計算機アーキテクチャ　128－3
ハイパフォーマンス　70－3
コンピューティング
（1998．3．5）

二次元FFTの並列処理手法の検討

水野 政治，宮田 裕行

三菱電機(株)　情報技術総合研究所

多くの信号処理，画像処理の分野において，二次元FFT(Fast Fourier Transform: 高速フーリエ変換)は頻繁に利用されている．この二次元FFTでは，いわゆるメモリアクセスにおけるコーナターンが発生するため，並列処理により高速化を図る場合，プロセッサ間で大量のデータ転送が生じ，システムの性能低下を招くという問題が指摘されてきた．

本稿では，コーナーターンによる性能低下を軽減する並列処理手法について検討し，その結果として複数のプロセッサをリング状に接続した並列アーキテクチャをベースとした並列処理手法を提案する．本手法は，一次元FFTを二分割し，プロセッサの演算とプロセッサ間データ転送を並行して行うパイプライン処理方式を採用するもので，これを可能とするためのネットワーク性能についても言及する．

A Proposal of a Parallel Processing Technique for 2D-FFT

Masaji Mizuno, Hiroyuki Miyata

Information Technology R&D Center,
Mitsubishi Electric Corporation

This paper describes a parallel processing architecture for the two-dimensional Fast Fourier Transform (2D-FFT). In the parallel processing for the 2D-FFT, "corner-turn" is known as a factor of performance bottleneck. In this paper, we propose the technique of parallel processing for the 2D-FFT on ring-based parallel architecture to solve "corner-turn" problem. Then we discuss the performance of ring-network to perform computation and data transfer concurrently.

1　まえがき

信号処理，画像処理において，高速フーリエ変換(Fast Fourier Transform, 略してFFT)の果たす役割は極めて大きい．本来の目的であるフーリエ変換の高速化の他に，畳込み演算，相関関数，フィルタリング等を高速に計算する手段として，多くの応用に適用されている[1)2)]．また，画像データなどの二次元データに対するFFT(以下，二次元FFT)においても，画像同士の相関演算，合成開口レーダ(Synthetic Aperture Radar, 略してSAR)における画像再生処理の畳込み演算など，多くの分野で用いられている．

しかし，FFT，特に二次元FFTは高速化の一手法として用いられながら，近年のデータ量の増大，処理時間の短縮などの要求にともない，最新の計算機技術をもってしても単一のプロセッサでは十分な処理時間が得られず，複数のプロセッサを用いた並列処理を必要とする場合が多い．二次元FFTでは扱う二次元データへのアクセスが行方向から列方向に変化する，いわゆるコーナーターンが発生するため，二次元FFTを並列に処理する場合には，複数のプロセッサ間で大量のデータ転送が生じ，システムの性能低下を招くという問題点がある．

そこで，本稿では，コーナーターンによる性能低下を軽減するため，複数のプロセッサをリング状に接続した並列アーキテクチャをベースとした，二次元FFTの並列処理手法を提案する．同様な研究としては，FFTのアルゴリズムと相性の良いハイパーキューブや二次元メッシュで接続された並列アーキテクチャをベースとしたものがある[3)4)5)6)]が，本稿のようにリング接続に関するものは少ない．

以下，2章では二次元FFTについて示す．次に，3章において二次元FFTの並列処理手法について検討し，4章でリング状に接続した並列アーキテクチャ上での二次元FFTの動作について示す．

2 二次元 FFT

高速フーリエ変換 (FFT) は, 有限個数のデータ系列に対する離散的フーリエ変換 (Discrete Fourier Transform, 略して DFT) の計算において演算の手順を合理化し, 全体の演算回数を大幅に減少させたものであり, J.W.Cooley と J.W.Tukey らによって 1965 年に初めて発表され, その後急速に広まると共に, 多くの研究者によって種々の改良アルゴリズムが発表されている [2].

