Vol.2020-BIO-61 No.3
2020/3/12

マルチノード・マルチ GPU 上での網羅的なタンパク質間相互作用予測の高速化

渡辺 紘生[1,2]　大上 雅史[1]　秋山 泰[1,a)]

概要：生命現象の解明や医薬品設計の高速化へ貢献するため，我々は網羅的なタンパク質間相互作用予測ソフトウェア MEGADOCK を開発している．本研究では MEGADOCK を対象として，マルチノード・マルチ GPU 上での効率的な予測計算実行を可能とするための，効率的なノード間タンパク質ペア分配スケジューリング手法を提案し実装した．提案スケジューリング手法を導入した MEGADOCK を用いて，東工大 TSUBAME 3.0 128 ノードでの 211,600 件の大規模なタンパク質間相互作用予測計算を行った結果，4 ノードを基準として強スケーリング 98.5%を達成した．また，産総研 AI 橋渡しクラウド (ABCI) 512 ノードでの 1,322,500 件の超大規模なタンパク質間相互作用予測計算の結果，16 ノードを基準として強スケーリング 96.7%を達成した．

キーワード：タンパク質間相互作用，PPI 予測，MEGADOCK，高性能計算，タンパク質ドッキング

Acceleration of protein-protein interaction prediction on HPC environments with multiple computation nodes and multiple GPUs

Hiroki Watanabe[1,2]　Masahito Ohue[1]　Yutaka Akiyama[1,a)]

Abstract: In order to contribute to accelerating elucidation of biological phenomena and drug discovery, we are developing MEGADOCK, a software for exhaustive protein-protein interaction (PPI) prediction. In this study, for the sake of efficient use of large-scale HPC enviroments that has a large number of computation nodes and GPUs, we proposed and implemented new scheduling methods of target protein pairs, which we newly introduced to MEGADOCK. For evaluating parallel performance of MEGADOCK with proposed scheduling methods, We conducted a large-scale PPI prediction calculations in which we use 211,600 target protein pairs on TSUBAME 3.0 and 98.5% parallel efficiency of strong scaling (vs. 4 nodes) was achieved at 128 nodes. We also conducted an ultra-large-scale PPI prediction calculations in which we use 1,322,500 target protein pairs on AI Bridging Cloud Infrastructure (ABCI) and 96.7% parallel efficiency of strong scaling (vs. 16 nodes) was achieved at 512 nodes.

Keywords: protein-protein interaction (PPI), PPI prediction, MEGADOCK, high-performance computation (HPC), protein-protein docking

1. 導入

生体内の多くのタンパク質は相互作用しながら機能を果たしている [1]．この相互作用はタンパク質間相互作用 (protein-protein interaction, PPI) と呼ばれており，生命現象の中核を担っている．また，近年では疾患の原因とされる PPI を阻害することで効果を得る PPI 阻害剤も開発

[1] 東京工業大学 情報理工学院 情報工学系
Department of Computer Science, School of Computing, Tokyo Institute of Technology
[2] 産業技術総合研究所 産総研・東工大実社会ビッグデータ活用オープンイノベーションラボラトリ (RWBC-OIL)
AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory (RWBC-OIL), National Institute of Advanced Industrial Science and Technology (AIST)
[a)] akiyama@c.titech.ac.jp

© 2020 Information Processing Society of Japan

されている [2,3]．しかし，タンパク質はヒトだけでも約 18 万種（2019 年 12 月時点，UniProt [4] (Universal Protein Resource) の AC エントリに基づく）が確認されており，それらのタンパク質に対して生化学的な実験によって網羅的に PPI を決定することは，時間的・金銭的な面で困難である．一方，計算機を用いて PPI を予測することにより，相互作用する可能性の高いタンパク質のペアを選出し，実際に生化学実験の対象とするタンパク質の数を減らし，時間的・金銭的コストを削減することが可能である．よって，計算機を用いた PPI 予測手法が注目されている．

計算機による PPI 予測手法には，タンパク質の立体構造を用いる手法 [5] のほか，アミノ酸配列 [6] や共進化情報 [7] に基づく手法がある．ここで，タンパク質立体構造情報を用い，かつ既知の PPI 情報を利用しない手法の 1 つに，我々が開発している MEGADOCK [8] が存在する．MEGADOCK は 2 つのタンパク質立体構造情報を入力としてタンパク質ドッキング計算を行い，その計算結果から PPI 予測を行うことができる．

MEGADOCK は，単純化された評価関数設計やマルチノード・マルチ GPU 上での計算を可能にする実装などの工夫による高速化がなされており，東工大 TSUBAME 3.0 の計算ノード 1 ノード，GPU 4 基使用時に，1 タンパク質立体構造ペアあたり約 2 秒でドッキング計算を行うことが可能である．しかし，タンパク質の相互作用ネットワークの予測には多数のタンパク質からなるタンパク質群の PPI 予測を行う必要があり，網羅的なドッキング計算に必要な計算時間は大きい．例えば，既知のヒトのタンパク質立体構造は 6,000 種類以上解明されており，それらのタンパク質立体構造に対して全対全の PPI 予測を行う場合，TSUBAME 3.0，70 ノード，GPU 280 基を用いても $2\ [\mathrm{s\cdot node}] \times 6{,}000^2/70\ [\mathrm{node}] \approx 12\ [\mathrm{day}]$ 以上要することとなる[*1]．さらに，近年 Protein Data Bank (PDB) [9] をはじめとする公共データベース中のタンパク質立体構造情報が増加している．以上より，ドッキング計算のさらなる高速化が必要とされる．

