社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

性能推定が容易な転送／演算分離型並列画像処理システムの構成と評価

吉田　昌司† 　池田　光二† 　高根　淳††

† (株) 日立製作所日立研究所 情報制御第一研究部　〒319–1225 茨城県日立市大みか町7–1–1
†† (株) 日立ハイテクノロジーズ システム開発部　〒312–0033 茨城県ひたちなか市市毛882
E-mail: †{syoshida,ikeda}@hrl.hitachi.co.jp, ††takane-atsushi@naka.hitachi-hitec.com

あらまし　並列画像処理システムにおいて、性能を不確定にする要素を低減するために、転送と演算を分離した並列画像処理アーキテクチャを提案する。(1) 転送と演算を分離した並列画像処理アーキテクチャ、(2) フィルタサイズを考慮した均等データ分割、(3) フラグを用いた同期制御方式、により、性能を不確定にする要素を極力低減する。本提案アーキテクチャの効果を実機で評価した結果、実用上問題のない性能推定が可能であることを確認できた。

Transfer/Execution-divided Parallel Image Processing System Easy to Estimate Performance

Shoji YOSHIDA†, Mitsuji IKEDA†, and Atsushi TAKANE††

† Hitachi Research Laboratory, Hitachi Ltd.　Omika 7–1–1, Hitachi, Ibaragi, 〒319–1225, Japan
†† System Develop Division, Hitachi High Technologies, Ichige 882, Hitachinaka, Ibaragi, 〒312–0033, Japan
E-mail: †{syoshida,ikeda}@hrl.hitachi.co.jp, ††takane-atsushi@naka.hitachi-hitec.com

Abstract　We propose the parallel image processing architecture that translation and execution are separated, in order to decrease the cause which make the performance uncertain. (1)parallel image processing architecture that translation and execution are separated, (2)even data division that the size of filter is considered, and (3)synchronous control method with flag, decrease the cause which make the performance uncertain as much as possible. Evaluating effect of this proposal architecture with real machine, it was confirmed that we were able to presume the performance without the problem on practical use.

Key words　並列処理、画像処理

1. まえがき

高速計算を実現する目的でさまざまな並列計算機が研究／開発されており、作成しようとする並列プログラムの性能を推定するのに用いられる並列計算モデルの研究が進んでいる [1] 〜 [8]。しかし、上記のような並列計算モデルでは、プロセッサ間の転送処理の性能推定に確率モデルが使用されたり、キャッシュやパイプライン処理などのプロセッサアーキテクチャの考慮が十分でなかったりするなど、性能を推定する上で不確定な要素が多く、実際のケースでは個々に性能が変動する。推定の精度を上げるためには、モデルを詳細にすることが求められるが、性能を知るために必要なパラメータ数が多くなり、推定がしにくくなる。従って使いやすいモデルにするためには、パラメータをあまり多くせずに主要なものに限定することが必要であるが、少ないパラメータでは精度が上がりにくいといったトレードオフがある。このような課題は、均等な処理分割が比較的やりやすく、並列化の効果があらわれやすいとされる画像処理の分野においても同様に生じている。

さらに、性能推定の後には、それをもとに演算削減などにより高速化の改善を行うが、性能向上のためのプログラム変更により、性能不確定要素の条件が変化し、高速化を図ったつもりがむしろ性能が悪化してしまう現象が発生し、並列プログラムの高速化を困難にする要因となっている。

このような課題の解決策として、筆者らは、性能を不確定にする要素を極力低減するために、転送と演算を分離した並列画像処理アーキテクチャを提案する。本提案のアーキテクチャでは、転送プロセッサによる転送処理と演算プロセッサによる演算処理を完全に分離することにより、性能推定の不確定要素が入りにくく、性能評価が容易であるのが特徴である。