二次元 FFT は, 二次元データに対する FFT であり, $N_1 \times N_2$ の要素からなる二次元データ系列 $g(k_1, k_2)$ に対する二次元 FFT $G(n_1, n_2)$ は, 次式により定義される.

$$G(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} g(k_1, k_2) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} \quad (1)$$

ここで, 回転子 (またはひねり係数)W は,

$$W_N^{nk} = exp(-j2\pi \frac{nk}{N}) \quad (2)$$

一方, $g(k_1, k_2)$ における k_2 方向での一次元 FFT $G'(k_1, n_2)$ は,

$$G'(k_1, n_2) = \sum_{k_2=0}^{N_2-1} g(k_1, k_2) W_{N_2}^{n_2 k_2} \quad (3)$$

で定義できるので, 式 1は次のように変形することができる.

$$G(n_1, n_2) = \sum_{k_1=0}^{N_1-1} G'(k_1, n_2) W_{N_1}^{n_1 k_1} \quad (4)$$

すなわち, 二次元 FFT は, N_1 組の N_2 点一次元 FFT(以下, これを行方向 FFT と呼ぶ) を実行し, 次に N_2 組の N_1 点一次元 FFT(以下, これを列方向 FFT と呼ぶ) を実行することにより実現できる.

この行方向 FFT と列方向 FFT との組合わせは, 二次元 FFT に特化したものではない. n 点の一次元 FFT は, $n = n_1 n_2$ と分解されるとき, (1) n_1 組の n_2 点 FFT を行い, (2) 適当な回転子を乗じ, (3) n_2 組の n_1 点 FFT を行う, ことにより実現できる. これは, "four step"FFT アルゴリズム, あるいはその変形である "six step"FFT アルゴリズム として知られている [7].

したがって, 二次元 FFT における高速化手法の確立は, 大規模な一次元 FFT へも適用可能であるため, 大きな意義を持つと考える.

3 二次元 FFT の並列処理手法の検討

二次元 FFT の並列処理手法としては, ハイパーキューブや二次元メッシュをベースとしたものが提案されている [3][4]. また, SAR の分野でも, SAR の画像再生処理の一環として, 二次元 FFT の並列処理手法についてハイパーキューブあるいは二次元メッシュ上での手法が提案されている [5][6].

ハイパーキューブや二次元メッシュなどのネットワーク・トポロジーは, FFT のアルゴリズムと相性が良く, 特にハイパーキューブでは隣接するプロセッサ間のみのデータ転送に限定できるという利点があるものの, プロセッサ数の増大に伴い, ハードウエア規模の増大, 配線やルーティングの複雑化という問題を抱える.

そこで, 本稿では, より単純なネットワーク・トポロジーとしてリングを取り上げ, これをベースとした二次元 FFT の並列処理手法について検討する. リングは, ハイパーキューブに比較しプロセッサ間の最長距離が長くなる [8] ものの, 配線やルーティングはより簡単となる.

同様にリングをベースとした手法は, Knight らにより, 8 台の Transputer をリング状に接続したアーキテクチャでの SAR の実現方式として提案されている [9]. 本手法は, 行方向 FFT をリング状に接続したプロセッサでパイプライン処理し, その結果を一旦メモリに蓄積し, 次に行方向 FFT をパイプライン処理するものであり, (1) 行方向 FFT の結果を蓄積するために大容量のメモリが必要となる, (2) 上記メモリから列方向のデータを取り出すときにコーナーターンが発生するため性能劣化が生じる, という問題があげられる.

本章では, まず, 複数の一次元 FFT を並列処理する方法について整理し, リング状に接続したアーキテクチャにおいて採用する二次元 FFT の並列処理手法について検討する.

3.1 複数の一次元 FFT の並列化

複数の一次元 FFT を複数のプロセッサで並列処理する方法を, ここでは以下のように分類することとする (図 1).

1. プロセッサ独立型: 1 つの FFT を 1 つのプロセッサ内で閉じて処理する方式.
2. プロセッサ協調型: 1 つの FFT を複数のプロセッサで協調して処理する方式.

   本方式はさらに次の 3 つに分類する.

(a) 並列処理型: 1つのFFTを全プロセッサで並列処理する方式.

(b) クラスタ型: プロセッサを幾つかのプロセッサ群(クラスタ)に分割し, 各クラスタ内では1つのFFTを並列処理すると共に, 複数のクラスタで同時に複数のFFTの処理を行う方式.

(c) パイプライン型: 1つのFFTにおける各ステージのバタフライ演算を, 複数のプロセッサでパイプライン処理する方式.