本研究では網羅的な PPI 予測のためのタンパク質ドッキング計算を高速化するために，従来の MEGADOCK の MEGADOCK のマルチノード環境下における，各ノードへのタンパク質立体構造ペア分配に関するスケジューリング手法を新たに提案し，実装した．スケジューリング手法の改良により，大規模なマルチノード・マルチ GPU 環境である，東工大 TSUBAME 3.0 および産総研 AI 橋渡しクラウド (ABCI) で高い並列化効率を達成することを目的とする．

[*1] MEGADOCK によるドッキング計算においてはペアを構成する 2 つタンパク質立体構造の扱いが非対称であり可換でないことから，ドッキング計算対象となるタンパク質立体構造ペア数は $6{,}000 \times 6{,}001/2$ ペアではなく $6{,}000^2$ ペアとなる．

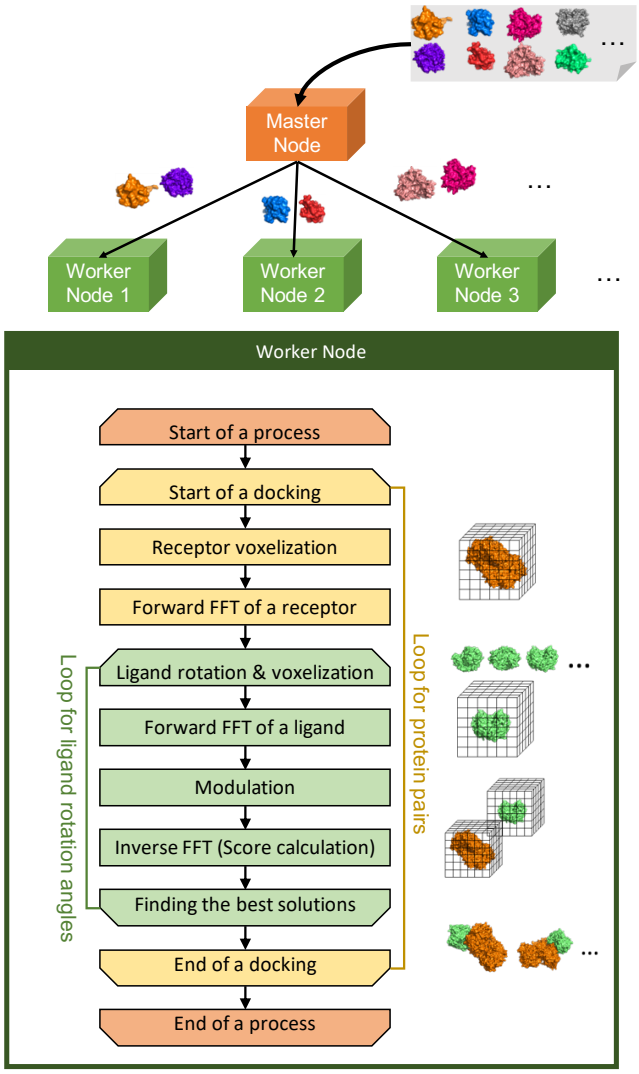

図 1　MEGADOCK の計算フローの概要図

2. スケジューリング手法の改良

2.1 従来手法

MEGADOCK は，MPI を用いたノード間並列計算機能により，数千規模のタンパク質からなる系の網羅的な PPI 予測などの大規模なドッキング計算の高速化を行っている [8]．ノード間並列におけるタスクの単位はタンパク質立体構造ペアであり，各ノードへのタンパク質立体構造ペアの分配スケジューリングにはマスター・ワーカー方式が用いられている（**図 1**）．マスター・ワーカー方式によるタンパク質立体構造ペア分配アルゴリズムを**アルゴリズム 1** に示す．

アルゴリズム 1 に示す，従来の MEGADOCK のタンパク質立体構造ペア分配スケジューリングには，以下の 2 点の特徴がある．

- マスターノードはタンパク質立体構造ペアの分配のみ行い，ドッキング計算を行わないため，計算資源を有効に活用できていない．
- 各ワーカーノードからのタンパク質立体構造ペアの要

