Vol.2014-ARC-213 No.4
Vol.2014-HPC-147 No.4
2014/12/9

圧縮センシングに基づく高分解能DoA推定アルゴリズムの高速実装方式の検討

後町 将人[1,a)]　高橋 善樹[1]　尾崎 敦夫[1]

概要：電波の到来方向を高分解能に推定する手段として, 圧縮センシングに基づく推定アルゴリズムの利用が有効である. この手法は, 波源分布に対応した疎行列を解とする最適化計算で構成され, 収束に至るまで高負荷な演算を反復する. このため, 高分解能化に伴う行列の巨大化は, 演算時間の爆発的な増加を招き, 数秒の制約時間では推定可能な分解能が低い. この問題に対処するため, GPU 向けの並列化を施した上で, 未知行列の零成分に対応する演算結果を, 最適化ループ中に再利用する演算量削減法を考案した. この高速化効果を見積り評価したところ, 従来法と CPU6 コアによる角度点数 1000 の波源分布の推定と, 提案法と GPU による角度点数 3,600 の波源分布の推定が, 同等の約 2.6 秒で完了し, 演算時間の増加を招くことなく, 約 3.6 倍の高分解能化を実現できる見込みが得られた.

1. はじめに

アレーアンテナで受信した電波の到来方向を, 走査する空間方向の波源分布から求める DoA (direction of arrival) 推定技術 (図 1) は, 不法電波源の位置特定や, アンテナ性能に影響する不要波の方向検出等に用いられる. この DoA 推定をオンライン解析で扱うためには, 数秒程度の時間で正確な推定結果を得る推定アルゴリズムが求められる.

より正確な DoA を推定するためには, 高分解能な推定アルゴリズムを利用し, 波源分布の角度点数を細分化する手段が有効である. 一方, Beam Forming [15] や MUSIC (multiple signal classification) [16] といった従来の推定アルゴリズムは, 低分解能であったり, 相関波を正確に推定できないといった課題を持つ. これに対し, 波源分布に対応した疎ベクトルを解とする最適化問題を構成し, これらの課題を解決した圧縮センシング (CS: compressive sensing) [5] に基づく推定アルゴリズム (図 2) が提案された [11]. しかしながら, この圧縮センシングに基づく推定アルゴリズムは, 相関波に対して高分解能な推定を実現する一方で, 従来のアルゴリズムとは異なり, 最適化における反復毎の計算に, 角度点数に対して多項式時間を要する. このため, 推定精度を向上するために角度点数を増やした場合, DoA 推定処理に長時間を要することが課題となっている.

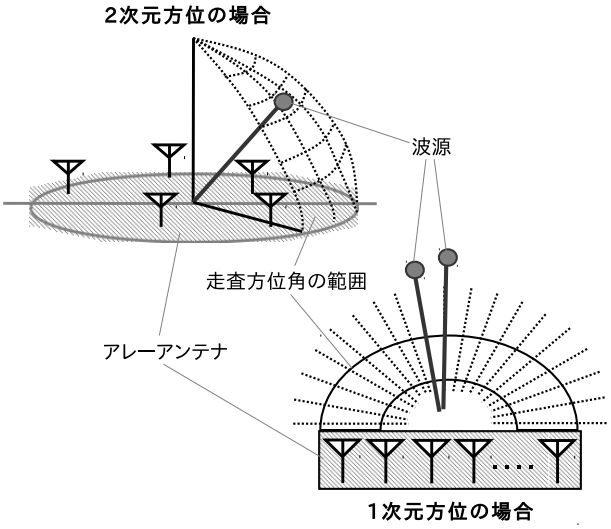

図 1 DoA 推定: アンテナの受信信号と走査情報から, 走査方位に対する波源分布を計算して, 波源の存在する方位を推定する.

