールについて，代表的なものについて以下に述べる．

(1) 入力ファイル

リポジトリに登録した入力ファイルのパス名は次のようになる．

```
COSMOS/2014-03-28/00817/HSC-I/HSC-0001226-000.fits
```

これはリポジトリの先頭ディレクトリからの相対パスである．名前の付け方は，ディレクトリについては，`COSMOS` は観測対象の名前，`2014-03-28` は観測日，`00817` はポイティング番号，`HSC-I` はフィルタ名である．ファイル名の部分は，`0001226` が visit と呼ばれる撮影露出ごとの ID，最後の `000` が CCD 番号である．

(2) Detrend 出力ファイル

Bias データの出力ファイルのパスの例を次に示す．

```
CALIB/BIAS/2014-03-28/NONE/all/BIAS-000.fits
```

これもリポジトリ先頭からの相対パスである．CALIB/BIAS で Bias データであることを示し，2014-03-28 は観測日である．NONE の部分はフィルタ名だが，Bias や Dark の場合にはフィルタ情報は不要であるから NONE が入る．all は処理のバージョンであり，入力ファイルセットや処理方法が異なるデータ保存するときはこの名前を変更する．ファイル名の 000 は CCD 番号である．

(3) Single frames 出力ファイル

下記は Single frames の出力ファイルのパスの例である．

```
rerun/cosmos/00817/HSC-I/output/SRC-0001226-000.fits
```

パス名の構成要素のうち，ディレクトリ名の rerun は固定，cosmos はコマンドのオプションで変更可能な先頭ディレクトリ名，00817 は pointing 番号，HSC-I はフィルタ名である．ファイル名の中の 0001226 が visit 番号，000 が CCD 番号である．Single frames の処理では，他にも多くの中間ファイルを生成するが，それらのパス名は output と SRC の部分を変えたものである．

(4) Coadd 出力ファイル

次に Coadd の処理で出力されるファイルパスの例を示す．

```
rerun/cosmos/deepCoadd/HSC-I/0/5,6.fits
```

パス名中の HSC-I がフィルタ名であり，その次の 0 は tract 番号である．ファイル名の 5,6 は，tract の中の patch のグリッド位置であり，これが patch の ID となっている．Coadd の処理においても，他に多くの出力ファイルを生成する．

3. Pwrake ワークフローの記述

HSC パイプラインをワークフローシステム Pwrake で実行するため，Rakefile 形式でパイプラインのワークフローを記述した．記述する際，できるだけ処理を変えないようにしたが，現状では出力ファイルの数やサイズが若干異なっており，そのチェックは今後の課題である．

HSC パイプラインでは，並列処理の機能をサポートしており，コマンドラインインタフェースを提供する．一方，Pwrake でワークフローを記述するためには，個々のタスクを実行するコマンドが必要である．以下では，detrend, single frame, coadd 処理の書き換えについて述べる．タスクレベルで並列化できない makeDiscreteSkyMap.py と mosaic.py については前述のパイプラインから書き換えは不要である．

3.1 Detrend 作成タスク

reduceBias.py などの Detrend 作成コマンドは，Toruqe にジョブを投入するインタフェースであり，パイプラインシステムには CCD 毎に処理するコマンドインタフェースが用意されていない．そこで，次のように python にタスクを実行するスクリプトを与え，処理対象の CCD 番号を引数で指定する．

```
python -c 'import hsc.pipe.tasks.detrends;
 hsc.pipe.tasks.detrends.BiasTask.parseAndRun();'
 ../repo --rerun all_bias
 --detrendId calibVersion=all
 --id field=BIAS dateObs=2014-03-28 ccd=30
```

改行は表示の都合上入っているが，実際は入らない．
他の dark, flat, fringe についても同様である．

3.2 Single frame processing

reduceFrames.py の処理を frame 毎に行うコマンドとして，hscProcessCcd.py というコマンドが用意されている．起動の例は下記の通り．

```
hscProcessCcd.py ../repo --rerun cosmos
 --id visit=1226 ccd=0 field=COSMOS filter=HSC-I
 dateObs=2014-03-28 --config isr.doFringe=False
```

