Vol.2022-HPC-187 No.11
2022/12/2

x-means法のハイブリッドMPI-OpenMP並列による性能評価

定方 翼[1,a)]　沼波 政倫[2,3]　片桐 孝洋[4]　大島 聡史[4]　永井 亨[4]

概要：本稿では，クラスタリング手法の 1 つである x-means 法を MPI と OpenMP を利用してハイブリッド並列化し，その性能評価について報告する．x-means 法は，k-means 法をベイズ情報量規準に従って妥当な回数繰り返すことで，クラスタリング対象のデータセットに対して適切なクラスタ数を推定するアルゴリズムである．そのため，k-means 法を複数回計算する必要があり，データセットのサイズや次元数，収束までの分割数に応じて実行時間が増加する．そこで，本研究では x-means 法をハイブリッド並列化し，OpenMP ディレクティブの挿入箇所，通信方式，ノード数，プロセス数，スレッド数を変化させた時の実行時間や並列化効率について評価を行った．その結果，1 ノード，1 プロセス，48 スレッドのハイブリッド実行に対して，128 ノード，1 ノードあたり 24 プロセス，2 スレッドのハイブリッド実行は約 170 倍高速であった．また，32 ノード以下のときはピュア MPI 実行が高速であり，64 ノード以上ではハイブリッド MPI-OpenMP 実行が高速になることを確認した．

Performance Evaluation of the x-means Clustering with Hybrid MPI-OpenMP Parallelization

Abstract: In this paper, we propose a hybrid parallelization of the x-means, one of the clustering methods, using MPI and OpenMP. In addition, its performance evaluation is reported. x-means clustering is an algorithm that estimates the appropriate number of clusters for a dataset to be clustered by repeating the k-means clustering a suitable number of times according to Bayesian information criterion. Therefore, we evaluated the execution time and parallelization efficiency by changing the insertion point of OpenMP directives, communication method, number of nodes, number of processes, and number of threads, using a hybrid parallelization of the x-means clustering. As a result, the hybrid execution with 128 nodes, 24 processes and 2 threads per node was about 170 times faster than the hybrid execution with 1 node, 1 process and 48 threads per node. We also confirmed that pure MPI execution is faster when the number of nodes is 32 or less, and hybrid MPI-OpenMP execution is faster when the number of nodes is 64 or more.

1. はじめに

近年，センシング技術，ソーシャル・ネットワーキング・サービス，シミュレーションなどの進歩にともない，大規模かつ高次元の未解析データが蓄積されており，これらのデータを自動的に理解し，要約するための手法が求められている．クラスタリングは，ラベル付けされていないデータセットからクラスタと呼ばれる自然なグループ分けを発見する教師なし機械学習の 1 種である．数あるクラスタリング手法の中で k-means 法 [1], [2], [3] は，最初の提案から長期間経過しているにもかかわらず，単純さや効率，経験的な成功から，現在最も広く使用されているクラスタリング手法の 1 つである [4]．しかし，k-means 法はクラスタ数を表す k を使用者が事前に指定する必要がある．そこで，Pelleg and Moore[5] によって k-means 法を拡張し，クラスタ数を自動的に推定する x-means 法が提案されている．x-means 法はデータセットを 2 分割する k-means アルゴリズムを情報量規準に沿って妥当な回数繰り返す手法である．k-means 法は貪欲法であるため，得られる解は

[1] 名古屋大学 大学院情報学研究科
Graduate School of Informatics, Nagoya University
[2] 核融合科学研究所
National Institute for Fusion Science
[3] 名古屋大学 大学院理学研究科
Graduate School of Science, Nagoya University
[4] 名古屋大学 情報基盤センター
Information Technology Center, Nagoya University
[a)] sadakata@hpc.itc.nagoya-u.ac.jp

局所解であり，初期値を変更して複数回実行される．よって，x-means 法は k-means 法を複数回繰り返すため，大規模データでは長時間を要し，高速化が求められる．そこで我々は，x-means 法を MPI と OpenMP を用いたハイブリッド並列化し，ノード数，プロセス数，スレッド数を変化させて x-means 法を適用することでその性能評価を行った．

本稿の構成は以下である．まず 2 章で k-means 法と x-means 法のアルゴリズムについて紹介する．次に 3 章で x-means のハイブリッド並列化実装方法について説明し，4 章で並列化による性能評価を述べる．5 章では，関連研究を紹介し，最後に 6 章で本研究のまとめを述べる．

