アルゴリズム 61－5
（1998．3．20）

再構成メッシュ上の並列アルゴリズムの視覚化ツール

渡辺 智雄　中野 浩嗣　林 達也

名古屋工業大学 電気情報工学科

近年，再構成メッシュ(Reconfigurable Mesh) を用いた並列アルゴリズムに関する研究が盛んにおこなわれ，数多くの有効な並列アルゴリズムが提案されている．しかし，再構成メッシュはアルゴリズムの実行中に動的にバスのトポロジが変化するなど動作の理解が困難であるといえる．したがって，本稿では再構成メッシュ上の並列アルゴリズムを視覚化するツールを提案し，ツールの利用者の支援を目指す．ユーザは視覚化部分のコードを意識することなくCライクな言語を用いることによって並列アルゴリズムを記述することが可能である．さらに，内部ポート接続の変更回数やブロードキャストの回数を表示し，厳密な性能評価もおこなうことが可能となる．

A Visualizing Tool for Parallel Algorithms on the Reconfigurable Mesh

TOMOO WATANABE, KOJI NAKANO, TATSUYA HAYASHI

Department of Electrical and Computer Engineering,
Nagoya Institute of Technology

Many parallel algorithms on the reconfigurable mesh have been developed so far. However, it is hard to understand the behavior of parallel algorithms, because the bus topology is dynamically changing during the execution of the algorithm. The main contribution of this work is to develop a tool for visualizing parallel algorithms on the reconfigurable mesh. It accepts parallel algorithms written by users in C-like language, without requiring the visualization codes. It also evaluates the exact performance of the algorithms.

1　はじめに

再構成メッシュ (Reconfigurable Mesh) はアルゴリズムの実行中に動的に再構成可能なバスをもつメッシュ状の並列 SIMD 型計算機モデルである．再構成メッシュの研究は，近年盛んにおこなわれており，画像処理やグラフ問題，ソーティングなど数多くの有効なアルゴリズムが提案されている [4].

このように再構成メッシュは強力な並列計算機モデルであるが，複数のプロセッサが同時に動作し，バスのトポロジーが動的に変化するため，アルゴリズムの設計や解析が困難となる．また，再構成メッシュはバスを用いてデータのブロードキャストをおこなうことができるため，通信の衝突が起こる可能性が

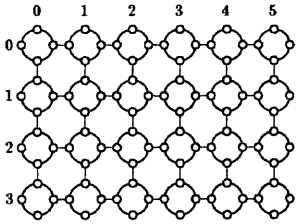

図 1: 4 × 6 の再構成メッシュ

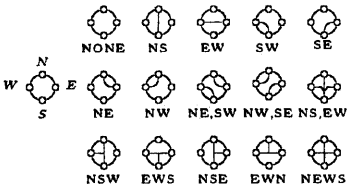

図 2: ポート内部の接続パターン

ある．アルゴリズムを設計する際に通信の衝突が起こらないように注意してアルゴリズムを設計しなくてはならないが，再構成メッシュはプロセッサごとに異なる動作をさせることが可能なため，通信の衝突の検出が困難である．このようにアルゴリズムの設計や解析を紙上でおこなうことはアルゴリズムが複雑になるにつれて困難になるといえる．この問題を解決するため，本稿では再構成メッシュ上での並列アルゴリズムの視覚化ツールを提案する．本ツールを用いることによって，再構成メッシュ上のアルゴリズムの設計やアルゴリズムの学習をおこなう者の支援をおこなう．

2 再構成メッシュモデル (Reconfigurable Mesh)

再構成メッシュ(Reconfigurable Mesh) はメッシュ上に並べられたプロセッサ (PE) とそれらを接続する通信用リンクにより構成される並列計算機である．4 × 6 の 2 次元の再構成メッシュを図 1 に示す．以降，本稿では 2 次元の再構成メッシュのみを対象とする．2 次元の場合，各プロセッサは 4 つの通信ポートをもち，それぞれの物理的な位置によって $N(ORTH)$ ポート，$E(AST)$ ポート，$W(EST)$ ポート，$S(OUTH)$ ポートと呼ばれる．これら 4 つのポートを介して上下左右に隣接するプロセッサと接続する．各プロセッサは同一のプログラムを実行し，完全に同期して動作する SIMD 型プロセッサであるが，入力データやプロセッサの物理的な位置によって異なる動作をさせることも可能である．

