計算機アーキテクチャ 79－11
（１９８９．１１．２１）

マルチPSI上の最短経路問題の実現と評価

和田 久美子
沖電気工業株式会社
総合システム研究所

市吉 伸行
(財) 新世代コンピュータ
技術開発機構

概要

最短経路問題は重要なグラフ問題のひとつであり，今まで様々な逐次アルゴリズム及び並列アルゴリズムが開発されている．しかし，大規模汎用MIMDマシン上で大規模なグラフを対象に実行する場合，プロセッサへの効率の良い負荷分散方法については，現在まであまり研究されていない．本稿では，最短経路問題を解く1つの分散アルゴリズムを提案し，疎結合マルチプロセッサであるマルチPSI/V2上での大規模な格子状グラフを用いた実行における負荷分散方式について考察する．実験した負荷分散方式は，グラフの局所性を多く保つ2次元単純マッピング方式，プロセッサ稼働率の高い2次元多重マッピング方式，そして理論的にはプロセッサ数の3分の2の台数効果を得ることのできる1次元単純マッピング方式である．それぞれのマッピング方式を用いた実行結果を示し，解析を行う．

A Distributed Shortest Path Algorithm and Its Mapping on the Multi-PSI

Kumiko Wada
Knowledge Information Processing Dept.
Systems Laboratory
Oki Electric Industry Co., Ltd.
4-11-22 Shibaura, Minato-ku, Tokyo 108, Japan
Nobuyuki Ichiyoshi
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan

Abstract

The shortest path problem is an important and well-studied graph problem, and a number of sequential and parallel algorithms have been developed. However, there seems to have been little study on how to map a large-scale graph on a large-scale general-purpose MIMD machine in solving the problem. In this paper, we present a distributed shortest path algorithm, and study strategies of mapping a large-scale grid graph on the Multi-PSI/V2, a loosely-coupled multiprocessor. The mappings experimented are the two-dimensional simple mapping which best preserves the locality of the graph, the two-dimensional multiple mapping which has a higher processor utilization rate, and the one-dimensional mapping which theoretically has the speedup which is half the number of processors. We give the performance results of each of the mappings and their analysis.

1 はじめに

最短経路問題は, 輸送やパイプライン網の建設, 集積回路の設計などにおいて非常に重要な問題である. そのため, 最短経路問題を高速に解くアルゴリズムの様々な研究がされてきた. 逐次アルゴリズムでは, n 個の点と e の辺を持つ与えられた有向グラフに対して,$O(n^2)$ 時間で 1 点から全点への最短経路問題を解く Dijkstra のアルゴリズム [6] がある. Warshall-Floyd のアルゴリズム [12, 7] は, $O(n^3)$ 時間で全点から全点への最短経路問題を解く. Johnson は, e が n^2 よりずっと小さい時, データ構造を工夫することによって Dijkstra のアルゴリズムの実行時間が $O(e \log n)$ に改良できることを示した [9]. Fredman と Tarjan は, Fibonacci heap を用いて, $O(e + n \log n)$ 時間で 1 点から全点への最短経路問題を解くアルゴリズムを提案した [8]. 並列 / 分散アルゴリズムも同様に研究されている [11]. Deo ら [5] は, 8 プロセッサから成る密結合 MIMD マシン上で効率良く動作するよう, 逐次アルゴリズムを並列化した. Chandy と Misra[3] は, 分散最短経路アルゴリズムを与えた. 最近では, Driscoll らが, n 個の点と $e \geq n \log n$ 個の辺を持つグラフに対して relaxed heap を用いれば, $P \geq e/(n \log n)$ 台のプロセッサから成る EREW PRAM マシンでは, $O(e/p)$ 最適実行時間を得ることができることを示した [1].

しかし, 10^2 程度のプロセッサ数の大規模汎用 MIMD マシン上で 10^3 以上の点を持つ大規模なグラフに対する最短経路問題に関しては, まだあまり研究されていないようである. このような大規模マシンは密結合することができず, データへのアクセスはコスト高である. 従ってグラフをどのようにプロセッサに割当てるかが実行効率を左右する.

本稿では, 最短経路問題を解く分散アルゴリズムを提案し, 並列推論マシンの実験機であるマルチ PSI/V2[10] 上での大規模な格子状グラフに対するマッピング方式について考察する. マルチ PSI/V2 は疎結合マルチプロセッサで, 8×8 のメッシュネットワーク上に最大 64 プロセッサまで接続される. 実験したマッピング方式は, グラフの局所性を多く保つ 2 次元単純マッピング方式, プロセッサ稼働率の高い 2 次元多重マッピング方式, そして理論的にはプロセッサ数の 3 分の 2 の台数効果を得ることのできる 1 次元単純マッピング方式である. それぞれの方式を用いての実行結果を示し, 実際の台数効果と通信オーバヘッドについて解析する.

