May 14, 2021

ITOシステムにおけるコード結合フレームCoToCoAを用いた宇宙プラズマ連成計算シミュレーションの計算・電力性能評価

深沢 圭一郎[1]、加藤 雄人[2]、三宅 洋平[3]、南里 豪志[4]、三吉 郁夫[5]、周 嘉程[6]、中澤 和也[7]

1. 京都大学学術情報メディアセンター
2. 東北大学大学院理学研究科
3. 神戸大学計算科学教育センター
4. 九州大学情報基盤研究開発センター
5. 富士通株式会社次世代TC開発本部
6. 京都大学大学院情報学研究科
7. 神戸大学大学院システム情報学研究科

Background 1

宇宙プラズマ

- 我々が住む宇宙の99.99%の以上の体積はプラズマと呼ばれる電離気体で占められている。
- このプラズマは密度が非常に小さいため無衝突状態にあり、無衝突プラズマ（宇宙プラズマ）を理解することで、我々が住む宇宙の本質的な理解に繋がる。
- 太陽の様々な活動によって宇宙プラズマが動き、宇宙環境変動、地上への電磁的影響が引き起こされる（宇宙天気）。

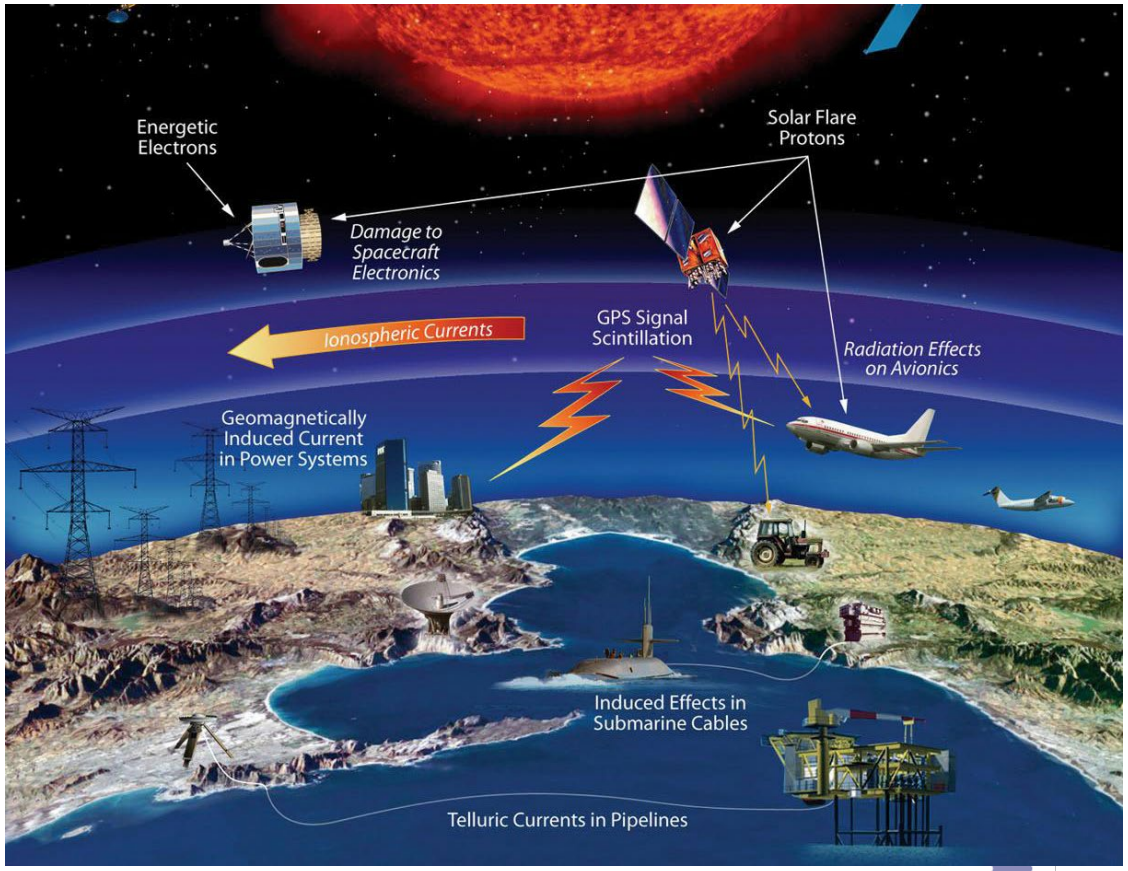


Fig. 1. Space weather affects to the environment [From NASA]

Background 2

宇宙プラズマシミュレーション

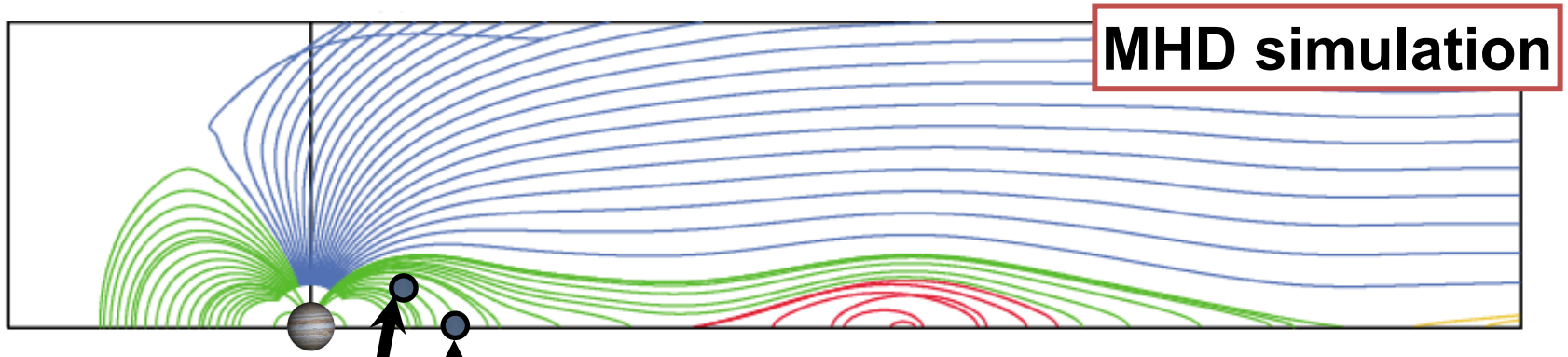
- 宇宙プラズマを流体近似した磁気流体力学（MHD）コードは磁気圏と呼ばれる巨大な惑星磁場の勢力範囲をシミュレーションすることができ、流体近似と粒子を合わせたハイブリッドコードや、粒子コードは磁気圏内やその周辺領域の局所的な運動論を含む物理過程をシミュレーションすることができる。
- 広範囲の宇宙天気を予報する場合、MHDシミュレーションが用いられるが、プラズマの振る舞いを理解するために重要な波動粒子作用などMHDシミュレーションで扱えない現象は、ハイブリッドコードや粒子コードを用いて研究開発が進んでいる。

Motivation

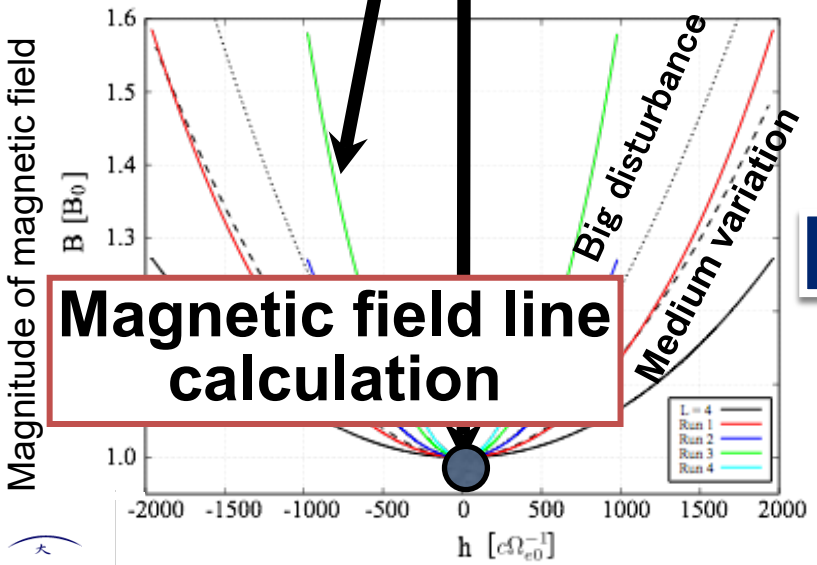
連成計算動作・性能評価と消費電力削減研究

- 本研究ではITOシステムにおいて、この宇宙プラズマの振る舞いを解く粒子コード、ハイブリッドコード、MHDコードにCoToCoAを用いて連成計算させた新しいシミュレーションコードの動作・性能評価を行うことを目的としている。
- CoToCoAは本研究グループが開発しているコード連結フレームワークである。
- また、このような宇宙プラズマコードなどをITO上で走らせる際に消費電力を削減させる手法の評価も行った。

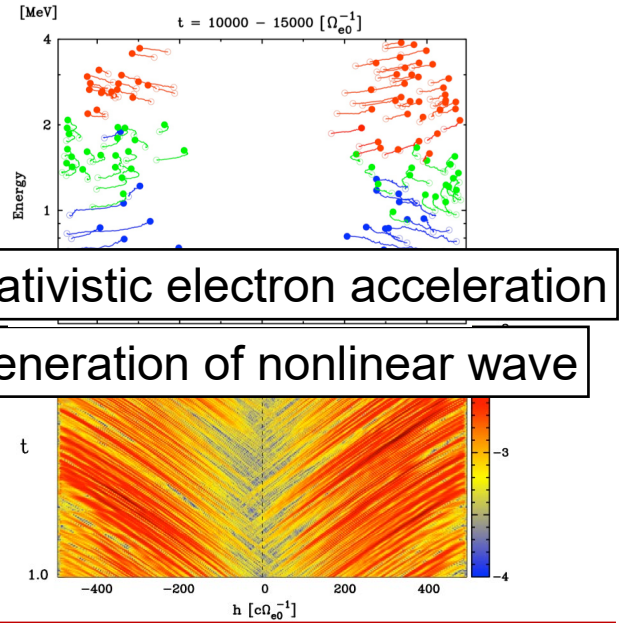
連成計算の例



MHD simulation



Magnetic field line calculation



Relativistic electron acceleration

Generation of nonlinear wave

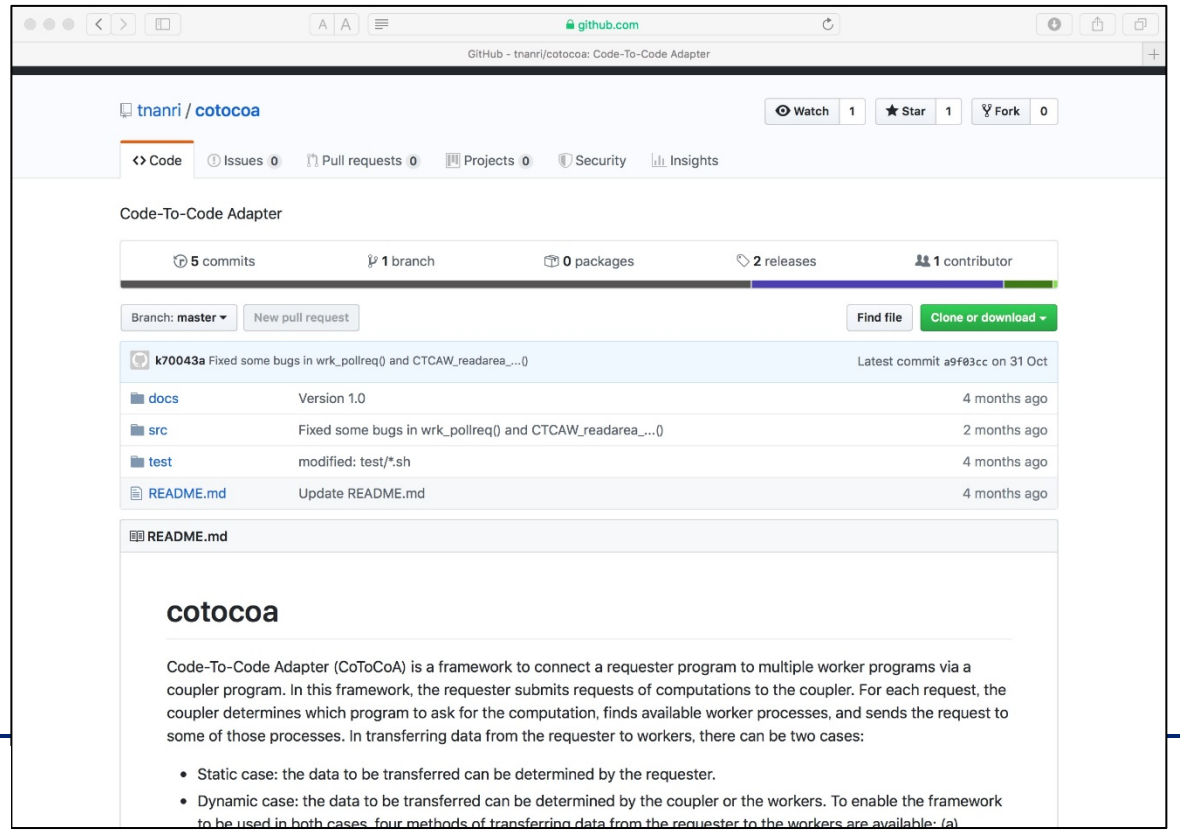
Electron Hybrid simulation

Code-To-Code Adapter (CoToCoA) Library

連成計算のためのフレームワーク

開発の方針

- ✓ 超並列計算で実用に耐える連成計算コードを開発する
- ✓ 連成計算のために必要なコード変更の手間を最小化する



挑戦的研究（萌芽）2017–2019年度
「スケーラブル通信ライブラリを用いた惑星圏
次世代連成計算技術の創出」により開発。

科研費 KAKENHI

京都大学 KYOTO UNIVERSITY

https://github.com/tnanri/cotocoa

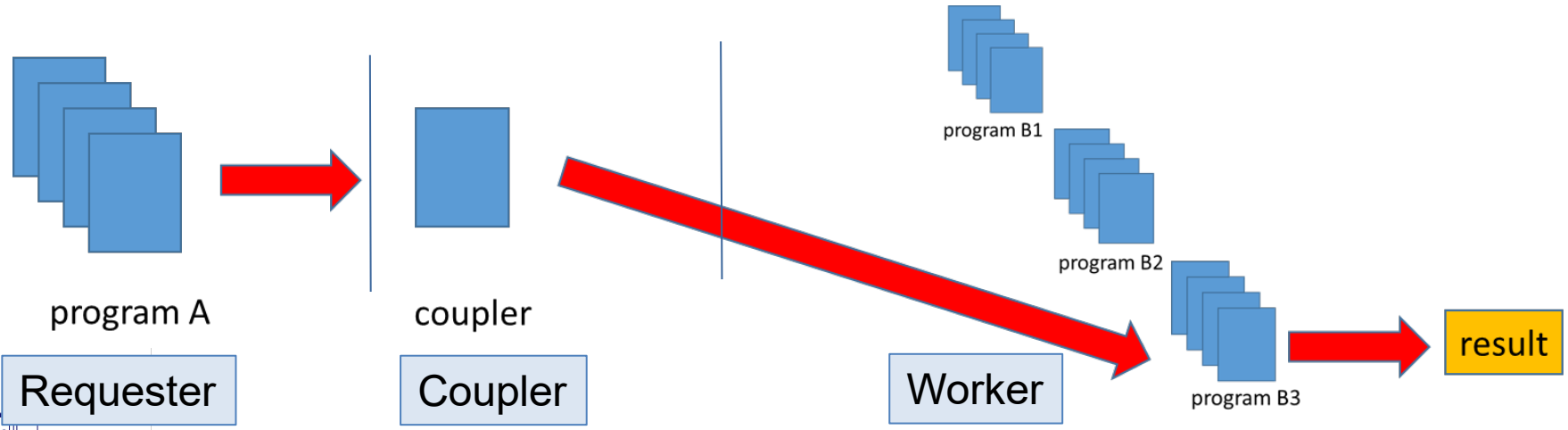
CoToCoAの概略

連成計算のための3種類のプログラム向けインタフェース

Requester : Couplerに計算依頼（request）を送信

Coupler : Requesterからの requestに応じてプログラムを選択
空いている Worker担当プロセス群に requestを転送

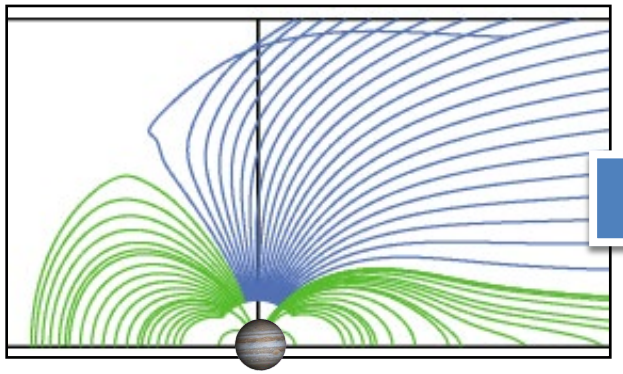
Worker : Couplerからの requestに応じてプログラムを実行



CoToCoAの使われ方

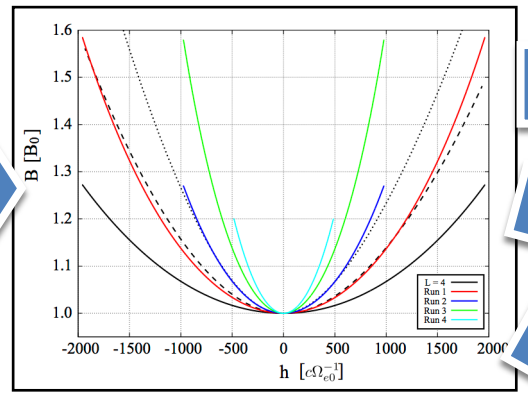
Requester

MHD

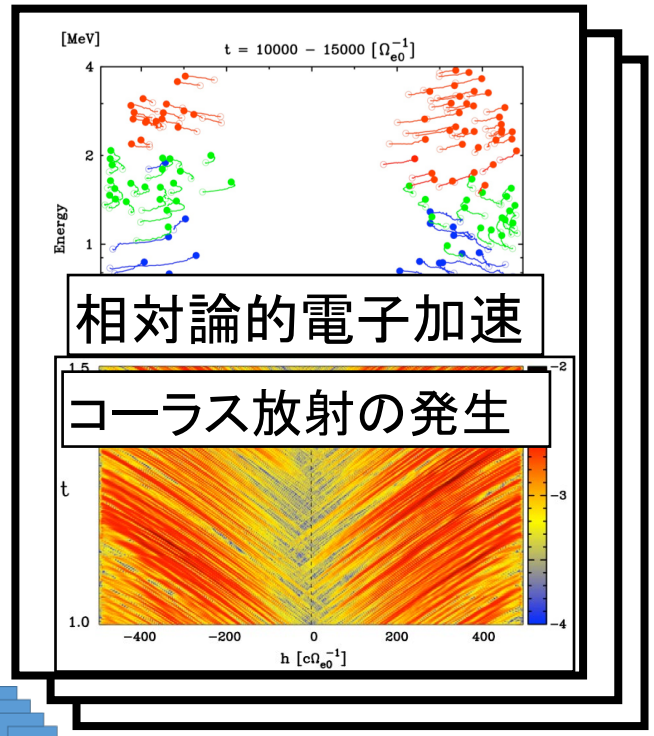


Coupler + α

磁力線抽出



Worker



Requester

Coupler

Worker

京都大学
KYOTO UNIVERSITY

実装の概略

Coupler reads the data of Requester and transfer it to Worker

Requester

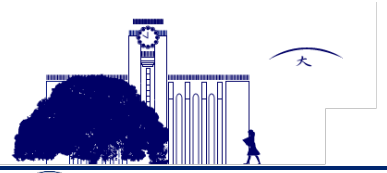
```
call CTCAR_init()

! Initialization part

call CTCAR_regarea_real8( &
     data, size, areaid)

do step = 1, nsteps
!  main calculation part

  call CTCAR_sendreq(dataint, &
       dataintnum)
end do

call CTCAR_finalize()
```

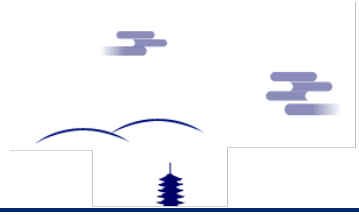
Coupler

```
call CTCAC_init()

! Initialization part

call CTCAC_regarea_real8(areaid)

do while
  call CTCAC_pollreq(reqinfo, fromrank, &
       dataint, dataintnum)

  if (CTCAC_isfin()) then
    exit
  end if

! select the program of Worker and rank,
offset
! calculation part

  call CTCAC_readarea_real8(areaid, rank, &
       offset, size, data)

  call CTCAC_enqreq_withreal8(reqinfo, progid,
&
       dataint, dataintnum, data, size)
end do
call CTCAC_finalize()
```

Worker

```
call CTCAW_init(progid, procsperreq)

call CTCAW_regarea_real8(areaid)

do while
  call CTCAW_pollreq_withreal8( &
       fromrank, dataint, dataintnum, &
       data, size)

  if (CTCAW_isfin()) then
    exit
  end if

! Initialization part

! main calculation

  call CTCAW_complete()

end do
call CTCAW_finalize()
```



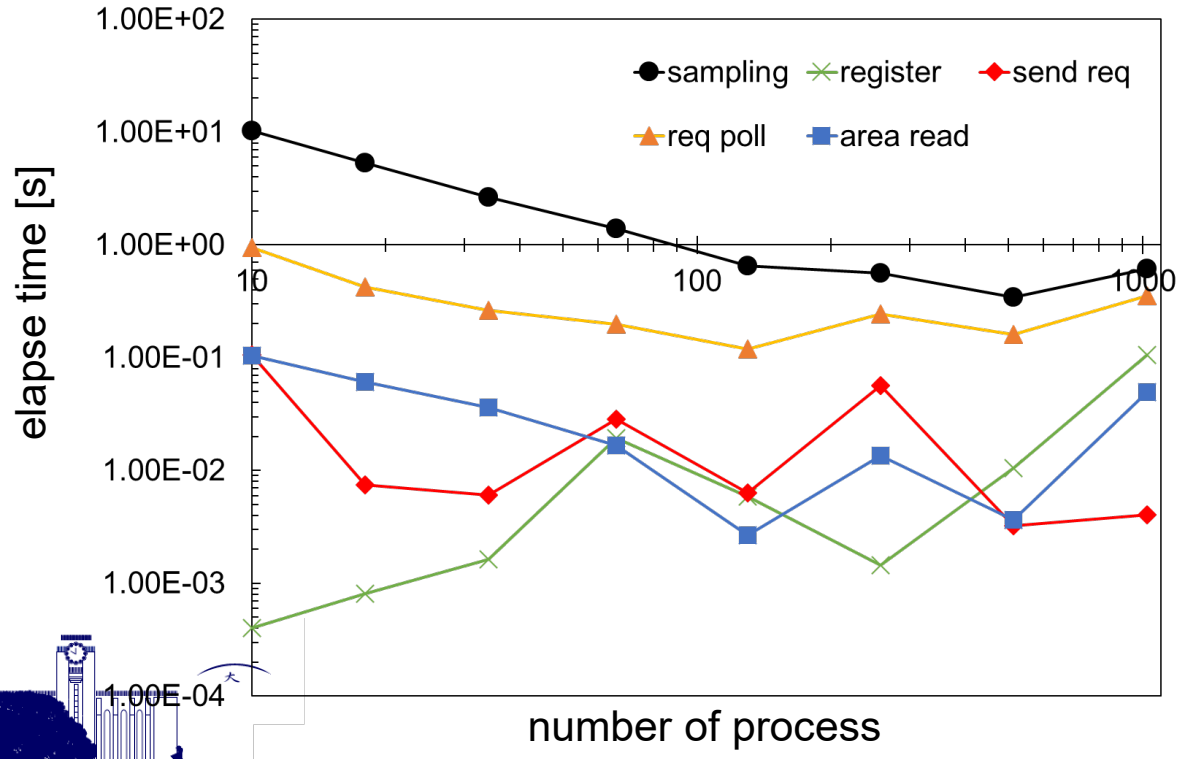


CoToCoA Performance 1

Parallel Scaling評価

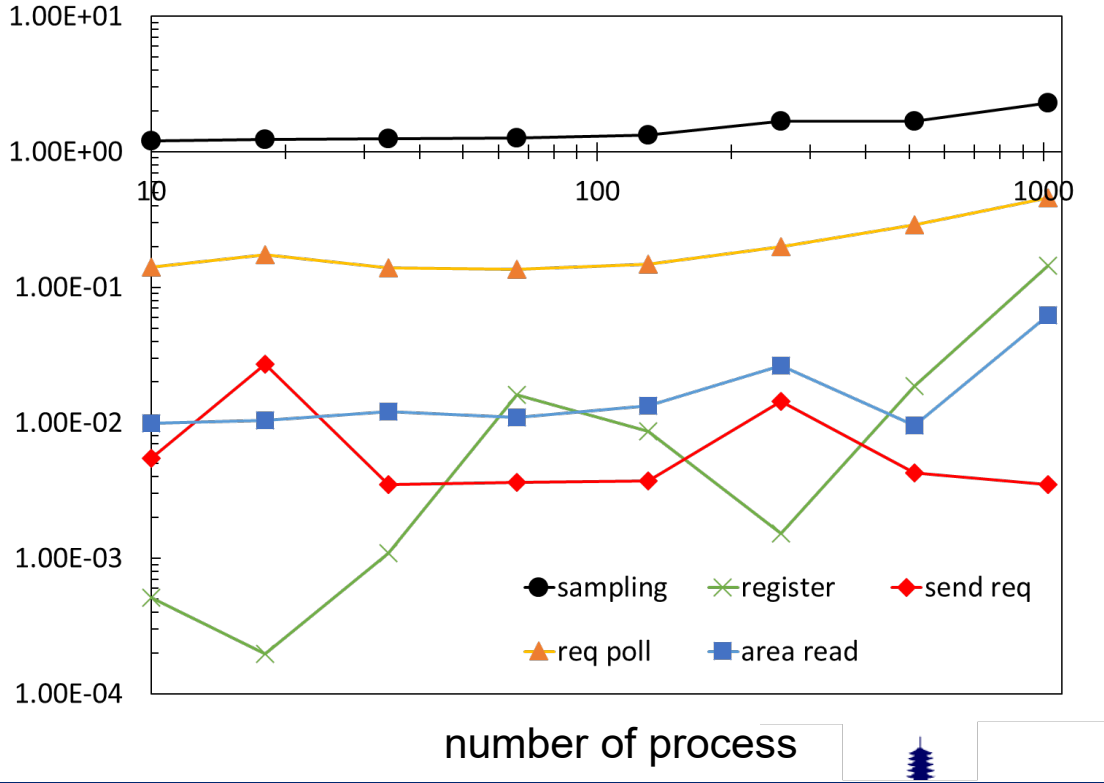
Strong

- 10~66 processes parallel (1 process/node)
- Grid size: 400x400x400



Weak

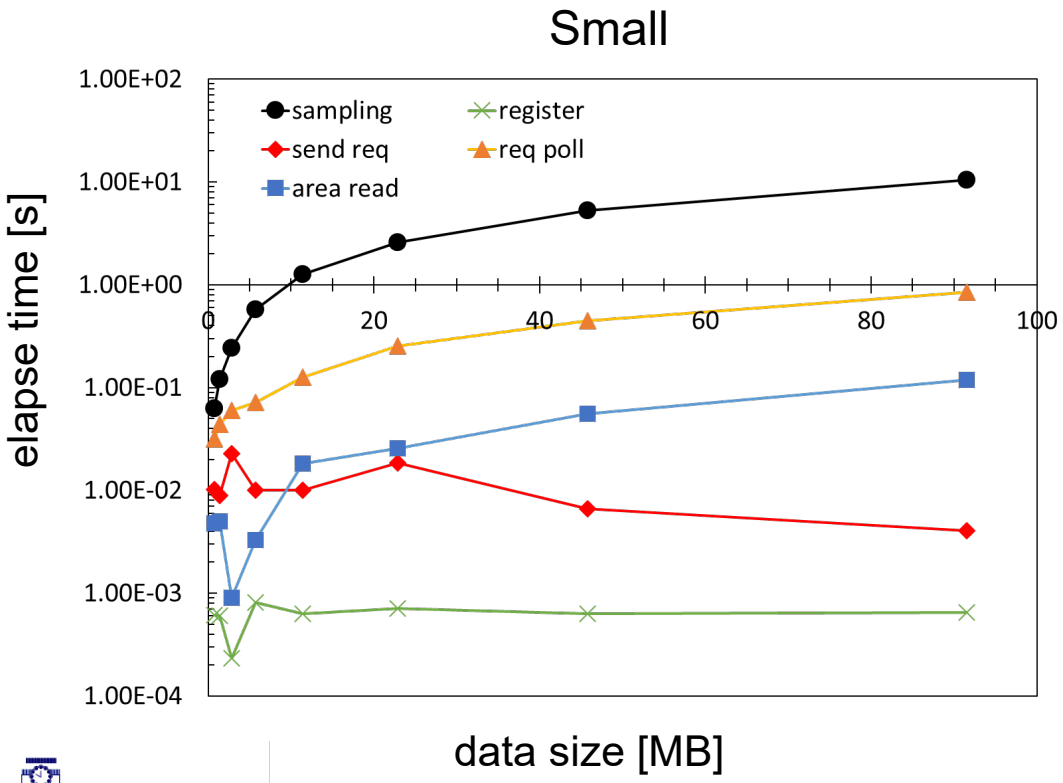
- 10~1026 processes parallel (1 process/node)
- Grid size: 100x100x100/process

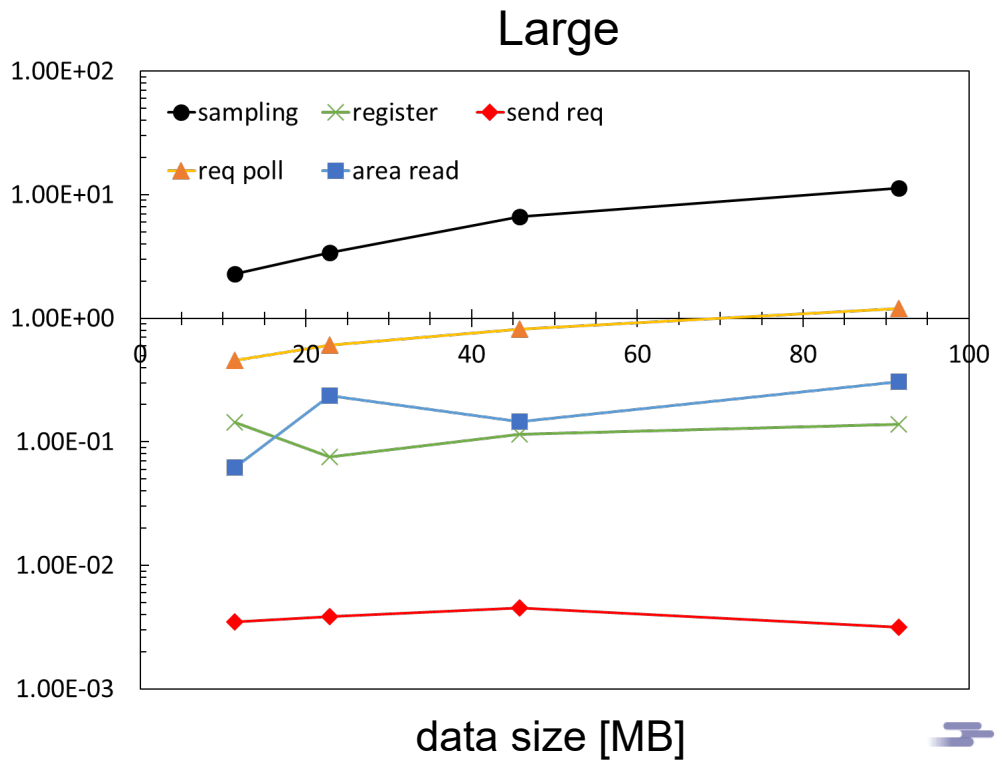


CoToCoA Performance 2

Data size scaling評価

- 18 (small) and 1026 (large) processes parallel (1 process/node)



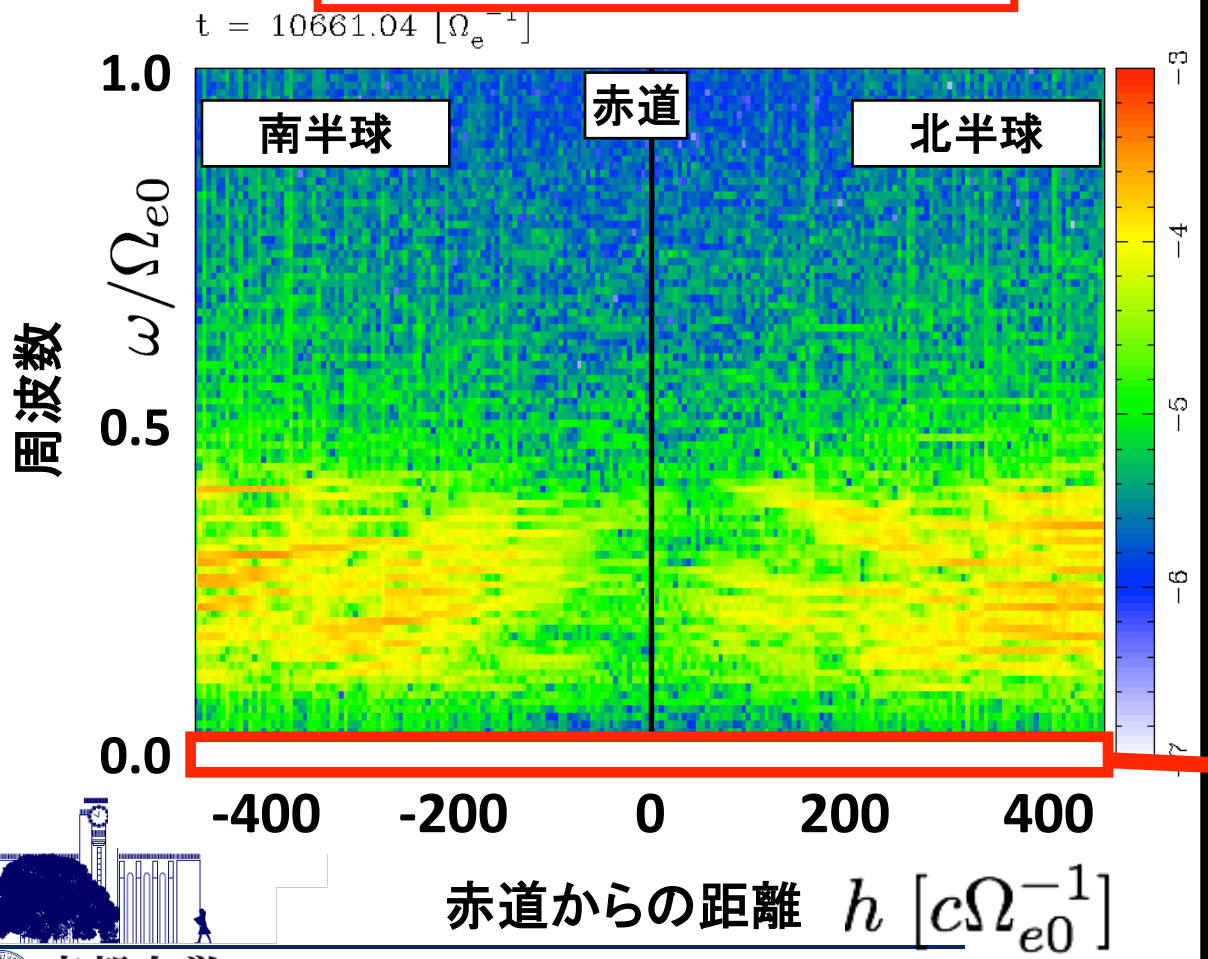


京都大学 KYOTO UNIVERSITY

地球磁気圏でのプラズマ波動の発生(粒子計算)と伝搬(流体計算)過程の連成計算

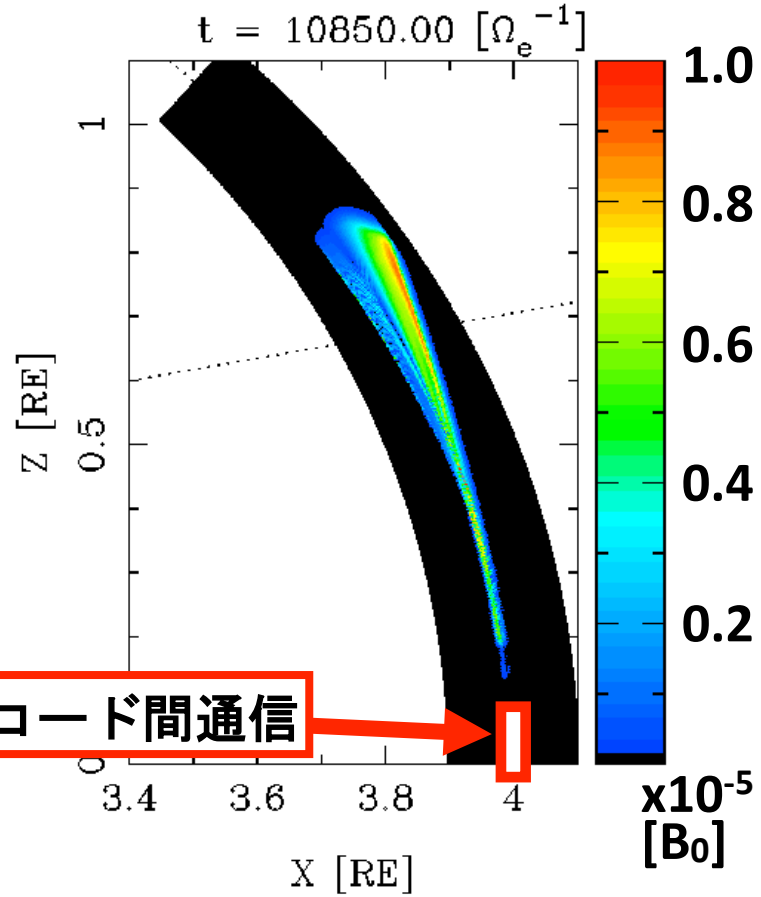
磁力線に沿った空間1次元計算

プラズマ粒子コード



磁気圏子午面の空間2次元計算

プラズマ流体コード



コード間通信

京都大学
KYOTO UNIVERSITY

プラズマ波動の発生(粒子計算)と伝搬(流体計算)過程の連成計算

ITOシステム全ノード利用プロジェクトでの動作検証・ベンチマーク実施状況

連成計算コード（粒子＋流体）の性能評価：最大1901プロセスまで実施
（1 MPI process / node）

粒子コード・流体コード単体での性能評価：最大16384プロセスまで実施
（複数MPI processes / node）

- 32768プロセスまでの実行を試みるも、ネットワークカードの接続数を超過して実施できず
- 連成計算コード、単体コードでの性能評価結果を確認中

連成計算による宇宙プラズマ粒子速度分布解析手法の開発

プラズマ粒子シミュレーション

プラズマとは？

-物質の第4の状態

→荷電粒子(電子、イオン)の集合体

宇宙理工学での重要性

→宇宙開発、産業応用に不可欠

(例:はやぶさイオンエンジンの設計)

宇宙プラズマの実験・観測は困難

→計算機シミュレーションが有用

→PIC計算

テスト粒子法による荷電粒子速度分布解析

粒子群の全運動状態を知る

→プラズマ粒子1つずつを調べることは困難

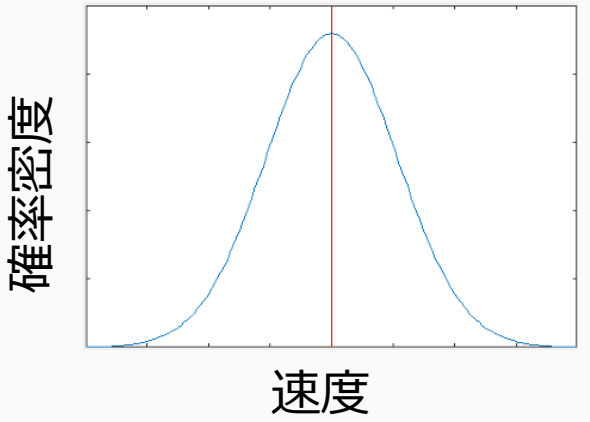
→ある速度の粒子の割合に注目(テスト粒子計算)

荷電粒子速度分布解析手法を発展

→より大量の分布を取得したい

→計算を高速化したい

→一連の解析を効率化したい

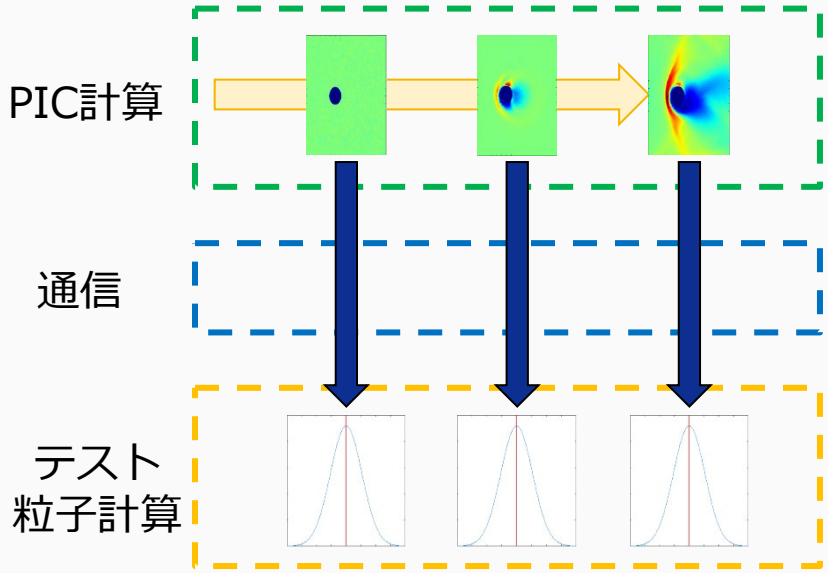


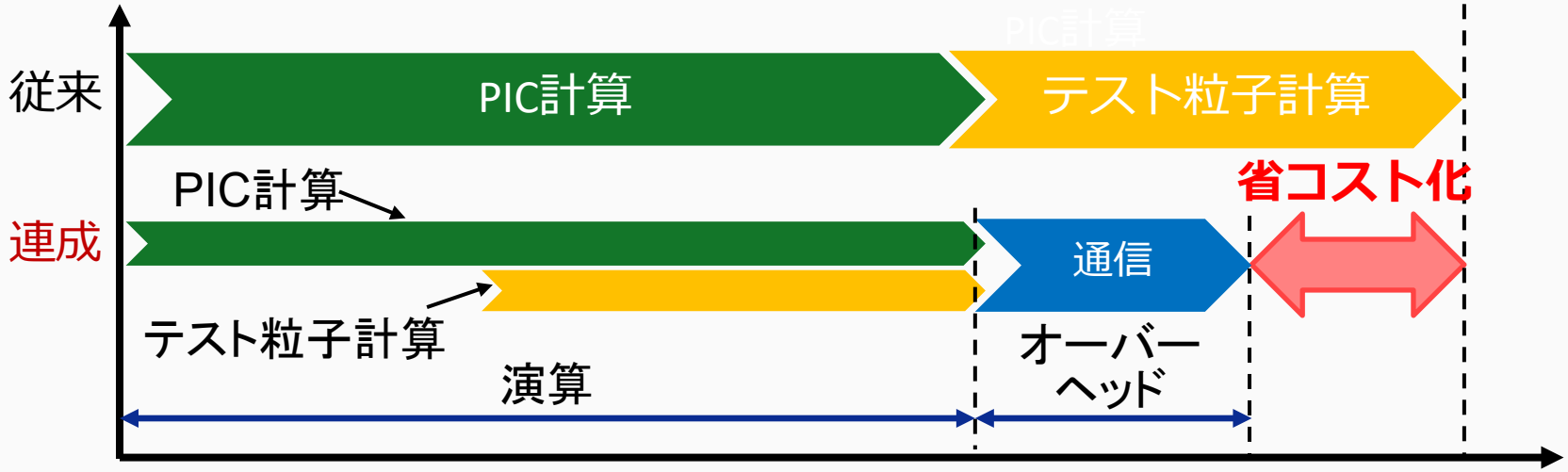
「プラズマ粒子シミュレーション」⇒「速度分布解析」
の一連の流れを連成計算により、高効率化

PIC計算とテスト粒子計算の非同期並行実行

- 従来のアプローチ
  →PIC (Particle-In-Cell) 計算とテスト粒子計算を順番に実行
  →別のプログラムを逐次実行するため非効率

- 提案手法
  →コード間結合フレームワークCoToCoAによる非同期・並行な連成計算
  →PIC計算とテスト粒子計算をオーバーラップし、全体の処理に要したコストを削減
  →PIC計算で得られた電場・磁場をテスト粒子計算に送信(request)

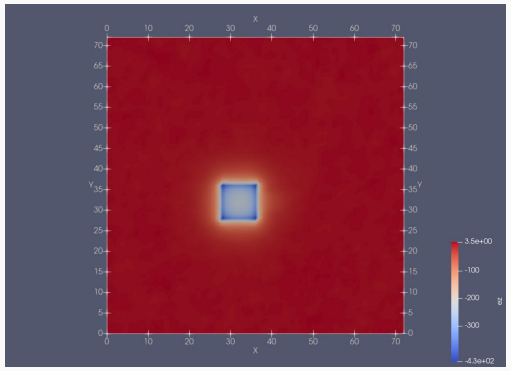




オーバーヘッドは演算部分に比べて1/100未満のコスト

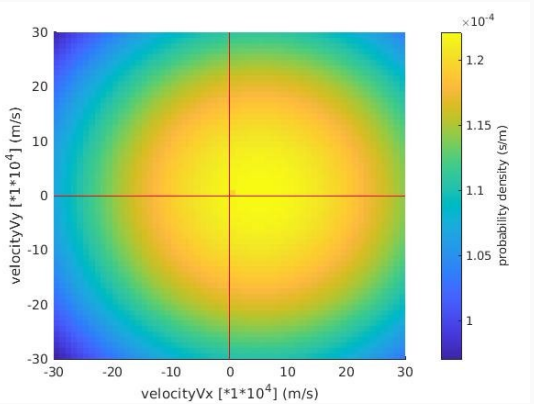
ITOシステム全ノード利用プロジェクトでの動作検証

- プログラム全体で最大26,065MPIプロセスでの実行を確認
  →30,000プロセス前後にハードウェア的上限がある可能性



PIC計算 with 11,664プロセス
実行時間: 1.42×10^{1} s

CoToCoA通信(1.79MB) with 1プロセス
処理時間: 1.43×10^{-2} s/req



テスト粒子計算 with 14,400プロセス
実行時間: 4.86 s/req

テスト粒子計算に割り当てた資源が使用されていない時間が存在

今後の展望

- プログラム全体での最大プロセス数の制約の原因調査

- 動作確認ができた26,065プロセス程度での最適化
  →PIC計算とテスト粒子計算へのプロセス配分
  →テスト粒子計算の並行実行(CoToCoAによるグループ化)

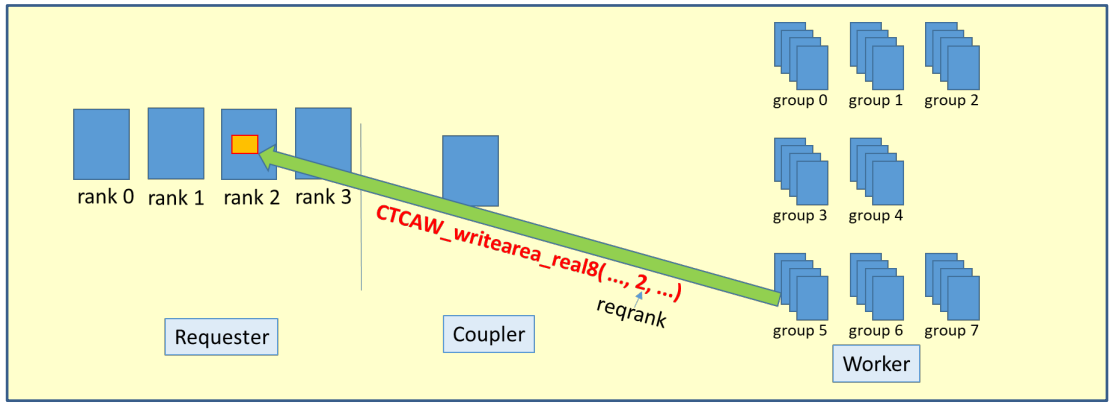
- スケーラビリティの調査
  →要素計算毎の並列性能調査
  →性能特性に基づき、自動最適プロセス配分手法を検討

- 実アプリへの提案フレームワークの適用

CoToCoA新規機能追加

1年間の連成計算評価によって、CoToCoAに新規機能が必要と提案された。
下記2件の機能が新たに追加されている。
評価は2021年度の予定

Workerから Requesterへのフィードバック　Requesterでのタスク処理状況確認




```
hdl_ctr = 1

do i = 1, 10
     ...
     call CTCAR_sendreq_hdl(dataint, 2, hdl(i))
     ...

     if (myrank == 0) then
        if (CTCAR_test(hdl(hdl_ctr))) then
           ! Check Result of hdl_ctr
            ...

           hdl_ctr = hdl_ctr + 1
        end if
     end if
end do
```

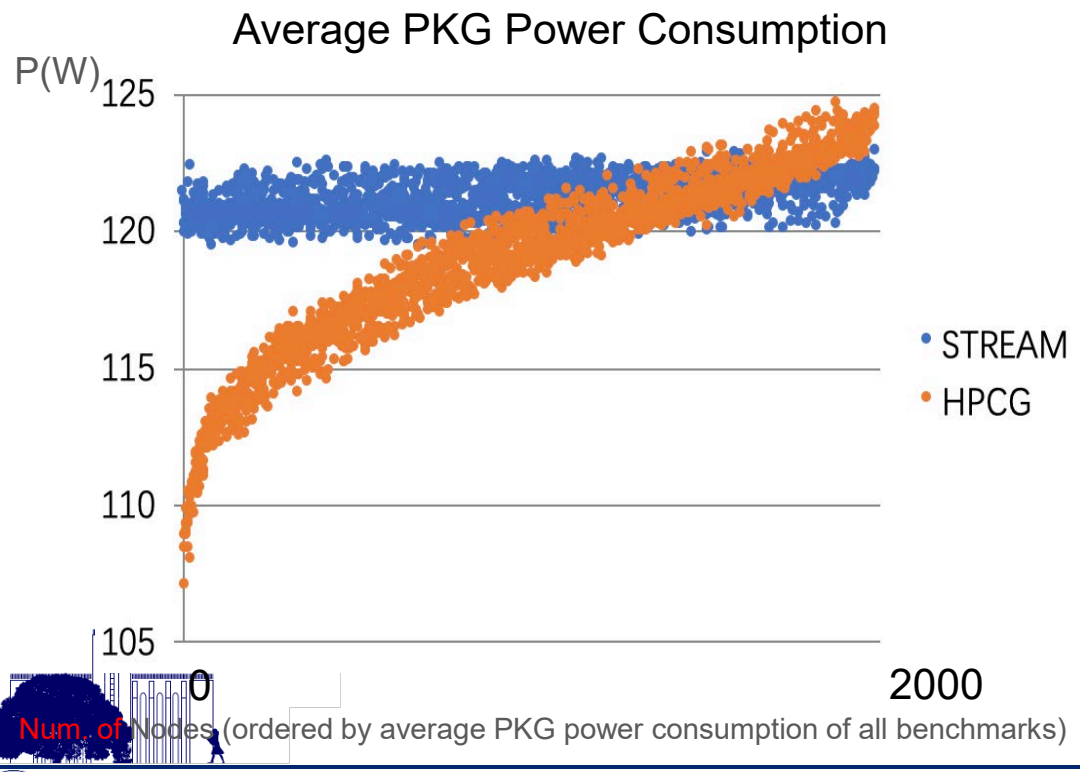
Power Evaluation

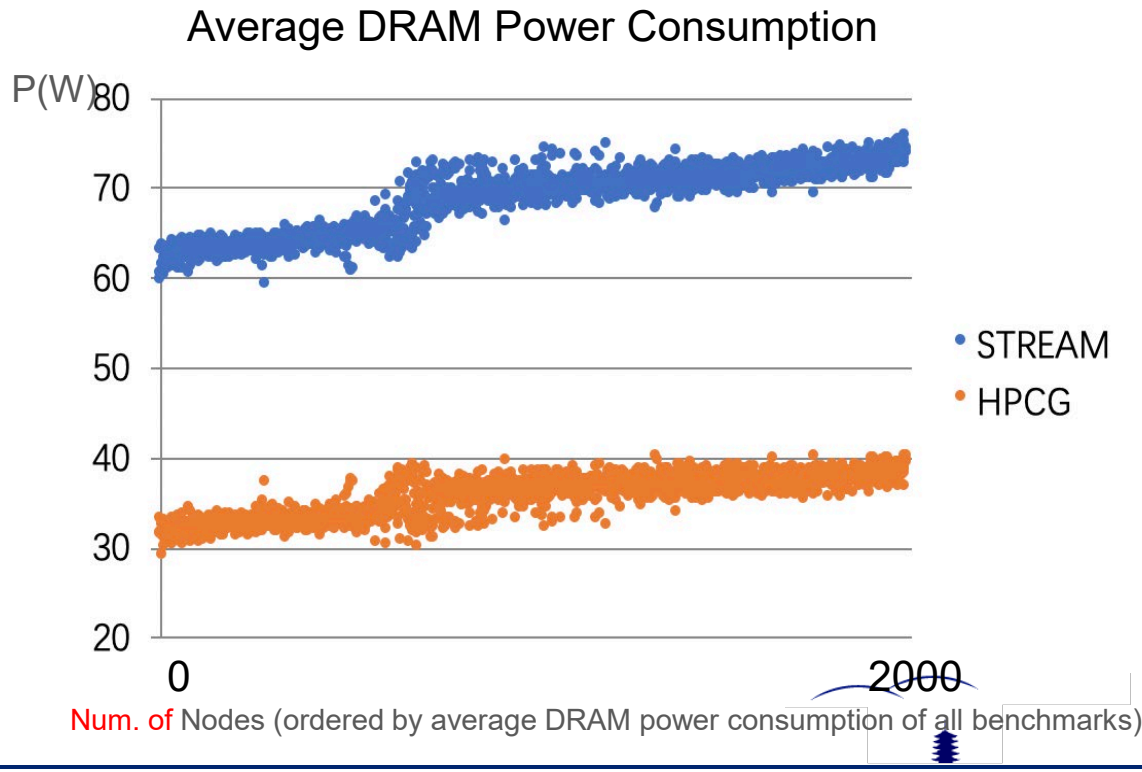
消費電力削減スケジューリング

- ITO全体を利用し、ノード毎の電力性能を測定。
- ノードにおける消費電力性能のばらつきを利用して、通常のスケジューリングに比べて、省電力なスケジューリングができないか実験した。
- ここでは、Inadomiらが開発したRAPL利用インターフェースであるRICを用いて、消費電力の測定、CPU消費電力に制限をかけた場合の電力性能の評価を行った。

Validation of Node-level Power Heterogeneity

Two benchmarks are chosen: HPCG (computation-intensive), STREAM (memory-intensive).
Benchmarks show different power behaviors in the supercomputer.
The DRAM power ranking of nodes has no obvious relationship with PKG power ranking.
Scheduler should consider both the power efficiency of nodes and jobs in the job queue.





Combinatorial Optimization Algorithm

Power Aware Algorithm (PAA): Always assign job to the most efficient node. The energy saving capability is proved in previous research [1]. PAA does not always choose the allocation plan with minimum energy consumption because it does not consider the property of jobs and jobs in the job queue.

There are usually two or more jobs to be scheduled at the next scheduling interval in the head segment of the job queue, which means jobs to be scheduled is known.

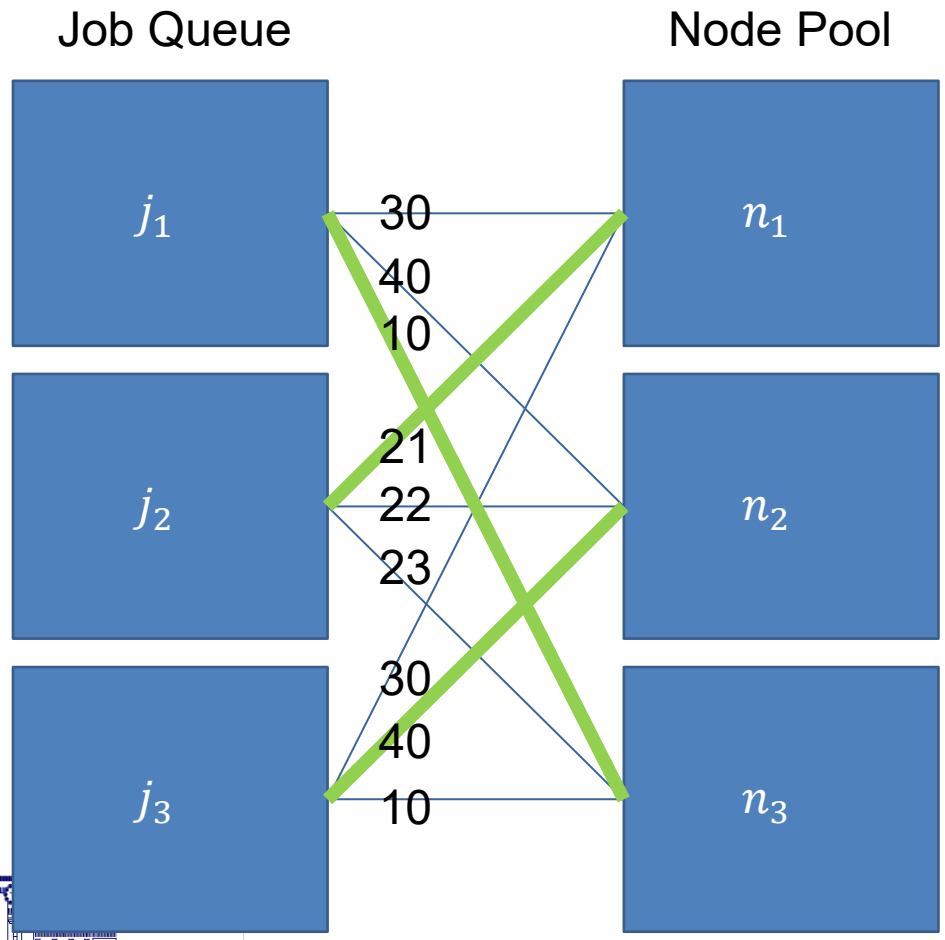
COA aims at using information of the property of jobs and power efficiency of nodes to calculate the allocation plan with the minimum energy cost.

The process of scheduling can be transformed into solving an optimal matching problem in a graph with KM algorithm.

[1] LE LI, Keiichiro FUKAZAWA, Hiroshi NAKASHIMA, and Takeshi NANRI. A node-level performance/power efficiency aware resource management technique.

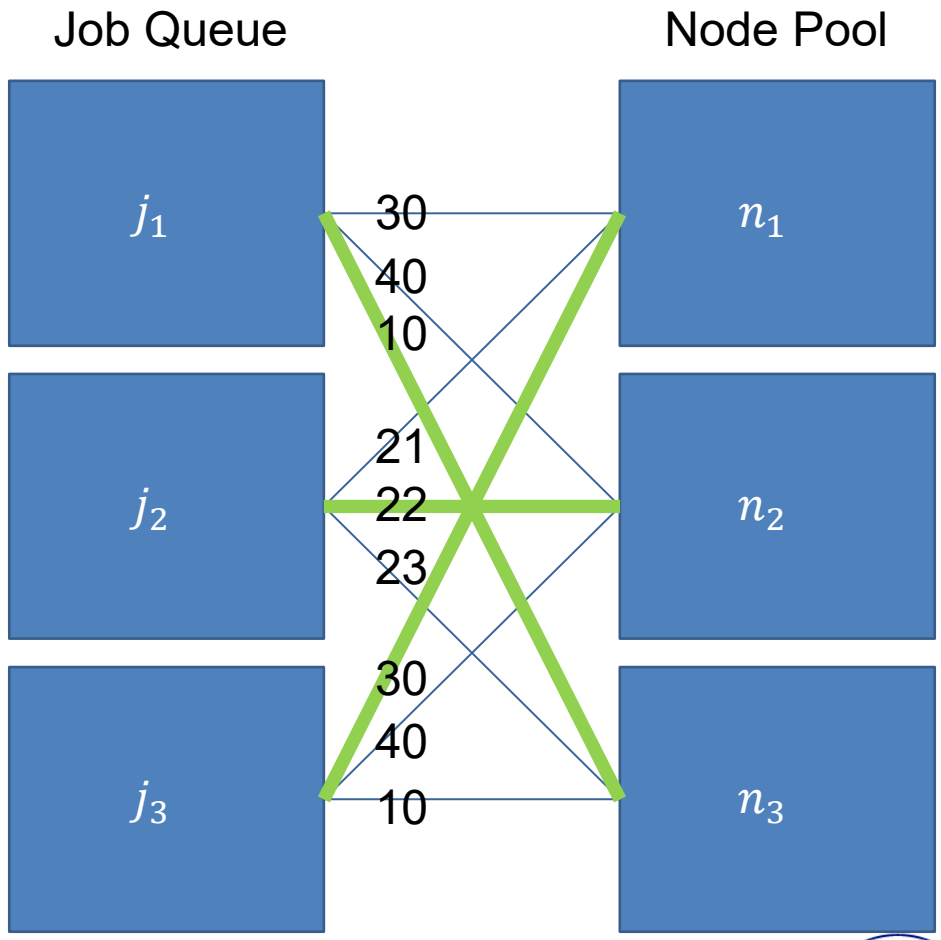
Solving Matching Problem in the Scheduling

Power Aware Algorithm (PAA)



Weight (Energy)=71

Combinatorial Optimization Algorithm (COA)



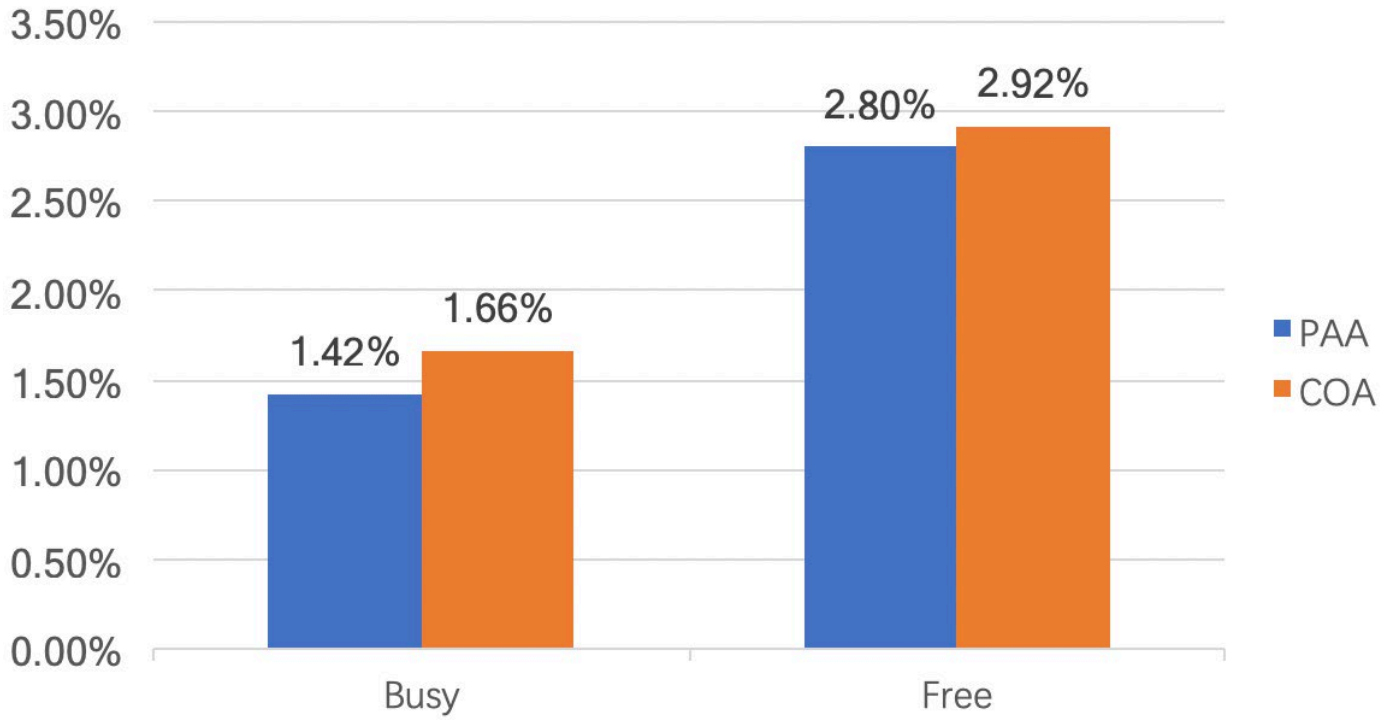
Weight (Energy)=62

Evaluation of Combinatorial Optimization Algorithm

Energy saving rate: $Saving_A = \frac{E_{Naive} - E_A}{E_{Naive}}$

In all scenarios, COA saves more energy than PAA.

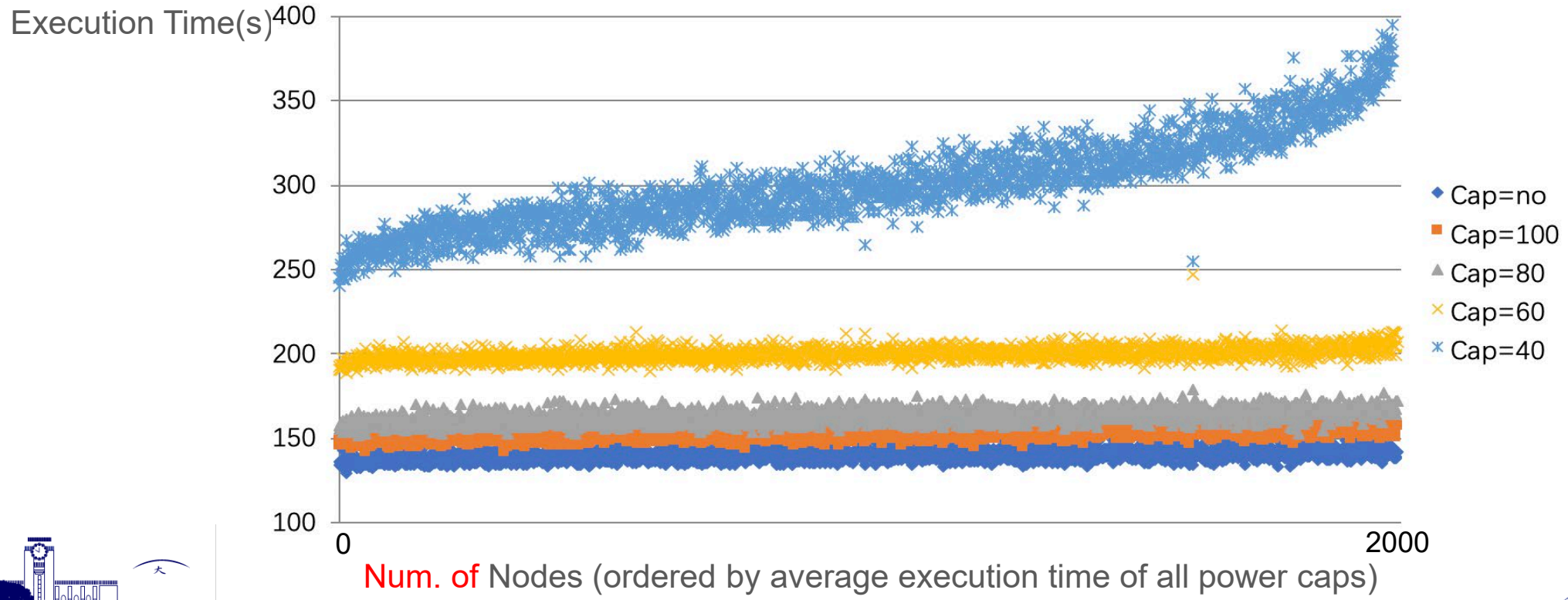
COA achieved up to 2.92% energy saving in the real supercomputer.



Node-level Performance Heterogeneity Under Power Caps

Power heterogeneity is transformed into performance heterogeneity under power caps.

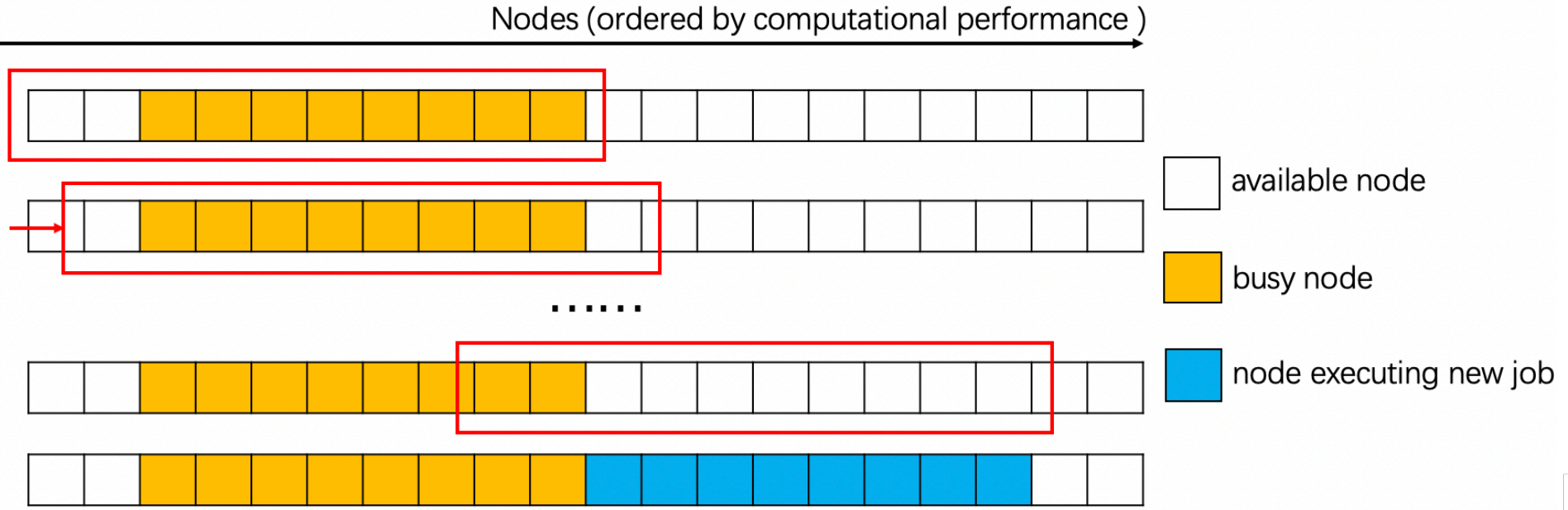
Execution time increases significantly under tight power cap (40W), and execution time shows a large variation.



Sliding Window Algorithm

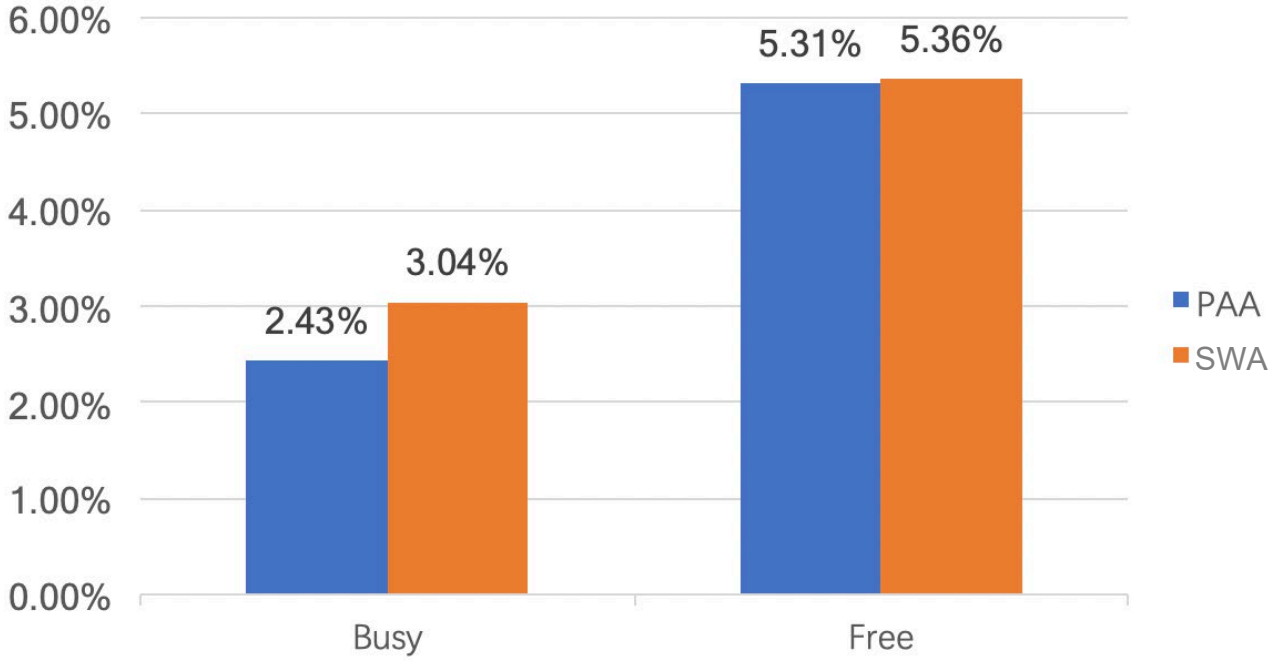
When jobs assigned to a mixture of good nodes and bad nodes, good nodes cannot take the advantages of performance since execution time of multi-node jobs is determined by the worst node.

Sliding Window Algorithm (SWA) performs node assignment for a q–node job by sliding a window wider than q nodes over all nodes, ranked by the computational performance, until q available nodes are included in the window.



Evaluation of Sliding Window Algorithm

- Jobs in a historic workload of Laurel 2 (from Sept. 18, 2019 to Sept. 25, 2019) are classified by execution time and required nodes. In the experimental setup, the number of jobs is set according to the result of classification.
- In all scenarios, SWA saves more energy than PAA.
- SWA achieves up to 5.36% energy saving compared to Naïve.



Summary

ITO全ノードを使い、連成計算の実行確認を行った。

- ✓ 連成計算ライブラリにはCoToCoAを利用し、MHDコードと磁力線計算の連成では、シミュレーション時間に比べて連成のオーバヘッドは十分に低いことが確認された。
- ✓ PICシミュレーションとテスト粒子計算の連成では、最大26,065MPIプロセスでの実行が確認できたが、それ以上では動かなかったため、何かしらの制限があると思われる。

ITO全ノードを使い、ノードの電力性能ばらつきを利用した省電力スケジューリング実験を行った。

- ✓ シングルノードでは、アプリケーションの電力特性も考慮することで、最大2.92%の省電力が可能だった。
- ✓ マルチノードでは、割り当てノードを一定のWindowサイズにより探査することで、効率的に省電力ノードを活用でき、最大5.36%電力削減が可能だった。