また，通常のメッシュ結合計算機と異なり，各プロセッサはプログラムの実行中にローカルにコントロール可能なスイッチを用いて図 2 の 15 パターンの中から 1 つのパターンを選ぶことにより，プロセッサ内部の接続パターンを変更することができる．これにより，プロセッサ間の通信リンクとプロセッサ内部の接続によりバスが構成される．プロセッサは 1 単位時間にこのバスを介してバス上の 1 つのプロセッサがデータをブロードキャストできる．また，プロセッサは 1 単位時間にバス上のデータを読むことができる．したがって，プロセッサは 1 単位時間に

- ポート内部の接続を変更する
- バス上にデータをブロードキャストする
- バスからデータを読む
- CPU 内部で算術演算などのローカルな処理をする

のいずれかを実行することができる．

3 並列アルゴリズム視覚化ツールの設計

再構成メッシュ上の並列アルゴリズムを開発する者，あるいは，学習する者にとって，より柔軟で強力な支援が可能な並列アルゴリズム視覚化ツールの設計と実装をおこなため，以下の点を考慮して設計をおこなった．

1. ユーザは視覚化部分を意識せずに並列アルゴリズムを記述できること．
2. ポートの内部接続の変更など再構成メッシュ特有の動作をわかりやすくユーザに提示すること．

3. 開発されたアルゴリズムのデバッグがおこなえるように，通信時に衝突がおこった場合はユーザに警告を促す．

4. 開発されたアルゴリズムのブロードキャストの回数などを測定し，アルゴリズムの性能を測定すること．

再構成メッシュのような並列計算機上の並列アルゴリズムの視覚化をおこなう場合，1 台の計算機上で並列アルゴリズムをシミュレーション実行するため，図 3 に示すように並列アルゴリズムにおける 1 ステップをプロセッサ (PE) の台数分だけ繰り返してから視覚化コードを埋め込む方法が考えられる．しかし，この視覚化コードを埋め込む作業をユーザに負担させる場合，本来の目的であるアルゴリズムの検証や設計，開発が困難になる．また，並列処理の教育用教材として使用されることも想定されるため，並列処理の初学者に対して，できるかぎり容易にアルゴリズムを記述できることが望ましいといえる．そこで，ユーザは視覚化コードをまったく意識せずにアルゴリズムを記述できるようにする必要がある．そのため，本ツールではユーザがアルゴリズムを記述したプログラムを視覚化コードを埋め込んだプログラムに変換する**トランスレータ**を用意することとした．これにより，ユーザは視覚化コードを意識することなくアルゴリズムの設計，あるいは理解に専念することができると考えられる．また，ユーザが並列アルゴリズムを記述するのに用いる言語は C ライクな言語を用いるものとする．

次に視覚化部分に関する設計について述べる．本ツールでは視覚化部分を**ビジュアライザ**と呼ぶこととする．本ツールでは表示を UNIX 上の X Window System によってビットマップディスプレイ上におこなうこととする．X Window System を用いることによって，オペレーティングシステム間の違いをできるだけ吸収し，どのようなプラットフォーム上でも実行できることを目指した．

再構成メッシュは 2 節で述べた通り，PE の物理的な位置や PE がもつ変数の値によって，1 単位時間 (ステップ) に (1) ポートの内部接続を変更する (2) バス上にデータをブロードキャストする (3) バスからデータを読む (4)CPU 内部で算術演算などのローカルな処理をするといった異なる動作をさせることが可能である．ここで，各ステップにおける動作の視覚