2 分散アルゴリズム

この節では, 1 点から全点への最短経路問題に対する分散アルゴリズムを与える. 最短経路問題は, グラフ理論の用語を用いて次のように表される. 与えられた n 個の点の集合 V と e 個の辺の集合 E によって, 有向グラフ $G = (V, E)$ を定義する. 辺は点の順序対である. 点の対から非負の実数値へのコスト関数 c を与える. V の点 v_i から点 v_j への辺に対して, $c(v_i, v_j)$ をその辺のコスト, あるいは長さという. 考察を容易にするため, 点 v_i から点 v_j への辺が存在しない場合, $c(v_i, v_j) = +\infty$, また, 各点 v_i に対して, $c(v_i, v_i) = 0$ であるとする. $l + 1$ 個の点 $v_0, v_1, \ldots, v_l$ において, すべての $i(= 0, 1, \ldots, l-1)$ に対して点 v_i から点 v_{i+1} への辺 e_i が存在する時, 辺の列 $e_0, e_1, \ldots, e_{l-1}$ を点 v_0 から点 v_l への経路と呼ぶ. ある経路に対して, その経路に含まれる辺のコストの和をその経路のコストという. グラフのある 2 点間を結ぶ経路のうち, コスト最小のものを最短経路という. 以上の仮定のもとで, 出発点と呼ぶ特定の 1 点 $v_0 \in V$ から V の各点への最短経路を求める問題を, 1 点から全点への最短経路問題と呼ぶ.

与えられた有向グラフ $G = (V, E)$ に対して, 任意に通信可能な p 個のプロセッサ $P_0, \ldots, P_{p-1}$ を用いて最短経路問題を求める分散アルゴリズムを与える. 点集合 V を適当に直和分解する. すなわち, 部分点集合 $V_i (i = 0, 1, \ldots, p-1)$ は, $V = V_0 \cup V_1 \cup \cdots \cup V_{p-1}, V_i \cap V_j = \emptyset (i \neq j)$ なる性質を持つ.

各プロセッサは, 割当てられた各点への最短となる可能性のある経路とそのコストの情報を保持する. これらはコスト経路情報によって更新される. コスト経路情報は, $cp(cost, v_j, v_i)$ の形をしており, 出発点から点 v_j への経路でコストが $cost$ のものが存在し, その経路で v_j の直前の点が v_i であることを示す. アルゴリズムの実行中, 各点 v_j は出発点から自分までの経路でその時点までに分かっているもののうち, 最も低コストの経路のコストとその経路での自分の直前の点をそれぞれ変数 $cost_j$, $path_j$ に保持している. 今, コスト経路情報 $cp(cost, v_j, v_i)$ が与えられたとする. この時, $cost < cost_j$ ならば, $cost_j$, $path_j$ の内容をそれぞれ $cost$, v_i に更新し, v_j のすべての隣接点 v_k へのコスト経路情報 $cp(cost + c(v_j, v_k), v_k, v_j)$ を生成する. コスト経路情報は, 各プロセッサ内で優先度付き待ち行列に格納され, それに含まれるコスト値に従って小さいものから順に取り出される. プロセッサ P_i が v のすべての隣接点 u に対してコスト経路情報を生成した時, 点 u が部分点集合 V_i に属しているならば, その情報をプロセッサ P_i の優先度付き待ち行列に格納する. そうでなければ, 情報はプロセッサ間メッセージ通信によって $u \in V_j$ を割当てられたプロセッサ P_j に送信され, プロセッサ P_j の優先度付き待ち行列に格納される.

各点に対するコスト経路情報の初期値は $cost = +\infty, path =$ 未定義, すべての優先度付き待ち行列は空である. アルゴリズムは, 最初に, 出発点 v_0 を割当てられたプロセッサが, v_0 のすべての隣接点 v_j に対してコスト経路情報 $cp(c(v_0, v_j), v_j, v_0)$ を生成し, 優先度付き待ち行列に格納する. すべての優先度付き待ち行列が空になった時, アルゴリズムは終了する. この時点で, 各点 v_j の $cost_j$, $path_j$ にはそれぞれ最短経路のコストと

```
プロセッサ P_0 の初期化:
    begin
        for プロセッサ P_0 に属するすべての点 v_i do cost_i := ∞, path_i := 未定義 ;
        優先度付き待ち行列を初期化 ;
        cost_0 := 0 ;
        for v_0 のすべての隣接点 v_j do
            cp(c(v_0, v_j), v_j, v_0) を v_j が属するプロセッサ P_b に送信する
    end

プロセッサ P_a(a ≠ 0) の初期化:
    begin
        for プロセッサ P_a に属するすべての点 v_i do
            cost_i := ∞ ;
        優先度付き待ち行列を初期化 ;
    end

プロセッサ P_a の解探索:

    優先度付き待ち行列が空でないならば,
        begin
            優先度付き待ち行列から cp(cost, v_j, v_i) を取り出す ;
            if cost_j > cost then
                begin
                    cost_j := cost ;
                    path_j := v_i ;
                    for v_j のすべての隣接点 v_k do
                        cp(cost + c(v_j, v_k), v_k, v_j) を v_k の属するプロセッサ P_b に送信する
                end
        end

    メッセージ cp(cost, v_j, v_i) を受信したら,
        begin
            優先度付き待ち行列に cp(cost, v_j, v_i) を格納する
        end
```