この処理は frame ごとに行うため，プロセスの延べ数は CCD 数×visit 数となる．

3.3 Coadd processing

stack.py に対応する処理として，warp, assemble, process の 3 つの処理を patch 毎に行うコマンドが用意されている．

```
makeCoaddTempExp.py ../repo --rerun cosmos
 --id tract=0 patch=5,6 filter=HSC-I
 --selectId ccd=0..103

assembleCoadd.py ../repo --rerun cosmos
 --id tract=0 patch=5,6 filter=HSC-I
 --selectId ccd=0..103

hscProcessCoadd.py ../repo --rerun cosmos
 --id tract=0 patch=5,6 filter=HSC-I
```

3.4 入力ファイル名の取得

Rakefile は Makefile と同様に，入出力ファイルによってタスクの依存関係を指定する必要がある．HSC パイプラインの各タスクは複数ファイルを入力し，複数ファイルを出力するが，今回はすべてを指定せず，依存関係の構築に必要な代表ファイルを指定した．入力ファイルのリストを得るために，リポジトリ内のパス名のルールを活用し，Rakefile に次のように記述して入力ファイルのリストを得る．

```
FRAMES = FileList["#{TOP}/#{FIELD}/#{DATE}/"+
         "#{POINTING}/#{FILTER}/*.fits"]
```

前出のように HSC パイプラインのコマンドには，ファイル名ではなく，visit 番号および CCD 番号などを指定する仕様である．その番号はファイル名から抽出する．

Coadd 処理については，入力ファイルのリストを得るために工夫が必要であった．この処理は，図 2 で示したような patch 領域に重なる CCD のデータを足し合わせる処理であるから，各 patch で必要となる入力ファイルは CCD の座

標によって決まる．ところが CCD の位置は mosaic.py で決まるため，Coadd 処理の入力ファイルリストが得られるのはその後である．また，各 patch の入力ファイルリストを得るコマンドがパイプラインには用意されていない．そこで，パイプラインのパッケージに含まれる Python スクリプトを利用して，patch に重なる入力ファイルを書き出すスクリプトを作成し，その出力を Rakefile で読むことによって，Coadd タスクを定義できるようにした．

3.5 ワークフローのグラフ

Rakefile で定義した HSC パイプラインの処理フローのグラフの一部を図 3 に示す．この図では，丸の頂点でタスクを表し，四角の頂点でファイルを表す．どのタスクも複数のファイルを出力するが，この図では便宜上 1 つの四角の頂点で表すことにする．タスクによって並列化の単位が異なり，Bias, Dark, Flat タスクが CCD 単位，Frames タスクが frame 単位，Coadd タスクが patch 単位で並列化可能である．Coadd 処理では，patch 領域と交差する frame データを結合するため，一般に複数の CCD のデータが入力ファイルとなる．また，使用コア数が mosaic.py とそれ以外のタスクで異なるため，Rakefile は mosaic.py とその前後の 3 つに分割して記述した．

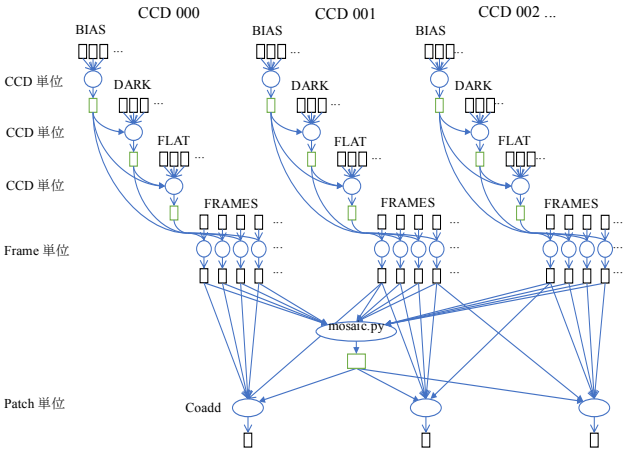

図 3　HSC パイプラインのワークフロー

3.6 絶対パスの相対化

HSC のパイプラインを Gfarm/Pwrake で実行する際の問題の 1 つに，ファイルの絶対パスを使用することがある．例えば，データをリポジトリに登録する際，リポジトリ内から元のファイルを指すシンボリックリンクを作るとき，元のファイルを絶対パスで指す．（ただし，データを移動する方式と，シンボリックリンクを張る方式で選択できる．）しかし，Pwrake では Gfarm ファイルに対して相対パスを用いることを想定している．というのは，シングルスレッドの gfarm2fs を並列動作させるため，Pwrake ではコアごとにマウントポイントを作るからである．複数のマウントポイントはそれぞれディレクトリ名も異なる．したがって，Pwrake のタスクで絶対パスを使う場合，gfarm2fs の並列化ができず，並列処理性能を発揮することができない．今回の性能調査では，シンボリックリンクを作成する処理の後で，シンボリックリンクを相対パスに変更ことによって対応した．