© 2020 Information Processing Society of Japan

アルゴリズム 1 マスター・ワーカー方式によるタンパク質立体構造ペア分配アルゴリズム

Require: Q_p: タンパク質立体構造ペアのグローバルキュー,
S_w: ワーカーの集合
Ensure: 最終的に Q_p は空である
if このノードはマスターである **then**
  for each $w \in S_w$ **do**
    if empty(Q_p) **then**
      break
    end if
    タンパク質立体構造ペア $p \leftarrow$ dequeue(Q_p)
    ワーカー w に p を割り当てる
  end for
  while not empty(Q_p) **do**
    ワーカーからのタンパク質立体構造ペアの要求を待つ
    if 要求をワーカー w から受け取る **then**
      $p \leftarrow$ dequeue(Q_p)
      w に p を割り当てる
    end if
  end while
else ▷ このノードはワーカーである
  while まだドッキング計算していないタンパク質立体構造ペア p を持っている **do**
    p についてドッキング計算する
    マスターにタンパク質立体構造ペアを要求する
    if 要求が失敗する **then** ▷ Q_p が空である
      break
    end if
  end while
end if

求が 1 つのマスターノードに集中するため，要求の衝突によりワーカーノードへのタンパク質立体構造ペアの割り当ての待ち時間が発生することがあり，計算効率が低下する可能性がある．特に，ノード数が増加するほど衝突が発生する可能性が高くなる．

これらの特徴により，マルチノード・マルチ GPU 環境下での並列化効率が低下することが懸念されたため，本研究ではタンパク質立体構造ペア分配スケジューリング手法を改善し，網羅的なドッキング計算の効率を向上することを目的とする．

2.2 提案手法

本研究では，MEGADOCK に対して以下の 3 つのタンパク質立体構造ペア分配スケジューリング手法を実装した．なお，a) および b) の手法名は本稿独自のものである．

a) 固定チャンク方式

マスター・ワーカー方式に以下の変更を行う．また，概要図を**図 2** に示す．

- マスターノードで複数の OpenMP スレッドを起動する．1 つのスレッドはマススタースレッドとしてタンパク質立体構造ペア分配機能を果たし，その他のスレッドはワーカースレッドとしてドッキング計算を行う．
- 従来の MEGADOCK はマスターノードが 1 回のタンパク質立体構造ペア要求あたり 1 ペアずつワーカーノードに割り当てを行っていたが，1 回の要求あたり k ペアずつ割り当てることとする．本稿では k のことをチャンクサイズと呼ぶこととする．チャンクサイズは実行時にコマンドライン引数から指定可能とする．なお，チャンクサイズを 1 に指定すると，従来の MEGADOCK で用いられているマスター・ワーカー方式との差異は，マスターノードがドッキングを行う点のみとなる．

従来手法は 1 タンパク質立体構造ペアのドッキング計算を終了するごとにマスターノード・ワーカーノード間に通信が発生していたが，a) 固定チャンク方式により，$k > 1$ の場合においてノード間通信頻度を低下させ，衝突が発生する可能性が低下することが期待される．

b) 動的チャンク方式

a) 固定チャンク方式を基本とし，チャンクサイズをプログラム実行時に動的に決めるように変更する．マスターノードがワーカーノードからのタンパク質立体構造ペア要求を受信した時点でグローバルキューに残っているタンパク質立体構造ペア数を N_G，同時使用するノード数を P として，チャンクサイズ k を式 (1) のように定める．k はプログラム実行中に動的に変化する値である．

$$k = \max\left\{1, \left\lfloor \frac{N_G}{P} \right\rfloor\right\} \tag{1}$$

a) 固定チャンク方式では，チャンクサイズを大きい値に設定した場合，プログラムの実行が進行してマスターノードがもつタンパク質立体構造ペア数が少数となったときに，タンパク質立体構造ペアのワーカーへの割り当てが不均等になり負荷分散効率が低下する可能性がある．そのため，b) 動的チャンク方式は，プログラム実行初期はチャンクサイズを大きめにすることでノード間通信頻度を低下させ，プログラムの実行が進行するとともにチャンクサイズを小さくすることで，負荷分散効率の向上が期待できる．

c) ワーク・スティーリング [10] 方式

タンパク質立体構造ペア分配にマスター・ワーカーアルゴリズムではなくワーク・スティーリングアルゴリズムを用いる．ワーク・スティーリングアルゴリズムは動的なタスクスケジューリングのためのアルゴリズムの一つで，計算機による電力システム分析の分野ではスケジューリング手法にワーク・スティーリングアルゴリズムを用いることでマスター・ワーカーアルゴリズムを用いる場合よりも高い並列実行性能を示したことが報告されている [11].

ここで，c) ワーク・スティーリング方式について説明する．ワーク・スティーリングアルゴリズムは，並列計算に

© 2020 Information Processing Society of Japan

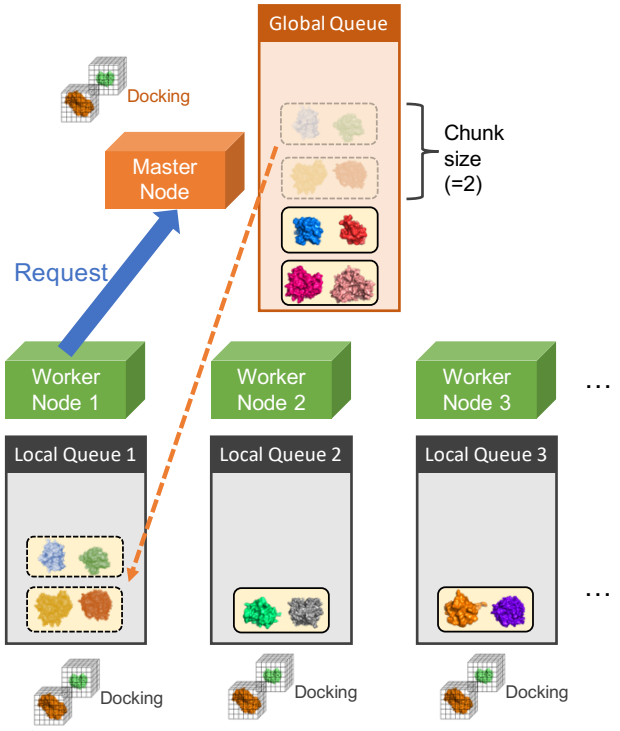

図 2 a) 固定チャンク方式の概要図（図はチャンクサイズ 2 の場合）

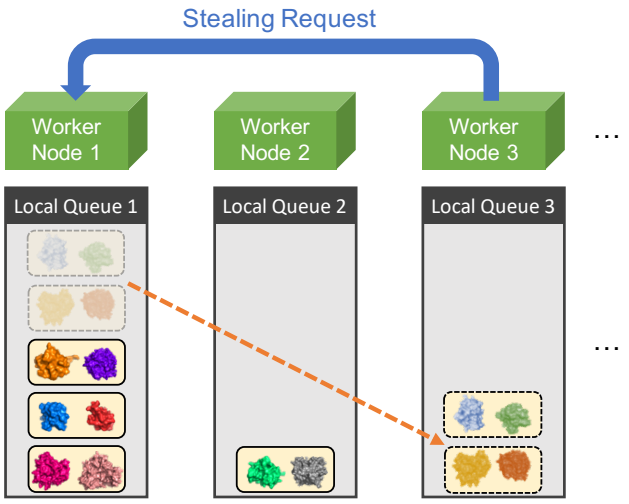

図 3 ワーク・スティーリングアルゴリズムによるタンパク質立体構造ペア分配の概要図

おける動的タスク割り当てのためのアルゴリズムである．ワーク・スティーリングアルゴリズムでは全てのノードがワーカーとして機能し，タスクを処理する．未処理のタスクをもたないノードが未処理のタスクをもつノードからタスクを奪うことをスティーリングといい，スティーリングによりノード間の効率的な負荷分散が可能となる．ワーク・スティーリングによるタンパク質立体構造ペア分配の概要図を**図 3** に示す．

ワーク・スティーリングを用いたタンパク質立体構造ペア分配アルゴリズムを**アルゴリズム 2** に示す．ただし，各ノードには $0,\ldots,N-1$ のノード番号が付されているものとする．また，各ノードがもつタンパク質立体構造ペアのキューをローカルキューと呼ぶこととする．

アルゴリズム 2 c) ワーク・スティーリング方式によるタンパク質立体構造ペア分配アルゴリズム

Require: Q_p: タンパク質立体構造ペアのグローバルキュー，S_w: ノードの集合，$Q_p^{my\_id}$: タンパク質立体構造ペアのローカルキュー，S_w: ノードの集合
Ensure: 最終的に Q_p は空である

procedure EXECUTE_DOCKINGS(my_id)
  while not empty($Q_p^{my\_id}$) **do**
    while not empty($Q_p^{my\_id}$) **do**
      タンパク質立体構造ペア $p \leftarrow$ dequeue($Q_p^{my\_id}$)
      p についてドッキング計算する
    end while
    while 1 つ以上のノードがドッキング計算していないタンパク質立体構造ペアをもつ **do**
      w を $S_w \backslash \{w_{my\_id}\}$ から無作為に選択する
      w からの スティーリングを試みる
      if スティーリングに成功し $p_0,\ldots,p_{M-1}$ を得る **then**
        for each $p \in \{p_0,\ldots,p_{M-1}\}$ **do**
          enqueue($Q_p^{my\_id}, p$)
        end for
        break
      end if
    end while
  end while
end procedure

procedure WAIT_REQUESTS(my_id)
  while *True* **do**
    if w' からのスティーリング要求を受信する **then**
      if empty($Q_p^{my\_id}$) **then**
        NULL を w' に送信する
      else
        $M \leftarrow \lfloor$length($Q_p^{my\_id}$)$/2 \rfloor$
        $Q_p^{my\_id}$ から $p_0,\ldots,p_{M-1}$ を取り出す
        $p_0,\ldots,p_{M-1}$ を w' に送信する
      end if
    end if
  end while
end procedure

procedure MAIN
  $my\_id \leftarrow$ get_my_id()　▷ ノード番号を取得する
  if $my\_id = 0$ **then**　▷ 初期化
    Q_p の要素を全て取り出し $Q_p^0,\ldots,Q_p^{N-1}$ に N 等分する
    for each $i \in \{0,\ldots,N-1\}$ **do**
      ノード i に Q_p^i を割り当てる
    end for
  end if
  spawn EXECUTE_DOCKINGS(my_id) ▷ ドッキング計算を行うスレッド
  spawn WAIT_REQUESTS(my_id)　▷ スティーリング要求待ちを行うスレッド
  全てのスレッドが終了するまで待つ
  ノード間の同期をとる
end procedure

© 2020 Information Processing Society of Japan

アルゴリズム 2 では 1 つのノード内でドッキング計算を行うスレッドと，他のノードからのスティーリング要求を待つスレッドを立ち上げる（それぞれアルゴリズム 2 の 42, 43 行に相当する）．

c) のアルゴリズムにおける終了検知について述べる．c) のアルゴリズムでは，ノードの集合をノード k がノード $2k+1, 2k+2$ を子ノードとしてもつ二分木とみなす．各ノードは初期状態が 0 のカウンターをもち，ドッキング計算を行うごとにカウンターの値をインクリメントする．各ノードはローカルキューが空になるごとに二分木の親に相当するノードに対してカウンターの値を送信し，カウンターの値を 0 にする．子ノードからカウンターの値を受け取ったノードは，自らがもつカウンターに，受け取った値を加算する．以上を繰り返し，最終的にノード 0 のカウンターの値が初期状態のグローバルキューの長さと一致したときに終了を検知する．

3. 実験 1 : TSUBAME 3.0 での大規模 PPI 予測計算

本章では，TSUBAME グランドチャレンジ大規模計算制度を利用した，新たに提案したタンパク質立体構造ペア分配スケジューリング手法の評価について述べる．なお，TSUBAME グランドチャレンジ大規模計算制度は，東京工業大学が大規模計算の研究課題を公募し，TSUBAME 3.0 の一部また全てのノードが占有可能な利用環境を提供する制度である．本章で説明する実験は 2019 年秋期の課題（カテゴリ B，180 ノード）として採択されたもので，同年 10 月中旬に実施した．

3.1 実験方法

TSUBAME 3.0 グランドチャレンジでは MEGADOCK でのドッキング計算を以下の条件で行い，並列化効率を評価した．

- データセットは，以下で定義する，211,600 ペアのタンパク質立体構造ペアからなるベンチマークセット A（大規模）を用いた．

ベンチマークセット A（大規模）

ZLab Benchmark 5.0 [12] に含まれる 230 タンパク質立体構造ペアをレセプターとリガンドに分け，230 個のレセプターと 230 個のリガンドから作り得るレセプター・リガンドのペアを 4 倍複製した，合計 $230^2 \times 4 = 211{,}600$ タンパク質立体構造ペアからなるデータセット

- TSUBAME 3.0 の計算ノードの構成を**表 1** に示す．
- MEGADOCK の実行に用いたソフトウェアの構成を**表 2** に示す．
- MEGADOCK は 1 ドッキング計算ごとに約 500 KB の結果ファイルを出力するが，ファイル出力先は TSUBAME 3.0 のローカルスクラッチ領域を指定した．
- f_node のノード数が 4, 8, 16, 32, 64, 128 の場合について実験を行った．
- OpenMP スレッドの最大並列数は f_node 1 ノードあたり 20 スレッドとした．

表 1 TSUBAME 3.0 の計算ノードの構成（f_node ノード）

項目	説明	個数
CPU	Intel Xeon E5-2680 v4, 2.4 [GHz]	×2
GPU	NVIDIA Tesla P100 for NVLink	×4
Memory	256 [GB]	
SSD	NVMe SSD, 2,0 [TB]	×1
Interconnects	Intel Omni-Path HFI, 100 [Gbps]	×4

表 2 TSUBAME 3.0 において MEGADOCK の実行に用いたソフトウェアの構成

項目	説明
OS	SUSE Linux Enterprise Server 12 SP2
Linux カーネル	4.4.121
CUDA	cuda/10.0.130
OpenMPI	openmpi/2.1.2
GCC	4.8.5

3.2 実験結果

はじめに，スケジューリング方式に b) 動的チャンク方式を用いた場合の実行速度を**図 4** に示す．ただし，測定結果は 3 回の実行結果の中央値とし，図 4 のエラーバーは 3 回の実行結果の標準偏差を表す．ここで，N ノードでの実行時間を T_N とし，以下の強スケーリング S_N を導入する．導入した S_N を用いて，N_{base} ノードを基準としたときに N ノードで強スケーリング S_N%であるという．

$$S_N = \frac{T_{N_{\text{base}}} N_{\text{base}}}{T_N N} \times 100 \qquad (2)$$

表 3 および図 4 から，ベンチマークセット A（大規模）の 211,600 件のドッキング計算をした場合，f_node 4 ノードを基準としたときに f_node 128 ノードで強スケーリング 98.5%を達成したことが分かる．

表 3 TSUBAME 3.0 でベンチマークセット A（大規模）のドッキング計算を行った際の強スケーリングの値（基準 : f_node 4 ノード）

ノード数 N	実行時間 T_N [sec.]	S_N
4	54,202	100
8	27,110	100
16	13,561	99.9
32	6,796	99.8
64	3,409	99.4
128	1,719	98.5