図 1: 最短経路を求める分散アルゴリズム

その経路での自分の直前の点が保持されている．アルゴリズムを図1に示す．

プロセッサ台数を 1 とした時，アルゴリズムは Dijkstra の逐次アルゴリズムと同等に動作する．一方，n 個の点をプロセッサ n 台に 1 つずつ割当てた場合，アルゴリズムは Chandy と Misra の分散アルゴリズムでメッセージの上書きが可能なバッファを用いる場合 [3] とほぼ同様となる．一般に複数プロセッサを用いた場合，コスト経路情報を格納する優先度付き待ち行列も複数存在することになり，優先度管理が分散化されることになる．従って，プロセッサ P_i で優先度付き待ち行列から取り出されて処理中のコスト経路情報より本来なら優先度が低く，後で処理されるべきものが，他のプロセッサで同時或いは先に処理されてしまう恐れがある．これによって，本来 Dijkstra のアルゴリズムでは生成されない無駄なコスト経路情報が発生する．しかし，コスト経路情報の優先度管理をしない分散アルゴリズムの最悪の場合の計算時間は，点の次数 (隣接する辺の数) が有界であっても点の総数の指数オーダになることもあり，たとえ管理が分散化されたとしてもコスト経路情報の優先度管理は探索の枝刈りをするために非常に有効である．またアルゴリズムは，Dijkstra のアルゴリズムと同等の計算時間オーダで最短経路を求めることが証明される．このアルゴリズムの計算時間オーダは，点の次数が n に依存しないある値で抑えられる時，$O(e \log n)$ である．

3 疎結合マルチプロセッサでの負荷分散

ここでは，疎結合マルチプロセッサ上で我々の分散アルゴリズムを効率良く実行するためのマッピング方式について考察する．10^4 以上の点を持つ大規模なグラ

フに対して，10^2 以上のプロセッサを持つ大規模な疎結合マルチプロセッサ上で問題を解くことを前提とする．

疎結合マルチプロセッサ上で効率良く負荷分散を行うには，主に2つの要素，すなわち，高プロセッサ稼働率とプロセッサ間通信の抑制が鍵となる．プロセッサ稼働率を上げるには，各プロセッサへ均等に仕事が割り付けられるように，問題全体を細かい部分問題に分割し，プロセッサ当たり多くの部分問題を割当てるようにする．一方，疎結合マルチプロセッサではプロセッサ間通信は非常にコスト高なので，プロセス間通信や一般のデータアクセスはできるだけ局所的に処理できるようにするのが望ましい．しかし，これは多くの場合，より多くの部分問題への分割と相反する結果となる．次の節では実験から，負荷分散方式がこれらの2つの要素に及ぼす影響を示す．

4 実験と解析

我々は，適当なグラフを設定し，疎結合マルチプロセッサであるマルチPSI上でいくつかのマッピング方式の実験を行った．その実験結果を以下に示す．

4.1 実験で用いたグラフ

実験で用いたグラフは，200 × 200 の正方格子上に配置された4万個の点に対してすべての隣り合う2点間に双方向の辺を与えた有向グラフである．隅の点のひとつを出発点とした．各辺のコストの与え方に従って，2種類のグラフを実験対象とした．

グラフ1: 各辺のコストを一定値としたもの．

グラフ2: 各辺のコストとして1から99までの非負整数の擬似乱数を与えたもの．

4.2 実験で用いた並列マシンと言語

実験は，並列推論マシンの実験機であるマルチPSI/V2[10]を用いて行った．これは，並列ソフトウェアの研究開発環境として新世代コンピュータ技術開発機構(ICOT)で開発された疎結合型のマルチプロセッサで，現在，8 × 8 のメッシュネットワークによって最大64台の要素プロセッサを接続することが可能である．各プロセッサの性能は130K append LIPS[1]である．