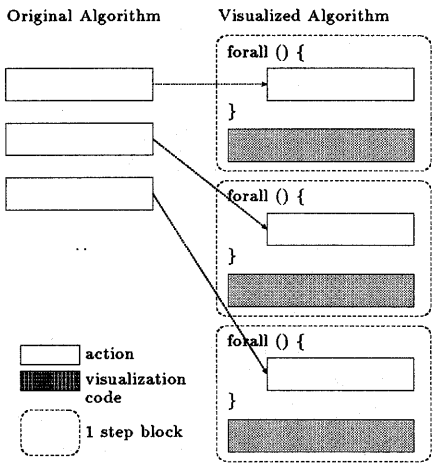

図 3: 視覚化コードの埋め込み

化について考える．上記の PE の動作をわかりやすくユーザに提示するため，PE の動作を 1 台の計算機上でシミュレートする．PE の台数を n とすると，図 3 に示したように並列アルゴリズムの 1 ステップに対して 各 PE の動作を n 回おこなってから表示をおこなう．このため，各 PE における状態を保存しておき，これをもとにして表示をおこなうこととした．次に，PE の各動作における表示の方法について述べる．まず，ポートの内部接続の変更に関してはその接続に対応した図 2 の 15 パターンのうちの 1 つを表示すればよい．さらに，ブロードキャストに関してはブロードキャストをおこなう PE の色を変更することによってどの PE がブロードキャストをおこなっているかをわかるようにした．ここでは，ブロードキャストをおこなう PE の色を赤に設定した．さらに，ブロードキャストされている様子を提示するために，ブロードキャストされた値が通るバスの色と幅を変更することによってバス上にデータが流れる様子を表現することとした．ここでは，ブロードキャストされた値が通るバスの色を青に設定した．ここで，通信の衝突が起こる可能性があるが，通信の衝突が起きた場合は警告ウィンドウを表示することによってユーザに注意を促すこととした．これにより，本ツールをアルゴリズムのデバッグに利用できると考えられる．また，CPU 内部でローカル

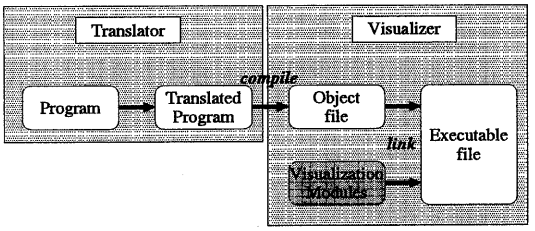

図 4: 本ツールの構成

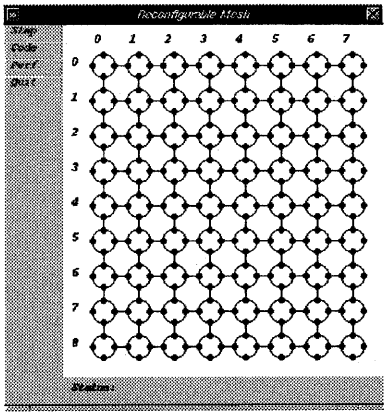

図 5: 起動直後のメインウィンドウ

な計算をおこなう場合，その過程を有効に表示することは困難であるといえる．これまで数多く提案されているソーティングアルゴリズムの視覚化の場合は値の比較に主眼がおかれているため，棒グラフなどが用いられているが，グラフ問題などでは棒グラフを利用することができない．このため，今回は変数名と変数の値を提示することとした．また，計算内容は表示できないため，ユーザの要求によって現在実行中のソースコードも表示することとした．次に，アルゴリズムの時間評価について考える．アルゴリズムやそれをもとにプログラムを記述すれば，時間評価はおこなうことは可能だが，ここではさらにポートの内部接続の変更回数とブロードキャストをおこなった回数を記録しておき，ユーザの要求時に提示することとした．これにより，例えば実行時間が定数時間の 2 つのアルゴリズムにおいて，どちらがよりブロードキャストの回数を少なくできるかといったより厳密な性能評価が容易におこなうことが可能となる．このため，ビジュアライザは PE の状態や変数を表示するメインウィンドウ，ソースコードの一部分を表示するソースコードモニタウィンドウ，ポートの内部接続の変更回数，ブロードキャストの回数を表示するパフォーマンスモニタウィンドウから構成される．

以上より，本ツールを実際に使用する際は以下の手順を踏むことになる．ツールのユーザは以下に示す手順を踏むことによって，アルゴリズムの正しさを検証することができる．図 4 に本ツールの構成を示す．

1. 再構成メッシュ上の並列アルゴリズムを C ライクな言語を用いてプログラムを記述する．このとき，表 1に示す関数群を用いて各 PE の動作を記述する (この記述はプロセッサの座標やローカルメモリの値による分岐を含む).
2. 記述したプログラムをトランスレータを用いて本ツール用のプログラムに変換する．
3. 変換されたプログラムを UNIX の make コマンドを用いてコンパイルし，本ツールで提供される視覚化モジュールとリンクすることによって実行可能ファイルを得る．
4. 得られた実行可能ファイルを実行することによってアルゴリズムを視覚的に確認することが可能となる．

