富岳の使い方

～富岳で機械学習～

2021年10月20日　新規作成
2021年10月28日 第4版
株式会社エクサ
テクノロジー・イノベーション部
堀　扶
tasuku-hori@exa-corp.co.jp

目次

《サンプルコード》
https://github.com/coolerking/fugaku_sample

《注意》

- 2021年10月28日時点の環境の利用所感をもとに記述しています
- 登場する会社名、製品名およびサービスは、各社の商標または登録商標です
- 本資料の評価は発表者の主観で記述しています

スーパーコンピュータの定義

（実は明確な定義はない）

「科学技術計算において
同世代で抜きん出て高速な
計算機」

東京大学出版会　「スパコンを知る」より

スーパーコンピュータ富岳は「京」の後継機

京といえば..

（2009年事業仕分けにおいて、スーパーコンピュータ「京」の予算に対し）

２位じゃダメなんですか

研究・開発者側の要件だけでなく
利用者側の要件を盛り込む傾向が加速

AIブームの影響を受けた「富岳」

理研松岡センター長

（富岳は）

「世界最大のAI基盤と位置づけ

GAFAに対抗し、

追い越すことも可能」

※中公新書ラクレ「スパコン富岳後の日本 科学技術立国は復活できるか」より引用

《仮説》

①ビジネスユース向け機械学習開発は技術的に可能か？

②顧客むけSI案件に適用可能か？

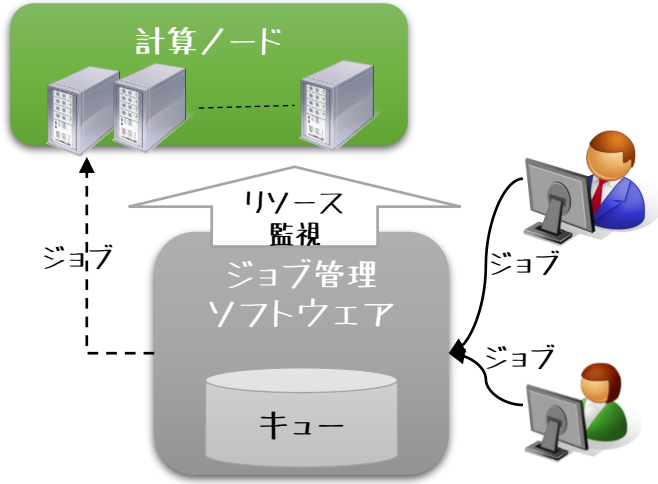
科学技術計算 vs ビジネスシステム

科学技術計算	ビジネスシステム
浮動小数点演算中心	整数演算中心
バッチ	OLTP
速度・精度重視	フェールセーフ
正常系テスト	例外系・異常系もテスト
短いライフサイクル	長いライフサイクル
1枚板のようなコード	構造化されたコード

富岳では科学技術計算の作法で設計・実装がベストプラクティス

《富岳ユーザのバッチ投入を管理》

ジョブ管理ソフトウェア

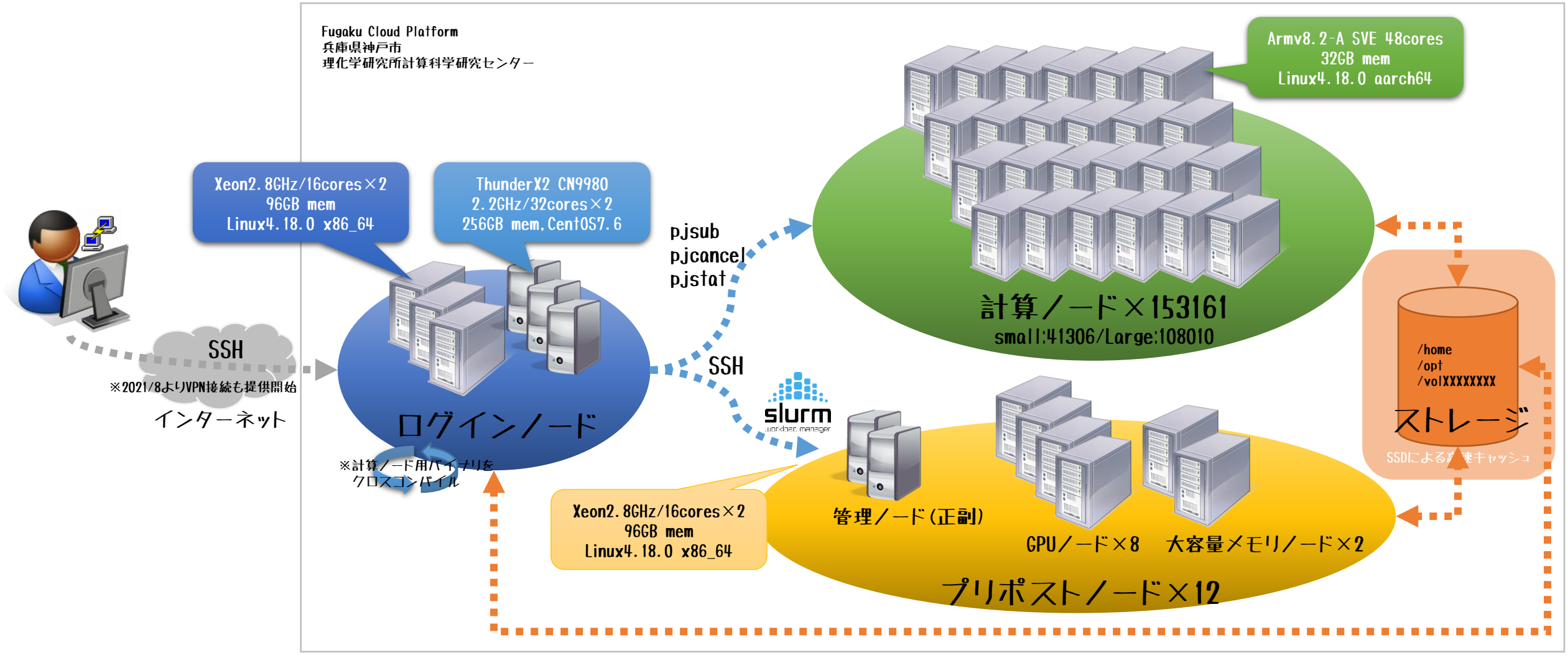


- 多様なニーズを持つユーザからのジョブを統合管理
- HPC環境リソースを最大限に活用
- スパコン全体のリソースから優先順位に従って順次実行
- 一般的な方法としてシェルスクリプトを作成しバッチジョブを投入
- 会話型ジョブ(SSH接続)も可能だがデバッグ・テスト用としての活用
- 富岳計算ノード管理
  - Fujitsu Software Technical Computing Suite
    - 計算ノードのコンピュータリソースを管理
    - ユーザはpjsub/pjdel/pjstatコマンドでジョブ操作
- プリポストノード管理
  - Slurm Workload Manager
    - プリポストノードのコンピュータリソースを管理
    - ユーザはsrun/sexec/scancel/sinfoコマンドでジョブ操作

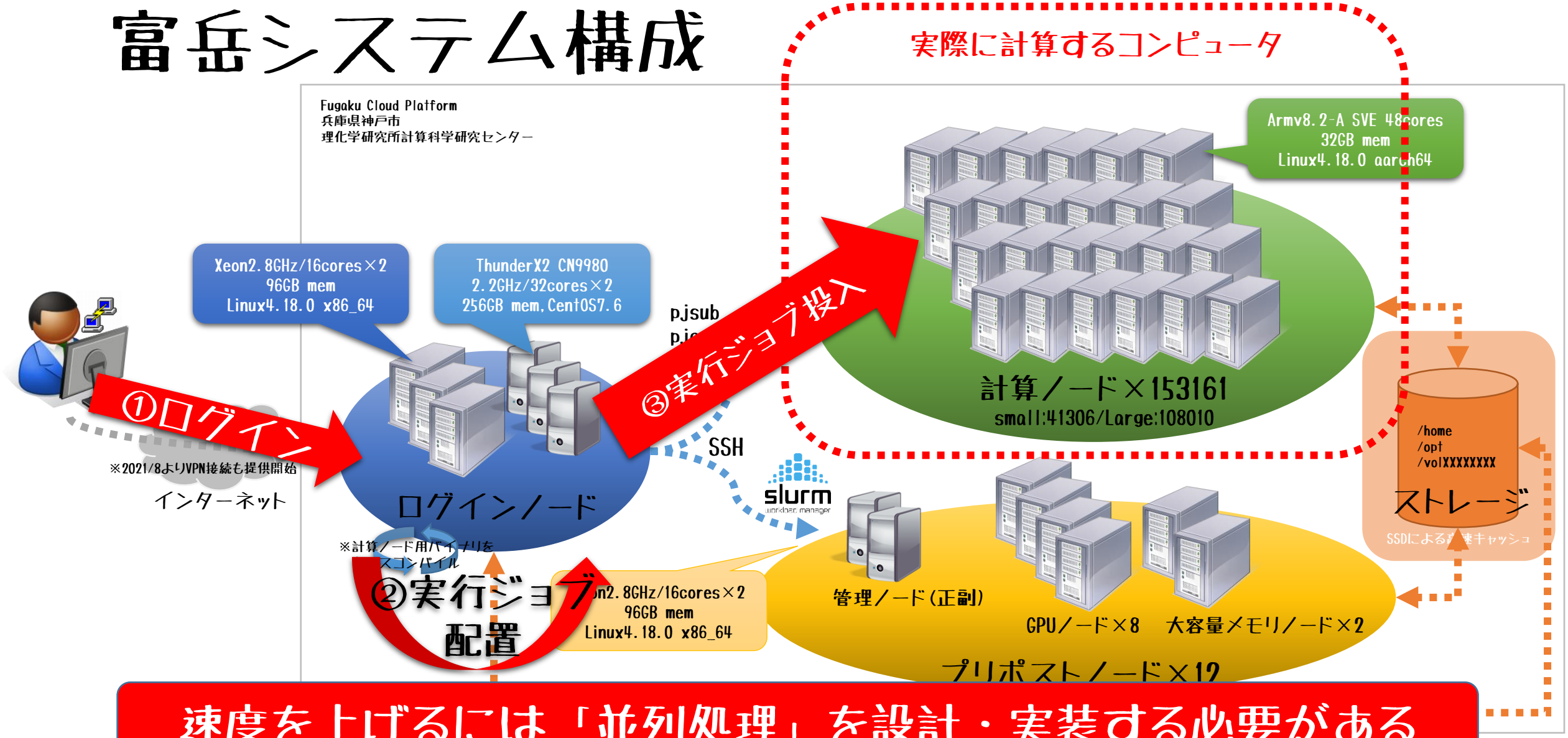
```
 1 #!/bin/bash↓
 2 #PJM --rsc-list "node=4"↓
 3 #PJM --rsc-list "rscunit=rscunit_ft01"↓
 4 #PJM --rsc-list "rscgrp=small"↓
 5 #PJM --rsc-list "elapse=72:00:00"↓
 6 #PJM --mpi "max-proc-per-node=4"↓
 7 #PJM -m b,e,s↓
 8 #PJM -S↓
 9 export OMP_NUM_THREADS=12↓
10 module purge↓
11 module load lang/tcsds-1.2.33↓
12 ↓
13 llio_transfer ./a.out↓
14 mpiexec -n 16 ./a.out↓
```

並列処理ジョブスクリプトサンプル

富岳システム構成

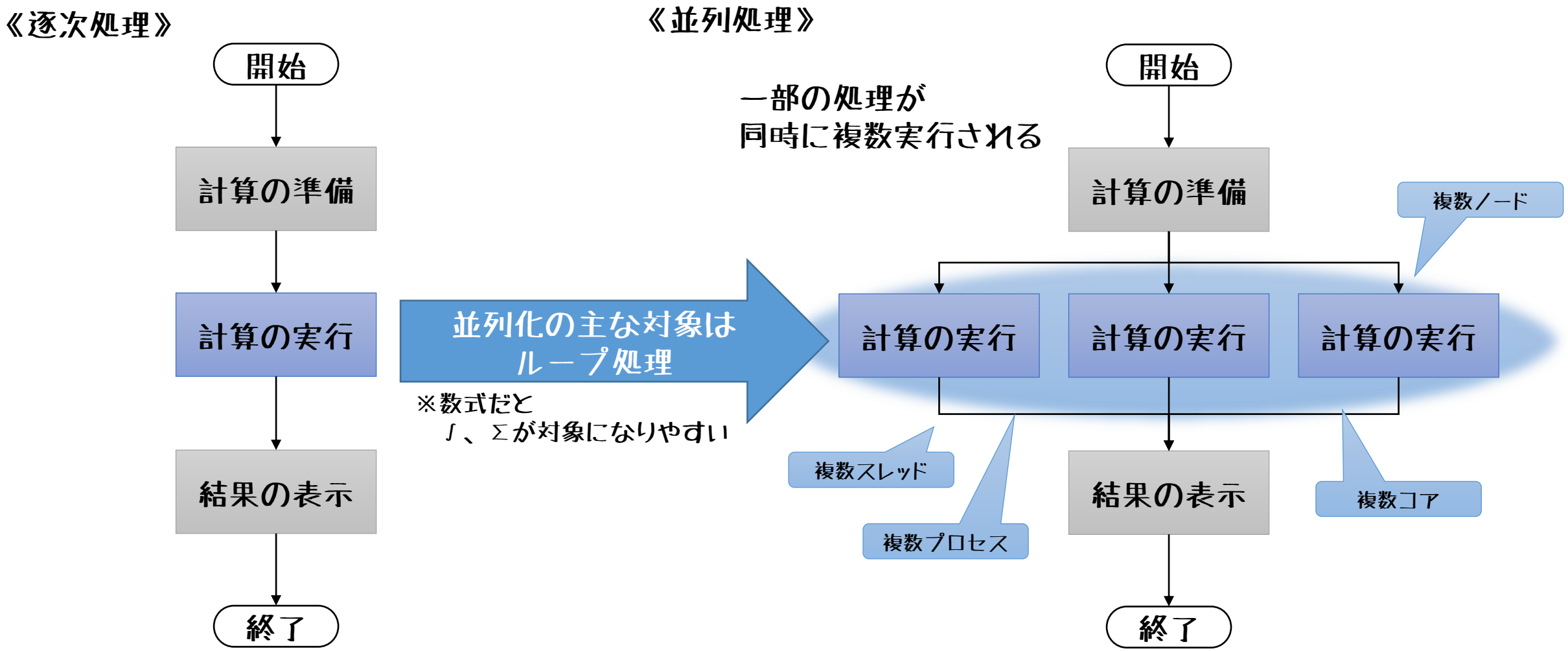


富岳システム構成



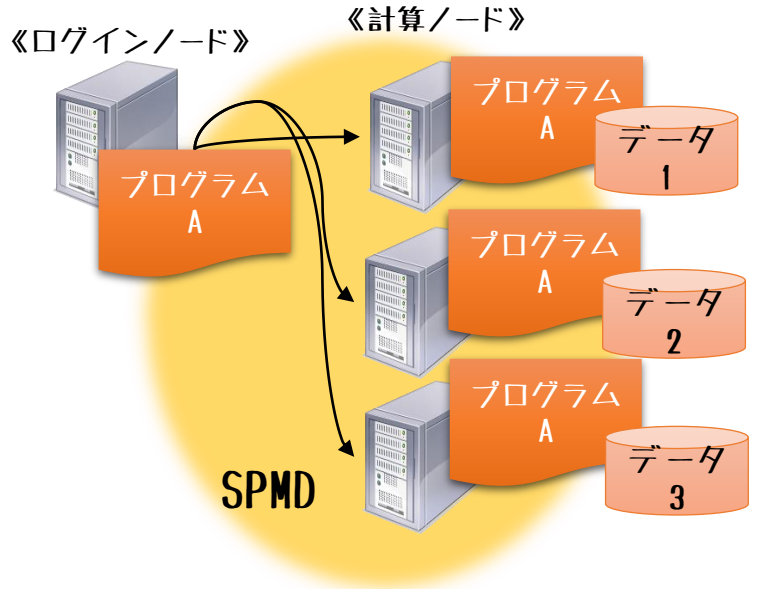
速度を上げるには「並列処理」を設計・実装する必要がある

バッチ：逐次処理から並列処理へ

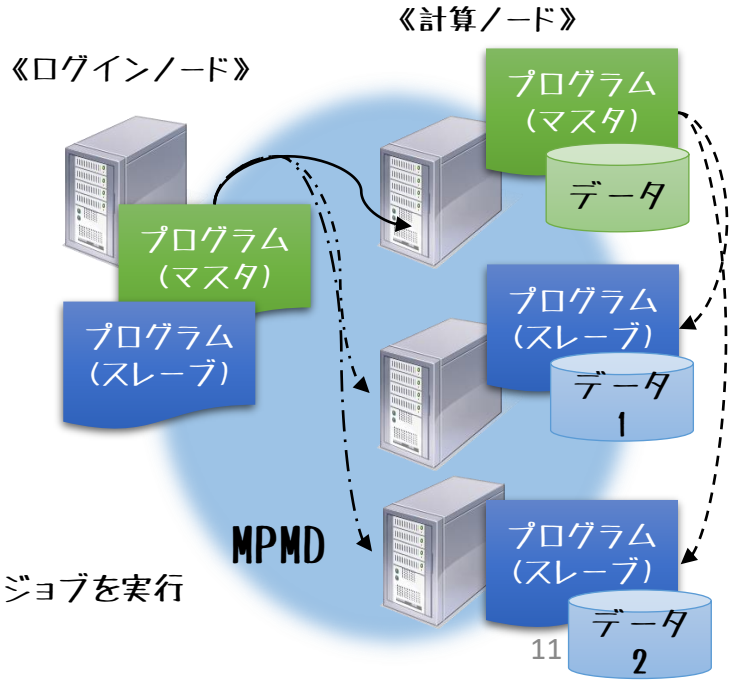


SPMDとMPMD

- SPMD: Single Program Multiple Data
  - 同一のプログラムが、各ノードにロードされ実行される
  - 全プロセスが主従関係を持たない、並列化が簡単



- MPMD: Multiple Program Multiple Data
  - 異なるプログラムが各ノードにロードされ実行される
  - マスタースレーブ型
  - １つのノードでは親プロセスが実行される
  - 他のノードでは同一の子プロセスが実行される