プログラムは，並列論理型言語KL1[4]を用いて記述された．グラフの各点とコスト経路情報はKL1のプロセスとして表現された．KL1では，本来のプログラムの意味を記述する部分と，並列実行に伴う負荷分散や実行優先度などを指定する部分が分離されている．後者はプラグマと呼ばれ，プログラムが効率良く実行されるために付加的に記述されるものであり，それによってプログラムの意味が変わることはない．プラグマは，プログラム中のボディゴールに書き添えることによって実現される．プロセッサへの点の割当てや優先度管理は，プラグマによって簡単かつ効率良く実現された．すなわち，優先度付き待ち行列はKL1処理系が提供する優先度制御機構を用いた．負荷分散方式はプロセッサ番号を算出するルーチンにより簡単に実現され，異なる負荷分散方式もプログラムの起動時にパラメータを変更することにより簡単に設定することができた．マルチPSI/V2で最初に実行されたのは優先度制御をしない最短経路アルゴリズムで，プロセッサ64台で $k = 4 \times 4$ の2次元多重マッピング方式を用いた場合，100 × 100 の正方格子状グラフ(辺のコストは乱数を用いた)の生成と最短経路の探索に約80秒を要した．その後，優先度管理を導入した新しい分散アルゴリズムが実現され，同様の問題を7秒(グラフ生成に6秒，解探索に1秒)で解いた．

[1] LIPS は *Logical Inference Per Second* の省略である．130K append LIPS は，1秒間に 130K 個のコンスセルをアペンドすることができることを意味する．

4.3 マッピング方式と実験結果の解析

試みたいくつかの負荷分散方式とその実験結果について述べ，簡単な解析を行う．

4.3.1 2次元単純マッピング方式

まず，我々は2次元単純マッピング方式と呼ぶ非常に単純な方式を試みた．$p = q^2$ 台のプロセッサを用いる場合，グラフを $q \times q$ 個のブロックに分割し，各ブロックを1つずつプロセッサ配置に従ってプロセッサに割当てる．例えば，16台構成の疎結合マルチプロセッサの論理プロセッサ配置が図2のようになっている時，グラフは 4×4 ブロックに分割される(図3)．プロセッサ P_0 には斜線部分のブロックが割当てられる．この方式は，同数の点を各プロセッサに割当てながら，グラフ内の局所性を最も良く保つものである．

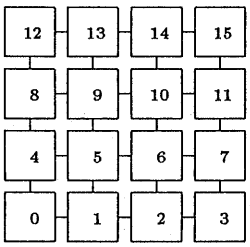

図2: 16台構成のマルチPSIの論理プロセッサ配置

グラフ2を用いた場合の実験結果を図9，10に示す．図9は，マッピング方式とプロセッサ台数を変化させて実行した場合の実行時間を表す．縦軸は実行時間(秒)，横軸は台数である．図10は，図9における台数効果を表したものである．縦軸は台数効果，横軸は台数である．2

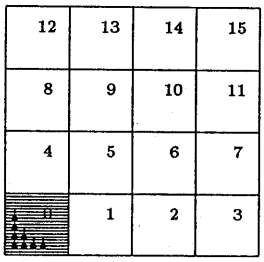

図 3: 2次元単純マッピング方式におけるグラフ分割

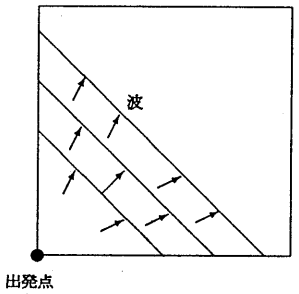

図 4: コスト経路情報生成の波

次元単純マッピング方式では, 結果はあまり良くないことがわかる.

理由について考えてみよう. コスト経路情報は最初に出発点の属するプロセッサで生成され, その後他のプロセッサへ波のように次第に広がっていく. 各プロセッサは波が来るまで暇な状態であり, 波が去った後再び暇な状態となるので, 暇な時間が比較的多いのである.

このマッピング方式における理想的な状態での台数効果を考える. プロセッサ間通信に要するオーバヘッドは無視できる位小さく, 部分問題への分割によって生じる無駄なコスト経路情報は一切発生しないと仮定する. すると, コスト経路情報の処理の波上に存在する点の集合は, 出発点からのマンハッタン距離[2]が互いに等しいものの集合となる. 出発点がグラフのある隅に位置する時, コスト経路情報の処理の波は, 図4のように進む.

このような仮定のもとでは, $p = q^2$ 個のプロセッサを用いて問題を解いた場合の実行時間と台数効果を以下のように評価することができる. 問題はプロセッサ台数と同数の部分問題に分割されるので, 部分問題は全部で q^2 個である. 図5で斜線で示された三角形の中の点に対して1プロセッサがコスト経路情報を処理するのに要する時間を T とする. 1プロセッサが問題全体を解くのに要する時間はグラフの点の総数に比例するので, 1プロセッサのみで実行した場合の実行時間は $2Tp$ となる. p 台のプロセッサで実行した場合に問題全体を解くのに要する時間は, $2Tq$ である. 従って台数効果は,

$$\frac{2Tp}{2Tq} = \frac{q^2}{q} = q = \sqrt{p}.$$

すなわちプロセッサ稼働率は $1/\sqrt{p}$ で, あまり良い値ではない. しかも, この値はプロセッサ台数の増加に従ってますます悪くなるのである. 実験結果から得られた台数効果は, グラフ1を用いた場合, 4台で 1.97, 16台で 3.87, 64台で 7.77, グラフ2を用いた場合はそれぞれ 1.69, 3.15, 6.10 となっており, プロセッサ数が4倍になる時, 台数効果は2倍弱となっていることがわかる.