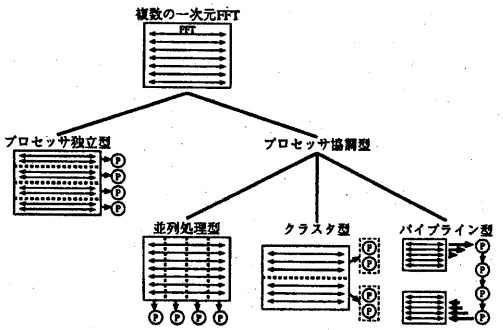


図 1: 並列処理手法の分類

プロセッサ独立型では, 各プロセッサ内で閉じてFFTを処理するので, プロセッサ間のデータ交換は必要なく, またプロセッサ数に対してFFTの個数が十分大きければ, 各プロセッサの計算負荷はほぼ均等となるため, 最も効率よく並列処理することができる.

一方, プロセッサ協調型では, プロセッサ間のデータ交換が必要となるため, 十分な効果を得るためにはデータ交換に対する何らかの対策が必要である.

3.2 二次元FFTの並列化

ここでは, 本稿で提案するシステムの適用を想定するアプリケーションのアルゴリズムより, 以下のように処理が進むものと仮定する (図2).

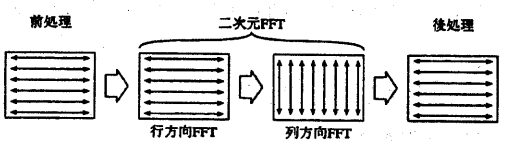


図 2: 想定する処理の流れ

1. 前処理: 同一行にあるデータを使って処理する
2. 二次元FFT: 行方向FFT + 列方向FFT
3. 後処理: 同一行にあるデータを使って処理する

行方向FFT及び列方向FFTについて, 先に分類した複数の一次元FFTの並列処理手法を適用した場合について, 特にプロセッサ間のデータ転送に着目して検討する.

まず, 前処理から行方向FFTに移行する場合のデータ転送量は,

- プロセッサ独立型を適用する場合, 前処理と同一のプロセッサにデータを配置すればよく, 特にプロセッサ間のデータ転送は必要としない.
- 並列処理型を適用する場合, コーナーターンが発生するため, 各プロセッサの保有するデータ $\frac{N_1 N_2}{P} = N_p$ のうち, $\frac{P-1}{P}$ を他のプロセッサに転送するので, データ転送量は $N_p(P-1)$.
- クラスタ型を適用する場合, 各プロセッサの保有するデータ N_p のうち, $\frac{p-1}{p} = \frac{P-C}{P}$ を他のプロセッサに転送するので, データ転送量は $N_p(P-C)$.
- パイプライン型を適用する場合, 前処理の結果を一旦1つのプロセッサに集約する必要があるので, データ転送量は $N_p(P-1)$.

同様の検討をまとめたものが表1である. 表1に示すように, 行方向FFTをプロセッサ独立型で, 列方向FFTを並列処理型で行う場合が, 各処理の間のプロセッサ間のデータ転送を最低限に押さえることができる. ただし, 並列処理型では1つのFFTを複数のプロセッサで並列処理するため, FFTのアルゴリズムの特性上, プロセッサ間のデータ転送が発生するので, この問題を解決する必要がある.

4 リング接続をベースとした二次元FFT用並列アーキテクチャ

以上の検討を踏まえ, 本章ではリング接続をベースとした並列アーキテクチャを示し, このアーキテクチャにおける二次元FFTの動作を説明する.

4.1 システム構成

図3は, 本稿で提案する二次元FFT用の並列アーキテクチャを示すものである.

各々のプロセッサは, 図3に示すように, リング状のネットワークで接続される. 以下, 最も単純な場合として単方向リングを想定する (双方向リングとしても特に問題はない). 隣接するプロセッサからのデータ転送 (入力) と隣接するプロセッサへのデー

表 1: プロセッサ間のデータ転送量

並列処理手法		データ転送量		
行方向 FFT	→ 列方向 FFT	前処理→行方向 FFT	行方向 FFT →列方向 FFT	列方向 FFT →後処理
独立型	→ 独立型	0	$N_p(P-1)$	$N_p(P-1)$
	→ 並列処理型	0	0	0
	→ クラスタ型	0	$N_p(P-1)$	$N_p(P-1)$
	→ パイプライン型	0	$N_p(P-1)$	$N_p(P-1)$
並列処理型	→ 独立型	$N_p(P-1)$	0	$N_p(P-1)$
	→ 並列処理型	$N_p(P-1)$	$N_p(P-1)$	$N_p(P-1)$
	→ クラスタ型	$N_p(P-1)$	$N_p(P-C)$	$N_p(P-1)$
	→ パイプライン型	$N_p(P-1)$	$N_p(P-1)$	$N_p(P-1)$
クラスタ型	→ 独立型	$N_p(P-C)$	$N_p(P-1)$	$N_p(P-1)$
	→ 並列処理型	$N_p(P-C)$	$N_p(P-C)$	0
	→ クラスタ型	$N_p(P-C)$	$N_p(P-1)$	$N_p(P-1)$
	→ パイプライン型	$N_p(P-C)$	$N_p(P-1)$	$N_p(P-1)$
パイプライン型	→ 独立型	$N_p(P-1)$	$N_p(P-1)$	$N_p(P-1)$
	→ 並列処理型	$N_p(P-1)$	$N_p(P-1)$	0
	→ クラスタ型	$N_p(P-1)$	$N_p(P-1)$	$N_p(P-1)$
	→ パイプライン型	$N_p(P-1)$	0	$N_p(P-1)$