© 2020 Information Processing Society of Japan

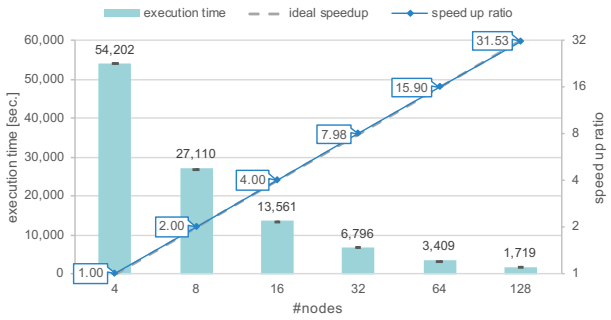

図 4　TSUBAME 3.0 グランドチャレンジで f_node 128 ノード，ベンチマークセット A（大規模）を用いた際の MEGADOCK の並列化効率評価実験結果

次に，新たに提案した 3 種類のタンパク質立体構造ペア分配スケジューリング方式，a) 固定チャンク方式，b) 動的チャンク方式，c) ワーク・スティーリング方式の並列化効率比較を行った．また，a) 固定チャンク方式については，チャンクサイズが $k = 1, 20$ の 2 通りについて実験を行った．なお，a) 固定チャンク方式でチャンクサイズが 1 の場合，従来の MEGADOCK が採用しているマスター・ワーカー方式との差異が，マスターノードがドッキング計算を行う点のみとなることに注意する．

結果を**表 4** に示す．ただし，測定結果は 3 回の実行結果の中央値とした．また，表 4 中では，WS はワーク・スティーリング，k は固定チャンク方式のチャンクサイズを表す．表 4 より，a) 固定チャンク方式，b) 動的チャンク方式，c) ワーク・スティーリング方式の間に並列化効率の差はないことが分かる．

表 4　タンパク質立体構造ペア分配スケジューリング方式間の並列化効率比較結果（数値は秒数を表す，また，WS はワーク・スティーリング方式，k は固定チャンク方式のチャンクサイズを表す）

ノード数	a) 固定チャンク		b) 動的チャンク	c) WS
	$k = 1$	$k = 20$		
4	54,198	54,204	54,202	54,221
8	27,107	27,114	27,110	27,131
16	13,566	13,573	13,561	13,576
32	6,791	6,799	6,796	6,805
64	3,408	3,417	3,409	3,422
128	1,721	1,722	1,719	1,755

4. 実験 2：ABCI での超大規模 PPI 予測計算

本節では，産総研「ABCI グランドチャレンジ」プログラムを利用した，新たに提案したタンパク質立体構造ペア分配スケジューリング手法の評価について述べる．なお，産総研「ABCI グランドチャレンジ」プログラムは，産業技術総合研究所が大規模計算の研究課題を公募し，AI 橋渡しクラウド (AI Bridging Cloud Infrastructure, ABCI) の一部また全てのノードが占有可能な利用環境を提供する制度である．本節で説明する実験は ABCI グランドチャレンジ 2019 第 2 回の課題として medium クラス（512 ノード使用）に採択されたもので，2019 年 9 月下旬に実施した．

4.1 実験方法

ABCI グランドチャレンジでは MEGADOCK でのドッキング計算を以下の条件で行い，並列化効率を評価した．

- データセットは，以下で定義する，1,322,500 ペアのタンパク質立体構造ペアからなるベンチマークセット B（超大規模）を用いた．

ベンチマークセット B（超大規模）

ZLab Benchmark 5.0 [12] に含まれる 230 タンパク質立体構造ペアをレセプターとリガンドに分け，230 個のレセプターと 230 個のリガンドから作り得るレセプター・リガンドのペアを 25 倍複製した，合計 $230^2 \times 25 = 1{,}322{,}500$ タンパク質立体構造ペアからなるデータセット

- ABCI の計算ノードの構成を**表 5** に示す．
- MEGADOCK の実行に用いたソフトウェアの構成を**表 6** に示す．
- MEGADOCK は 1 ドッキング計算ごとに約 500KB の結果ファイルを出力するが，ファイル出力先は ABCI のローカルスクラッチ領域を指定した．
- rt_F のノード数が 16, 32, 64, 128, 256, 512 の場合について実験を行った．
- OpenMP スレッドの最大並列数は rt_F 1 ノードあたり 20 スレッドとした．

表 5　ABCI の計算ノードの構成（rt_F ノード）

項目	説明	個数
CPU	Intel Xeon Gold 6148, 2.4 [GHz]	×2
GPU	NVIDIA Tesla V100 for NVLink	×4
Memory	384 [GB]	
SSD	NVMe SSD, 1,6 [TB]	×1
Interconnects	InfiniBand EDR, 100 [Gbps]	×2

表 6　ABCI において MEGADOCK の実行に用いたソフトウェアの構成

項目	説明
OS	CentOS 7.5.1804
Linux カーネル	3.10.0
CUDA	cuda/10.0.130
OpenMPI	openmpi/2.1.6
GCC	4.8.5