我々は, 各マッピング方式について16台のプロセッサを用いてグラフ2に対して実行した場合の, プロセッサ稼働率と通信オーバヘッドを測定した (図11). 棒グラフは稼働率をパーセントで表したもの ((稼働時間 / 総実行時間) $\times 100$) で, 斜線部分がそのうちの通信に費やされた時間を表している. 実験結果から, この方式での平均プロセッサ稼働率は 24% という値が得られており, 期待される値 25% ($= 1/\sqrt{16}$) にかなり近い.

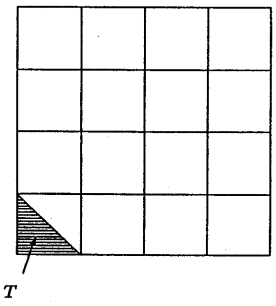

図 5: 2次元単純マッピング方式に対する評価

4.3.2 2次元多重マッピング方式

2次元単純マッピング方式では良いプロセッサ稼働率を得ることはできなかった. これは, 各プロセッサがグラフのある1部分のみを割当てられていたためである. 各プロセッサがグラフ中の分散した幾つかの部分を受け持てば問題は解決されるだろう. 2次元多重マッピング方式では, グラフはまず k 個の大きなブロックに分割され, その後さらに各ブロックが2次元単純マッピング方式と同様に p 個のブロックに分割される. 各プロセッサは大きなブロック当たり1部分ずつ, 合計 k 個のブロックを割当てられることになる. 図6は, $k = 4 \times 4$, $p = 4 \times 4$ の場合を示している. グラフはまず16個の大きなブロックに分割され, その後さらに大きなブロック

[2] 2次元平面上では, マンハッタン距離は点 $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ に対して, $d(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$ によって定義される.

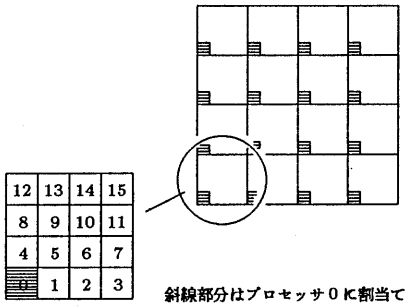

図 6: 2 次元多重マッピング方式におけるグラフ分割

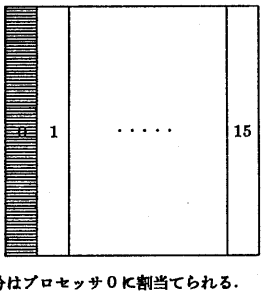

図 7: 1 次元単純マッピング方式におけるグラフ分割

が, 2 次元単純マッピング方式と同様にそれぞれ 16 個に分割される. プロセッサ P_0 は, 図の斜線部分を担当する.

このようにすることにより, 各プロセッサはコスト経路情報の処理の波がグラフ上に散らばった k 箇所の点集合を通り過ぎる際に稼働する. 従って, 各プロセッサの暇な時間は減少することが期待される.

実験結果から $k = 4 \times 4$ の時の台数効果は, グラフ 1 を用いた場合, 4 台で 3.14, 16 台で 9.39, 64 台で 26.14, グラフ 2 を用いた場合はそれぞれ 2.56, 7.25, 18.26 であった. 同様に $k = 8 \times 8$ の時の台数効果は, グラフ 1 を用いた場合, 4 台で 2.88, 16 台で 8.91, 64 台で 21.34, グラフ 2 を用いた場合それぞれ 2.71, 8.13, 20.25 となっており, プロセッサ台数に依存せず, 2 次元単純マッピング方式に比べてずっと良いプロセッサ稼働率が得られたことがわかる. 実際, プロセッサ 16 台で実行した場合のプロセッサ稼働率は $k = 4 \times 4$ の時 80%, $k = 8 \times 8$ の時 94.4% となっている (図 11). しかし, 実行時間は驚くほどには改良されていない. これは, 後で述べる無駄な見込み計算 (*speculative computation*) と通信オーバヘッドがかなり増大したためである.

4.3.3 1 次元単純マッピング方式

2 次元多重マッピング方式は, 1 プロセッサ当たり複数個のブロックを割当てることによってプロセッサ稼働率を改良しようとするものだった. 1 次元単純マッピング方式は, 最小のグラフ分割でプロセッサ稼働率を上げようとする方式である. ブロックと波ができるだけ垂直に交わるようにグラフを分割すれば, やはりプロセッサ稼働率を改良することができるだろう. そこで, グラフを単に p 個の細い短冊の形に分割し, それらを 1 つずつプロセッサに割当てる. 図 7は $p = 16$ の場合である.