※スレーブプログラムをバックグラウンド実行してから、マスタジョブを実行

補足

《並列処理の設計》

例：Σの並列化（SPMD並列処理/Fortran）

$$\sum_{I=1}^{9} I$$

1から9までの整数の総和を求めるサンプル

《逐次処理（1ノードで実行）》

```
            PROGRAM MAIN
C 1から9までの総和
            ISAM = 0
            DO I = 1, 9
              ISUM = ISUM + I
            ENDDO
C 出力
            PRINT *,'ISUM=',ISUM
C
            END
```

MPIを使った並列化

《SPMDによる並列処理（3ノードで実行）》

```
            PROGRAM MAIN
            INTEGER N(9)
            INTEGER NBUF(4)
            EXTERNAL I_VADD
C NTASK:全タスク数(ここでは3)
C MYID:自分のタスクNo(0,1,2)
            CALL MP_ENVIRON(NTASK,MYID)
C 全タスクのグループIDを取得
            CALL MP_TASK_QUERRY(NBUF,4,3)
            IALLGRP = NBUF(4)
C 自分のタスクNoから分担するグループ処理
C の範囲を計算
            ISTA = MYID*3 + 1
            IEND = ISTA + 2
C 計算対象の作成
            DO I = ISTA, IEND
              N(I) = I
            ENDDO
            ISUM = 0
C 並列化されたループ
            DO I = ISTA, IEND
              ISUM = ISUM + N(I)
            ENDDO
C 他のCPU上で実行されているプロセスとの通信
C しながら部分和の総和計算
            CALL MP_REDUCE(ISUM, ITMP,4,0,I_VADD,IALLGRP)
C 出力
            PRINT *,'ISUM=', ISUM
C
            END
```

並列処理

逐次処理から並列処理へ

例：πを計算する

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

グラフの［0，1］範囲の面積を求めるとπになる
積分計算部分を四則演算で記述

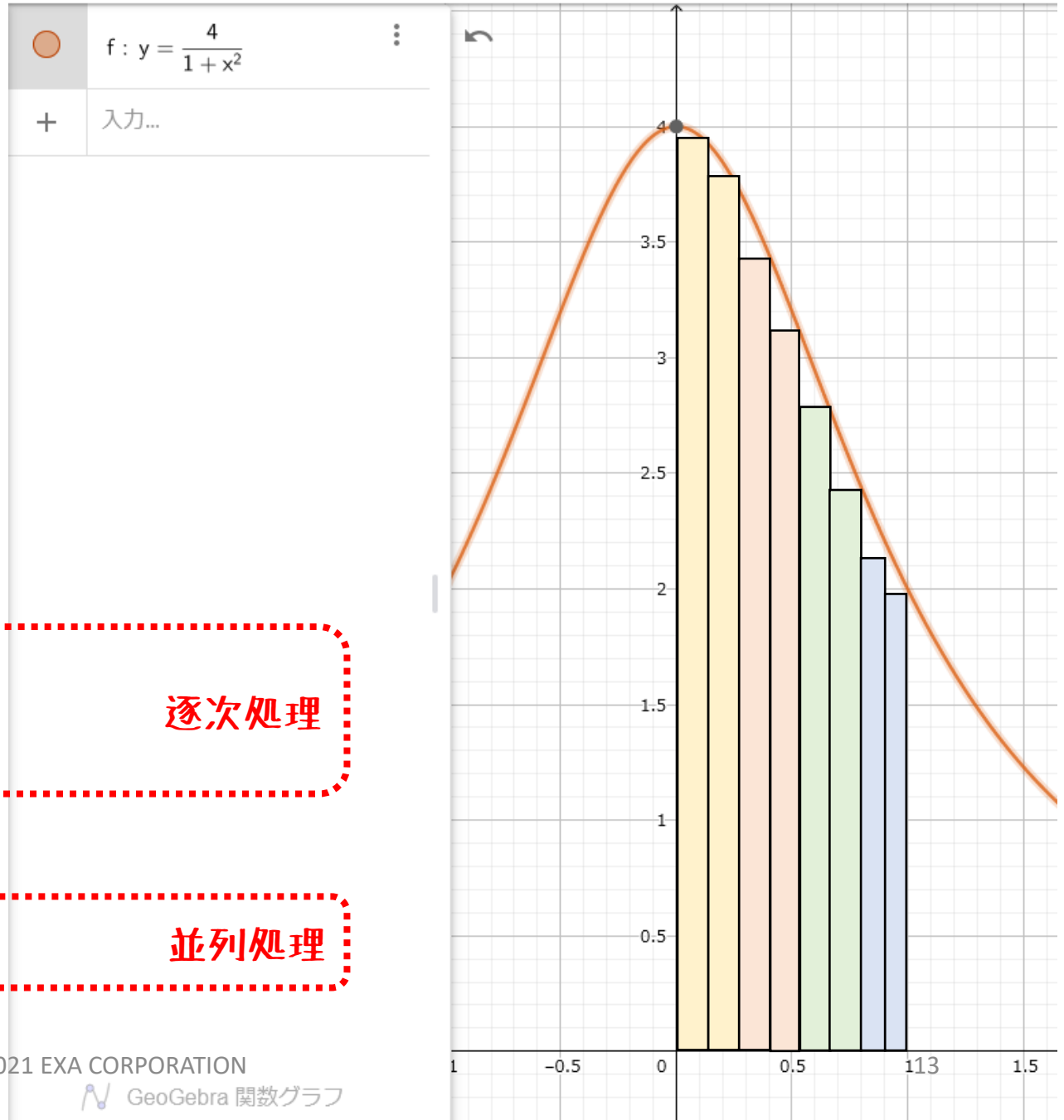
1. X座標0から1の範囲を等間隔で短冊にする
2. すべての短冊の高さ（Y座標値）を集計
3. 高さ合計に短冊の幅（X座標）を掛ける

逐次処理

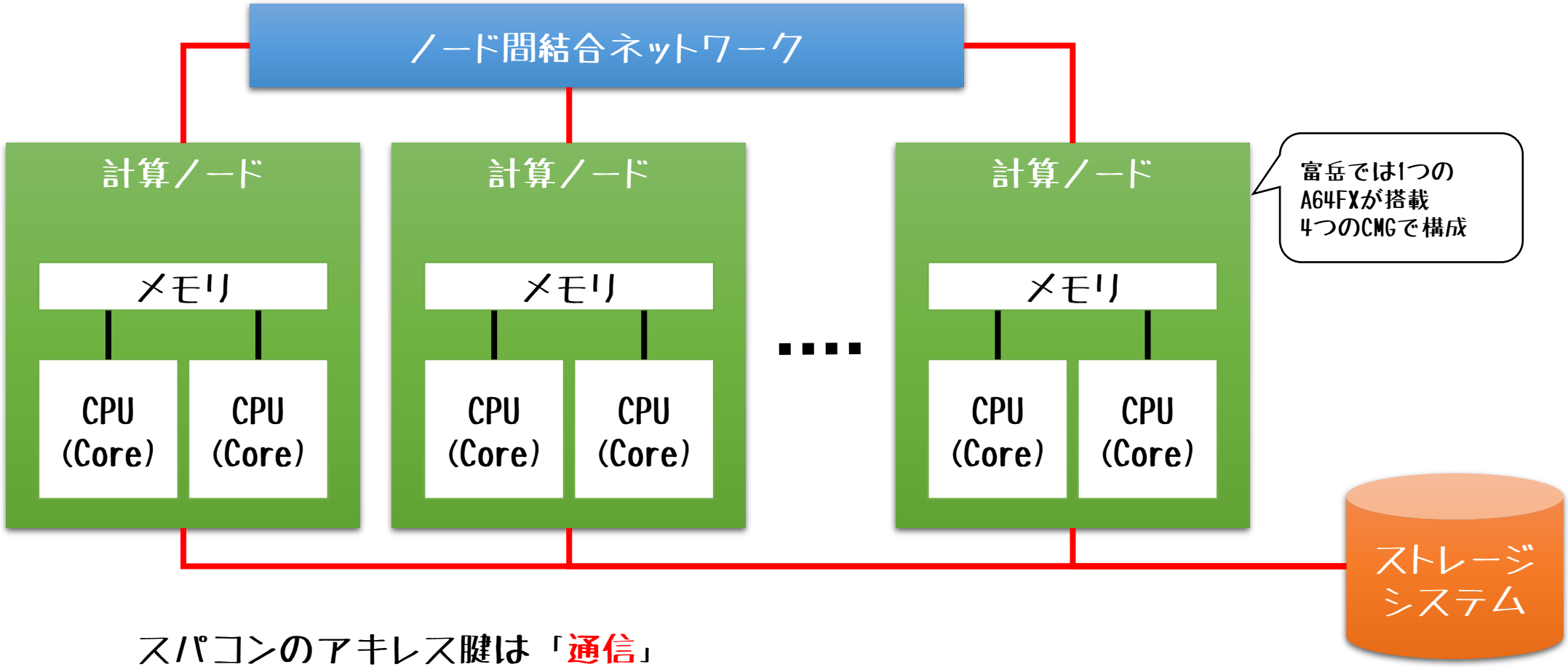
短冊の幅をできるだけ小さくすれば精度があがる

処理2を計算ノードごとに振り分ける

並列処理



スパコンの基本アーキテクチャ



スパコンのアキレス腱は「**通信**」

HPC分野における”通信”仕様

MPI: Message Passing Interface

- 標準化されたメッセージ伝達インターフェイス仕様
- 分散メモリ型並列計算機向き
- プロセッサ数が多い並列システム向き
- スケーラビリティ、性能が高い
- 言語とは独立した通信プロトコル仕様を持つ
  - Fortran、C、C++
  - 実装ライブラリ経由でC#、Python、Javaなど
- 主な実装
  - MPICH、MVAPICH、OpenMPI、ベンダ提供のMPIライブラリ
  - 富岳にはOpenMPIベースの富士通製、MPICHベースの理研製が導入済み
    - コンパイラオプションで指定
- mpiexec/mpirunコマンドをつかって実行