4 実行例と本ツールの評価

4.1 入力 n ビット中の 1 の数のカウント

ここでは，入力 n ビット中の 1 の数をカウントするアルゴリズム [6] を用いて並列アルゴリズム視覚化ツールの実行例を示す．

ビジュアライザを $ncol = 8$, $nrow = 9$ で起動した場合，図 5 のメインウィンドウが表示される．背景の白い部分にプロセッサとそのインデックス (座標) が表示される．さらに，現在おこなわれている動作が左下に，現在表示している変数名が右下に表示される．また，左側には各操作ボタンが表示される．ここで，*Code* ボタンと *Perf.* ボタンを押すこ

表 1: 本ツールが提供する関数

connect_port()	指定にしたがってポートの内部接続を変更
reset_port()	ポートをリセット
broadcast()	指定されたポートから値をブロードキャスト
read_bus()	バスから指定されたポートにデータを読む
show_data()	変数の値を表示する
hide_data()	変数の値の表示を隠す
make_input()	入力データの生成
row()	行方向のインデックス (座標)
col()	列方向のインデックス (座標)

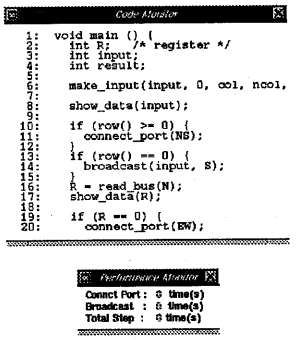

図 6: 起動直後のコードモニタウィンドウとパフォーマンスモニタウィンドウ

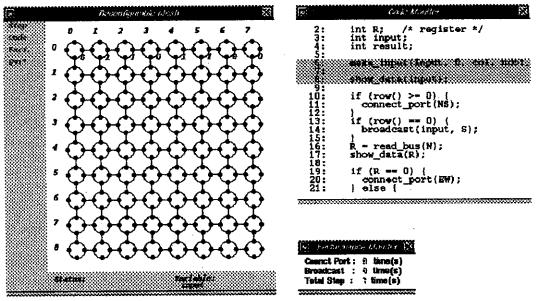

図 7: 入力が変数 input にストアされる

とにより，コードモニタウィンドウとパフォーマンスモニタウィンドウがあらわれる (図 6).

以下，*Step* ボタンを押していくことにより，アルゴリズムがステップ実行される．

まず，**make_input()** によってランダムに生成された 2 進数が図 7 に示すように第 0 行の変数 input にストアされる．これは各プロセッサの右下に変数が表示されていることによって確認できる．

次に，各プロセッサが N ポートと S ポートを接続することにより，列バスが形成される (図 8)．この列バスを用いて縦方向にブロードキャストをおこない，入力データを分配する．

列バスが形成されたのち，図 9 に示すように第 0 列の PE が変数 input を S ポートからブロードキャストする．このとき，ブロードキャストをおこなうプロセッサは赤色になり，データの通り道となるバスは青色に変化し，バス幅も太くなる．

次のステップで，ブロードキャストされた値をバスから読み，各 PE の変数 input にストアする (図 10)．これで，第 i $(0 \leq i \leq n-1)$ 列の PE は i 番目の入力をもつことになる．

次に，変数 input の値によってポートの内部接続を変更する．変数 input の値が 0 であれば，E ポートと W ポートを接続し，1 であれば，N ポートと E ポート，S ポートと W ポートをそれぞれ独立に接続する (図 11)．

次に，図 12 に示すように，最も左上のプロセッサ，つまり，PE(0,0) が 値 1 を W ポートからブロードキャストする．

最後に，図 13 に示すように最も右端の列の PE，つまり，PE $(i, n-1)(0 \leq i \leq n-1)$ がバスから値を読み，変数 result にストアする．