4. 性能調査

4.1 実行環境

表 1　測定環境

クラスタ	IPMU クラスタ
CPU	Intel Xeon E5-2650 2.30GHz
主記憶容量	64GiB
コア数/ノード	20
計算ノード数	60
ネットワーク	InfiniBand
OS	CentOS release 6.5
HSC pipeline	hscPipe 3.8.1
Gfarm	ver. 2.5.8.14
Ruby	ver. 2.2.2
Pwrake	ver. 2.pre

評価環境について表 1 に示す．用いた計算機クラスタは，Kavli-IPMU に設置されたクラスタである．計算ノードの他に Gfarm メタデータサーバノードが設置されている．計算ノードは 60 ノードであり，それぞれ 20 コアを搭載するが，本稿の調査ではその一部のコアで実行する．Gfarm で書き込むノードをすべてローカルにするため，gfarm2.conf のオプションにて schedule_idle_load_thresh 100 と設定した．また，Gfarm のバックエンド DB である PostgreSQL の保存先を RAM ディスクに設定した．これはメタデータサーバの性能の問題によるもので，詳細は 4.4 節で述べる．

ネットワークの速度の iperf による実測値と，Gfarm ファイルに対する読み書きのバンド幅の dd による実測値（ブロックサイズ 64KB，書込サイズは 1GB，読込サイズはディスクが 100GB，キャッシュが 1GB）を表 2 に示す．このクラスタではネットワークが十分速いため，ローカルとリモートの転送速度に差はない．

表 2　使用クラスタにおけるネットワーク速度および Gfarm ファイル転送速度の実測値

InfiniBand Network		4.5 Gbps
Gfarm	local disk write	79 MB/s
	local disk read	173 MB/s
	remote disk read	173 MB/s
	local cache read	1.7 GB/s
	remote cache read	471 MB/s

4.2 入力データ

性能調査に用いた入力データを表 3 に示す．これらのデータはすべて FITS ファイルである．観測データは，HSC による 1 時間の観測データである．パイプライン処理を実行する前に，これらの入力ファイルを Gfarm ファイルシステムにコピーする際，カタログデータについては Mosaic タスクを実行するノードに配置し，他は処理ノードにラウンドロビンで配置した. その後 hscIngestImages.py でリポジトリに登録した.

表 3　パイプライン入力データ

	Data size	# of fiels	# of shots
bias データ	10.2 GB	560	5
dark データ	10.2 GB	560	5
flat データ	20.4 GB	1120	10
観測データ	24.5 GB	1344	12
カタログデータ	6.9 GB	2724	

4.3 経過時間の測定結果

この入力データに対する HSC パイプライン処理性能を調べた．ここで使用した計算機ノード数は 16 であり，各ノードで 7 コアを使用した．HSC パイプラインのファイル共有のため，Gfarm ファイルシステムを用いた．Mosaic 以外のタスクについては，1 ノードあたり 7 コア使用し，元のパイプラインに対しては 8, 16 ノード，Pwrake に対して 8, 16, 32 ノードで測定した．Mosaic タスクについては，複数ノードでの実行をサポートしていないため，1 ノードを使用し, OpenBLAS の dgesv によって 20 コアで実行している．以上の設定で HSC パイプラインを実行し, 測定した経過時間を表 4 に示す．オリジナルの HSC パイプラインの並列システムで実行したときの経過時間を見ると，Bias, Dark, Flat タスクの実行時間が約 1 分から 4 分である一方, Frames タスクが 40 分，Mosaic タスクが 16 分，Coadd タスクが 64 分と，後半のタスクの経過時間が長く，これらのタスクの高速化が課題であることがわかる．しかしこの測定においてすでに使用コア数は 16×7=112 に対して，HSC パイプラインでの並列実行の最大数は，実装上，Frames タスクでは CCD 数の 112，Coadd タスクでは patch 数の 85 であり，これ以上の数のコアを使用しても性能は向上しない.