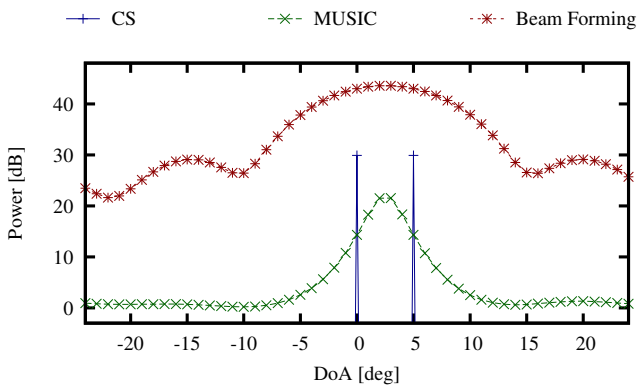

図 2 波源分布の推定結果: CS で推定した相関性の高い 2 つの波源は, Beam Forming と MUSIC では正確に推定できていない.

[1] 三菱電機 情報技術総合研究所
[a)] Gocho.Masato@ds.MitsubishiElectric.co.jp

我々は, この課題に対処し, 高分解能な DoA 推定の結果を短時間で得る手段を構築するために, 圧縮センシングに基づく高分解能 DoA 推定アルゴリズムの高速化に取り組んだ. この高速化の手段として, まず, GPU (graphics processing unit) を利用する並列プログラムを実装した. 次に, 本推定アルゴリズムが, 波源分布に対応する未知行列を最適化計算で求める方式であり, 最適化の反復毎に未知行列が零成分を増やしていく点に着目し, この零成分に対応する演算結果を再利用する演算量削減方式を考案した. 本稿では, 並列高速化の方式と演算量削減法について説明すると共に, 高速化効果の見積り結果について報告する.

2. 圧縮センシングに基づく高分解能DoA推定

アンテナ素子数を L, 走査する空間方向の角度点数を N, そして, 受信信号の時間サンプル数を M と置くと, 走査情報を表す $L \times N$ の複素行列 $\mathbf{A}$ から, $L \times M$ の複素行列 $\mathbf{Y}$ が受信信号として得られる. 両者の関係は, 以下の式で表される.

$$\mathbf{Y} = \mathbf{AX} + \mathbf{W} \tag{1}$$

ここで, $\mathbf{W}$ は雑音, $\mathbf{X}$ は時間サンプル毎の波源分布であり, それぞれ $N \times M$ の複素行列で表される. 式 (1) において, 既知行列 $\mathbf{Y}$, $\mathbf{A}$ から, 連立方程式を計算して未知行列 $\mathbf{X}$ を直接的に求めることは, 方程式の数に対して未知数が膨大 ($L \ll N$) のため, 困難である. このため, 任意の DoA 推定アルゴリズムを用いて, 既知行列 $\mathbf{A}$, $\mathbf{Y}$ から, 未知行列 $\mathbf{X}$ を推定することになる. 圧縮センシングに基づく推定アルゴリズムでは, 電波の到来方位角に対応する行列成分のみ値を持った疎行列 $\mathbf{X}$ を, 最適化問題を解くようにして推定する. この推定結果である疎行列 $\mathbf{X}$ は, 更に, 各行ベクトル ($[X_{n,1}, X_{n,2}, \ldots, X_{n,M}]$, $n = 1, 2, \ldots, N$) に対して, 以下の式に示すように, ℓ_2-ノルムを計算することで疎ベクトル $\mathbf{x}$ へと縮約される.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} \|X_{1,1} & X_{1,2} & \ldots & X_{1,M}\|_2 \\ \|X_{2,1} & X_{2,2} & \ldots & X_{2,M}\|_2 \\ \vdots & \vdots & \ddots & \vdots \\ \|X_{N,1} & X_{N,2} & \ldots & X_{N,M}\|_2 \end{bmatrix} \tag{2}$$

この縮約された疎ベクトル $\mathbf{x}$ が, 図 2(凡例:CS) で示される最終的な波源分布の推定結果となる.