© 2020 Information Processing Society of Japan

4.2 実験結果

スケジューリング方式に b) 動的チャンク方式を用いた場合の実行速度を図 **5** に示す．なお，ABCI グランドチャレンジでは，ノード占有時間の都合上，b) 動的チャンク方式のみで実験を行った．ただし，測定結果は 3 回の実行結果の中央値とし，図 5 のエラーバーは 3 回の実行結果の標準偏差を表す．**表 7** および図 5 から，ベンチマークセット B（超大規模）の 1,322,500 件のドッキング計算をした場合，rt_F 16 ノードを基準としたときに rt_F 512 ノードで強スケーリング 96.7%を達成したことが分かる．

表 7 ABCI でベンチマークセット B（超大規模）のドッキング計算を行った際の強スケーリングの値（基準：rt_F 16 ノード）

ノード数 N	実行時間 T_N [sec.]	S_N
16	51,113	100
32	24,469	100
64	12,795	99.9
128	6,415	99.6
256	3,224	98.5
512	1,652	96.7

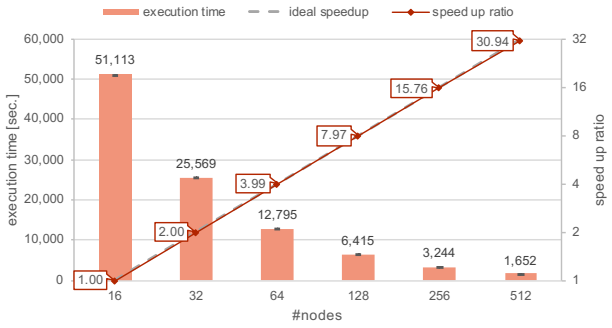

図 5 ABCI グランドチャレンジで rt_F 512 ノード，ベンチマークセット B（超大規模）を用いた際の MEGADOCK の並列化効率評価実験結果

5. 考察

ベンチマークセット A（大規模）の 211,600 件のドッキング計算に対して TSUBAME 3.0 f_node 128 ノードを用いた場合，タンパク質立体構造ペア分配スケジューリングに b) 動的チャンク方式を用いると強スケーリング 98.5%（対 f_node 4 ノード）を達成することができた．また，新たに提案した 3 種類のタンパク質立体構造ペア分配スケジューリング方式，a) 固定チャンク方式，b) 動的チャンク方式，c) ワーク・スティーリング方式の間に並列化効率の差が見られないという結果を得た（**表 8**）．

また，ベンチマークセット B（超大規模）の 1,322,500 件のドッキング計算に対して ABCI rt_F 512 ノードを用いた場合，タンパク質立体構造ペア分配スケジューリングに b) 動的チャンク方式を用いると強スケーリング 96.7%（対 rt_F 16 ノード）を達成することができた．さらに，同条件でタンパク質ペア分配スケジューリング手法を a) 固定チャンク方式に変更し，チャンクサイズを 1 に設定して，1 回のみ実行時間計測を行った結果，1,265 秒という結果を得た．この結果は 1 点計測の結果であり，単純に b) 動的チャンク方式の実行時間と比較できるものではないものの，TSUBAME 3.0 での並列化効率評価実験も考慮すると，a) 固定チャンク方式且つチャンクサイズが 1 の場合においても ABCI 512 ノードの規模で高い並列化効率を得られることが予想される．なお，ABCI では c) ワーク・スティーリング方式を用いた実験が行えていないが，a) 固定チャンク方式（チャンクサイズ：1）および b) 動的チャンク方式の ABCI での並列化効率評価結果から，c) ワーク・スティーリング方式でも高い並列化効率を得られると考えられる．さらに，a) 固定チャンク方式でチャンクサイズが 1 の場合，従来の MEGADOCK が採用しているマスター・ワーカー方式との差異が，マスターノードがドッキング計算を行う点のみとなることから，本研究で用いた計算機環境においては，従来の MEGADOCK が採用しているマスター・ワーカー方式を用いても高い並列化効率を得られることも類推される．

表 8 TSUBAME 3.0 f_node 128 ノード使用時のタンパク質立体構造ペア分配スケジューリング方式間の並列化効率比較結果

スケジューリング方式	実行時間 [sec.]
a) 固定チャンク方式（チャンクサイズ：1）	1,721
b) 動的チャンク方式	1,719
c) ワーク・スティーリング方式	1,755

これらのことから，ABCI 512 ノードの規模（ABCI の全ノード数は 1,088 であるため，512 ノードは ABCI の約半系である）においても，タンパク質立体構造ペア分配スケジューリング方式間で並列化効率に差は生じないと考えられる．特に，ノード間通信頻度の高い，チャンクサイズを 1 としたときの a) 固定チャンク方式でも b) 動的チャンク方式と同程度の実行性能であるということから，チャンクサイズを 1 としたときの a) 固定チャンク方式，さらには従来手法のマスター・ワーカー方式でも ABCI 半系程度の規模の計算機システムでも高い並列化効率が得られることが予想される．つまり，現在の MEGADOCK におけるノード間通信の時間的コストは，1 タンパク質立体構造ペアのドッキング計算速度と比較して十分小さいと考えられる．

ただ，より大規模な計算機システムでドッキング計算をする場合や，ドッキング計算がより高速化された場合には，1 タンパク質立体構造ペアのドッキング計算速度に対してノード間通信コストが無視できなくなることは考えられる．その場合には，タンパク質立体構造ペア分配アルゴリズムの違いによる並列実行性能の差が生じる可能性がある．

© 2020 Information Processing Society of Japan

6. 結論

6.1 本研究の結論

本研究では，従来の MEGADOCK のマルチノード環境下における，各ノードへのタンパク質立体構造ペア分配に関するスケジューリング手法を新たに以下の 3 種類提案した．

- a) 固定チャンク方式
- b) 動的チャンク方式
- c) ワーク・スティーリング方式

東工大 TSUBAME 3.0 を用いた実験の結果，3 種類の提案手法による並列化効率の差は生じなかった．

一方，新たに提案した手法のうち，b) 動的チャンク方式を用いた PPI 予測計算実験では，東工大 TSUBAME 3.0 128 ノードでの 211,600 件の大規模な PPI 予測計算で強スケーリング 98.5%，および産総研 AI 橋渡しクラウド (ABCI) 512 ノードでの 1,322,500 件の超大規模な PPI 予測計算で強スケーリング 96.7%と，どちらの環境においても強スケーリング 95%を達成した．

6.2 今後の課題

本研究では，異なるタンパク質立体構造ペア分配スケジューリング方式の間に並列化効率の差が見られなかったが，その原因に MEGADOCK のドッキング計算においてノード間通信の時間的コストが無視できるほど小さいことが考えられる．そのため，将来的に MEGADOCK のドッキング計算がさらに高速化された状態，あるいは本研究で用いた計算機環境よりも大規模な計算機環境において，異なるタンパク質立体構造ペア分配スケジューリング方式間でによる並列化効率比較を行い，スケジューリング方式の違いにより並列化効率に差が生じるかを検証する必要がある．

謝辞 本研究は，東工大の TSUBAME グランドチャレンジ大規模計算制度（2019 年秋期の課題として採択，同年 10 月中旬実施）により提供を受けた TSUBAME 3.0 の計算リソース，および産総研「ABCI グランドチャレンジ」プログラム（2019 年 第 2 回の課題として採択，同年 9 月下旬実施）により提供を受けた AI 橋渡しクラウド (ABCI) の計算リソースを用いて行われた．

参考文献

[1] U. Stelzl *et al.* A human protein-protein interaction network: A resource for annotating the proteome. *Cell*, Vol. 122, No. 6, pp. 957–968, 2005.
[2] T. Oltersdorf *et al.* An inhibitor of bcl-2 family proteins induces regression of solid tumours. *Nature*, Vol. 435, No. 7042, pp. 677–681, 2005.
[3] G. M. Popowicz *et al.* Structures of low molecular weight inhibitors bound to mdmx and mdm2 reveal new approaches for p53-mdmx/mdm2 antagonist drug discovery. *Cell Cycle*, Vol. 9, No. 6, pp. 1104–1111, 2010.
[4] A. Bateman *et al.* UniProt: The universal protein knowledgebase. *Nucleic Acids Research*, Vol. 45, No. D1, pp. D158–D169, 2017.
[5] R. Chen, L. Li, Z. Weng. ZDOCK: An initial-stage protein-docking algorithm. *Proteins*, Vol. 52, No. 1, pp. 80–87, 2003.
[6] Y. Murakami, K. Mizuguchi. Homology-based prediction of interactions between proteins using averaged one-dependence estimators. *BMC Bioinformatics*, Vol. 15, No. 213, 2014.
[7] H. Zhou, E. Jakobsson. Predicting protein-protein interaction by the mirrortree method: Possibilities and limitations. *PLOS ONE*, Vol. 8, No. 12, 2013.
[8] M. Ohue *et al.* MEGADOCK 4.0: An ultra-high-performance protein-protein docking software for heterogeneous supercomputers. *Bioinformatics*, Vol. 30, No. 22, pp. 3281–3283, 2014.
[9] H. M. Berman *et al.* The Protein Data Bank. *Nucleic Acids Research*, Vol. 28, No. 1, pp. 235–242, 2000.
[10] R. D. Blumofe, C. E. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 356–368, 1994.
[11] S. K. Khaitan, J. D. McCalley. Scale: A hybrid mpi and multithreading based work stealing approach for massive contingency analysis in power systems. *Electric Power Systems Research*, Vol. 114, pp. 118–125, 2014.
[12] T. Vreven *et al.* Updates to the integrated protein–protein interaction benchmarks: Docking benchmark version 5 and affinity benchmark version 2. *Journal of Molecular Biology*, Vol. 427, No. 19, pp. 3031–3041, 2015.

© 2020 Information Processing Society of Japan