参考までに，パイプライン実行中に読み書きしたファイルに関する統計を表 5 と
表 6 に示す．表 5 が元のパイプラインでの結果，
表 6 が Pwrake での結果であり，どちらも 112 コアのケースである．この統計は，パイプラインでデータ入出力を扱う Butler というクラスのコードに，ファイル入出力の開始・終了時刻，ファイル名とファイルサイズを出力するパッチを挿入することによって取得した.

表 4　パイプライン実行経過時間

	Original		Pwrake		
nodes*cores	8*7	16*7	8*7	16*7	32*7
total cores	56	112	56	112	224
Bias	4m04s	1m21s	2m08s	1m09s	1m07s
Dark	4m10s	1m25s	2m19s	1m18s	1m06s
Flat	10m12s	4m10s	6m59s	3m58s	3m16s
Frames	**45m36s**	**40m38s**	**39m43s**	**20m58s**	**11m47s**
Mosaic	11m30s	16m00s	23m10s	20m10s	22m22s
Coadd	71m04s	64m24s	65m34s	47m37s	48m06s

表 5　元のパイプライン実行時のファイル入出力

		ファイルサイズ (GB)	ファイル数	経過時間（秒）/コア数	処理時間に対する割合
Bias	read	432	5500	9	11%
	write	51	672	5	6%
Dark	read	451	6058	10	12%
	write	51	672	5	6%
Flat	read	951	13344	31	12%
	write	105	1232	12	5%
Frames	read	231	5392	40	2%
	write	128	11419	39	2%
Mosaic	read	12	3747	439	46%
	write	0.14	2496	22	2%
Coadd	read	5369	36558	341	9%
	write	194	1469	38	1%

表 6　Pwrake による実行時のファイル入出力

		ファイルサイズ (GB)	ファイル数	経過時間（秒）/コア数	処理時間に対する割合
Bias	read	432	5724	1	1%
	write	51	673	3	4%
Dark	read	452	6284	6	8%
	write	51	673	3	4%
Flat	read	952	13576	36	15%
	write	105	1233	7	3%
Frames	read	215	9984	38	3%
	write	239	17472	29	2%
Mosaic	read	12	3747	466	39%
	write	0.14	2497	22	2%
Coadd	read	2097	18549	68	2%
	write	176	1197	16	1%

表 5 と表 6 から，ファイル入出力の経過時間の処理時間に対する割合は，元のパイプライン，Pwrake どちらも Mosaic 以外は数%から多くて 15%であり，処理時間に対して支配的にはなっていないことがわかる．これは Gfarm によるストレージアクセス分散の効果である．Mosaic タスクのみ，ファイルアクセスが逐次であるため，処理時間に対するファイル読み込みの割合が大きい．

また，表 5 と表 6 で，ファイル入出力の数とサイズがかなり異なる結果となった．この違いの原因として，異なるコマンドラインインタフェースを用いたこと考えられる．特に，ログを見ると，同じファイルを異なる回数読む現象がみられる一方，作成されたファイルの数とサイズについては，これらの表より差が小さいことから，必要な処理結果は得られていると思われる．今回の調査では同一の条件を見つけられなかったため，この条件での比較を行った．

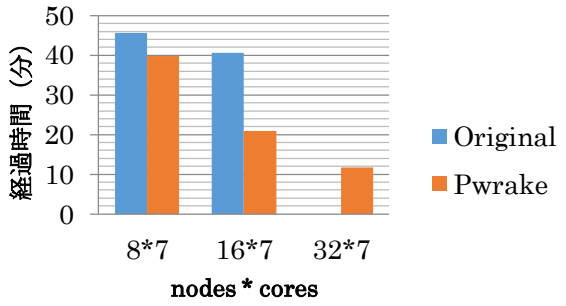

図 4　Frames タスクの経過時間

一方，Frames タスクに着目すると，Pwrake で実行したパイプラインの経過時間は，ノード数を 8, 16, 32 と倍にすることによって，経過時間がそれぞれ約 40 分，21 分，12 分と約半分になるという結果が得られた．Frames タスクの実行時間のグラフを図 4 に示す．Frames タスクについては，Rakefile で Frame ごとにタスクを定義したため，frame 数である 1344 まで並列化が可能である．そのため，32×7=224 コアの場合でもスケールする結果となった．他方，Bias, Dark, Flat, Coadd タスクについては，CCD 数または patch 数ごとにタスクを定義しているため，8 ノード（56 コア）から 16 ノード（112 コア）にすると実行時間が減少しているが，16 ノード（112 コア）から 32 ノード（224 コア）にしても経過時間に変化がない．