ここで, N_1, N_2: データサイズ, P: プロセッサ数, C: クラスタ数, p: クラスタ内のプロセッサ数 $(= P/C)$, $N_p = \frac{N_1 N_2}{P}$

タ転送 (出力) は同時に行えるものとする. また, 入力と出力, 隣接するプロセッサからの他の隣接プロセッサのデータ転送 (ルーティング) は, プロセッサの処理と並行して行えるよう構成されるものとする.

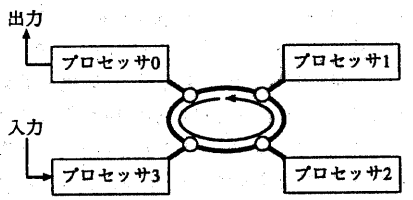


図 3: システム構成 (プロセッサ数 =4 の場合)

4.2 処理全体の動作

3.2で示したように, 本システムで想定する処理は, (1) 前処理, (2) 二次元 FFT, (3) 後処理と行われる.

まず, 処理の対象とするデータは, 同一行のデータが同一のプロセッサに格納されるように入力される. プロセッサの台数を P とすると, 各プロセッサへは P だけ離れた行データを入力する (図 4).

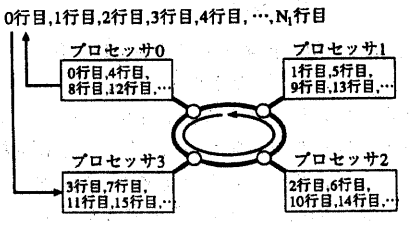


図 4: プロセッサへのデータ分配 ($P = 4$ の場合)

前処理及び行方向 FFT は, 各プロセッサ内で閉じて行われる (プロセッサ独立型). 次に, 列方向 FFT については 1 列分の FFT を全プロセッサで並列処理に処理し (並列処理型), その結果に対する後処理は各プロセッサ内で閉じて処理する (プロセッサ独立型) (図 5).

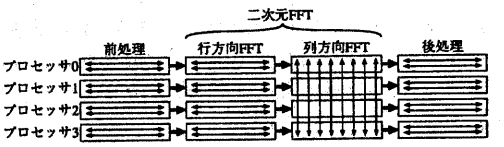


図 5: 処理全体の動作

4.3 列方向 FFT に対する動作

次に, 列方向 FFT について詳しく説明する.

本システムでは, 一列分の FFT の処理を (1)FFT の前半, (2) プロセッサ間データ転送, (3)FFT の後半に分割し, N_2 個の一列分の FFT をパイプライン処理することにより, 列方向 FFT を完了する (図 6).

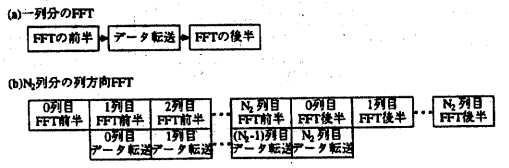


図 6: 列方向 FFT の動作

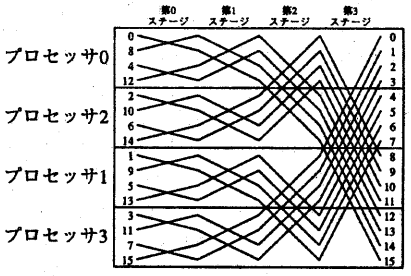
以下, まず一列分の FFT の分割方式について示し, 上記パイプライン処理を可能とするための条件について言及する.