例：πを計算する(SPMD:並列処理/C/MPI)

```c
  MPI_Init(&argc, &argv);                    // MPI初期化
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);  // プロセス数の取得
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);      // 現在のプロセスIDの取得
  MPI_Get_processor_name(hostname, &length); // ホスト名の取得

// セグメント数(n)をすべてのプロセスにブロードキャスト
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

// このループは最大反復回数をインクリメントするため、
// プロセッサの計算速度をテストするための追加の作業が提供される
  for(total_iter = 1; total_iter < n; total_iter++)
  {
    sum=0.0;
    width = 1.0 / (double)total_iter; // セグメント幅(dx)
//    width = 1.0 / (double)n; // width of a segment

    for(i = rank + 1; i <= total_iter; i += numprocs)
//    for(i = rank + 1; i <= n; i += numprocs)
    {
      x = width * ((double)i - 0.5); // i番目セグメントのx座標中央値
      sum += 4.0/(1.0 + x*x); // 与えられたランクにおける個々のセグメントの高さ(y)合計
    }

// 与えられたランクにおけるセグメントのおおよその面積(π)
    rank_integral = width * sum;

// すべてのプロセスから部分面積（π）値を収集して追加
    MPI_Reduce(&rank_integral, &pi, 1, MPI_DOUBLE,MPI_SUM, 0, MPI_COMM_WORLD);

  } // for(total_iter = 1; total_iter < n; total_iter++) の終わり


  // MPIのクリンナップと終了処理
  MPI_Finalize();
```

①《MPI》
初期化処理
メタ情報取得

②《MPI》
全ノードへデータ送信
（ここでは全ノード数n）

《Mapステップ》
マスタから
ワーカへ

③各ランクの担当する処理
X座標範囲を算出
範囲内全セグメントの
4/(1+x^2)値合計を計算
担当範囲の面積算出

MapReduce
モデル

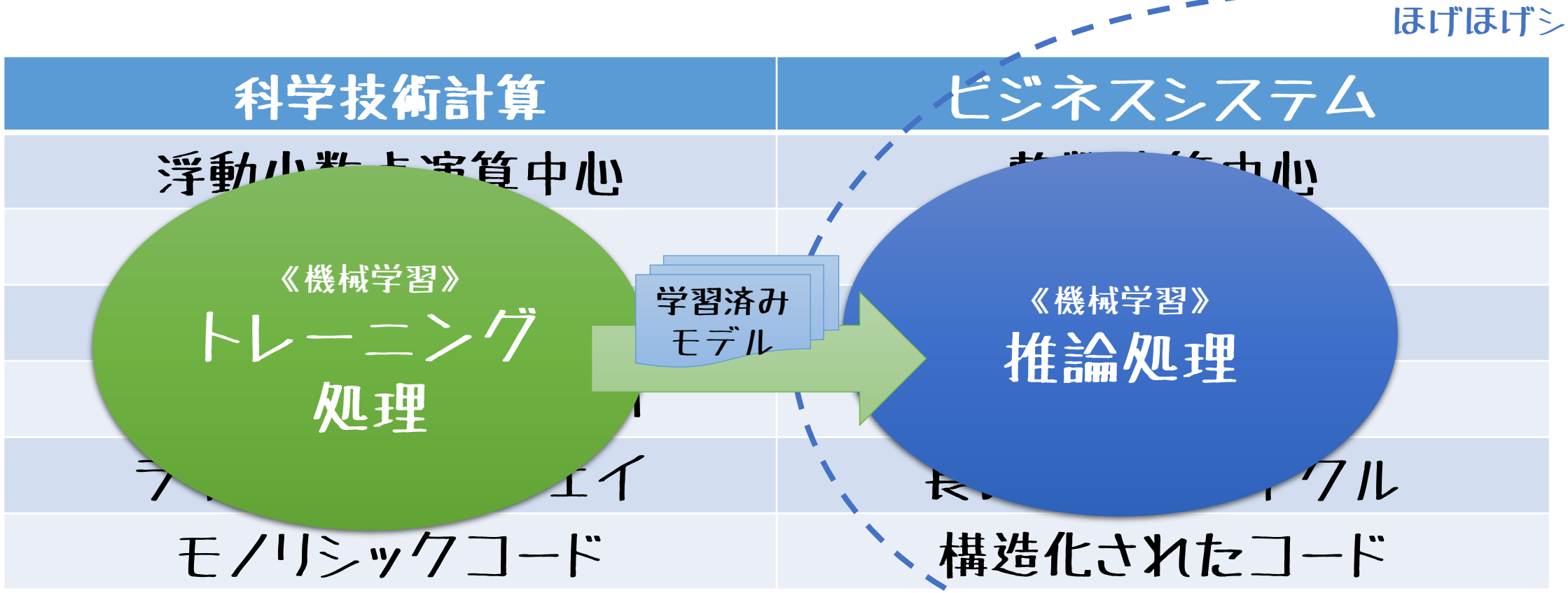
MPIメタ情報を元に
並列化処理をロジック化

④《MPI》
全計算ノードから集計（面積合計）

《Reduceステップ》
マスタへ集約

⑤《MPI》
終了処理

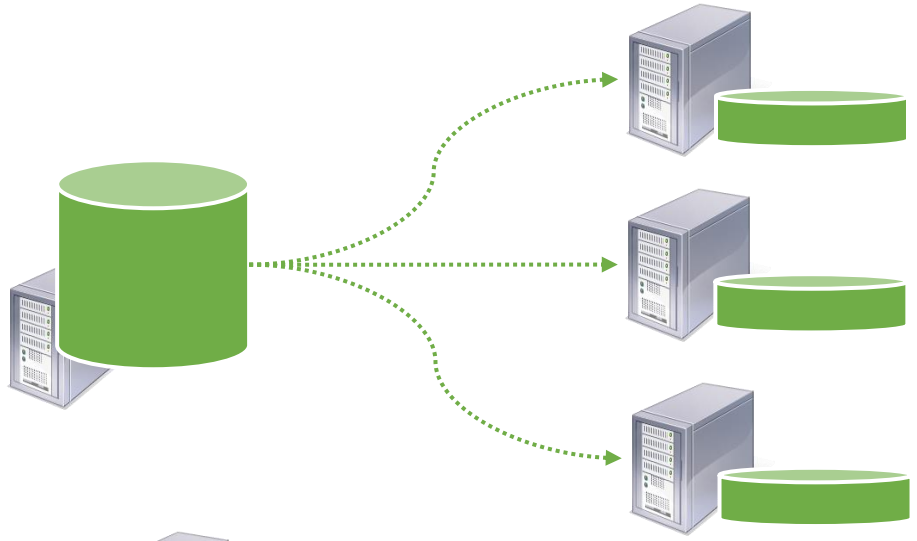
機械学習のトレーニング処理



トレーニング処理における並列処理化

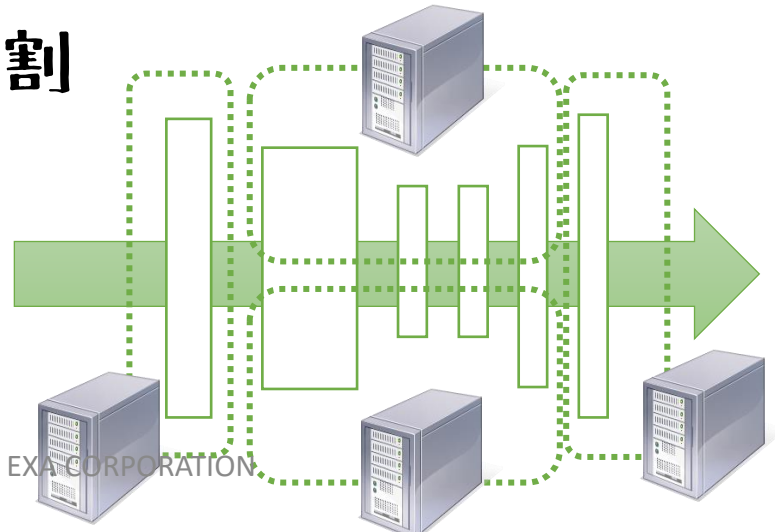
①データ並列化

- 学習データを計算ノード別に分割



②モデル並列化

- 学習モデルを計算ノード別に分割



機械学習トレーニング処理

- Pre-Train
  - 学習データの編集
    - データを増やす、入力・出力(正解)データの整形、学習・テストデータに分ける
    - 機械学習フレームワークにあったデータセット/バッチサイズに加工
  - 学習対象となるモデルの初期化
- トレーニングループ(エポック単位)
  - Forward
    - 入力データをモデルにかけ出力(推論)データを取得
    - 出力(推論)データ、出力(正解)データを元に誤差関数を求める
  - Backward
    - テストデータを使って評価、基準値を超えていれば週トレーニング終了
    - トレーニング継続の場合、誤差伝播法などで勾配を算出
  - Update
    - モデルパラメータを更新する
- Post-Train
  - 学習済みモデルを保存する