2. アルゴリズム

2.1 k-means

k-means アルゴリズムを以下で説明する．ここで，クラスタ数を k，i 番目のクラスタを C_i，C_i に属するデータ数を N_i，データを x とする．

(i) データセットとクラスタ数 k，アルゴリズムの終了条件であるしきい値を設定する．
(ii) データセットの中から，ランダムに k 個のデータを初期重心として選択する．
(iii) 式 (1) で定義するように，重心と各データの距離を計算し，その総和を J とする．

$$J = \sum_{i=1}^{N} \sum_{j=1}^{k} \|x_i - \mu_j\|^2 \tag{1}$$

(iv) 各データに対し，最も近い重心をもつクラスタにデータを割り当てる．
(v) 式 (2) に定義する重心を計算する．

$$\mu_k = \frac{1}{N_k} \sum_{x_n \in C_k} x_n \tag{2}$$

(vi) 手順 (iii) から (v) を J の変化量があらかじめ設定したしきい値以下となるまで繰り返す．

2.2 x-means

本研究では，石岡 [6] によって提案されたベイズ情報量規準（BIC）の計算に近似を適用し，計算速度を向上させた x-means 法を採用した．以下では，文献 [6] を参考に，x-means アルゴリズムについて説明する．

(i) クラスタリング対象のデータセットを n 個の p 次元データとする．
(ii) クラスタ数の初期値 k_0 を与える．
(iii) $k = k_0$ として，k-means 法を適用する．分割後のクラスタを $C_1, C_2, ..., C_{k_0}$ とする．
(iv) $i = 1, 2, ..., k_0$ として，以下の手順 (v) から (viii) を繰り返す．
(v) クラスタ C_i に対して $k = 2$ として k-means 法を適用し，分割後のクラスタを C_i^1, C_i^2 とする．
(vi) C_i に含まれるデータ x_i に p 変量正規分布

$$f(\theta_i; x) = \frac{1}{\sqrt{(2\pi)^k |V_i|}} \exp\left(-\frac{1}{2}(x-\mu_i)^t V_i^{-1}(x-\mu_i)\right) \tag{3}$$

を仮定し，そのときの BIC を以下のように定義する．

$$\mathrm{BIC} = -2\log L(\hat{\theta}_i; x_i \in C_i) + q\log(n_i) \tag{4}$$

ここで，$\hat{\theta}_i = [\hat{\mu}_i, \hat{V}_i]$ は，p 変量正規分布の最尤推定値とする．また，μ_i は p 次元の平均値ベクトル，V_i は $p \times p$ の分散共分散行列である．q は独立なパラメータ空間の次元数で，$q = p(p+3)/2$ である．x_i はクラスタ C_i に含まれている p 次元のデータとし，n_i は C_i に含まれるデータ数とする．L は尤度関数 $L(\cdot) = \prod f(\cdot)$ である．
(vii) C_i^1, C_i^2 のそれぞれに対して，パラメータ θ_i^1, θ_i^2 をもつ p 変量正規分布を仮定し，2 分割モデルにおいてデータが従う確率密度とする．あるデータがどちらのパラメータを用いるかは属しているクラスタによって決まる．

$$x_i = \begin{cases} \alpha_i[f(\theta_i^1); x], & x_i \in C_i^1 \\ \alpha_i[f(\theta_i^2); x], & x_i \in C_i^2 \end{cases} \tag{5}$$

α_i は上式を確率密度とするための基準化定数で

$$\alpha_i = \begin{cases} 1/\int[f(\theta_i^1; x_i)]dx, & x_i \in C_i^1 \\ 1/\int[f(\theta_i^2; x_i)]dx, & x_i \in C_i^2 \end{cases} \tag{6}$$

となる．厳密に求めると p 次積分が必要となり，多くの計算量が必要となるため，近似値として

$$\alpha_i = 0.5/K(\beta_i) \tag{7}$$

を用いる．$K(\cdot)$ は標準正規分布の下側確率であり，β_i は $f(\theta_i^1; x_i)$ と $f(\theta_i^2; x_i)$ の分離の程度を表す指標で

$$\beta_i = \sqrt{\frac{\|\mu_i^1 - \mu_i^2\|^2}{|V_i^1| + |V_i^2|}} \tag{8}$$

と定義される．これらを用いて 2 分割モデルにおける BIC$'$ を以下のように計算する．

$$\mathrm{BIC}' = -2\log L(\hat{\theta'}_i; x_i \in C_i) + q'\log(n_i) \tag{9}$$

ここで，$\hat{\theta}_i = [\hat{\theta_i^1}, \hat{\theta_i^2}]$ は，2 つの p 変量正規分布の最尤推定値である．各々の p に対して分散と平均のパラメータが存在するので，パラメータ空間の次元数は $q' = 2q = p(p+3)$ となる．
(viii) BIC $>$ BIC$'$ ならば，2 分割したクラスタが好ましいと

判断し，$C_i \leftarrow C_i^1$ とする．C_i^2 をスタックにプッシュし，手順 (v) へ進む．BIC $\leq$ BIC$'$ ならば，分割前クラスタが好ましいと判断し，2 分割したクラスタは破棄する．スタックが空でなければポップし C_i として手順 (v) へ，空なら次の手順へ進む．