4.3.1 一列分の FFT の分割方式

一列分の FFT は, プロセッサ数 $P = 4$, 一列のデータ数 $N_1 = 16$ とした場合, 単純にプロセッサと FFT のアルゴリズムをマッピングすると, 第2ステージのバタフライ演算でプロセッサ0とプロセッサ2, プロセッサ1とプロセッサ3で, 第3ステージにおいてプロセッサ0とプロセッサ1, プロセッサ2とプロセッサ3で, データ転送が必要なことがわかる (図7(a)).

これに対し, FFT では出力されるデータ系列を入力したデータ系列と同様の並びにするためにビットリバースが必要となることが一般に知られている. そこで, このビットリバースを第1ステージと第2ステージの間で適用し, アルゴリズムを整理したものが図7(b) である. このように, FFT の処理の中間でビットリバースを適用することにより, プロセッサ間のデータ転送を一個所に集約することができ, その前後においては各プロセッサは独立に処理を進めることができる.

(a)ビットリバースを行わない場合



(b)ビットリバースを行う場合

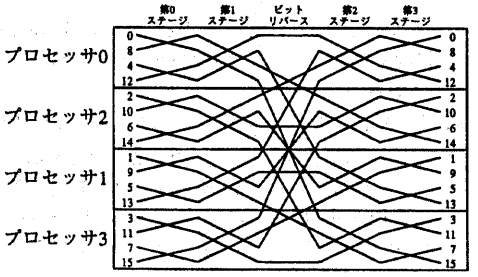


図 7: プロセッサと FFT とのマッピング

このような並列処理が可能とするには, $N_1 = 2^{n_1}$ とすると, n_1 が偶数の場合には第 $\frac{n_1}{2}$ ステージの前で, n_1 が奇数の場合には第 $\frac{n_1-1}{2}$ ステージの前でビットリバースを適用する必要がある. この場合のプロセッサ数 P は,

$$P = \begin{cases} \sqrt{N_1} & (n_1\text{が偶数の場合}) \\ \sqrt{N_1/2} & (n_1\text{が奇数の場合}) \end{cases} \quad (5)$$

4.3.2 パイプライン処理の条件

列方向 FFT のパイプライン処理を可能とするためには, 図6で示すように, n 列目の FFT の前半を計算する間に (n-1) 列目のデータ転送を行う必要がある.

まず, 各プロセッサにおける FFT の前半の処理時間 C_{proc} は, $\frac{N_1}{P}$ のデータに対する FFT の処理と同等であるため, 1つのバタフライ演算の処理時間を C_{bfy} とすると,

$$C_{proc} = C_{bfy}\frac{N_1}{2P}log_2(\frac{N_1}{P}) \quad (6)$$

ここで, 実数演算の処理時間を C_p と仮定すると, バタフライ演算は1回の複素数乗算と2回の複素数加減算によって成り立つので, 単純には $C_{bfy} = 10C_p$ となる. よって, 式6は次のように変形できる.

$$C_{proc} = 10C_p\frac{N_1}{2P}log_2(\frac{N_1}{P}) \quad (7)$$

次に, ビットリバースにおけるプロセッサ間のデータ転送に着目する. ビットリバースでは, 各プロセッサの保有するデータ数 $\frac{N_1}{P}$ のうち, 他の $(P-1)$ 個のプロセッサに対して, 各々 $\frac{1}{P}$ 個のデータを転送することとなる. 1つのデータを n 個隣りのプロセッサへのデータ転送時間を nC_t とすると, リング状のネットワークでは複数のプロセッサで同時にデータ転送が行なえる (図8) ので, ビットリバースに必要なデータ転送時間 C_{trans} は,

$$\begin{aligned} C_{trans} &= \frac{N_1}{P}\frac{1}{P}(C_t + 2C_t + \cdots + (P-1)C_t \\ &= \frac{N_1}{P}\frac{1}{P}C_t\frac{P(P-1)}{2} \\ &= C_t\frac{N_1}{2P}(P-1) \end{aligned} \quad (8)$$

ここで, パイプライン処理を可能とするためには, $C_{trans}/C_{proc} \leq 1$ を満たす必要があるので,

$$\frac{C_{trans}}{C_{proc}} = \frac{C_t\frac{N_1}{2P}(P-1)}{10C_p\frac{N_1}{2P}log_2(\frac{N_1}{P})} \leq 1 \quad (9)$$