②モデル並列

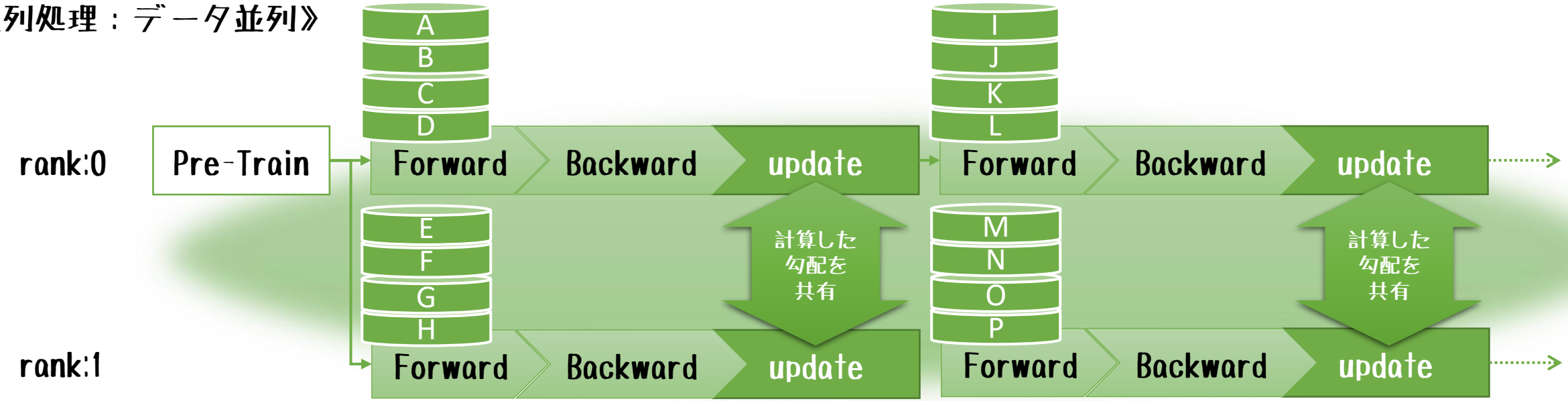
①データ並列

①データ並列化

異なるデータで同じモデルを並行処理→実行効率向上

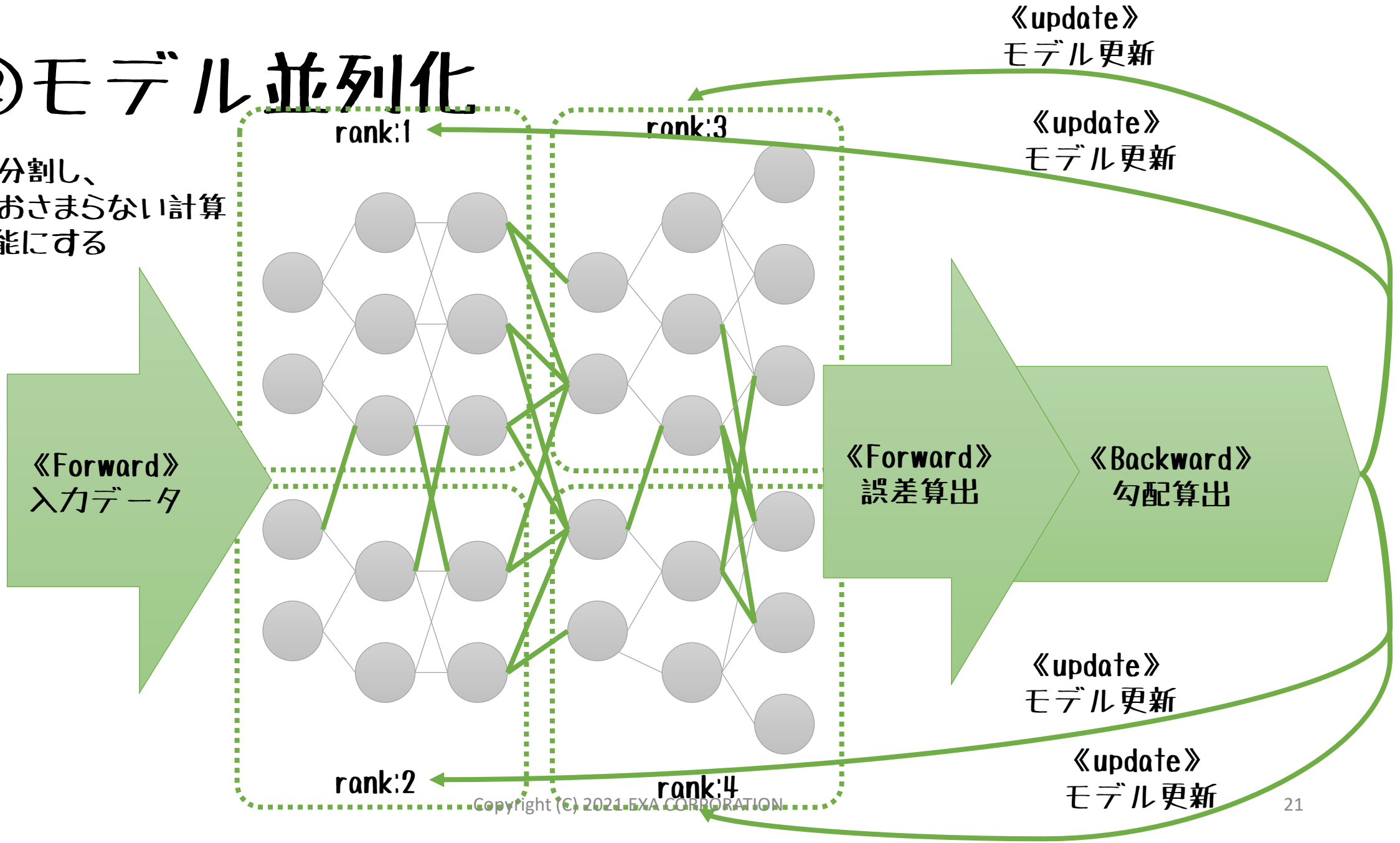
《逐次処理》

《並列処理：データ並列》



②モデル並列化

モデルを分割し、
１コアにおさまらない計算
を実行可能にする



①データ並列 vs ②モデル並列

①データ並列	②モデル並列
トレーニング効率向上	大規模モデルが扱える
実装が楽	実装は困難
モデルに並列化コード混在	トレーニング処理が複雑
分散度が上がると精度低下	分散度が上がると通信頻度増加
非同期更新時の精度低下	分割方法はトライ＆エラー

大規模データセット向け
データ大規模化→精度向上

大規模モデル向け
モデル大規模化→精度向上

富岳が提供する機械学習フレームワーク

- TensorFlow 2.2.0
  - oneDNN対応、NCCL/cuda非対応
  - 以下の環境変数を変更する

```
export PATH=/home/apps/oss/TensorFlow-2.2.0/bin:$PATH
export LD_LIBRARY_PATH=/home/apps/oss/TensorFlow-2.2.0/lib:$LD_LIBRARY_PATH
```

  - 2.1.0も存在するが一部機能動作せず
- PyTorch 1.7.0
  - 以下の環境変数を変更する

```
export PATH=/home/apps/oss/TensorFlow-2.2.0/bin:$PATH
export LD_LIBRARY_PATH=/home/apps/oss/TensorFlow-2.2.0/lib:$LD_LIBRARY_PATH
```

- PyTorch1.5.0、1.6.0、ChainerK-4.5.0も存在するが動作するかは不明

該当バージョン以外を使用する場合は、
ソースコードからコンパイルする必要がある

2021/10/28確認
「富岳AIフレームワーク利用ガイド1.0」にChainer-4.5.0の記載あり

Distributed TensorFlow

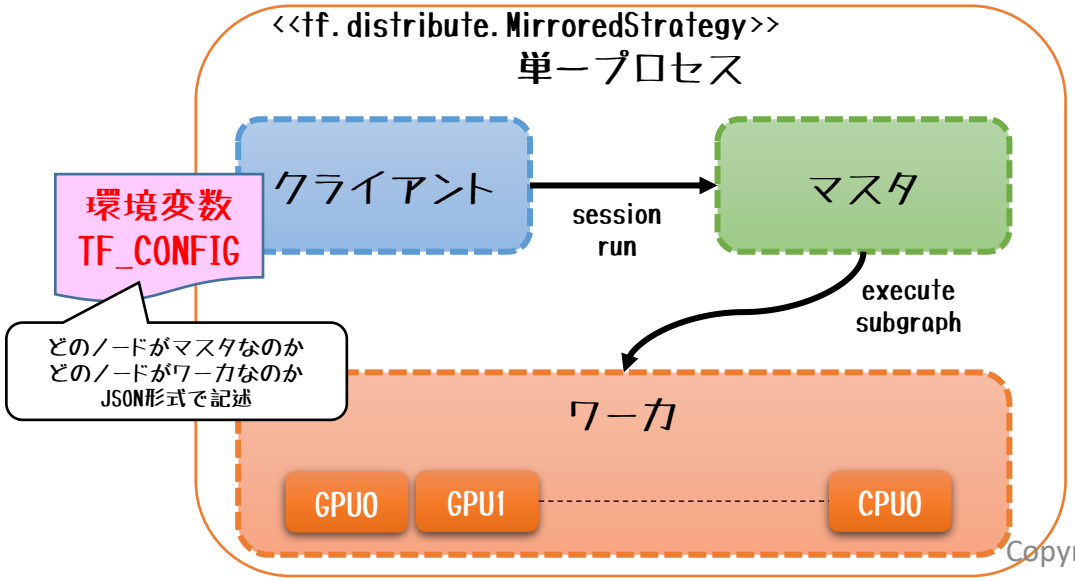
- TensorFlowのサブパッケージ
  - tf.distribute（v2.4以降）
  - 富岳（TensorFlow 2.2.0)の場合は tf.experimental.distribute
- トレーニング処理の並列化を支援する機能を提供
  - データ並列
  - MapReduceモデル
  - GPU、TPU対応
  - 単一ノード、複数ノード対応
    - Strategyクラス
  - クラスタ対応
    - Resolverクラス

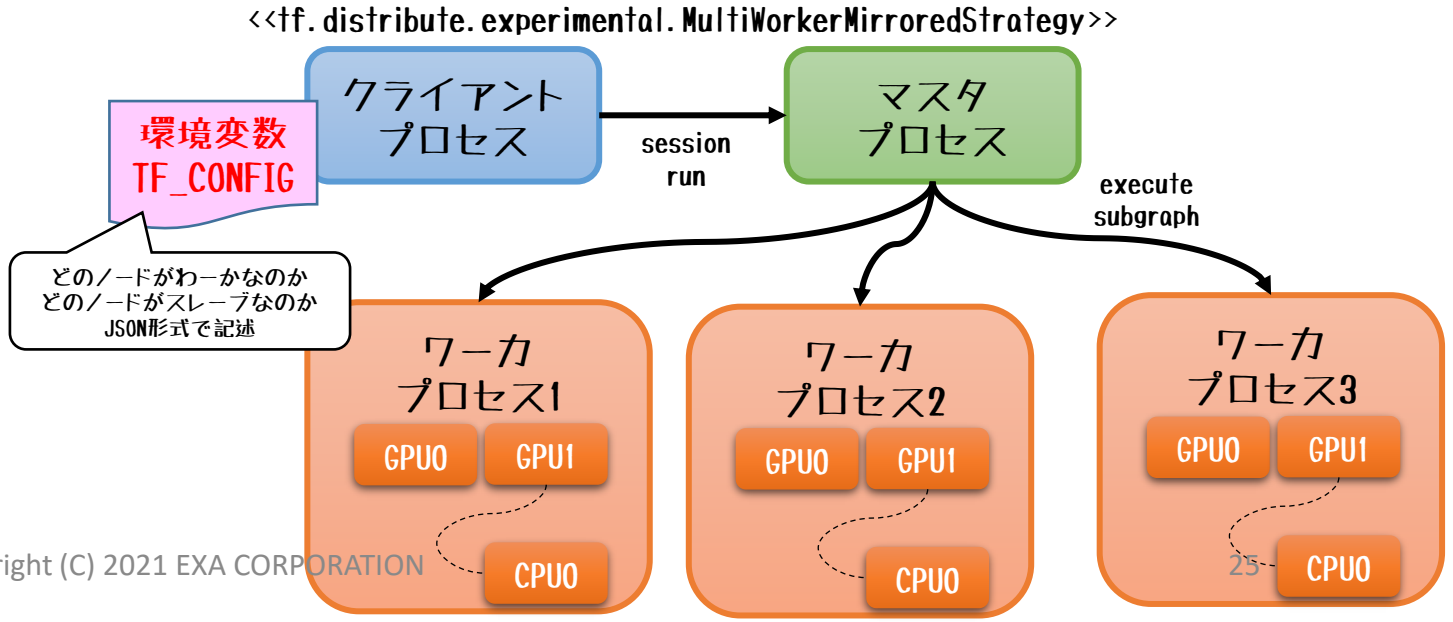
Strategyクラス

- 分散環境にてTensorFlowを使ったトレーニング処理を行う機能
  - 単一ノード上の複数GPU/CPU → MirroredStrategy
  - 複数ノード上の複数GPU/CPU → MultiWorkerMirroredStrategy

TensorFlow2.2.0では使用できない！

- 最初の計算ノード(rank0)をマスタ、のこりをワーカにする定義
  →Strategyクラス初期化時に環境変数TF_CONFIGを読み込む





Horovod

- 並列化を支援する外部のPythonパッケージ
- 分散型深層学習トレーニングフレームワーク
  - TensorFlow、PyTorch、Apache MXNet などに対応
  - 富岳では TensorFlow-2.2.0 環境にて提供
    - ~~PyTorch-1.7.0 環境では各自でコンパイル~~
- MPI風の実装が可能
  - 科学技術計算系技術者には理解しやすい
  - horovodrunラッパで実行
    - 富岳では動作しない→mpiexecコマンドで実行する

2021/10/28 確認
PyTorch-1.7.0でも horovod 使用可能に

例：MNIST（TensorFlow/Horovod）


```python
import tensorflow as tf
import horovod.tensorflow as hvd

# Horovod の初期化
hvd.init()

# ローカルランクの処理に使用されるGPUをピン付け（プロセスごとに1つのGPU）
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# モデルのビルド...
loss = ...
opt = tf.train.AdagradOptimizer(0.01 * hvd.size())

# Horovod 分散オプティマイザを追加
opt = hvd.DistributedOptimizer(opt)

# 初期化中にランク0から他のすべてのプロセスに変数をブロードキャストするためのフックを追加
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# トレーニング操作を構築
train_op = opt.minimize(loss)

# Save checkpoints only on worker 0 to prevent other workers from corrupting them.
# 他のワーカがチェックポイントを破損しないように、ワーカ0のみでチェックポイントを保存
checkpoint_dir = '/tmp/train_logs' if hvd.rank() == 0 else None

# MonitoredTrainingSession は、セッションの初期化、チェックポイントからの復元、
# チェックポイントへの保存、完了またはエラーの発生時に閉じる処理を行う
with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                       config=config,
                                       hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # 同期トレーニングを実行する
    mon_sess.run(train_op)
```

《Horovod》
初期化処理
メタ情報取得
（メタ情報をもとにハイパパラメータ調整）

《Horovod》
全ノードへデータ送信
（フックオブジェクトとしてモデルに登録）

追加するHorovodコードはMPIに近い
※TensorFlowコードの癖に合わせるので挿入箇所は異なる

モデルの実行

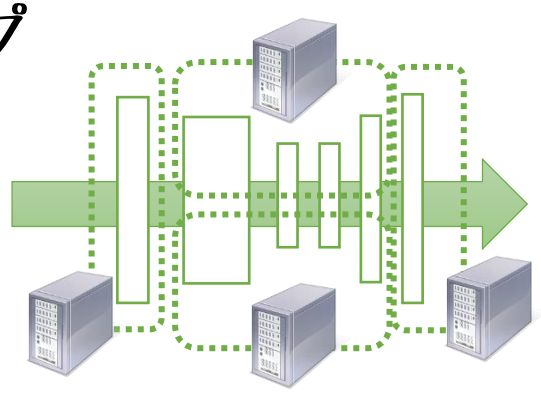
Mesh TensorFlow

- モデル並列に対応した並列トレーニングを行うためのライブラリ
- Google提供なのにTensorFlow外部Pythonパッケージ
- 複数の同一プロセッサで動作
  - TPU-ポッド、マルチGPU、マルチCPU
- 主な用途
  - データ並列（バッチ分割）
  - モデル並列（モデル分割）
  - 大きな入力空間の分割(イメージ/ビデオ)
  - 上記の組み合わせ

Mesh TensofFlowの使用

- TensorFlow 1.13で動作
  - TensorFlow 2.2.0 でも動作するが今のところは1.x互換モードでの実装
- メッシュTensorFlowグラフをビルド
  - TensorFlowライク（名前付き tensor-dimensions を使用）
- “mesh” を定義する
  - プロセッサの論理n次元配列を物理ハードウェアへマップ
- “layout” を選択する
  - tensor-dimensions を mesh-dimensions にマップ

富岳ではmesh と layout はジョブ実行時に決定される
→ジョブスクリプト内でパラメータを決定し実行コマンド引数に渡す



《Mesh TensorFlow》

富岳TensorFlow2.2.0では使用できない

pip install --user mesh-tensorflow を実行すると以下のエラーが発生

```
PermissionError: [Errno 13] Permission denied: '/vol0004/apps/oss/TensorFlow-
2.2.0/lib/python3.8/site-packages/tensorboard_plugin_profile-2.4.0-py3.8.egg/EGG-
INFO/entry_points.txt'
```

- 一般ユーザでは /vol0004/apps/oss/TensorFlow-2.2.0 以下のディレクトリへの書き込み権限がない
- venv 環境下でのインストールも×だった

現時点(2021/10/18)では、Mesh TensorFlowを富岳にて使用するには
TensorFlowを各自のホームディレクトリ上でインストールする必要がある