(ix) C_i における 2 分割が全て終了する．

(x) はじめに k_0 分割したクラスタ全てについて 2 分割が終了し，全データに対してそれらの属するクラスタ番号が一意になるようにクラスタ番号を振り直す．

3. 並列化

3.1 並列化の概要

x-means 法において計算の大部分を占めるのは，各クラスタの重心とそのクラスタに割り当てられたデータとの距離計算部分である．各プロセスにデータを分散して保持することで，この距離計算部分を MPI によって並列化することができる．r 番目のプロセスが持つデータ番号の先頭 $head$ と末尾 $tail$ はそれぞれ，全データ数を N，プロセス数を P,

$$n = \lfloor N/P \rfloor \tag{10}$$

$$t = N - (n+1)P \tag{11}$$

として，$r < t$ のとき

$$head = nr \tag{12}$$

$$tail = n(r+1) - 1 \tag{13}$$

$r \geq t$ のとき

$$head = nt + (n+1)(r-t) \tag{14}$$

$$tail = nt + (n+1)(r-t+1) - 1 \tag{15}$$

とする．また，距離計算部分は OpenMP を利用してスレッド並列化も可能である．各プロセスで計算した重心をマージし，プロセス全体の重心を計算するために，プロセス間で各クラスタに属するデータ数と重心を転送する必要がある．また，k-means 法によって得られたクラスタの BIC を計算するために必要な分散共分散行列と対数尤度関数の計算も MPI によって並列化することができる．x-means 法の並列化の概要を**図 1** に示す．

3.2 MPI 実装

並列 x-means における重心のマージと分散共分散行列の計算の具体的な実装を説明する．重心のマージは，各クラスタの重心とそのクラスタに割り当てられたデータ数をプロセス間で転送し，全体の重心を計算する．ここで，プロセス間通信は**図 2** に示すように番号が 1 のプロセスから順に番号が 0 のプロセスに転送する逐次通信方式と，番号が隣り合うプロセス間で転送する二分木通信方式の 2 通

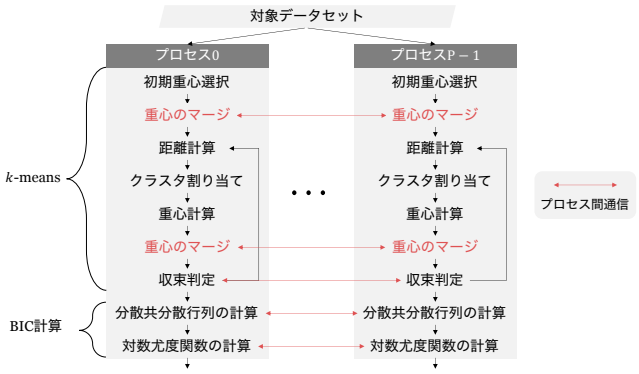

図 1 x-means の並列化の概要

りを検討した．重心のマージの擬似コードを**図 3** に示す．また，クラスタの BIC を求めるために，**図 4** に示すように分散共分散行列をデータ並列で計算する．さらに，対数尤度関数の計算も**図 5** に示すように実装し，データ並列化する．

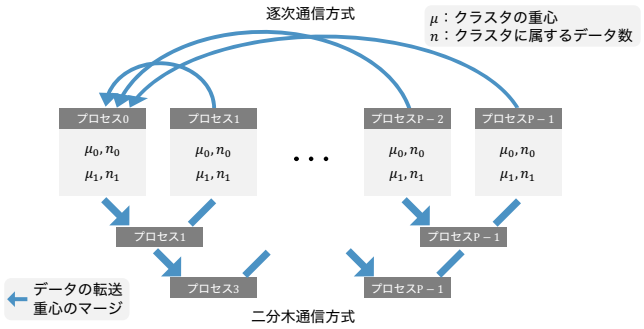

図 2 重心のマージの概要図

```
1  void Centroid::MergeSum(int recv_num, vector<double>
       recv_mu){
2    for(int i=0; i<dim; i++){
3      mu[i] += recv_mu;
4    }
5    num += recv_num;
6  }
7
8  void Centroid::MergeDiv(){
9    for(int i=0; i<dim; i++){
10     mu[i] /= (double)num;
11   }
12 }
```

図 3 重心のマージの擬似コード．転送されたクラスタの重心とデータ数を足し (`MergeSum()`)，すべてのプロセスが足された後，全体の重心を求め (`MergeDiv()`)，全プロセスにブロードキャストする．