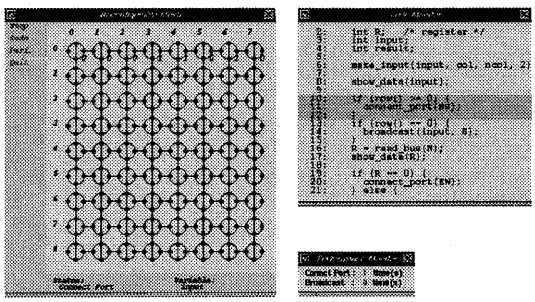

図 8: 列バスの形成

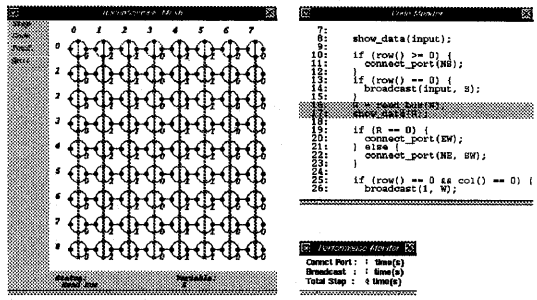

図 10: バスから値を読み，変数 input にストア

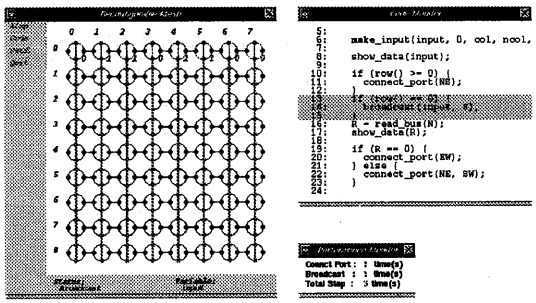

図 9: 入力のブロードキャスト

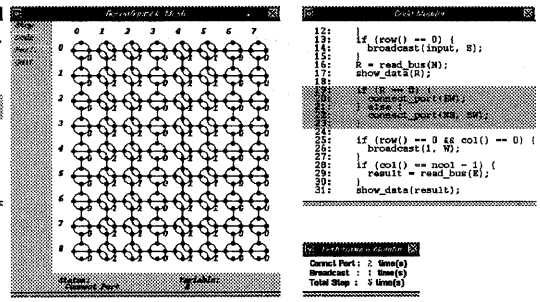

図 11: ポートの内部接続の変更

もう一度，*Step* ボタンをおすと，図 14 に示すように *Step* ボタンが *Done* ボタンにかわり，アルゴリズムが終了したことがわかる．

4.2 アルゴリズムの性能評価

本ツールを用いることにより，再構成メッシュ上の並列アルゴリズムの厳密な性能評価をおこなうことが可能となる．以下の 6 つのアルゴリズムを考える．

- サイズが $(n+1) \times n$ の再構成メッシュ上で入力 n ビット中の 1 の数を数えるアルゴリズム [6].
- サイズが $2dn \times 2n$ の再構成メッシュ上で n 個の d ビット整数の加算をするアルゴリズム [3].
- サイズが $n \times n$ の再構成メッシュ上で n 個の数の最大値を求めるアルゴリズム [1].
- サイズが $n \times n^2$ の再構成可能メッシュ上で n 個の数のソートをおこなうアルゴリズム [6].
- サイズが $n \times e$ の再構成メッシュ上で頂点数 n，辺数 e のグラフの到達可能頂点問題アルゴリズム [5].
- サイズが $ne \times e$ の再構成メッシュ上で頂点数 n，辺数 e のグラフの最小全域木を求めるアルゴリズム [5].

これらのアルゴリズムはすべて定数時間のアルゴリズムであるが [5]，表 2 に示すように同じ定数時間のアルゴリズムでもポートの内部接続の変更回数やブロードキャストの回数，総ステップ数が異なる．このようにパフォーマンスモニタを用いることによって実際に必要なステップ数でアルゴリズムの評価をおこなうことが可能となる．これにより，同じ時間複雑度のアルゴリズムでもより少ないステップ数のアルゴリズムを設計する際の支援をおこなうことが

表 2: 定数時間のアルゴリズムの比較

アルゴリズム	ポートの内部接続の変更 (回)	ブロードキャストの回数 (回)	総ステップ数 (ステップ)
n ビットの 1 の数をカウント	2	2	7
n 個の d ビット整数の加算	2	3	10
n 個の数の最大値	2	3	12
n 個の数のソート	6	7	29
グラフの到達可能頂点問題	2	3	12
最小全域木	4	8	29