次に，112 コアのケースについて，元の HSC パイプラインの並列処理システムで実行した場合と，Pwrake で実行した場合について比較する．Bias, Dark, Flat については経過時間に大きな差がないものの，Frames についてはオリジナルでの経過時間が約 41 分であったものが，Pwrake での経過時間は約 21 分と半分近くになっている．この原因は HSC パイプラインの並列処理システムの動作にある．112 コアを使用する場合，パイプラインシステムは，112 個のタスクをコアに振り分けて実行し，そのすべてのタスクが終了するまで待ち合わせてから，次の 112 個のタスクを実行する，というように実装されている．そのため，時間が長いタスクを待ち合わせる際に使用しないコアが発生し，全体の経過時間が長くなっていると考えられる．一方，Pwrake はタスクの実行が終了すると，すぐに次に実行可能なタスクを割り当てるため，コアを有効活用できる．今回の調査によって，Frames タスクに関しては，並列処理システムとして Pwrake を使うことで高速化が可能であることがわかった．

Mosaic タスクについては，オリジナルの経過時間 16 分に対し，Pwrake での経過時間が 22 分に増加している．この違いは，解く行列の次元が，オリジナルのケースでは 3 万以下であるのに対し，Pwrake のケースでは 4 万以上と異なるためである．その原因は調査中である．

4.4 Gfarm メタデータサーバ性能

以上の性能調査では，Gfarm のメタデータサーバのバックエンド DB である PostgreSQL のデータ保存先を RAM ディスクとしている．保存先をハードディスクにした場合のパイプライン処理時間について，Pwrake で実行した場合の Frames タスクのみの結果を表 7 に示す．ノード数を 8 ノードから 32 ノードに増加しても経過時間は横ばいとなり，その原因が個々の hscProcessCcd.py タスクの経過時間の増加にあることがわかる．RAM ディスクの使用によって処理時間が 224 コアで 12 分に短縮されていることから，この場合は Gfarm メタデータの処理がボトルネックであると考えられる．ただし RAM ディスクを使用する場合，マシンの再起動によってデータが失われるため，SSD などの高速なストレージの導入が望ましい．HSC パイプラインの処理においてメタデータアクセスの頻度について，今後調査する予定である．

表 7　Frames タスク(hscProcessCcd.py)の実行時間

nodes*cores	8*8	16*8	32*8
経過時間(分)	44.5	44.2	44.6
タスクの平均経過時間（秒）	133	254	472

5. 関連研究

天文データ処理の並列化の例として頻繁に取り上げられるアプリケーションに Montage[7]がある．Montage は天文画像の加工処理を行う汎用的なソフトウエアパッケージであるが，一次処理済みの画像が対象である．一方，HSC パイプラインは CCD が出力した画像の一次処理が目的である．Montage は Pegasus[8], Swift[9], Pwrake[10]など多くのワークフローシステムの実証に用いられている．

Yamamoto ら[11]は，すばる Suprime-Cam のデータに対する Gfarm による並列処理を提案している．このときの Gfarm は現在と異なるプロトタイプ実装（Gfarm version 1）であったため，システムコールの hook や gfrun の使用といった工夫が必要であった．今回の Gfarm version 2 では，gfarm2fs によって Gfarm ファイルシステムをマウントすることにより，通常のファイルシステムと同じようにファイルにアクセスしたり処理プログラムを実行したりすることができる．

6. まとめと今後の課題

すばる望遠鏡に搭載された焦点面カメラ Hyper Suprime-Cam (HSC) による超新星爆発の早期発見に挑戦すべく，本研究では HSC が出力するデータの一次処理を行うパイプラインシステムの高速化の面から取り組みを開始した．現状のパイプラインシステムを高並列で動作させる場合の問題点を明らかにするため，クラスタに構築した Gfarm ファイルシステム上で実行し，性能について調査した．HSC パイプライン独自の並列処理システムとの比較のため，パイプライン処理を Rakefile で記述し，ワークフローシステム Pwrake で実行して性能を比較した．その結果，Single frames 処理については，並列処理システムにおける 2 つの問題点（並列処理数が CCD 数までであることと，不必要なタスク開始同期を行っていること）が存在しない Pwrake システムによってスケールする性能が得られ，高性能化が可能であることがわかった．また，Gfarm メタデータサーバのバックエンド DB の IO 性能も重要になることもわかった．今後の課題は，メタデータアクセスの詳細な調査， Mosaic 処理と Coadd 処理の高速化，今回より多くのコア数を使用した高速化である．