2 次元単純マッピング方式に対して述べたのと同様の仮定のもとで, このマッピング方式の理想的な状態における台数効果を見積もる. 図 8で斜線で示された三角形ひとつの中の点に対して 1 プロセッサがコスト経路情報を処理するのに要する時間を T とする. すると, $7T$ 時間めに各プロセッサが処理するのは, 図中の点で

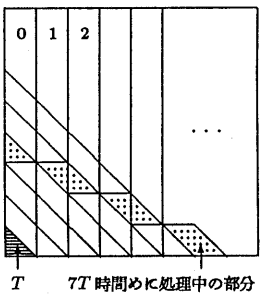

図 8: 1 次元単純マッピング方式に対する評価

示された部分となる. 結局, p 台のプロセッサを用いて問題を解くのに要する時間は, $T(3p-1)$ である. 1 プロセッサが問題全体を解くには $2Tp^2$ 時間かかるから, 台数効果は,

$$\frac{2Tp^2}{T(3p-1)} = \frac{2p^2}{3p-1} \approx \frac{2}{3} \cdot p \ (p >> 1\text{の時})$$

となり, プロセッサ台数の約 3 分の 2 の台数効果が得られることになる. これは, $\sqrt{p}$(2 次元単純マッピング方式) ではなく p に比例する, すなわち, プロセッサ台数が増加しても効果の率は下がらないことを意味している.

実験結果から, このマッピング方式では 2 次元多重マッピング方式での実行結果とほぼ同じ程度の性能が得られたことがわかる. 台数効果は, グラフ 1 を用いた場合, 4 台で 2.86, 16 台で 10.12 というように議論された値にほぼ近い値となっているが, 64 台では 29.34 で議論された値より悪い. これは, ブロックの境界で生じるプロセッサ間通信のオーバヘッドが原因と考えられる. グラフ 2 を用いた場合では, 4 台で 1.97, 16 台で 5.69, 64 台で 17.34 と, プロセッサ台数が増加するに従って次第に悪くなっている.

プロセッサ 16 台で実行した場合のこのマッピング方式での平均プロセッサ稼働率も 45% となっており, 期待される値 68%($= 2 \cdot 16/(3 \cdot 16 - 1)$) より悪い (図 11).

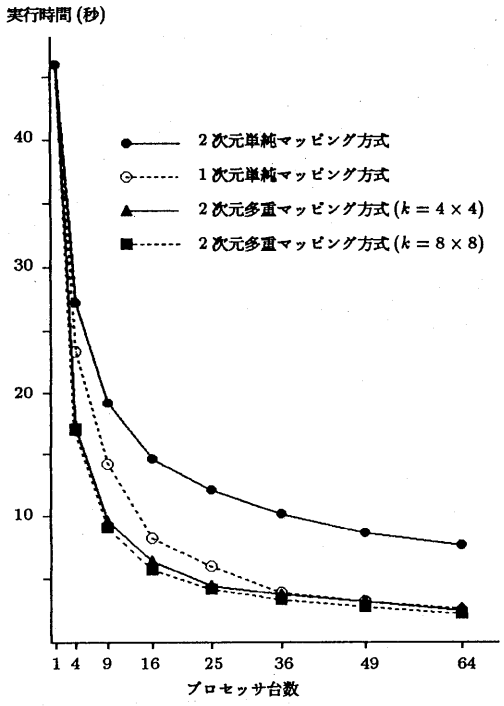

図 9: マッピング方式, プロセッサ台数と実行時間

4.4 議論

実験結果から分かるように, 最も単純なグラフと言えるグラフ 1 では殆ど無駄な見込み計算が生じないため, ほぼ理論的な台数効果を得ることができたといえる. 理論値より若干悪い値となった主な理由は, 通信オーバヘッドである. 一方, グラフ 2 での実験では, 台数効果は理論値よりも悪い値となった. 理由のひとつは通信オーバヘッドであり, もうひとつは無駄な見込み計算である.

グラフがマッピングのためにブロックに分割される時, ブロックの境界ではプロセッサ間通信が発生する. グラフが細かいブロックに分割されるほど, ブロックの境界の長さの合計は長くなり, プロセッサ間通信が増大する. 通信オーバヘッドが実際に占める割合はグラフの大きさやプログラム, また, 言語の実装方法, マシンなどに依存するが, ブロックの境界の長さの合計に比例する.