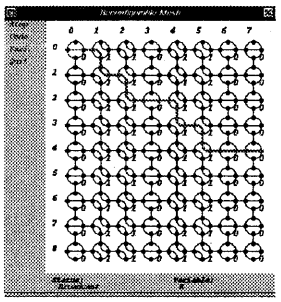

図 12: PE(0,0) が値 1 をブロードキャスト

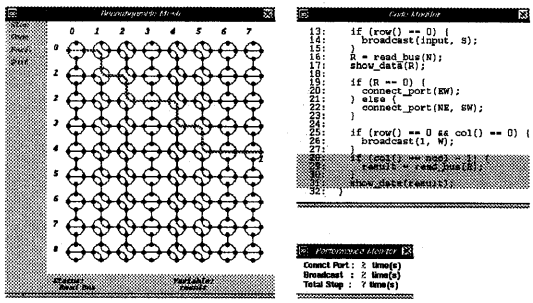

図 13: バスから値を読み，変数 result にストア

可能となる．

さらに，ここでは示さなかったが通信の衝突が検出された場合，図 15 に示すようなワーニングウィンドウが表示される．これによって，通信の衝突の検出をおこなうことが可能となる．

5 まとめ

再構成メッシュ上の並列アルゴリズムの視覚化ツールを提案した．本ツールはユーザがアルゴリズムを記述したプログラムをツール用のプログラムに変換するトランスレータと記述されたアルゴリズムの表示をおこなうビジュアライザから構成される．本ツールを用いることによって以下の利点が挙げられる．

- ユーザが自由に並列アルゴリズムを記述することができ，アルゴリズム記述の際に視覚化部分は意識する必要がない．
- ポートの内部接続の変更の様子やブロードキャストといった再構成メッシュ特有の動作が視覚的に確認できる．
- 並列アルゴリズムを記述したソースコードを参照しながらアルゴリズムの確認をすることが可能である．
- ポートの内部接続の変更回数とブロードキャストのおこなわれた回数，アルゴリズムの総ステップ数を表示させることができ，アルゴリズムの性能評価にも利用できる．

しかし，以下に示すような欠点もある．

- 変数は表示されるが，ソースコードを読む以外に内容を確認する手段がない．
- ステップ実行しかサポートされていないため，デバッグをおこないたい場合は若干不便である．

このようにいくつかの欠点もあるが，本ツールを用いることによって，並列アルゴリズムを記述する際に視覚化部分を意識せずに記述できるため，理解しようとしている，または，開発しようとしているアルゴリズムに専念することができる．また，記述されたアルゴリズムをわずかな手順で視覚化して表

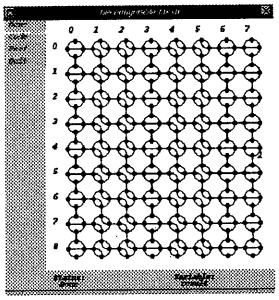

図 14: アルゴリズムの終了

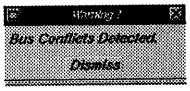

図 15: 通信の衝突がおきた場合のワーニングウィンドウ

示することができるため，再構成メッシュ上のアルゴリズムの開発や設計をおこなうものに対して，設計や開発の支援をおこなうことができ，アルゴリズムの設計，開発の効率向上が期待できる．さらに，C 言語の知識があれば，容易に並列アルゴリズムを記述することが可能であるため，並列アルゴリズムの学習者がアルゴリズムを学ぶ際の理解の手助けになるのではないかと考えられる．なお，本ツールは，`http://camphor.elcom.nitech.ac.jp/Software/` から入手可能である．

謝辞

本研究の一部は文部省科学技術研究補助金奨励研究 (A)(09780262) 及びカシオ科学振興財団の助成金による．

参考文献

[1] S. G. Akl: Parallel Computation: Models and Methods, *Prentice Hall,* 1997.

[2] Y. Ben-Asher, D. Pelog, R. Ramaswami, and A. Shuster: The Power of Reconfiguration, *Journal of Parallel and Distributed Computing,* 13:139-153, 1991.

[3] J. - W. Jang, and V. K. Prasanna: An optimal sorting algorithm on reconfigurable mesh, *Proc. 6th International Parallel Processing Symposium,* 130-137, IEEE, 1992.

[4] K. Nakano: A Bibliography of Published Papers on Dynamically Reconfigurable Architectures, *Parallel Processing Letters,* Vol. 5 No. 1, 1995

[5] K. Nakano: Constant-time algorithms on the reconfigurable meshes, *Journal of IPSJ,* Vol. 38, No. 11, 1019-1025, 1997

[6] B.-F. Wang, G.-H. Chen, and F.-C. Lin: Constant time sorting on a processor array with a reconfigurable bus system, *Information Processing Letters,* 34(4): 187-192, April 1990.