3.3 OpenMP 実装

OpenMP のディレクティブを挿入することで，各クラスタの重心と各データ点との距離計算をする関数をスレッド並列化する．ディレクティブの挿入場所は**図 6** に示すように距離計算関数内と，**図 7** に示すように距離計算ループの

```
1  vector<vector<double>> Cluster::cov(int rank, int size)
       {
2
3    vector<vector<double>> cov_(dim, vector<double>(dim,
         0.0));
4
5    for(int i=0; i<dim; i++){
6      for(int j=0; j<dim; j++){
7        for(int k=0; k<Dataset.size(); k++){
8          cov_[i][j] += (Dataset[k][i] - centroid[i]) * (
               Dataset[k][j] - centroid[j]);
9        }
10     }
11   }
12
13   /* Communication */
14
15   return cov_;
16 }
```

図 4 分散共分散行列計算のデータ並列化．7 行目の `for` ループが並列化される．

```
1  double Cluster::multivariate_normal_logpdf(vector<
       vector<double>> cov_){
2      double ll = 0.0;
3      vector<double> flattened = flatten(cov_);
4      double det_cov = 0.0;
5      double *inv_cov = flattened.data();
6      lu_inv_det(&det_cov, inv_cov, 0);
7      int k = centroid.size();
8      vector<double> mean = centroid;
9      vector<double> tmpmean;
10     tmpmean.resize(mean.size());
11     copy(mean.begin(), mean.end(), tmpmean.begin());
12     double *mean_ = tmpmean.data();
13     for(int i=0; i<Dataset.size(); i++){
14         vector<dtype> x = Dataset[i].read();
15         vector<double> tmpx;
16         tmpx.resize(x.size());
17         copy(x.begin(), x.end(), tmpx.begin());
18         double *x_ = tmpx.data();
19         double y[dim];
20         cblas_daxpy(k, -1.0, mean_,  1, x_, 1);
21         cblas_dgemv(CblasRowMajor, CblasNoTrans, dim,
               dim, 1, inv_cov, dim, x_, 1, 0.0, y, 1);
22         double tmp = cblas_ddot(k, x_, 1, y, 1);
23         ll += log(1 / (sqrt(pow(2*M_PI, k) * det_cov))
               * exp(-0.5 * tmp));
24     }
25     return ll;
26 }
27
28 double Cluster::log_likelihood(int rank, int size){
29   double gll = 0.0;
30   vector<vector<double>> gcov = cov(rank, size);
31   double ll = multivariate_normal_logpdf(gcov);
32   MPI_Allreduce(&ll, &gll, 1, MPI_DOUBLE, MPI_SUM,
         MPI_COMM_WORLD);
33   return gll;
34 }
35
36 double Cluster::bic(int rank, int size){
37     int gnum = 0;
38     MPI_Allreduce(&num, &gnum, 1, MPI_INT, MPI_SUM,
           MPI_COMM_WORLD);
39     double bic_ = -2 * log_likelihood(rank, size) + df(
           dim) * log(gnum);
40     return bic_;
41 }
```

図 5 BIC 計算の MPI 実装．13 行目の `for` ループがデータ並列化される．プロセス全体の BIC を計算するために `MPI_Allreduce()` で総和を計算する．

外側の 2 通りを検討する．

```
1  template<typename T, typename U>
2  double Distance(const vector<T> &data, const vector<U>
       &centroid){
3    vector<double> diffs(data.size());
4    #pragma omp parallel for
5    for(int i=0; i<data.size(); i++){
6      diffs[i] = (data[i] - centroid[i])*(data[i] -
           centroid[i]);
7    }
8    return accumulate(diffs.begin(), diffs.end(), 0.0);
9  }
```

図 6 距離計算関数内における OpenMP ディレクティブの挿入

```
1    while(iter < maxiter){
2      /* ... */
3      #pragma omp parallel for
4      for(int i=0; i<ndata; i++){
5        for(int j=0; j<ncluster; j++){
6          dist[i][j] = Distance(Dataset[i], Centroidset[j
               ]);
7        }
8      }
9      /* ... */
10   }
```

図 7 距離計算ループ外側の OpenMP ディレクティブの挿入

4. 性能評価

4.1 問題設定

評価対象として，Python のオープンソース機械学習ライブラリである scikit-learn 0.24.2 [7] に含まれる sklearn.datasets.make_blobs を利用して，クラスタ数 7，データ数 1,000,000 の 2 次元データセットを作成した．

4.2 評価環境

評価環境として，名古屋大学のスーパーコンピュータ「不老」Type I サブシステムを利用した．計算機の構成を**表 1** に示す．

表 1 計算機構成

ハードウェア構成	
CPU	A64FX (Armv8.2-A + SVE)
コア数/ノード数	48 コア+2 アシスタントコア
動作周波数	2.2GHz
理論演算性能	倍精度 3.3792 TFLOPS 単精度 6.7584 TFLOPS
ソフトウェア構成	
開発環境	C++
コンパイラ	mpiFCCpx -Kfast, -Kopenmp, -SSL2

4.3 結果

4.3.1 距離計算ループの内側に OpenMP ディレクティブを挿入したときの結果

距離計算のループの内側に OpenMP ディレクティブを挿入し，プロセス間通信に逐次通信方式を採用した並列 x-means の実行時間と並列化効率を**図 8** に示す．ここで，並列化効率は 1 ノード 1 プロセス 48 スレッド実行に対する速度向上率をプロセス数で割った値である．最も実行に時間を要したのは，1 ノード 1 プロセス 48 スレッドで実行した時で 6070.96 秒かかった．最速は 4 ノード 48 プロセスでピュア MPI 実行したときで，9.68 秒かかった．各ノード数毎の最速の並列実行形態を**表 2** に示す．

表 2 ノード数毎の最速実行形態と実行時間

ノード数	実行形態 (p: プロセス，t: スレッド)	実行時間 [秒]
1	48p ピュア MPI	17.39
2	48p ピュア MPI	11.25
4	48p ピュア MPI	9.68
8	48p ピュア MPI	13.07
16	16p3t ハイブリッド MPI-OpenMP	14.64
32	12p4t ハイブリッド MPI-OpenMP	13.30
64	8p6t ハイブリッド MPI-OpenMP	13.61
128	6p8t ハイブリッド MPI-OpenMP	13.54
256	3p16t ハイブリッド MPI-OpenMP	14.69
512	2p24t ハイブリッド MPI-OpenMP	16.19

4.3.2 距離計算ループの外側に OpenMP ディレクティブを挿入したときの結果

距離計算のループの外側に OpenMP ディレクティブを挿入し，プロセス間通信に逐次通信方式を採用した並列 x-means の実行時間と並列化効率を**図 9** に示す．最も実行に時間を要したのは，1 ノード 1 プロセス 48 スレッドで実行した時で 998.25 秒かかった．最速の実行は OpenMP ディレクティブを距離計算ループの内側に挿入したときと同様に 4 ノード 48 プロセスのピュア MPI 実行であった．各ノード数毎の最速の並列形態を**表 3** に示す．

表 3 ノード数毎の最速実行形態と実行時間

ノード数	実行形態 (p: プロセス，t: スレッド)	実行時間 [秒]
1	48p ピュア MPI	17.39
2	48p ピュア MPI	11.25
4	48p ピュア MPI	9.68
8	24p2t ハイブリッド MPI-OpenMP	11.52
16	16p3t ハイブリッド MPI-OpenMP	10.63
32	12p4t ハイブリッド MPI-OpenMP	10.18
64	6p8t ハイブリッド MPI-OpenMP	10.23
128	4p12t ハイブリッド MPI-OpenMP	10.47
256	2p24t ハイブリッド MPI-OpenMP	10.74
512	1p48t ハイブリッド MPI-OpenMP	11.76

4.3.3 二分木通信方式へ変更したときの結果

各距離計算のループの外側に OpenMP ディレクティブを挿入し，プロセス間通信に二分木通信方式に変更した時の並列 x-means の実行時間と並列化効率を**図 10** に示す．最も実行に時間を要したのは，これまでと同様 1 ノード 1 プロセス 48 スレッドで実行した時であった．最速の実行は 128 ノード 24 プロセス 2 スレッドのハイブリッド実行で，5.85 秒を要した．各ノード数毎の最速の並列形態を**表 4** に示す．

表 4 ノード数毎の最速実行形態と実行時間

ノード数	実行形態 (p: プロセス，t: スレッド)	実行時間 [秒]
1	48p ピュア MPI	17.56
2	48p ピュア MPI	11.32
4	48p ピュア MPI	8.27
8	48p ピュア MPI	6.83
16	48p ピュア MPI	6.26
32	48p ピュア MPI	6.14
64	24p2t ハイブリッド MPI-OpenMP	5.87
128	24p2t ハイブリッド MPI-OpenMP	5.85
256	16p3t ハイブリッド MPI-OpenMP	6.14
512	12p4t ハイブリッド MPI-OpenMP	6.57

4.4 考察

OpenMP ディレクティブは，距離関数ループの内側に挿入すると，呼び出し回数が増加し，特に高スレッド実行時にスレッド並列化のオーバーヘッドが大きくなる．そのため，**表 5** に示すように距離関数ループの外側に OpenMP ディレクティブを挿入した時と比較し，実行時間が増加すると考えられる．よって表 5 に示した条件の時，最高速の実行時間は OpenMP ディレクティブを距離計算ループ外に入れる並列化方式であった．また，距離計算ループ内に OpenMP ディレクティブを入れる場合は 6 プロセス 8 スレッドが最速で 13.54 秒であったが，距離計算ループ外に入れる場合は 4 プロセス 12 スレッドの 10.47 秒が最速となるため，1.29 倍の高速化が実現できた．

表 5 128 ノード実行の OpenMP ディレクティブ挿入箇所の変更による実行時間 [秒] の比較

実行形態	距離計算ループ内	距離計算ループ外
1p48t	57.58	14.42
2p24t	22.95	11.03
3p16t	17.62	10.48
4p12t	13.98	10.47
6p8t	13.54	11.58
8p6t	14.99	13.63
12p4t	20.54	19.36
16p3t	27.62	28.65
24p2t	48.23	52.76

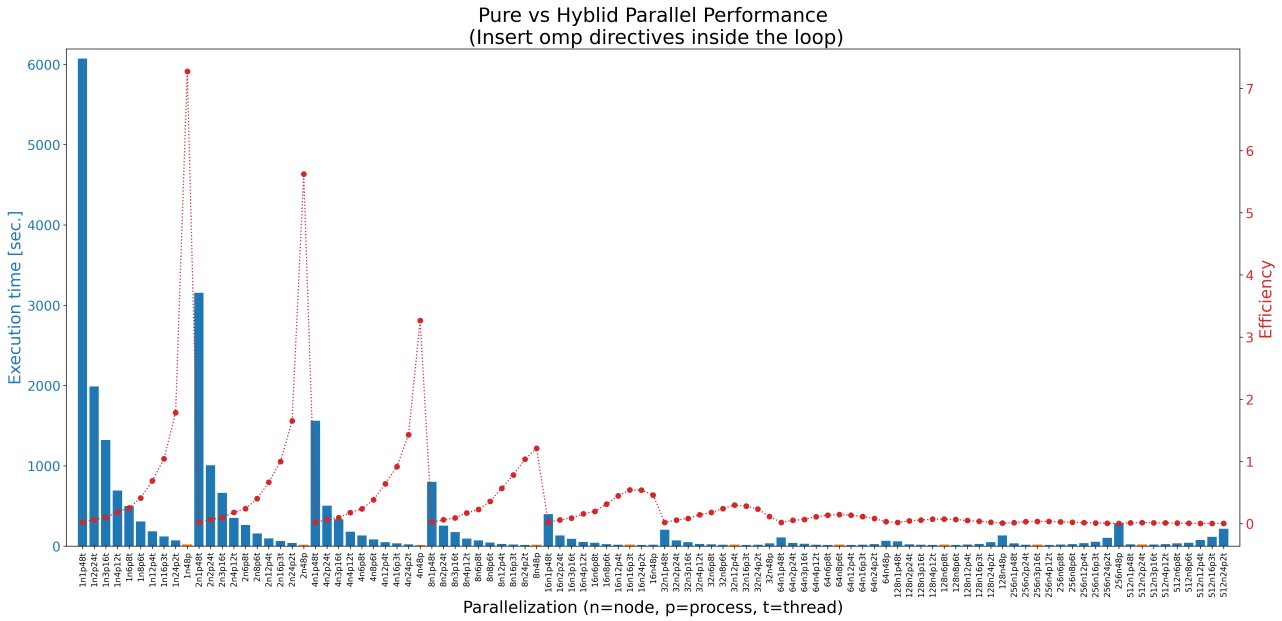

図 8 並列 x-means の性能評価

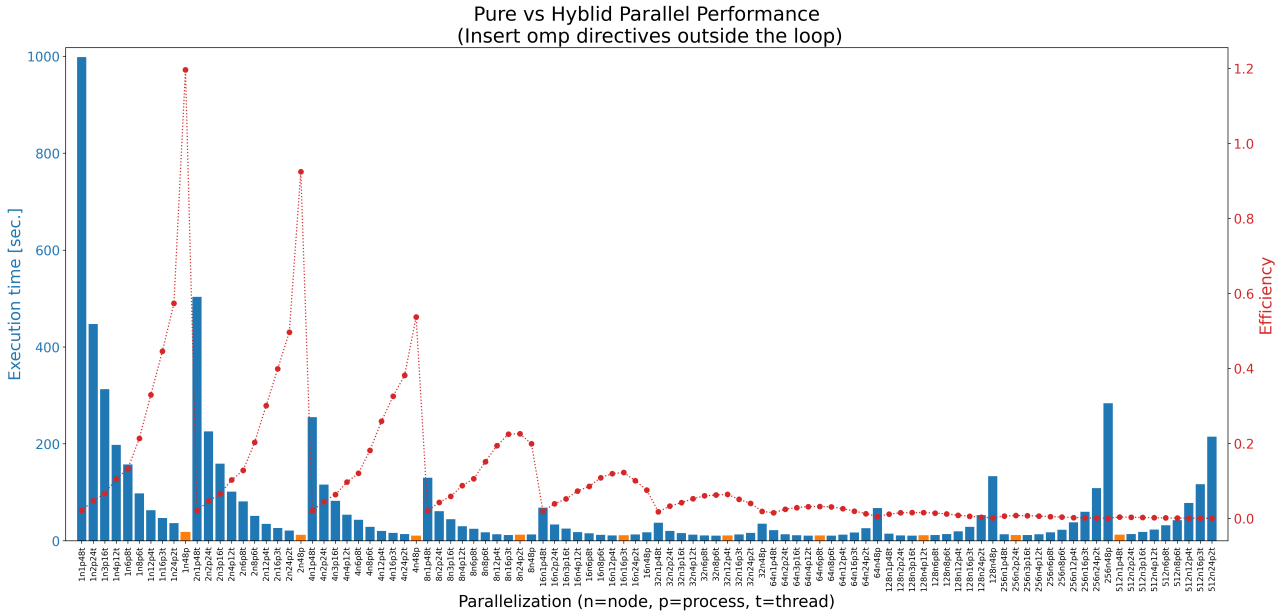

図 9 並列 x-means の性能評価

次にプロセス間通信について考察する．プロセス間通信は，逐次通信方式の場合，プロセス 0 の負荷が大きくなり，負荷バランスの悪さにより実行時間の増加につながると考えられる．一方，二分木通信方式では，プロセス数 P としたときの通信回数が，$\log(P)$ 回であり，逐次通信方式の P 回と比較して削減できる．このため，**表 6** に示すように，プロセス数を増加させたときの通信のオーバーヘッドが小さくなり，実行時間が減少したと考えられる．表 6 から，各方式で最速の実行形式は，逐次通信方式では 4 プロセス 12 スレッドの 10.47 秒に対して，二分木通信方式では 24 プロセス 2 スレッドの 5.85 秒である．二分木通信方式により，より高いプロセス数でも高速化が実現できるようになり，逐次通信方式に対して 1.78 倍の高速化が実現できた．

表 6 128 ノード実行の異なる通信方式における実行時間 [秒] の比較

実行形態	総プロセス数	逐次通信方式	二分木通信方式
1p48t	128	14.42	13.97
2p24t	256	11.03	9.15
3p16t	384	10.48	7.96
4p12t	512	<u>10.47</u>	7.72
6p8t	768	11.58	6.68
8p6t	1024	13.63	6.39
12p4t	1536	19.36	6.36
16p3t	2048	28.65	5.93
24p2t	3072	52.76	<u>5.85</u>
48p	6144	133.28	6.17

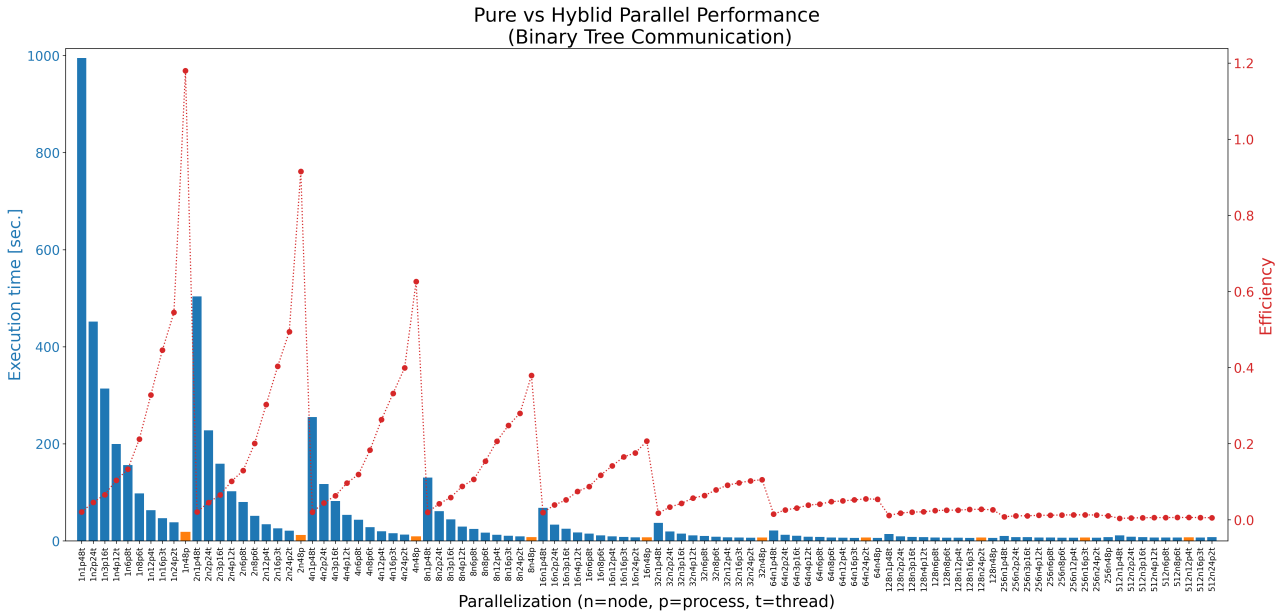

図 10 並列 x-means の性能評価

5. 関連研究

k-means の MPI や OpenMP を利用した並列処理に関する研究は，Zhang らによる文献 [8] や Chandramohan による文献 [9]，Othman らの文献 [10]，Barajas らによる文献 [11] がある．文献 [8] では，k-means を MPI を利用して並列化するときの実装方法と実行時間について説明がされ，比較的小規模な複数のデータセットを対象に議論がされている．一方で，並列数を変化させたときの効率などについては，議論されていない．文献 [9] では k-means を OpenMP と MPI のそれぞれを利用して，並列数を変えた時の実行時間や速度向上率を評価しているが，ハイブリッド実装については議論されていない．文献 [10] では，DNA データと人工データを対象に MPI を利用して並列化した k-means の速度向上率をクラスタ数と並列数を変化させて評価している．しかし，評価に使用した環境は，最大 8 ノードであり，並列化の評価範囲は限定的である．文献 [11] では，42 次元の衛星データを対象にして，OpenMP，MPI と OpenMP のハイブリッド，Spark を利用した k-means の並列化実装を比較している．その一方で，8 コア CPU を 2 個持つノードを 16 まで増やした評価を行なっており，本研究と比較して，並列数は少ない．また，いずれの文献においても OpenMP ディレクティブ挿入箇所の変更や通信方式の変更を考慮した評価は行われていない．

6. おわりに

本研究では，k-means 法を拡張しクラスタ数を自動推定する x-means 法を MPI と OpenMP によりハイブリッド並列化し，そのときの実行時間と並列化効率について評価した．その結果，OpenMP ディレクティブを距離計算ループの外側に挿入し，プロセス間通信を二分木通信方式にしたときに最も高速化され，1 ノード 1 プロセス 48 スレッドの時，994.85 秒の実行時間に対し，128 ノード 24 プロセス 2 スレッドの MPI-OpenMP ハイブリッド実行で，実行時間が 5.85 秒となり，速度向上率は約 170，並列化効率は 1.328 となることがわかった．

x-means の並列化によるさらなる高速化に向けて，データの配置の違いによって，各プロセスが担当するデータ数が変わるため，負荷バランスを考慮する必要がある．また，最適な並列化形態は，計算機やデータ数，データの次元数などに依存するため，対象の問題と環境に合わせて自動的にチューニングする技術 [12] の開発は今後の課題である．

謝辞 本研究は JSPS 科研費 JP19H05662 の助成を受けた．

参考文献

[1] Lloyd, S.: Least squares quantization in PCM, *IEEE Transactions on Information Theory*, Vol. 28, pp. 129–137 (1982).
[2] Ball, G. and Hall, D.: ISODATA, a novel method of data analysis and pattern classification, *Tech. rept. NTIS AD 699616. Stanford Research Institute, Stanford, CA.* (1965).
[3] MacQueen, J.: Some methods for classification and analysis of multivariate observations, *Fifth Berkeley Symposium on Mathematics, Statistics and Probability. University of California Press.*, pp. 281–297 (1967).
[4] Jain, A. K.: Data Clustering: 50 Years Beyond K-Means, *Pattern Recognition Letters* (2009).
[5] Pelleg, D. and Moore, A.: X-means: Extending K-means with Efficient Estimation of the Number of Clusters (2000).
[6] Ishioka, T.: An expansion of X-means for automatically determining the optimal number of clusters, *Proceedings*

of the Fourth IASTED International Conference Computational Intelligence, pp. 91–96 (2005).

[7] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research (2011).

[8] Zhang, J., Wu, G., Hu, X., Li, S. and Hao, S.: A Parallel Clustering Algorithm with MPI –MKmeans, *Journal of Computers*, Vol. 8, No. 1 (2013).

[9] Chandramohan, A. P.: Parallel K-means Clustering, The State University of New York (online), available from ⟨https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Chandramohan-Fall-2012-CSE633.pdf⟩ (accessed 2022-10-16).

[10] Othman, F., Abdullah, R., Nur' Aini AbdulRashid, Salam, R. A.: Parallel K-Means Clustering Algorithm on DNA Dataset, *Parallel and Distributed Computing: Applications and Technologies*, pp. 248–251 (2004).

[11] Barajas, C., Guo, P., Mukherjee, L., Hoban, S., Wang, J., Jin, D., Gangopadhyay, A. and Gobbert, M. K.: Benchmarking Parallel K-Means Cloud Type Clustering from Satellite Data, *Bench 2018: Benchmarking, Measuring, and Optimizing*, pp. 248–260 (2019).

[12] Katagiri, T. and Takahashi, D.: Japanese Autotuning Research: Autotuning Languages and FFT, *Proceedings of the IEEE*, Vol. 106, pp. 2056–2067 (2018).