図 11では, 実際の稼働時間のうちの通信に費やされた部分とそうでない部分の比は, 2 次元単純マッピング方式で 5.3%, 2 次元多重マッピング方式の $k = 4 \times 4$ の場合で 19.2%, $k = 8 \times 8$ の場合で 34.1%, 1 次元単純マッピング方式で 10.2% となっている. これらをそれぞれ境界の長さの合計 $6L$, $30L$, $62L$, $15L$ で割ると (L はグラフ全体の 1 辺の長さ), 8.8%, 6.4%, 5.5%, 6.8%

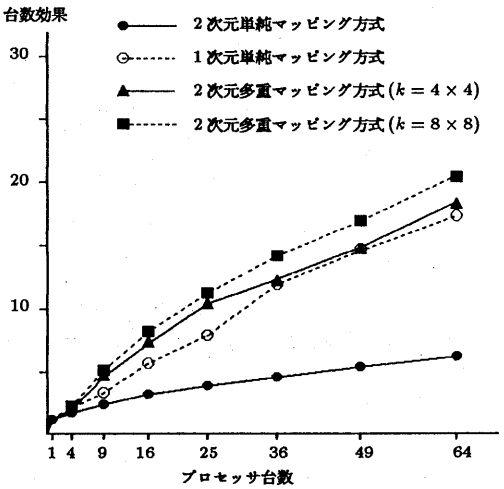

図 10: マッピング方式, プロセッサ台数と台数効果

で殆ど一定となっており, 期待される結果にほぼ一致している[3].

無駄な見込み計算量の解析は最も難しい. Dijkstra の逐次アルゴリズムではすべての計算が決定的で無駄がないのに比べ, 我々の分散アルゴリズムのコスト経路情報の処理では見込み計算が行われる. 何故なら, 生成されるコスト経路情報はもし後からもっと良い情報が得られた場合, 不必要となるかも知れないからである. そこで, 我々は, プロセッサ台数とマッピング方式を変化させた場合の, 実行中に生成されるコスト経路情報の総数を測定し, 1 台で実行した場合と比較した. 図 12はそれを示している. 超過分が最終的には無駄となったコスト経路情報の合計である. プロセッサ 16 台の場合, 2 次元単純マッピング方式では 20.2%, 2 次元多重マッピング方式の $k = 4 \times 4$ の時 39.1%, $k = 8 \times 8$ の時 20.2%, 1 次元単純マッピング方式では 17.2% となっている. 8×8 分割の時に 4×4 分割の時よりも無駄な見込み計算量が減少する理由ははっきりわかっていないが, 恐らくグラフの性質によるものであろう.

今回の結果の一般性については問題があるかも知れない. 今回の結果は, 局所性の高いグラフについて当てはまる. 例えば, 平面グラフや, 3 次元空間で点が一様な密度で配置され, 辺の長さが制限されているようなものがそうである. このようなグラフは, 実際の応用事例ではよく見られる. 辺が疎でも局所性のあまりないグラフは, マルチ PSI のようなメッシュネットワークで接続されたマルチプロセッサに割当てるのは難しいだろう. このようなグラフに対しては, ハイパーキューブ結合型マ

[3] 測定誤差は 5% ~ 10% である.

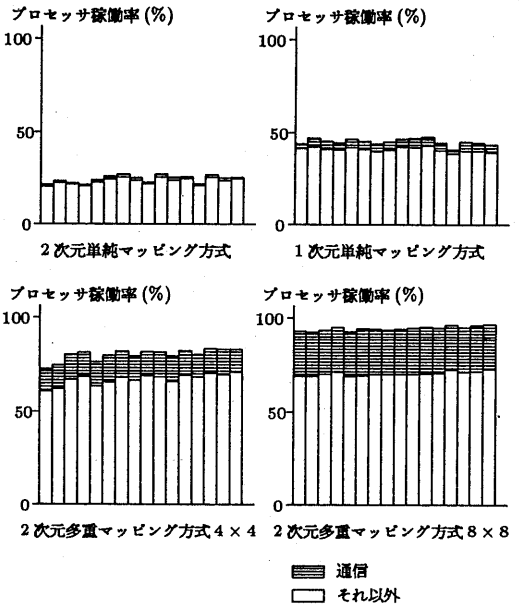

図 11: マッピング方式とプロセッサ稼働率 (プロセッサ 16 台の場合)

ルチプロセッサ[4]の方が良いマッピングがあると考えられる.