謝辞 本研究は，JST CREST「広域撮像探査観測のビッグデータ分析による統計計算宇宙物理学」，「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」および「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」の支援により行った．HSC パイプラインソフトウェアおよびデータの提供，クラスタの設置などを行っていただいた Kavli-IPMU の安田直樹氏に感謝します．

参考文献

[1] S. Miyazaki, Y. Komiyama, H. Nakaya, Y. Kamata, Y. Doi, T. Hamana, H. Karoji, H. Furusawa, S. Kawanomoto, T. Morokuma, Y. Ishizuka, K. Nariai, Y. Tanaka, F. Uraguchi, Y. Utsumi, Y. Obuchi, Y. Okura, M. Oguri, T. Takata, D. Tomono, T. Kurakami, K. Namikawa, T. Usuda, H. Yamanoi, T. Terai, H. Uekiyo, Y. Yamada, M. Koike, H. Aihara, Y. Fujimori, S. Mineo, H. Miyatake, N. Yasuda, J. Nishizawa, T. Saito, M. Tanaka, T. Uchida, N. Katayama, S.-Y. Wang, H.-Y. Chen, R. Lupton, C. Loomis, S. Bickerton, P. Price, J. Gunn, H. Suzuki, Y. Miyazaki, M. Muramatsu, K. Yamamoto, M. Endo, Y. Ezaki, N. Itoh, Y. Miwa, H. Yokota, T. Matsuda, R. Ebinuma, and K. Takeshi, “Hyper Suprime-Cam,” in *SPIE Astronomical Telescopes + Instrumentation*, 2012, p. 84460Z.

[2] M. Tanaka, T. J. Moriya, N. Yoshida, and K. Nomoto, “Detectability of high-redshift superluminous supernovae with upcoming optical and near-infrared surveys,” *Mon. Not. R. Astron. Soc.*, vol. 422, no. 3, pp. 2675–2684, May 2012.

[3] N. Tominaga, T. Morokuma, S. I. Blinnikov, P. Baklanov, E. I. Sorokina, and K. Nomoto, “SHOCK BREAKOUT IN TYPE II PLATEAU SUPERNOVAE: PROSPECTS FOR HIGH-REDSHIFT SUPERNOVA SURVEYS,” *Astrophys. J. Suppl. Ser.*, vol. 193, no. 1, p. 20, Mar. 2011.

[4] H. Furusawa, M. Tanaka, Y. Yasu, S. Y. Suzuki, R. Itoh, N. Katayama, Y. Komiyama, S. Miyazaki, Y. Utsumi, T. Uchida, H. Aihara, and N. Yasuda, “Hyper Suprime-Cam: data analysis and management system,” in *SPIE Astronomical Telescopes + Instrumentation*, 2008, p. 70161F–70161F–11.

[5] O. Tatebe, K. Hiraga, and N. Soda, “Gfarm Grid File System,” *New Gener. Comput.*, vol. 28, no. 3, pp. 257–275, Aug. 2010.

[6] M. Tanaka and O. Tatebe, “Pwrake: A parallel and distributed flexible workflow management tool for wide-area data intensive computing,” in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC ’10)*, 2010, pp. 356–359.

[7] J. C. Jacob, D. S. Katz, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, T. A. Prince, and R. Williams, “Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking,” *Int. J. Comput. Sci. Eng.*, vol. 4, no. 2, pp. 73–87, Jul. 2009.

[8] E. Deelman, G. Singh, M.-H. Su, J. Blythe, others, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz., “Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems,” *Sci. Program. J.*, vol. 13, no. 3, pp. 219–237, 2005.

[9] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, “Swift: Fast, Reliable, Loosely Coupled Parallel Computation,” in *2007 IEEE Congress on Services (Services 2007)*, 2007, vol. 0, pp. 199–206.

[10] M. Tanaka and O. Tatebe, “Workflow Scheduling to Minimize Data Movement Using Multi-constraint Graph Partitioning,” in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012, pp. 65–72.

[11] N. Yamamoto, O. Tatebe, and S. Sekiguchi, “Parallel and Distributed Astronomical Data Analysis on Grid Datafarm,” in *Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 461–466.