2. 性能推定の不確定要因

従来の並列処理アーキテクチャでは、正確な性能の推定は根

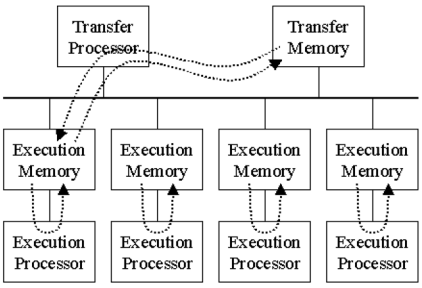

図1　転送／演算分離型アーキテクチャ
Fig. 1　Transfer/Execution-divided Architecture

本的に困難であった。その理由を挙げると以下のようになる。
［課題 1］ 並列処理のオーバヘッドとなるプロセッサ間の転送と、並列化すべき演算を明確に区別することができず、転送と演算にかかる時間を計算モデルにより推定している。しかし、計算モデルが前提としている諸々の条件が成り立たず性能の推定に大きな誤差を生じてしまうことが多い。
［課題 2］ 転送と演算の混合した処理をもとに、プロセッサへの負荷分担を決定しているため、複数のプロセッサへの負荷分散は大単位でしかできず、必ずしも均等に負荷を分散できない。
［課題 3］ 複数プロセッサの同期に割り込みなどを使用しているため、同期に要する時間がばらつきをおこし、性能推定が不確定な要因となる。

これらの課題に対して、並列処理のオーバヘッドとなる転送部分はすべて転送プロセッサが管理し、複数の演算プロセッサは、並列化されるべき正味の演算のみを行うアーキテクチャを提案する。次章で、それぞれの課題に対する解決策を示す。

3. 転送／演算分離型アーキテクチャ

3.1 アーキテクチャの構成

課題1に対しては、プロセッサ間の転送のみを行う転送プロセッサと演算のみを行う演算プロセッサに分けるアーキテクチャを採用することにより、解決できる。図1に、本提案の転送／演算分離型アーキテクチャの構成を示す。転送／演算分離型アーキテクチャは、転送メモリを持った転送プロセッサと、それぞれ演算メモリを持ったn個の演算プロセッサから構成する。さらに、転送部分はすべて転送プロセッサが管理し、演算プロセッサは正味の演算のみを行うように、役割を分担する必要性から、転送プロセッサ、演算プロセッサを以下のルールに従うように性格付ける。

- 転送プロセッサは転送メモリ、全演算プロセッサの演算メモリのいずれもアクセスできる。
- 演算プロセッサは、自分の演算メモリのみにアクセスする。転送メモリや他の演算プロセッサの演算メモリはアクセスできない。

つまり、プロセッサ間のデータの移動はすべて転送プロセッサに管理させる。

図2に、１つの演算プロセッサに対する基本的な処理手順を示す。処理の手順は以下のようになる。

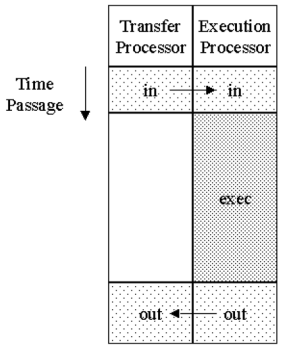

図2　基本処理手順
Fig. 2　Fundamental Process Sequence

［手順 1］ 転送プロセッサによる入力転送処理(in)
転送プロセッサは転送メモリから演算プロセッサの演算メモリへ、演算プロセッサの必要とするプログラムとデータを転送。
［手順 2］ 演算プロセッサによる演算(exec)
演算プロセッサは、演算メモリからプログラムとデータを読み出し、必要な演算処理を行い、演算結果を演算メモリに書きこみ、終了を転送プロセッサに報告。
［手順 3］ 転送プロセッサによる出力転送処理(out)
転送プロセッサは、演算プロセッサの演算メモリから演算結果のデータを読み出し、転送メモリへ転送し、転送メモリ上のしかるべき場所に格納。