5 おわりに

最短経路問題は今まで様々な研究がされており, 数々の逐次アルゴリズムや並列アルゴリズムが開発されている. しかし, 10^2 以上のプロセッサの大規模汎用 MIMD マシン上で, 点の総数が 10^3 以上の大規模なグラフに対する最短経路問題についてはこれまであまり研究されていない. 我々は, 最短経路問題を解く 1 つの分散アルゴリズムを提案し, 疎結合マルチプロセッサであるマルチ PSI/V2 上での大規模グラフに対する最短経路問題の負荷分散方式について述べた. グラフの局所性を最も良く保つ 2 次元単純マッピング方式や, 高いプロセッサ稼働率を得ることが可能な 2 次元多重マッピング方式, また, 理論的には線形の台数効果を得ることが可能な 1 次元単純マッピング方式について実験を行った. 各マッピング方式における性能を測定し, 実際の台数効果と通信オーバヘッドについて議論した. 我々の実験は今のところ正方格子状のグラフに限られているが, 考察結果は, 局所性の高いグラフ一般に対して適用することが可能である. マッピング方式に関して観察された特徴の多くは, 1 点から全点への最短経路問題に限らず, コスト最小の極大木を求めるものや, 木の深さ

[4] プロセッサ台数を p とする時, プロセッサ間メッセージ通信に要する routing の最大値は, 2 次元メッシュネットワークで $O(\sqrt{p})$, ハイパーキューブネットワークで $O(\log p)$ である.

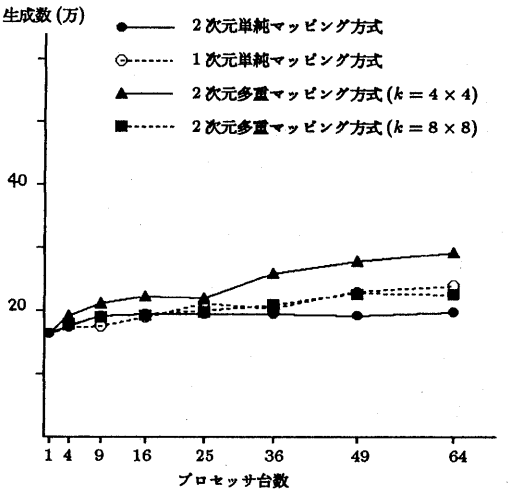

図 12: マッピング方式, プロセッサ台数とコスト経路情報の生成数

優先探索等, その他のグラフアルゴリズムに対しても同様に観察されるであろう.

KL1 言語について特記すべき事項は, 高い生産性と実験の容易さである. データ依存の同期機構は言語のセマンティクスとして用意され保証されているので, プログラマはプロセス間の同期について全く気を使う必要がない. KL1 では, プログラムの意味とマッピングは分離しているため, マッピング方式の変更がプログラムの意味の記述に影響することはないので, 異なるマッピング方式を簡単に試みることができた. 我々はまた, このように無駄な計算が生じるような場合にはアルゴリズムの計算量を減らすために優先度制御が必要不可欠であることを認識した.

6 謝辞

ICOT の淵一博所長, 内田俊一第 4 研究室室長にこの研究の機会を与えて下さったことを感謝します. 最初に優先度制御を用いた分散アルゴリズムの考えを提起された近山隆氏, 常に有益な示唆と助言をいただいた瀧和男氏, プロセッサ稼働率と通信オーバヘッドの測定をしていただいた中島克人氏 (現在, 三菱電機 (株) 情報電子研究所), この論文を査読して下さった六沢一昭氏, 様々な有益な助言を下さったその他の ICOT 第 4 研究室の研究員の方々に感謝します.

参考文献

[1] J. R. Driscoll, H. N. Gabow, R. Shrairman, and R. E. Tarjan. Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Compu-

tation. *Comm.ACM*, Vol.31, No.11 (Nov.1988), pp. 1343-1354.

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman. The Design and Analysis of Computer Algorithms, Addison-Wesley (1974).

[3] K. M. Chandy and J. Misra. Distributed Computation on Graphs: Shortest Path Algorithms. *Comm. ACM*, Vol.25, No.11 (Nov.1982), pp. 833-837.

[4] T. Chikayama, H. Sato and T. Miyazaki. Overview of the Parallel Inference Machine Operating System (PIMOS). In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988* (1988), pp. 230-251.

[5] N. Deo, C. Y. Pang and P. E. Lord. Two Parallel Algorithms for Shortest Path Problems. In *Proceedings of the 1980 International Conference on Parallel Processing*. IEEE, New York, 244-153, 1980.

[6] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1 (1959), 269-271.

[7] R. W. Floyd. Algorithm 97: Shortest Path. *Comm.ACM*, Vol.5, No.6 (1962), p. 345.

[8] M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J.ACM*, Vol.34, No.3 (July 1987), pp. 596-615.

[9] D. B. Johnson. Efficient Algorithms for Shortest Paths in Sparse Networks. *J. ACM*, Vol.24, No.1, pp. 1-13.

[10] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa, and T. Chikayama. Distributed Implementation of KL1 on the Multi-PSI/V2. In *Proceedings of the Sixth International Conference on Logic Programming* (1989), pp. 436–451.

[11] M. J. Quinn and N. Deo. Parallel Graph Algorithms. *ACM Computing Surveys*, Vol.16, No.3 (Sept.1984), pp. 319-348.

[12] S. Warshall. A Theorem on Boolean Matrices. *J.ACM*, Vol.9, No.1 (1962), pp. 11-12.