よって,

$$\frac{C_t}{C_p} \leq \frac{10log_2(\frac{N_1}{P})}{P-1} \quad (10)$$

表 2: C_t/C_p の計算値

プロセッサ数 P	データ数 N_1									
	64	128	256	512	1024	2048	4096	8192	16384	32768
2	50.00	60.00	70.00	80.00	90.00	100.00	110.00	120.00	130.00	140.00
4	13.33	16.67	20.00	23.33	26.67	30.00	33.33	36.67	40.00	43.33
8	4.29	5.71	7.14	8.57	10.00	11.43	12.86	14.29	15.71	17.14
16	—	—	2.67	3.33	4.00	4.67	5.33	6.00	6.67	7.33
32	—	—	—	—	1.61	1.94	2.26	2.58	2.90	3.23
64	—	—	—	—	—	—	0.95	1.11	1.27	1.43
128	—	—	—	—	—	—	—	—	0.55	0.63

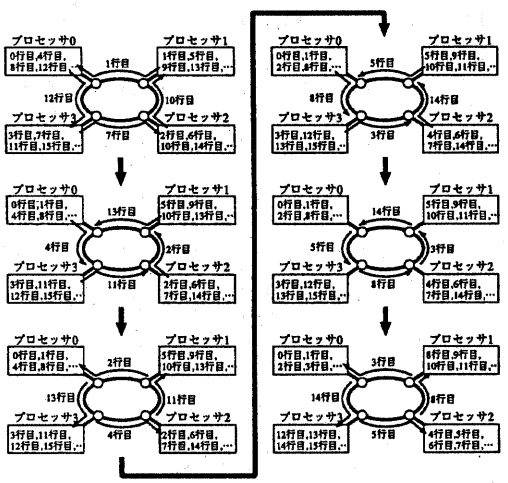


図 8: ビットリバースにおけるデータ転送の多重化

式 10における右辺の値を, 幾つかの N_1, P について計算したものが表 2である. 例えば, $N_1 = 4096, P = 8$ の場合, 演算性能 $(1/C_p)$ の約 12 倍の以上の性能を持つネットワークを用意すれば, プロセッサ数に見合った台数効果を得ることがわかる. 逆に 12 倍の性能を持つネットワークを用意できない場合には, データ転送が表面化するため, プロセッサ数より少ない台数効果のみしか得られないことを考慮する必要があることがわかる.

5 むすび

本稿では, 二次元 FFT の並列処理手法について検討し, 複数のプロセッサをリング状に接続した並列アーキテクチャと, これをベースとした並列処理手法を提案した. 本システムでは, 複数の一次元 FFT を並列処理するため, 各々の FFT を 2 つのパートに分割し, プロセッサの演算とプロセッサ間データ転送を並行して行うパイプライン処理方式を採用することとし, これを可能とするための条件についても示した.

本稿においてはリング接続を前提としたが, 今後, 他のネットワークトポロジーにおける比較検討を実施し, 最終的なアーキテクチャの決定を行う予定である.

参 考 文 献

1) 電子情報通信学会 (編) : 「ディジタル信号処理」, コロナ社 (1987).

2) 安居院 : 「FFT の使い方」電子科学シリーズ, 産報出版 (1982).

3) Angelopoulos, G. and Pitas, I. : "Two-dimensional FFT algorithms on hypercube and mesh machines", In *Signal Processing*, Vol.30, pp.355–371 (1993).

4) Aloisio, G., Fox, G. C., Kim, J. S., and Veneziani, N. : "A concurrent implementation of prime factor algorithms on hypercubes", In *IEEE Trans. Signal Processing*, Vol.39, No.1, pp.160–170 (1991).

5) Mastin, G. A., Plimpton, S. J., and Ghiglia, D. C. : "A massively parallel digital processor for spotlight synthetic aperture radar", In *Internat. J. Supercomput. Appl.*, Vol.7, No.2, pp.97–112 (1993).

6) Fabbretti, G., Farina, A., Laforenze, D., and Vinelli, F. : "Mapping the Synthetic aperture radar signal processor on a distributed-memory MIMD architecture", In *Parallel Computing*, Vol.22, No.5, pp.761–784 (1996).

7) Bailey, D. H. : "FFTs in External or Hierarchical Memory", In *The J. Supercomputing*, Vol.4, pp.23–35 (1990).

8) 富田眞治 : 「並列コンピュータ工学」, 昭晃堂 (1996).

9) Knight, A. R. and Lnggs, M. R. : "A SAR Processor Implemented on a Transputer Ring", In *International Geoscience and Remote Sensing Symposium(IGRASS)*, Vol.2, pp.900–902 (1994).