このように、並列処理のオーバヘッドとなる転送部分はすべて転送プロセッサが管理し、演算プロセッサは、正味の演算のみを行うという役割の分担が達成される。

3.2 データ分割

プロセッサ間の転送のみを行う転送プロセッサと演算のみを行う演算プロセッサに分けることにより、転送と演算の分離が可能になると、次には演算をn個の演算プロセッサに均等に分割することが必要となる（課題2)。

複数のプロセッサに処理を分割する方法には、データを分けるデータ分割とタスクを機能的に分けるタスク分割がある。タスク分割は、複数のプロセッサに対し負荷を均等に分けることが困難であり、この負荷のばらつきが性能推定の不確定要素となる。複数のプロセッサに均等に負荷を分けられ並列処理の効率を高めることのできる、データ分割を採用する。画像処理では分割されるデータは画像であり、処理すべき1枚の画像を均等にn個に分割する必要がある。画像処理のうち、フィルタ演算の場合、フィルタの大きさ分の入力画像がそろわないと出力画像が計算できないため、出力画像の処理領域の周囲には、フィルタサイズに応じて計算できない領域(undefine領域)ができる。このundefine領域を考慮しないと、均等分割にならない。

図3, に、均等な画像分割の概要を示す。

横分割において、入力画像の高さをH_{in}、出力画像の高さをH_{out}、フィルタのカーネルサイズをk(図では$k=3$の例を示している)とおくと、undefine領域の幅が上下左右$(k-1)/2$であることより、

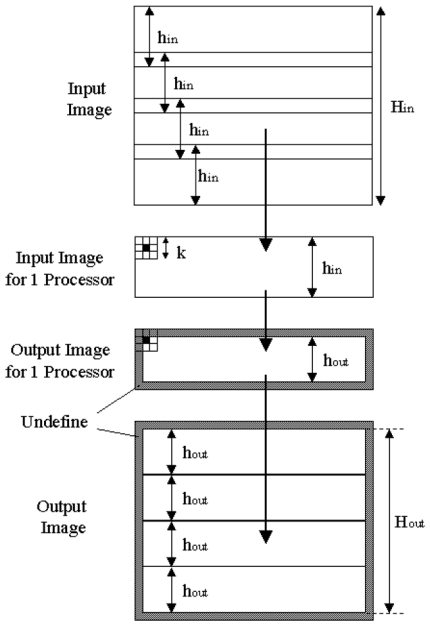

図3　横　分　割
Fig. 3　Horizontal Division

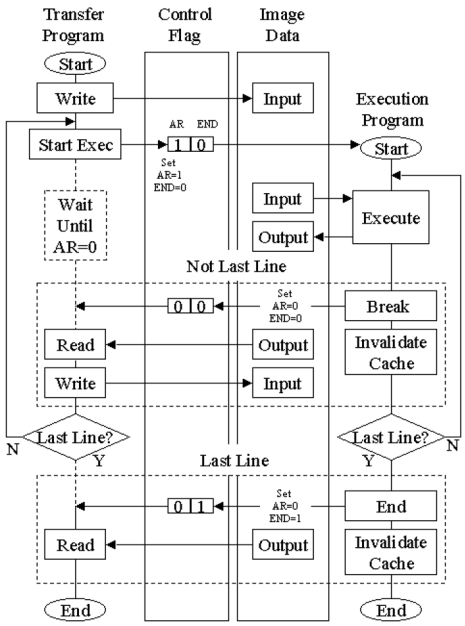

図4　同期制御のフローチャート
Fig. 4　Flowchart of Syncronous Control

$$H_{out} = H_{in} - (k-1)$$

これを、n個の演算プロセッサに均等に分割すると、1プロセッサが担当する出力画像の高さh_{out}は、

$$h_{out} = \frac{H_{out}}{n} = \frac{H_{in} - (k-1)}{n}$$

さらに、undefine領域を含む出力部分画像を集めて1つの出力画像に統合するためには、各演算プロセッサに割り当てられる入力部分画像はフィルタのカーネルサイズだけ重なる共通部分を持たせる必要があるため、1プロセッサが担当する入力画像の高さh_{in}は、