この圧縮センシングに基づいて高分解能な DoA 推定の結果を得るためには, EM (expectation maxmization) アルゴリズム [12] 等の最尤推定で用いられる推定項に, 疎性に関する制約項として ℓ_p-ノルムを設定する手法が有効である [8]. 具体的には, 以下の目的関数 $f(\mathbf{X})$ を用いる.

$$f(\mathbf{X}) = \|\mathbf{Y} - \mathbf{AX}\|_2^2 + \lambda \|\mathbf{x}\|_p^p \tag{3}$$

ここで, $\|\mathbf{x}\|_p$ は, $(\sum_{n=1}^{N} |x_n|^p)^{1/p}$ を表す ℓ_p-ノルムであり, $1 > p > 0$ を満たす任意の定数を p に設定して扱う. λ は, 推定項と制約項のバランスを取る正規化パラメータである. λ には, 解の精度向上に有効な変数値を設定する手法も提案されている [17][19] が, 本稿では, 文献 [11] と同様に任意の定数値を設定して扱う.

式 (3) は非線形関数のため, 本稿では, 解 $\mathbf{X}$ を求めるために, 非線形数値最適化アルゴリズムを使用する. これは, 反復 (k) 周目の各パラメータに識別子 (k) を付加すると, 以下の式による $\mathbf{X}$ の反復更新処理として表される.

$$\begin{aligned} \mathbf{X}^{(k+1)} &= \mathbf{X}^{(k)} - \alpha^{(k)} \mathbf{D}^{(k)} \\ \mathbf{D}^{(k)} &= \mathbf{H}_{(k)}^{'-1} \nabla f(\mathbf{X}^{(k)}) \\ \alpha^{(k)} &= \arg\min_{\alpha^{(k)}} f(\mathbf{X}^{(k)} + \alpha^{(k)} \mathbf{D}^{(k)}) \end{aligned} \tag{4}$$

ここで, $\mathbf{H}^{'}$ は, ヘッセ行列 $\nabla^2 f(\mathbf{X})$ を表す. また, α は, ヘッセ逆行列 $\mathbf{H}^{'-1}$ と勾配 $\nabla f(\mathbf{X})$ との積から得られる探索方向 $\mathbf{D}$ へと進むステップ幅であり, 一般に, 二分法等で近似計算した結果や, 定数値 1 を設定して扱われる. 本稿では, 文献 [11] と同様の手法を実装するため, ヘッセ近似行列を必要とする準ニュートン法を使用した. まず, 式 (3) の制約項を近似式に置き換え, 目的関数 $f(\mathbf{X})$ を以下の式 (5) へと変形する.

$$f(\mathbf{X}) \approx \|\mathbf{Y} - \mathbf{AX}\|_2^2 + \lambda \sum_{n=1}^{N} \left(|x_n|^2 + \epsilon\right)^{p/2} \tag{5}$$

この式 (5) の勾配 $\nabla f(\mathbf{X})$ は, 以下の式 (6) で与えられる.

$$\nabla f(\mathbf{X}) \approx \mathbf{HX} - \mathbf{B} \tag{6}$$

$$\mathbf{H} = \mathbf{H}_0 + \mathrm{diag}\{\mathbf{s}\} \tag{7}$$

$$\mathbf{H}_0 = 2\mathbf{A}^H \mathbf{A} \tag{8}$$

$$\mathbf{B} = 2\mathbf{A}^H \mathbf{Y} \tag{9}$$

$$\mathbf{s} = \lambda p \begin{bmatrix} \left(|x_1|^2 + \epsilon\right)^{p/2-1} \\ \left(|x_2|^2 + \epsilon\right)^{p/2-1} \\ \vdots \\ \left(|x_N|^2 + \epsilon\right)^{p/2-1} \end{bmatrix} \tag{10}$$

ここで, ϵ は, ゼロ除算を回避するための微小な定数値である. また, $\mathbf{H}$ は, ヘッセ行列 $\mathbf{H}^{'}$ を近似した $N \times N$ の行列, $\mathbf{A}^H$ は, 行列 $\mathbf{A}$ の複素共役転置をとった $N \times L$ の行列, $\mathbf{B}$ は, $N \times M$ の行列, $\mathrm{diag}\{\mathbf{s}\}$ は, ベクトル $\mathbf{s}$ を対角成分とした $N \times N$ の対角行列を表す. ステップ幅 α を 1 に設定すると, 式 (6–10) により, 未知行列 $\mathbf{X}$ の更新式 (4) は, 以下の式 (11) となる.