機械学習ライブラリベストプラクティス

- TensorFlow2.2.0 + Horovod
  - MapReduceモデルをHorovodを使って実現
  - モデル並列は支援機能なし（個別に実装する必要あり）
- TensorFlow2.2.0（+Distributed TensorFlow）
  - 単一計算ノード内で並列化する場合
  - モデル並列は支援機能なし（個別に実装する必要あり）

富岳における第１選択

- Mesh TensorFlowは現時点では使用しない

《HPC一般ユーザ向けOSSインストールツール》

Spack

- HPC環境にて一般ユーザが独自OSSを導入するときに使用する
  - 富岳ではsudo/docker使用不可
  - ただし、将来的にSingularityが使えるようになるらしい
- 富岳上にないOSSを使う場合に利用が推奨されている
  - spack コマンド経由でOSSをホームディレクトリ上にインストール
  - Spackリポジトリ上のパッケージ(Pythonコード)をもとにコンパイル
  - 富岳では独自Spack v0.61.2ブランチを使用(2021/10/14時点)
- Spackは実行時参照する環境変数も管理
  - PATH/LD_LIBRARY_PATH/RPATH/MANPATH..
- Spackパッケージを自分で作成することも可能
- 富岳におけるTensorFlow/PyTorchのSpack提供は準備中とのこと

Spackの主な機能

- 複数のコンパイラ、バージョンを管理
- OSSパッケージ間の依存関係はDAGで管理
- DAGを編集することで、Spack外のライブラリ使用も可能
- conda env に類似した個別環境が構築できる
  - 個別環境で富岳提供OSSも活用する場合はチェイニングを使う
- オフライン環境対応
  - ソースキャッシュミラー、バイナリキャッシュミラー
- moduleコマンドとの親和性..ほか
- 富岳で使用する場合、計算ノード上で実行する必要あり
- 日本語翻訳したSpack v0.16.2 チュートリアル

機能が豊富なため、使いこなすには時間がかかる

別バージョンのTensorFlowを使いたい

- Spack経由でのインストールは難易度高
  - エラー特定が難しい、基本手を出してはダメ
- 「富士通AIフレームワーク利用ガイド1.0」に従ってソースコードからインストールする(2021/10/15こっそり追加された)
  - PyTorch/Chainerについての記載もあり
  - 別バージョン構築するまえにTensorFlow2.2.0でまず試行を推奨
- TensorFlow2.2.0前提で記述なので、Gitリポジトリの内容をHookして、希望のバージョンに変更しつつビルドをすすめる
- bazel前提のOpenJDKはmodule load で導入可(spack installはエラーになる)
- 構築したバイナリはtarball形式にしてジョブバッチでllio_transferした後$HOME/.tmpなどに展開後環境変数指定

おそらくPyTorch/Chainer別バージョンも同様の方法で$HOME上に環境を構築可能

例：tkinterが有効なPython環境構築

- 富岳ガイドに従いプライベートSpack構築
- ${HOME}/.tmp は作成しておく
- 開始・終了時Slackへメール送信される
- Spack 環境 “test” に以下の環境を構築
  - tkinterパッケージが有効なpython
  - pip, py-numpyなどのPythonパッケージ
- Spack “test” 環境を使う場合以下実行

```
. ${HOME}/spack/share/spack/setup-env.sh
spack env activate -p test
```

動作するまでトライアンドエラーを繰り返すため時間がかかる
spack load で導入可能なOSSでおさめるべき

```
1  #!/bin/bash↓
2  #PJM --rsc-list "node=1"↓
3  #PJM --rsc-list "rscunit=rscunit_ft01"
4  #PJM --rsc-list "rscgrp=small"↓
5  #PJM --rsc-list "elapse=72:00:00"↓
6  #PJM --mpi "proc=1"↓
7  #PJM -m b,e,s↓
8  #PJM --mail-list "hogehoge@exa-ict.slack.com"↓
9  #PJM -S↓
10 export ENV_NAME="test"↓
11 PACKAGE_NAMES=( ¥↓
12     "python" ¥↓
13     "py-numpy" ¥↓
14     "py-pip" ¥↓
15     "py-h5py" ¥↓
16     "py-matplotlib" ¥↓
17     "py-tornado" ¥↓
18     "py-docopt" ¥↓
19     "py-pylint" ¥↓
20     "py-pytest" ¥↓
21     "py-pytest-cov" ¥↓
22     "py-progress" ¥↓
23     "py-prettytable" ¥↓
24     "py-mpi4py" ¥↓
25     "py-psutil" ¥↓
26     "py-plotly" ¥↓
27     "py-cremod" ¥↓
28     "py-pyyaml" ¥↓
29 )↓
30 export FLIB_CNTL_BARRIER_ERR=FALSE↓
31 export LD_PRELOAD=/usr/lib/FJSVtcs/ple/lib64/libpmix.so
32 export TMPDIR=${HOME}/.tmp↓
33 . ${HOME}/spack/share/spack/setup-env.sh↓
34 spack env remove -y ${ENV_NAME}↓
35 spack env create ${ENV_NAME}↓
36 spack env activate -p ${ENV_NAME}↓
37 for PACKAGE_NAME in "${PACKAGE_NAMES[@]}"; do↓
38     spack add "${PACKAGE_NAME}"↓
39 done↓
40 spack install↓
41 which python↓
42 python -c "import tkinter"↓
43 which pip↓
44 pip list↓
45 ###↓
46 # EOF↓
47 ###↓
```

バリアント(+tkinter)、Cコンパイラオプションはpackages.yaml上に別途定義

${HOME}/spack/linux/packages.yamlの最終行に追加：

```
    python:
        variants: +tkinter
```

まとめ

①富岳上での機械学習開発は技術的に可能か？

- 可能
  - TensorFlow2.2.0もしくはPyTorch1.7.0であれば環境構築は容易
  - 既存提供環境を外れる場合は、環境構築に時間がかかる

②顧客向け機械学習開発に富岳を活用できるのか？

- 新規モデル開発PoCに適している
  - システムライフサイクルへの富岳組み込みはNG
  - オンデマンドな環境（GCPやAWSなど）で実現すべき

まとめ

富岳での機械学習ベストプラクティス

- ビジネス利用では「新規モデル開発のPoC」向け
- できるだけデータ並列で
- TensorFlow2.2.0 + Horovod

- モデル並列は必要な場合のみ
  - 大規模モデルを使う必要がある場合のみ
- 独自環境は勧めない
  - どうしてもの場合、Spack使用せずにソースコードからのインストールを推奨

富岳利用に必要なスキル

- Linux（RedHat系）
- 機械学習の場合
  - Python/Tensorflow もしくは Python/PyTorch
  - 並列処理化の場合
    - Horovod、Distributed TensorFlow
    - サンプルコードが読める
- 機械学習以外の科学技術計算の場合
  - C/C++/Fortran、MPI/OpenMP

富岳AIフレームワークガイド

TensorFlow2.2.0の場合次のサンプルコード実行例あり：
ResNet/OpenNMT/Bert/Mask-R-CNN

- オプション
  - 富士通コンパイラ独自オプション
  - ジョブ管理ソフトウェア
  - Spack
  - OSS

富岳セミナー入門編

まとめ

自由に使えるテスト環境を別途用意する
・GCP/AWS上に→運用システム化ベースに

<<Master/Login>>
Raspberry Pi4/2GB+SDCard32GB
Raspberry Pi OS Lite
NIS Master/NFS Server
Slurm Master

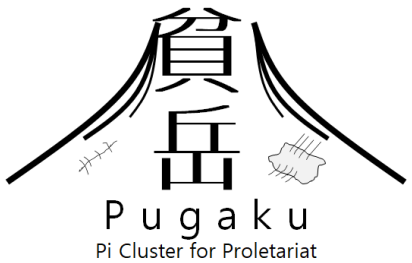
<<Worker/Compute>>
Raspberry Pi3B+/1GB+SDCard32GB
Raspberry Pi OS Lite
NIS Client/NFS Client
Slurm Worker

<<Worker/Compute>>
Raspberry Pi3B+/1GB+SDCard32GB
Raspberry Pi OS Lite
NIS Client/NFS Client
Slurm Worker

<<Worker/Compute>>
Raspberry Pi3B+/1GB+SDCard32GB
Raspberry Pi OS Lite
NIS Client/NFS Client
Slurm Worker

富岳で実行前に自由にテスト可能な環境として 富岳クローン？「貧岳」

おわり



https://github.com/coolerking/fugaku_sample/tree/main/pugaku

補足

サンプルコード

GitHub https://github.com/coolerking/fugaku_sample に以下のサンプルコードを公開

本セッションで使用した
サンプルコード

計算内容	種類	富岳	説明
Πの算出	逐次処理	○	Cによる実装
πの算出	並列処理SPMD	○	Cによる実装。mpiexecコマンドで実行する
MNIST	逐次処理	○	TensorFlow (Python)による実装
MNIST	並列処理SPMD	○	TensorFlow/Horovod (Python)による実装
MNIST	並列処理SPMD	×	Distributed TensorFlowによる実装、PCでは動作する

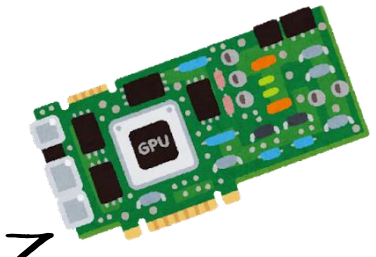
参考文献

- 東京大学出版「スパコンを知る　その基礎から最新の動向まで」
  - 著者：岩下武史・片桐孝洋・高橋大介
  - スパコンのアウトラインをざっくり理解したい方むけ
- 共立出版「Raspberry Piでスーパーコンピュータをつくろう!」
  - 著者：Carlos R.Morrison　訳者：齊藤哲哉
  - 貧岳をつくろうと思ったのはこの本から
  - Piクラスタ構築としては少し物足りないが、並列コンピューティングサンプルも掲載されており読みやすい
- オーム社「HPCプログラミング」
  - 著者：寒川光・藤野清次・長嶋利夫・高橋大介
  - Fortranベースの実践アルゴリズム学習向け
  - 科学技術計算コードを書きたい人向け
  - 機械学習しか使わない人は読まなくてもいい

補足

《富岳プリポストノード》

GPUノードで機械学習



- 富岳にはプロポストノード内にGPU搭載ノードが存在する
- 8ノードしかない
- X86_64 アーキテクチャなので簡単
  - エクサ支給PC上で動作するトレーニング処理はすぐ移植可能
- Python3 の venv を使えばPython環境を分離できる
- NVIDIA GPU/TensorFlowによるGPU内並列化しかできない
- Slurm によるジョブ管理
- バッチジョブの先頭コメント部の書き換えは必要
  - メール経由でのSlack通知も可能

補足

《富岳プリポストノード》

例：GPUノード上にvenv環境構築

- GPUノードにvenv “gpu_sample” 環境を構築
- TensorFlow/PyTorch両方使用可能に
- 他のPythonパッケージは適宜
- Slurmバッチスクリプト
  - sbatch xxxxx.shで実行
  - squeue でジョブ実行状況確認
- Slurmもメール通知可能

2021/10/19 確認
TensorFlowのインストールは成功する

```
#!/bin/bash
#SBATCH -p ppsq
#SBATCH --time=24:00:00
#SBATCH --gpus=1
#SBATCH --gpus-per-node=1
#SBATCH --gpus-per-task=1
#SBATCH --mail-type="ALL"
#SBATCH --mail-user="hogehoge@fugafuga.slack.com"
#SBATCH -J "ratf_env"
#SBATCH --comment="gpu_sample create venv"
export ENV_NAME=gpu_sample
export PATH=${PATH}:/usr/local/cuda-11.2/bin
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/local/cuda-11.2/lib64
mkdir -p ${HOME}/.local
mkdir -p ${HOME}/.local/avx512
cd ${HOME}/.local/avx512
python3 -m venv ${ENV_NAME}
source ${ENV_NAME}/bin/activate
pip3 install pip --upgrade
pip3 install tensorflow-gpu==2.5.0
pip3 install torch==1.8.1+cu111 torchvision==0.9.1+cu111 torchaudio==0.8.1 -f h
tps://download.pytorch.org/whl/lts/1.8/torch_lts.html
# etc
pip3 list
#####
# EoF
#####
```

PJMとはパラメータオプションが異なる

cudaパスは直前にマニュアルで確認のこと

補足

《富岳プリポストノード》

TensorFlowからGPUが参照できない

```
(tensorflow) pps02$ python
Python 3.6.8 (default, Dec  5 2019, 15:45:45)
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> print(tf.config.list_physical_devices())
2021-10-19 07:57:06.982982: E tensorflow/stream_executor/cuda/cuda_driver.cc:271] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
2021-10-19 07:57:06.983013: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: fn02sv02
2021-10-19 07:57:06.983025: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: fn02sv02
2021-10-19 07:57:06.983082: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:200] libcuda reported version is: 460.27.4
2021-10-19 07:57:06.983102: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:204] kernel reported version is: 460.27.4
2021-10-19 07:57:06.983109: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:310] kernel version seems to match DSO: 460.27.4
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
>>> quit()
(tensorflow) pps02$
```

現時点で導入されているcuda-11.2の
PATH/LD_LIBRARY_PATHを設定しても
GPUを参照することはできなかった

2021/10/19 確認
TensorFlowからはGPUがリストできない
）lspci |grep -I nvidia ではV100 2枚存在は確認できる）

補足

《並列処理の設計》

アムダールの法則

全経過時間のP(%)の部分を並列化できるプログラムの側道向上率の上限は $\frac{100-P}{100}$ 倍となる

- SPMDでは全プログラムを並列化できるわけではない
- アムダールの法則は並列数が∞での理論値に過ぎない

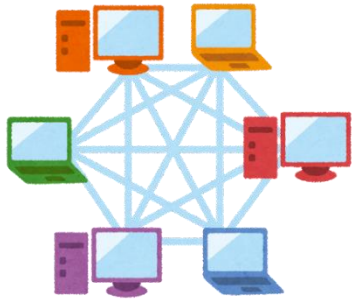
table3.3.1 アムダールの法則

P(%)	10	20	30	40	50	60	70	80	90	95
速度向上率(倍)	1.11	1.25	1.43	1.60	2	2.50	3.33	5.00	10.00	20.00

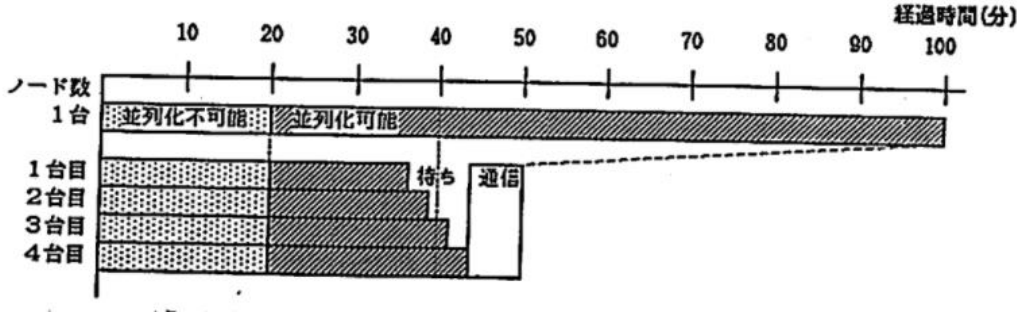
補足

並列処理の設計

待ち時間と通信時間の考慮



- 複数ノードの並列処理では通信時間がボトルネックに
- MapReduceモデルの場合、一番処理のかかった計算ノードに紐づく
- 待ち時間はノードが増加すればするほど問題になる
- 対策
  - 並列化部分をなるべく増やす
  - 漸化式は並列性がない事が多い
  - タスク間のロードバランスをなるべく均一にする
    - サイクリック分割
      - データ内容に依存するので汎用性がひくい
      - 分割した周囲のデータ通信が増える
      - コード量が増える
  - 通信時間をなるべく少なくする
    - 分割数をへらす（並列性が下がる）
    - 通信量がだめなら、通信回数をへらす



補足

HPCI: High Performance Computing Infrastructure

- High Performance Computing Infrastructure
- 全国の最先端計算資源を効率よく利用できる体制と仕組みを構築・整備・提供
- RIST(高度情報科学技術機構)はフラッグシップシステムである「富岳」を中核とするHPCIの利用促進
- 第2階層計算資源：フラッグシップへの橋渡しを担うHPC先端システム
- 全国12機関が様々なタイプの計算機を提供

※2021.9.14 第7回大型実験施設とスーパーコンピュータとの連携利用シンポジウム資料より引用

補足

HPCI 第2階層計算機

- 北海道大学
  - Grand Chaliot:Xeon/EPYC
  - Polaie:KNL
- 東北大学
  - AOBA-A:ベクトル搭載
  - AOBA-B:Xeon/EPYC
- 名古屋大学
  - 不老Type1:A64FX
  - 不老Type2:GPU搭載
- 京都大学
  - システムA(XC40):Xeon/EPYC
- 大阪大学
  - SQUID:ベクトル搭載/GPU搭載
  - OCTOPAS:GPU搭載/KNL
- 九州大学
  - ITO-A:Xeon/EPYC
  - ITO-B:GPU搭載
- 筑波大学
  - Cygnus:GPU搭載
- JCAHPC
  - Oakforest-PACS:KNL
- 東京大学
  - Reedbush-H,L:GPU搭載
  - Oakbridge-CX:Xeon/ePYC
  - Wisteria/BDEC-01:A64FX/GPU搭載
- 東京工業大学
  - TSUBAME3.0:GPU搭載
- 海洋研究開発機構
  - 地球シミュレータ:ベクトル搭載
- 産業技術総合研究所
  - ABCI:GPU搭載

補足

《富岳》

利用時の注意事項

- 富岳側のルールを守ること
  - アカウントの又貸し禁止
    - 法律で罰せられる場合あり
- アカウントをもらったらまず富岳セミナー入門編を受講する
  - ポータルサイトから申し込みが必要
- システム構成が使用中も変わる
  - 毎月第２木曜15:00からユーザブリーフィング（参加申込不要）
    - 参加できない場合は後日資料を確認する
- 運用スケジュールを常に確認
  - 管理側が大規模バッチを１週間まわすことも多い
- ３月後半、９月後半はほぼ使用できないと考えて良い
  - リソース貸し出し期限寸前は大混雑
  - 投入ジョブが実行されるのに１日以上待つことも..
  - プリポストノードはノード数がすくないため上記期間以外も混雑する