$$h_{in} = h_{out} + (k-1) = \frac{H_{in} - (n-1)(k-1)}{n}$$

となる。縦分割においても、同様である。

このように、フィルタの大きさまで考慮して画像データを均等に分割することにより、各演算プロセッサに与えられる負荷が均等になり、性能推定の不確定要素の一つである負荷のばらつきを低減することができる。さらに、縦分割と横分割を比較した場合、縦分割は、各演算プロセッサに割り当てられるべき転送メモリ上のデータの各ラインが転送プロセッサのキャッシュ上で離散的に配置されるのに対し、横分割は各ラインが転送プロセッサのキャッシュ上で連続的に配置されるため、横分割の方が転送プロセッサでキャッシュミスが起こりにくく、性能が高い上に性能のばらつきも小さくなり性能評価もやりやすくなる。

3.3　同 期 制 御

転送プロセッサと演算プロセッサが図2のように協調して処理をすすめていくためには、両者の同期をとることが必要である。演算メモリを効率的に使用するためには、1画像より小さい処理単位のデータをやりとりし合いながら処理を進め、それを積み重ねて1画像分の処理を実現することが望ましい。このとき、同期に要する時間がばらつきをおこすと、性能推定が不確定な要因となる。課題3の一つの解決案として、我々が実現したフラグを用いた同期制御方式を説明する。

転送単位ごとの転送プロセッサ・演算プロセッサの同期をとるために、演算メモリアクセス権ビットと処理終了ビットという2ビットの同期制御フラグを設ける方式とする。演算メモリアクセス権ビットは、転送プロセッサ、演算プロセッサどちらに演算メモリのアクセス権があるかを示しており、転送プロセッサ、演算プロセッサはそれぞれ自分の処理が終わると相手にアクセス権を渡し、次にアクセス権が戻ってくるまで待つ、というしくみにする。処理終了ビットは、処理すべき1画像分のデータの処理を終えたことを演算プロセッサが転送プロセッサに報告する。これにより転送プロセッサのデータ転送と、演算プロセッサの演算が順序よく行われていくことを保証する。

図4に、同期制御の基本フローチャートを示す。転送プロセッサは、処理すべき画像データを転送した後、演算プロセッサにアクセス権を渡し、待ち状態にはいる。演算プロセッサは、演算を終了すると、転送プロセッサにアクセス権を返し、待ち状態にはいる。処理を返された転送プロセッサは、演算結果データを演算プロセッサから転送し、次のラインの画像データを転送し、また演算プロセッサに制御をうつす、という繰り返しとなる。処理終了時には、演算プロセッサは、処理終了ビットを立てて返す。転送プロセッサは、演算プロセッサから処理を戻されるたびに処理終了ビットをチェックし、処理途中で演算終了ビットが返ってきたり、演算終了なのに処理終了ビットが返ってこなかったりした時はエラーと判定して、エラー処理に移行する。以上のように、たった2ビットの制御フラグで、転送プ

ロセッサと演算プロセッサの同期処理が実現できる。

このような簡単な同期制御方式により、同期に要する時間を短く、かつ時間のばらつきを小さくすることができ、性能推定の精度を向上させることができる。

4. 転送／演算分離による性能推定

転送／演算分離型アーキテクチャでは、転送と演算を分離したことにより、転送プロセッサによる転送時間T_{tran}と、演算プロセッサによる演算時間T_{exec}のみで全体の性能推定を決定することができる。以下にそれを示す。4.1では、性能推定の要素となる転送時間T_{tran}、演算時間T_{exec}を定義し、4.2で、転送時間T_{tran}、演算時間T_{exec}よりシステム全体の処理時間の見積りの計算を示す。

4.1 転送時間、演算時間の定義

転送プロセッサによる転送時間T_{tran}は、並列化に伴うオーバヘッドにかかる時間であり、1行分の転送時間、転送プロセッサと演算プロセッサの同期にかかる時間、キャッシュコヒーレンシのためのキャッシュ無効化（dirty書き出し)の時間を含む。これらは、画像サイズ、画像枚数が決まれば、固定時間となる。

演算プロセッサによる演算時間T_{exec}は、並列化されるべき部分であり、演算時間、演算メモリへのアクセス時間を含む。従って、演算プロセッサによる演算時間には、1プロセッサで実行する通常の逐次処理と全く同様となり、逐次処理時間T_1を見積もればよい。つまり、

$$T_{exec} = T_1$$

4.2 全体処理時間、速度向上比の見積り

前章のT_{tran}、T_{exec}をもとにして、本アーキテクチャ全体の処理時間T_{system}と速度向上比S_{system}を求める。速度向上比(speed up)は、並列処理の効果を示す一つの指標で、以下で定義される。[11]

$$S_{system} = \frac{T_1}{T_{system}}$$

（1） $T_{exec} < T_{tran}$の場合

$T_1 = T_{exec}$、$T_n = T_{tran}$ であるから、

$$T_1 < T_n$$

並列化するメリットがなく、逐次処理すべきである。従って、全体処理時間と速度向上比は、以下であらわされる。

$$T_{system} = T_1 = T_{exec}$$
$$S_{system} = 1$$

（2） $T_{tran} < T_{exec} < (n-1)T_{tran}$の場合

図5に示す。この時、転送プロセッサの転送により処理時間が決定し、転送ボトルネックとなる。演算プロセッサには、転送プロセッサの待ち状態となる空き時間が生じている。全体処理時間と速度向上比は、以下であらわされる。

$$T_{system} = T_{tran}$$
$$S_{system} = \frac{T_1}{T_{system}} = \frac{T_{exec}}{T_{tran}}$$

（3） $T_{exec} > (n-1)T_{tran}$の場合

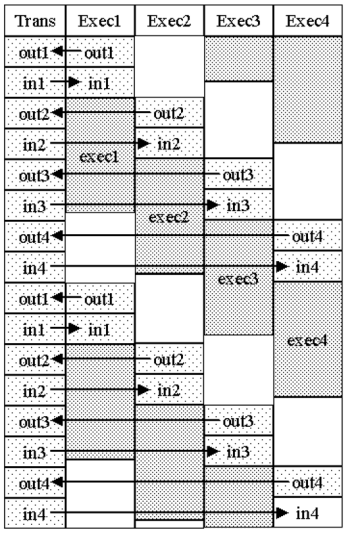

図5 転送ボトルネック
Fig. 5 Transfer Bottleneck

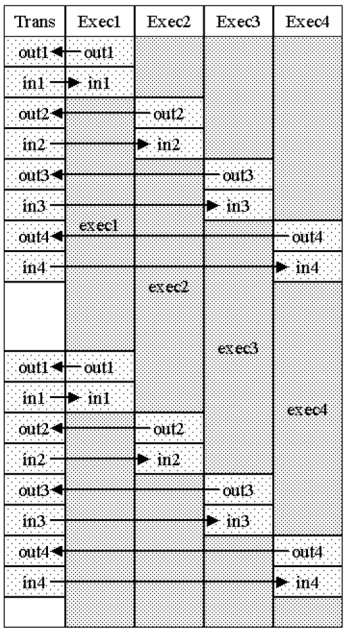

図6 演算ボトルネック
Fig. 6 Execution Bottleneck

図6に示す。この時、演算プロセッサの演算により処理時間が決定し、演算ボトルネックとなる。転送プロセッサには、演算プロセッサの待ち状態となる空き時間が生じている。全体処理時間と速度向上比は、以下であらわされる。

$$T_{system} = \frac{T_{tran} + T_{exec}}{n}$$
$$S_{system} = \frac{T_1}{T_{system}} = \frac{nT_{exec}}{T_{tran} + T_{exec}}$$