$$\begin{aligned} \mathbf{X}^{(k+1)} &= \mathbf{X}^{(k)} - \mathbf{H}_{(k)}^{-1} \left(\mathbf{H}_{(k)} \mathbf{X}^{(k)} - \mathbf{B}\right) \\ \mathbf{H}_{(k)} \mathbf{X}^{(k+1)} &= \mathbf{B} \end{aligned} \tag{11}$$

すなわち, この最適化計算は, 反復毎に, 未知行列 $\mathbf{X}^{(k)}$ から係数行列 $\mathbf{H}_{(k)}$ の対角成分を更新し, 更新後の係数行列

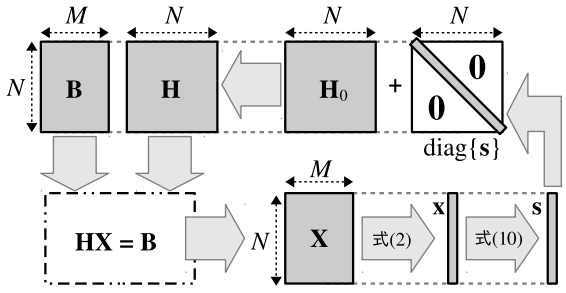

図 3 従来方式による高分解能 DoA 推定の処理手順

Algorithm 1 *estimate_sparsity*($\mathbf{A}$, $\mathbf{Y}$, $\mathbf{x}^{(1)}$, $\mathbf{x}_{\mathrm{opt}}$)

Input	$\mathbf{A}$	$\in \mathbb{C}^{L\times N}$	走査情報
	$\mathbf{Y}$	$\in \mathbb{C}^{L\times M}$	受信信号
	$\mathbf{x}^{(1)}$	$\in \mathbb{C}^{N}$	波源分布 (初期ベクトル)
Output	$\mathbf{x}_{\mathrm{opt}}$	$\in \mathbb{C}^{N}$	波源分布 (疎ベクトル)

1: $\mathbf{H}_0 \leftarrow 2\mathbf{A}^H\mathbf{A}$ ▷▷ 定数部の計算: 式 (8)
2: $\mathbf{B} \leftarrow 2\mathbf{A}^H\mathbf{y}$ ▷▷ 定数部の計算: 式 (9)
3: **repeat** $k \leftarrow 1 \ldots$
4: ▷▷ ヘッセ近似行列の対角成分を更新: 式 (10), 式 (7)
5: $\mathbf{s}^{(k)} \leftarrow \lambda p \left[\ (|x_1^{(k)}|^2+\epsilon)^{p/2-1} \ \ldots \ (|x_N^{k}|^2+\epsilon)^{p/2-1} \ \right]^T$
6: $\mathbf{H}_{(k)} \leftarrow \mathbf{H}_{(0)} + \mathrm{diag}\{\mathbf{s}^{(k)}\}$
7:
8: ▷▷ 連立方程式を計算: 式 (11)
9: solve $\mathbf{H}_{(k)}\mathbf{X}^{(k+1)} = \mathbf{B}$
10:
11: ▷▷ 行列からベクトルへの縮約: 式 (2)
12: $\mathbf{x}^{(k+1)} \leftarrow \left[\ \|\mathbf{X}_1^{(k+1)}\|_2 \ \ldots \ \|\mathbf{X}_N^{(k+1)}\|_2 \ \right]^T$
13:
14: ▷▷ 収束判定: 式 (12)
15: $\mathbf{x}_\Delta \leftarrow \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$
16: $\delta^{(k)} \leftarrow \dfrac{\|\mathbf{x}_\Delta\|_2}{\|\mathbf{x}^{(k)}\|_2}$
17: **until** ($\delta^{(k)} \geq \delta_{\mathrm{threshold}}$)
18: $\mathbf{x}_{\mathrm{opt}} \leftarrow \mathbf{x}^{(k+1)}$

$\mathbf{H}_{(k)}$ を用いて, 連立方程式 (11) を計算する構成となる (図 3). このため, 係数行列 $\mathbf{H}$ の対角成分が, 最適化計算の初期値として必要となる. この初期値には, 他の DoA 推定アルