数 値 解 析 42－4
(1992. 8. 20)

電力潮流計算の並列処理における負荷の最適化

堀川和雄†, 森 眞一郎†, 富田眞治†
†：京都大学 工学部

電力潮流計算を並列化してプロセッサにタスクを分配する際，各プロセッサに分配される負荷を最適にする方法を考察する．並列計算機において反復解法をおこなう場合，最も処理に時間がかかるプロセッサによって全体の性能は決まる．各プロセッサの処理時間は計算時間および通信時間によって定まるが，問題の最適な分割を行うことで処理時間を小さくすることが可能である．そのため，1) 並列度が最大になるように問題を分解し，2) クラスタ化によって処理粒度の最適化を行ない，3) クラスタ単位でプロセッサに割当を行うことで問題分割を行う．この最適化を実際の電力潮流計算に当てはめ，高速化および省プロセッサがはかれることを実機データに基づく計算により確認した．

Optimal Load Balancing for Parallel Load-flow Analysis

Kazuo HORIKAWA†, Shin-ichiro MORI†, Shinji TOMITA†

†: Department of Information Science
Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606-01 Japan

E-mail: {horikawa, moris, tomita}@kuis.kyoto-u.ac.jp

In this paper, we discuss the way to optimize the load balance of parallel load-flow analysis. Since the performance of parallel iterative calculation is limited by the processor which has largest load, we can obtain highest performance if we get optimal load distribution. Our approach to gain optimal load for each processors start with 1) problem partitioning into tasks so that we get the maximum parallelism. And then 2) we collect some tasks to make clusters which have optimal grain size. Finally 3) we allocate clusters to processors with one to one relation. Following this approach, we confirmed that, for practical load-flow analysis problems, we can get higher performance with less processors compare to the maximum parallelism implementation.

1 はじめに

発電機や負荷, およびこれらを結ぶ送電線や変圧器等の設備から構成される大規模かつ複雑なシステムである電力系統において, 安定かつ経済的に運用するための系統計画と系統運用業務に対する高度な技術の要求はいまだなお高いものである. このような電力系統の運用の効率化をはかる上から, 系統解析の高速化が求められている.

電力潮流計算は, 系統解析において最も基本となる計算であり, 非線形の連立方程式であるノード方程式が与えられた時にその解を求める問題である [1, 2, 4].

本稿では, 電力潮流計算を疎結合型マルチプロセッサ上に実装し, 並列処理による高速化を行う場合において, 計算負荷の適切な分配により, 問題の並列性を最大限利用した場合に比べ, 少ないプロセッサ資源で, より高速な処理が可能であることを示す.

第 2 章では, 電力潮流計算の概要を述べたあと, 並列マシンへの実装法を示し簡単にその評価を行う. その結果を基に 3 章で系統の性質を反映したクラスタ化による計算負荷の最適化手法を述べ, その効果を示す.

2 電力潮流計算の並列化

2.1 潮流計算の概要

電力系統は発電機、送電線、負荷などが組み合わされたシステムである. 発電機は、有効/無効電力を供給し、負荷はこれを消費する. 発電機と負荷を結ぶ送電系統は, 発電機からの供給電力を消費地に配送する.

潮流計算の目的は, 送電系統内の各点における電圧の分布を求めることにある.

2.2 潮流計算の解法

電力系統の発電端子と負荷端子を取りだし, 残りの送電線その他の送電系統は一括して M という回路で表す (図 1) [1].

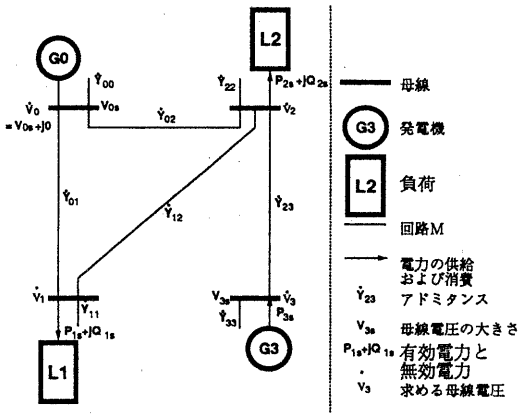


図 1: 電力系統の例

$Y_{ij} = Y_{ji}$の性質があり, 発電機母線, 負荷母線には任意の順序に 0,1,$\cdots$, i,$\cdots$,$N-1$ の記号をつけておく. これ以降これらを一括して母線と呼び, Nは母線数を表すものとする. 送電系統 M の内部には起電力は含まれず, 各母線と大地の間につながれた充電キャパシタンスや, 電力用コンデンサなどはすべて負荷として扱う. また母線はその特性が, 有効電力 P_{ks} と無効電力 Q_{ks}で表される **PQ 指定母線**と, 有効電力 P_{ls}と端子電圧の大きさ V_{ls}で表される **PV 指定母線**に分類できる.

電力潮流計算では各母線の特性と送電系統 M のアドミタンス行列 Y_{ij}が与えられ

$$\dot{I}_i = \sum_{m=1}^{N} \dot{Y}_{im}\dot{V}_m \quad (i = 0, 1, \ldots, N-1) \quad (1)$$

などの関係から非線形連立方程式を解くことで各母線の電圧を求める.

非線形連立方程式の解法としては, 主に Newton-Raphson 法および Gauß-Sidel 法

[1]ドットは複素数であることを表す

があるが, 1) 記憶用量が少なくてよい. 2) 各母線での演算処理に必要なデータは隣接する母線の情報のみでよい. 3) 各母線における一斉計算が可能であり並列化が容易である. 等の利点がある後者を採用した [5].

Gauß-Sidel 法では$\dot{V}$を求めるのに, $n+1$ 回目の反復においては n 回目の結果を用いて以下の演算を行う[2].

- PQ 指定母線では (kは PQ 指定母線番号)

$$\dot{V}_k^{(n+1)} = (\frac{P_{ks} - jQ_{ks}}{\overline{V}_k^{(n)}} - \sum_{j \neq k} \dot{Y}_{kj} \dot{V}_j^{(n)}) / \dot{Y}_{kk} \tag{2}$$

- PV 指定母線では (lは PV 指定母線番号)

$$\begin{cases} Q_l^{(n+1)} = -\Im((\sum_j \dot{Y}_{lj} \dot{V}_j^{(n)}) \overline{\dot{V}}_l) \\ \dot{V'}_l^{(n+1)} = (\frac{P_{ls} - jQ_l^{(n+1)}}{\overline{\dot{V}}_l^{(n)}} - \sum_{j \neq l} \dot{Y}_{lj} \dot{V}_j^{(n)}) / \dot{Y}_{ll} \\ \dot{V}_l^{(n+1)} = \dot{V'}_l^{(n+1)} \frac{V_{ls}}{|\dot{V'}_l^{(n+1)}|} \end{cases} \tag{3}$$

式(2)(3) を各母線の電圧計算と呼ぶ. 実装において式 (2)(3) の総和計算では, $\dot{Y}_{kj}$ が非 0 の要素の添字リストをあらかじめ作ることにより, 演算量を非 0 要素の数のオーダに削減している.

2.3 潮流計算の並列マシンへの実装とその評価

潮流計算を, 疎結合型のマルチプロセッサ・システム上に実装しその評価を行う. 並列処理の単位としては, 電力系統内の各母線毎の電圧計算 (式 (2) あるいは式 (3) による.) を最小単位とし, プロセッサ当たり 1 単位分 (1 母線分の電圧計算に対応) を割り当てた. したがって, 系統内の母線数が当該潮流計算の並列度を示すことになる.

このような並列化を行った潮流計算を 256 台構成の nCUBE2 および 64 台構成の AP1000 上に実装し並列化による速度向上比を求めた. その結果を表 1 に示す. 表には, 母線数ならびに最大枝数の異る 5 つの系統に関して, 各々のマシンで逐次処理した場合に対する, 上述の並列化を行った場合の速度向上比を示している.

いずれのマシンにおいても, 並列化による速度向上は見られるが, 十分な台数効果が得られているとはいいがたい. この原因は, 電力系統自体が非均質であり, 各母線での計算時間がその母線における枝数[3]に比例して変化するからである. また, 実マシンの特性を調べた結果, 通信のオーバヘッドが大きく, 母線単位の並列化では処理粒度が細か過ぎることがわかった.

そこで, 複数の母線で 1 つのクラスタを構成し, クラスタ単位で並列処理を行うことを考える.

表 1: Gauß-Sidel 法の並列化による電力潮流計算の速度向上比

母線数 (PE 数)	最大枝数	速度向上比	
		nCUBE2	AP1000
4	3	1.25	1.63
12	3	1.97	3.03
33	6	2.56	9.79
43	10	2.04	7.66
118	9	5.89	測定不可

[2] $\dot{V}^{(n)}$は n 回目の$\dot{V}$の近似値を示す. $\overline{\dot{V}}$は$\dot{V}$の共役複素数を, $\Im(\dot{X})$ は$\dot{X}$の虚部を, jは虚数単位をそれぞれ示す.

[3] 当該母線 i に関するアドミタンス・ベクタ $Y_i = \{Y_{xi} | x = 0 \ldots N-1\}$ のうち $Y\{xi\} = 0$ でないものの数

3 系統のクラスタ化による計算負荷の最適化

3.1 クラスタ化の概要

電力潮流計算では，各母線自体の特性の与え方 (PQ 指定/PV 指定) や，母線の枝数により，母線電圧の計算量が変化する．したがって，前節で述べたような母線単位での並列化では，各プロセッサの負荷のばらつきが大きくなる．このため，反復計算時に負荷の軽いプロセッサが，最大の負荷をもつプロセッサの計算の終了を待たなければならず，これが十分な台数効果が得られない原因となっている．

これらの背景を鑑み，電力潮流計算における負荷の最適化に関して，以下の2つを最適化の目標とする．

1. 最大負荷の軽減
2. 負荷の均衡化による，プロセッサ資源の削減

負荷の軽減は，通信を行う2つの母線を同一クラスタに配置し，クラスタ間通信をクラスタ内通信に置き換えること (通信オーバヘッドの削減) で実現する．ただし，クラスタ化により計算時間が増加するが，通信時間と計算時間のトータルが最大負荷を越えないような母線の組み合わせでのみクラスタ化を行う．この結果，系全体で負荷の均衡化が行われるとともに，最大負荷も軽減される．

以下では，用語の定義を行ったのちに，最適化アルゴリズムの詳細を示す．

なお，クラスタ化に基づく最適化により，母線数が実プロセッサ台数以上の問題も解くことが可能になるが，本稿で示すアルゴリズムはまだこれに対応していない．

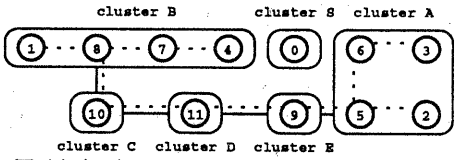
3.2 定義

[仮想プロセッサ] 1つの母線電圧計算が割り当てられる仮想的なプロセッサである．母線 i の電圧計算を行う仮想プロセッサを p_i と記述し，仮想プロセッサの集合全体を P で表す．$ADJ[p_i]$ を仮想プロセッサ i が通信を行う仮想プロセッサ集合を表し，その要素数を $|ADJ[p_i]|$ と表す．仮想プロセッサ集合全体を P と書く．複数の仮想プロセッサで1クラスタを構成し，1つの実プロセッサに配置する (図2参照)．

[クラスタ] ある実プロセッサに対応付けられている仮想プロセッサの集合であり，実プロセッサは必ずただ一つのクラスタだけに対応する．クラスタ i のことを C_i，割り当てられている仮想プロセッサ集合を $\mathbf{P}_{C_i}$ と書き，また，クラスタが含む仮想プロセッサの数を $|\mathbf{P}_{C_i}|$ と書く．ある仮想プロセッサは必ずどれかのクラスタに属し，かつ，ただ1つのクラスタにしか属さない．クラスタ i に隣接するクラスタ集合を，$ADJ[C_i]$ で表す．またその要素数を $|ADJ[C_i]|$ と書く．クラスタ集合全体を指すのに $\mathcal{C}$ と表し，クラスタ数は $|\mathcal{C}|$ と記す．

[計算時間および通信時間] 反復計算における1回の反復で，仮想プセッサが母線電圧を計算するのに要する時間のうち，他の仮想プロセッサとの間でデータの授受を行う時間を除いたものを計算時間と呼び，母線 i の計算時間を $T_{calc}(p_i)$ で表す．

また，2台の仮想プロセッサ，p_i ならびに p_j 間での1回のデータ授受に要する時間 (以下では，通信時間と呼ぶ) を $T_{comm}(p_i, p_j)$ で表す．しかしながら，同一クラスタ内の仮想プロセッ



円は仮想プロセッサ, ラウンドボックスはクラスタを表す. また仮想プロセッサ間の通信は破線で, クラスタ間の通信は実線で表す.

この例の場合,

$P = \{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}\}$
$|P| = 12$
$ADJ[p_1] = \{p_8\}$, $ADJ[p_8] = \{p_1, p_7, p_{10}\}$
$|ADJ[p_1]| = 1$, $|ADJ[p_8]| = 3$
$\mathbf{P}_{C_B} = \{p_1, p_4, p_7, p_8\}$, $\mathbf{P}_{C_C} = \{p_{10}\}$
$|\mathbf{P}_{C_B}| = 4$, $|\mathbf{P}_{C_C}| = 1$
$ADJ[C_B] = \{C_C\}$, $ADJ[C_C] = \{C_B, C_D\}$
$|ADJ[C_B]| = 1$, $|ADJ[C_C]| = 2$
$\mathcal{C} = \{C_A, C_B, C_C, C_D, C_E, C_S\}$
$|\mathcal{C}| = 6$

などがいえる.

図 2: 仮想プロセッサとクラスタの例

サ間の通信時間は異るクラスタ間にまたがる仮想プロセッサ間の通信時間に比べて非常に小さいことと, 実装の対象となるマシンにおいて, プロセッサ間通信の時間がプロセッサ間距離にほとんど依存しないことを考慮し, 以下では $T_{comm}(p_i, p_j)$ を単に T_{comm} という定数で代用する. また, 仮想プロセッサ間の通信のうち, 送信側クラスタと受信側クラスタが同一のものは, まとめて 1 回の通信として考えるものとする.

[クラスタの負荷]

クラスタ i の負荷 $W(C_i)$ は, クラスタ i が割り付けられた実プロセッサが 1 回の反復に要する時間であり, 次式で表される.

$$T_{comm} \cdot |ADJ[C_i]| + \sum_{p_j \in \mathbf{P}_{C_i}} T_{calc}(p_j) \equiv W(C_i) \quad (4)$$

[最大負荷] クラスタの負荷のうち最大のものを最大負荷 $W_{max}(\mathcal{C})$ と呼ぶ.

$$\max_{\forall C_i \in \mathcal{C}} \{W(C_i)\} \equiv W_{max}(\mathcal{C}) \quad (5)$$

最大負荷が最小となっている状態が, 最適に負荷分散されている状態である. また必要な実プロセッサ数は $|\mathcal{C}|$ で表される.

3.3 クラスタ化のアルゴリズム

[前準備] 2 つのクラスタを統合する関数 unify の定義.

関数 unify
(クラスタ集合: $\mathcal{C}$; integer: a, b):クラスタ集合
入力:統合前のクラスタ集合 $\mathcal{C}$,
　　統合されるクラスタの番号 a,b
出力:統合後のクラスタ集合 $\mathcal{D}$

STEP1:準備
クラスタ集合 $\mathcal{C}$ をクラスタ集合 $\mathcal{D}$ にコピーする. その際 $ADJ[C_i]$, $\mathbf{P}_{C_i}$ の関係は $ADJ[D_i]$, $\mathbf{P}_{D_i}$ でも保たれる.

STEP2:操作

1. $\mathbf{P}_{D_a} := \mathbf{P}_{D_a} \cup \mathbf{P}_{D_b}$ かつ
   $ADJ[D_a] := (ADJ[D_a] \cup ADJ[D_b]) - \{D_a\} - \{D_b\}$
   となるようにクラスタ D_a を変更する.
2. D_b を除去する.

/* D_b は D_a に吸収される */

STEP3:帰り値の設定
新たに生成したクラスタ集合 $\mathcal{D}$ を返す.
//

クラスタ化のすべての場合を尽くそうとすると, 組合せ爆発を起こし, 現実的ではない. ここでは 2 つのクラスタを 1 つに統合することを繰り返してクラスタ化していく. その際, 最も負荷の重いクラスタから統合の候補としていくことで, 最大不可が小さくなる効果を狙う. クラスタ化後の, クラスタと実プロセッサとのマッピングの関係はここでは問題としない. 以下にクラスタ化のアルゴリズムを示す.

アルゴリズム [**MAKECLST**]
入力:仮想プロセッサ集合:p
出力:クラスタ集合:$\mathcal{C}$

連結リスト L:integer /* クラスタ番号を示す */
　　/*クラスタの負荷順に連結されたリスト*/
変数 t:ポインタ /* リスト L のセルを指すポインタ */
　　/* t の指すセルの内容も t で参照する */
変数 s:integer /* 隣接するクラスタの番号 */
変数 i, j:integer /* 一時変数 */

STEP1:準備

1. クラスタ集合$\mathcal{C}$を設定する.
   for$(i = 1, N-1)$ **do**
   $\mathbf{P}_{C_i} := \{p_i\}$ かつ
   $ADJ[C_i] := \{C_{i_1}, C_{i_2}, \ldots\}$
   where $ADJ[p_i] = \{p_{i_1}, p_{i_2} \ldots\}$
   なるクラスタ集合$\mathcal{C}$を生成する.
   end do
   /*初めに各クラスタはそれぞれ1つの仮想プロセッサが割り当てられる*/

2. 負荷の大きいクラスタ順の，クラスタ番号の連結リスト L を作成.

STEP2:次のステップを繰り返す.

1. t はリスト L の先頭を指す.
   /*統合されるクラスタの候補を負荷の大きいクラスタから選ぶ*/

2. $\min_{C_i \in ADJ[C_t]} \{W_{max}(\text{unify}(\mathcal{C}, t, i))\}$
   $= W_{max}(\text{unify}(\mathcal{C}, t, s))$ なる C_s を選択する.
   /*C_tに隣接するクラスタのうち，もしC_tと統合したら最大負荷が最も小さくなるクラスタを選ぶ*/

3. **if** $(W_{max}(\text{unify}(\mathcal{C}, t, s)) \leq W_{max}(\mathcal{C}))$
   then

   (a) クラスタ集合$\mathcal{C}$をクラスタ集合 $\text{unify}(\mathcal{C}, t, s)$ に置き換える.

   (b) クラスタ番号が s であるセルをリスト L から取り除く.

   (c) クラスタの負荷が変わったので，リスト L を作り直す.

   (d) **if** (リスト L のセル数が1?)
   then go to STEP3
   else go to STEP2 の 1
   end if

   /*それがよりよいクラスタ集合を与えるなら，実際に統合する*/

   else
   /*そうでないなら次の候補クラスタを選ぶ*/

   (a) ポインタ t は次のセルを指す.

   (b) **if** (t が nil を指す?)
   then go to STEP3
   else go to STEP2 の 2
   end if

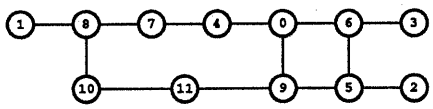
   end if

STEP3:$\mathcal{C}$を出力する. //

しかし，このアルゴリズムはローカルな最適解に落ちる可能性がある．アニーリングなどを用いれば改善できると思われる．

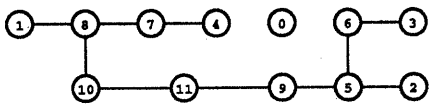
3.4 クラスタ化の例

実際にクラスタ化によってどのような効果が得られるか，例を用いて示そう．簡単のため計算時間一定とし，パラメタを $T_{comm} = 5, T_{calc}(p_i) = 2 \equiv T_{calc}$とする．図3の(a)の電力系統をとりあげる．



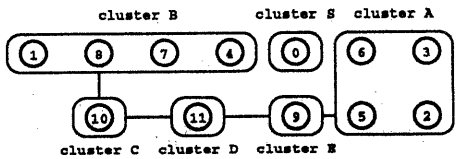
節は母線(番号は母線の番号)，枝は母線間に非0のアドミタンスが存在することを表す．また，母線0は基準母線であり，電圧一定である．

(a) 電力系統の例



節は母線電圧計算を割り当てられた仮想プロセッサ(番号は仮想プロセッサの番号，母線電圧 i の計算を割り当てられる仮想プロセッサは p_i である)，枝は仮想プロセッサ間に通信が存在することを表す．

(b) (a)の電力系統を割り当てた
仮想プロセッサの通信ネットワーク



節はクラスタ(クラスタ内の○はそのクラスタに割り当てられる仮想プロセッサを示す．また仮想プロセッサ0は特別に1つだけでクラスタを構成する)，枝はクラスタを割り当てられた実プロセッサ間で通信が存在することを示す．

(c) (b)の仮想プロセッサのクラスタ化

図3: クラスタ化の例

(a) の電力系統が与えられたとき，電力系統の1つの母線，即ち母線電圧計算1つを1つの仮想プロセッサに割り当てると，(b) の仮想プロセッサおよびその間の通信ネットワークが得られる．(b) では (a) で節0へつながる枝が切られている．潮流計算において母線0を基準母線にとるため$\dot{V}_0$が定数となり，母線0と接続している母線電圧を計算する仮想プロセッサは母線0と通信を行なう必要がなくなるからである．

さて,(b) の仮想プロセッサ1つ1つを実プロセッサに割り当てた場合, 最大負荷がどれだけの大きさになるかを計算すると, 仮想プロセッサ5または8を割り当てられる実プロセッサで最大になり, その値は

$$W_{max}(\mathcal{C}) = 5 \cdot 3 + 2 \cdot 1 = 17 \quad (6)$$

である．

次に,(c) のようにクラスタ化して, クラスタを実プロセッサに割り当てる場合に最大負荷の大きさを計算すると, クラスタAかBを割り当てられる実プロセッサで最大になり,

$$W_{max}(\mathcal{C}) = 5 \cdot 1 + 2 \cdot 4 = 13 \quad (7)$$

となり, クラスタ化せずに並列化した場合の1.3倍に高速化される．また必要な実プロセッサ数も11から6に減り, 約半分になる．

もし (c) でクラスタCとクラスタDをあわせてクラスタFとしたら，そのクラスタFが最大負荷を与え,

$$W_{max}(\mathcal{C}) = 5 \cdot 2 + 2 \cdot 2 = 14 \quad (8)$$

となってしまうので，クラスタCとクラスタDは統合しない．

3.5 クラスタ化の効果

実際にクラスタ化した仮想プロセッサ集合を実プロセッサに割り当てて，実行できるような実装は残念ながらまだ完成していないので，クラスタ化の効果を計るのに理論的な計算を用いて推定する．その際，演算時間，通信時間の指標としては実機のデータを用い，実電力系統についてクラスタ化を行なった場合について評価を行なう．

実機において$T_{calc}(p_i)$はPQ指定母線であるか,PV指定母線であるかによって変わり，$|ADJ[p_i]|$によっても変わるが，定数であることには変わりない．実験結果とプログラムの解析によると, 仮想プロセッサp_iでは，PQ指定母線の場合は$(3 + |ADJ[p_i]|) \cdot c$, PV指定母線の場合は$(6 + 2|ADJ[p_i]|) \cdot c$と見積もることができる．また，$T_{comm}$は送り手がまだ送信していないため受け手が待っているという, 通信を行う以前の待ち時間を除く大きさで考えている．T_{comm}の実測は困難であるため実験結果からT_{comm}を見積もる．表2にそれぞれの値を示す．

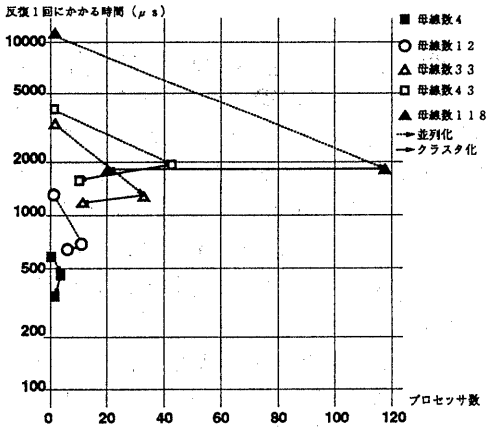
表2: パラメタの設定

計算機	演算の定数c	通信時間
nCUBE2	$17\mu s$	$190\mu s$
AP1000	$9\mu s$	$51\mu s$

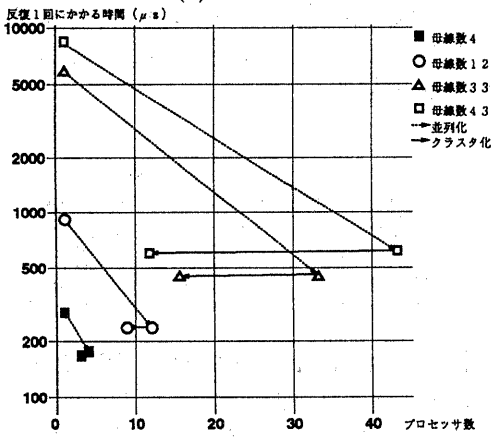
図4にnCUBE2，AP1000上に電力潮流計算を実装した場合の並列化ならびにクラスタ化の効果を示す．最大の並列度を引きだす並列化により数倍から10倍近い速度向上が得られている．次に本稿で述べたクラスタ化を行なうことで，並列度を最大限にした並列化に比べさらに数パーセント速度が向上するとともにプロセッサ数も数分の1に激減する効果が読みとれる．

どの程度速度が向上し，必要プロセッサ数が減るかというのは，与えられる仮想プロセッサ集合の通信形態や，プロセッサでの演算時間と通信時間の比率に依存するものである．今回の潮流計算の場合，クラス

タ化の効果は速度の向上よりもプロセッサ数の削減に顕著に現れている．実際に今回のクラスタ化では複数の仮想プロセッサをまとめることで，通信量を減らして演算量を増やす方向での最適化が行なわれた．



(a) nCUBE2



(b) AP1000

クラスタ化による実行時間が短くなったのを強調するため，時間軸は対数軸にしてある．

図 4: 電力潮流計算の並列化ならびにクラスタ化

4 まとめ

本稿では電力潮流計算を並列化して疎結合型マルチプロセッサに実装し，高速化をはかる場合において，計算負荷を適切に分配することにより，問題の並列度を最大限に引き出す場合に比べて，より少ないプロセッサ資源で，より高速な処理が可能であることを示した．

最適化の手段としては，まず潮流計算を並列度が最大となるように問題分割し，次に分割された問題を最大負荷が小さくなる方向でのクラスタ化によって処理粒度を最適にし，最後にクラスタ単位でプロセッサに割当を行なうという問題分割法を提案した．

今後の課題はクラスタ化したプログラムのインプリメンテーション，実プロセッサ数に制約がある場合のクラスタ化のアルゴリズムの実現，ならびにローカルな最適状態に落ち込まないようにアルゴリズムを改良することである．

謝辞

数々の有用な意見を戴いた三菱電機産業システム研究所の坂口敏明博士、田岡久雄博士に深く感謝致します。

参考文献

[1] 森 正武, “数値解析”, 第 2 章, 共立出版, 1973 年.

[2] 関根康次, “電力系統解析理論,” 第 3 章, 電気書院, 1971 年.

[3] 清水俊幸, 堀江健志, 石畑宏明. “高速メッセージハンドリング機構–AP1000 における実現–,” 並列処理シンポジウム JSPP, pp.195-202, 1992 年 6 月.

[4] 高橋一弘, “電力システム工学,” 第 3 章, コロナ社, 1977 年.

[5] 田岡久雄, “大規模電力系統解析の高速化に関する研究,” 東京大学工学部博士論文, 1985 年 10 月.