以上をまとめ、$n = 4$の時のグラフをかくと、図7、図8のようになる。図7は、演算転送比と全体処理時間の関係を表している。転送ボトルネックでは、全体時間が転送時間で決まるため固定時間となるが、演算ボトルネックでは、演算比率が上昇

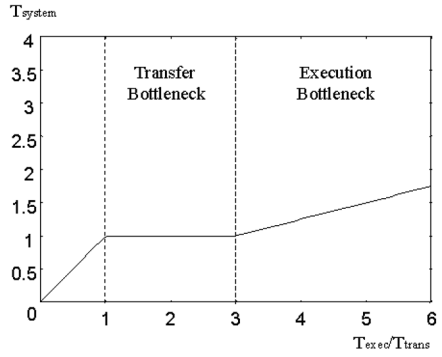

図7　全体処理時間

Fig. 7　Total Execution Time

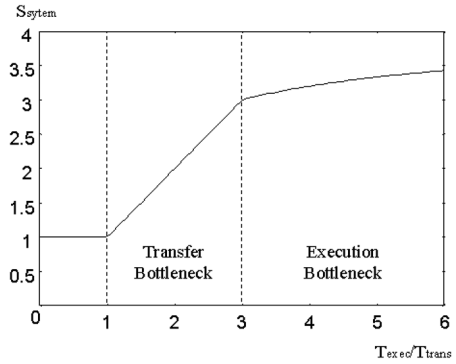

図8　速度向上比

Fig. 8　Execution Speed up

するにつれ全体時間もゆるやかに上昇する。図8は、演算転送比と速度向上比の関係を表している。転送ボトルネックでは、並列効果は1倍から3倍の間であり、演算ボトルネックでは、並列効果は3倍から4倍にむけ上昇することがわかる。

このように、転送／演算分離型並列画像処理アーキテクチャでは、演算時間と転送時間の比で、性能推定を完全に確定することができ、演算と転送のどちらがボトルネックになっているかを考えるだけで性能見積りを決定することができる。他のマルチプロセッサ方式より単純な方法で性能推定が可能である。

4.3　複合関数合成による性能向上

転送／演算分離型並列画像処理アーキテクチャでは、演算と、転送とを完全に分離して考えられるので、演算ボトルネックのときは演算のみを、転送ボトルネックの時は転送のみを対策すればよく、性能向上しやすい。さらに、転送ボトルネックの関数を複合すると、転送が削減されて性能が向上する。

図9、図10に、複合関数合成による性能向上の手法を示す。転送ボトルネックの関数A、Bがあり、Aの出力がBの入力となって、連続して処理されるとする。このとき、関数Aがマスタに結果画像を戻す処理と、関数Bでマスタが入力画像を転送する処理を削除し、関数Aと関数Bを合成した複合関数を作ることで、転送回数を減らし、転送処理にかかる時間を大幅に削減して高速化することが可能である。つまり、複合関数では、演算プロセッサが関数Aと関数Bの両方の演算を連続して行うことにより、演算プロセッサの稼働率が向上するということである。転送ボトルネックの関数を合成して性能を向上させる手法があるということは、転送／演算分離型並列画像処理アーキテクチャの特徴的な利点と考えられる。

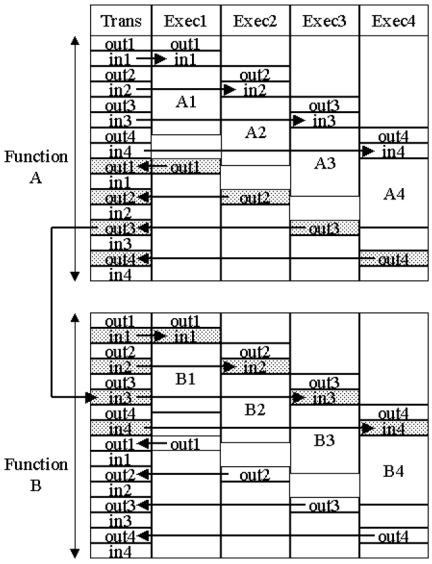

図9　複　合　前

Fig. 9　Before Compounded

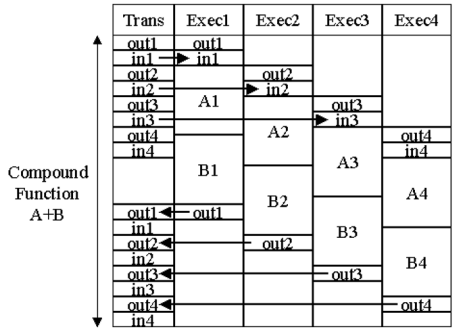

図10　複　合　後

Fig. 10　After Compounded

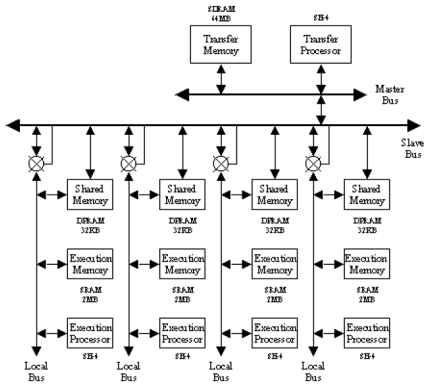

図11　ハードウェア構成

Fig. 11　Hardware Structure

4.4　ハードウェア構成と性能評価

図11に、1枚の基板に実装した画像処理装置のハードウェア

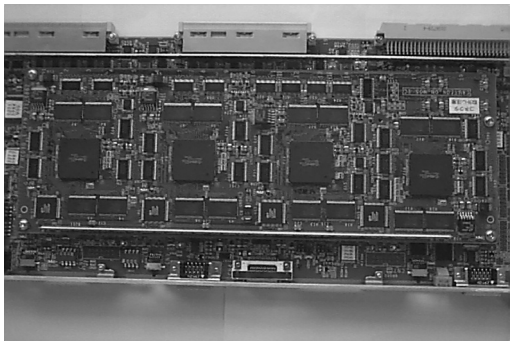

図12　ボード写真
Fig. 12　Board Picture

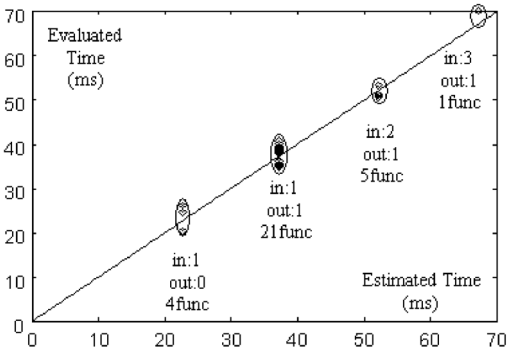

図13　見積り時間と実測時間の比較
Fig. 13　Comparison between Estimated Time and Evaluated Time

構成を示す。本並列画像処理基板は、転送メモリとして64MBのシンクロナスDRAM、演算メモリとして2MBのSRAM、転送プロセッサと演算プロセッサは、いずれも日立製作所の組み込み用RISCプロセッサSuperH-4を用いている。演算プロセッサの自ローカルメモリアクセスと転送プロセッサの他ローカルメモリが並行して行えるように、演算プロセッサと演算メモリを接続するスレーブバスにはバススイッチを備える。

同期制御フラグに代表される制御情報を格納するために、図11に示すように、転送プロセッサと演算プロセッサの両方からアクセスできる「共有メモリ」を用意する。制御情報は、量的には少ないが転送プロセッサと演算プロセッサで適宜必要になるので、共有メモリは、両方から頻繁にアクセスされても性能低下の少ない2ポートメモリを用いて実現する。

また、転送を行う際には、キャッシュのコヒーレンシを保つために、演算プロセッサのキャッシュを無効化する必要がある。図4に示したように、同期制御と同時にキャッシュの無効化を行う。SuperH-4には、キャッシュを無効化する命令が用意されており、命令レベルでキャッシュを制御することで実現する。

図12は、ボードの写真である。図11の全体を実際のボードに実装した。4つ横に並んでいるのが演算プロセッサである（いずれも日立製作所の組み込み用RISCプロセッサSuperH-4）。

提案アーキテクチャを具体化し、実機によって性能見積り値と実測値とを比較評価した。図13は、横軸が見積り時間(ms)、縦軸が対応する実測時間(ms)で、両者の対応関係を示しており、1入力0出力4関数、1入力1出力21関数、2入力1出力5関数、3入力1出力1関数のデータがプロットしてある。両者が一致していれば45度線上になるが、そこからのずれが見積り誤差である。さらに、表1には、見積り時間と実測時間の差（誤差時間）を表にまとめたものを示す。

表1　誤差時間
Table 1　Error Time

func. type	num. of func.	estimated time	error of time ave.	error of time max
1in-0out	4	23ms	2.5ms	3.2ms
1in-1out	21	38ms	1.4ms	3.1ms
2in-1out	5	53ms	1.7ms	2.3ms
3in-1out	1	68ms	2.0ms	2.0ms
all	31	-	1.6ms	3.2ms

誤差時間は平均1.6ms、最大でも3.2msの範囲内にあり、見積りの精度が高いことがわかる。

5. おわりに

並列画像処理システムにおいて、(1)転送と演算を分離した並列画像処理アーキテクチャ、(2)フィルタサイズを考慮した均等データ分割、(3)フラグを用いた同期制御方式、により、性能を不確定にする要素を極力低減することを試みた。本提案アーキテクチャの効果を実機で評価した結果、見積りとの誤差3.2ms以下となり、実用上問題のない性能推定が可能であることを確認できた。

文　献

[1] Matias, Y.: Parallel Algorithms Columns: On the Search for Suitable Models, SIGACT News, Vol. 28, No. 3, pp. 21-29(1997).

[2] 星合　隆成: 密結合マルチプロセッサシステムにおける排他制御方式とその性能解析, 信学論, Vol. J78-D-1, No. 2, pp. 248-259 (1995).

[3] D. Yen et al: Memory Interference in Synchronous Multiprocessor Systems, IEEE Trans. Computers, Vol. C-31, No. 11, pp. 1116-1121 (1982).

[4] 古市　実裕、永松　礼夫、出口　光一郎: 高並列計算機の性能評価のための挙動予測モデル, 情処学論, Vol. 38, No. 9, pp. 1766-1774 (1997).

[5] 松永　俊雄、福村　好美：バス結合型マルチプロセッサ方式の性能評価, 信学論, Vol. J73-D-1, No. 9, pp. 737-745 (1990).

[6] 布谷　嘉章他：マルチプロセッサシステムの性能評価, 信学誌, Vol. J66, No. 12, pp. 1261-1266 (1983).

[7] David Culler: A Practical model of parallel computation, Commun. ACM, Vol. 39, No. 11, pp. 78-85 (1996).

[8] 當山　孝義、堀口　進: 並列アルゴリズムの動作解析のための実用並列計算機モデルLogPQ, 情処学論, Vol. 39, No. 6, pp. 1766-1774 (1998).

[9] Sahni, Sartaj: Scheduling master-slave multiprocessor systems, IEEE Trans. Computers, Vol. 45, No. 10, pp. 1195-1189 (1996).

[10] 大上　靖弘, 杉本　和英, 北村　徹, 角　保志, 富田　文明: ネットワーク型並列計算環境における物体認識, 信学論, Vol.J82-D2 No.12 p.2307-2315 (1999).

[11] J. Hennesy and D. Paterson: Computer Architecture - A Quantitative Approach, Morgan-Kaufmann (1990).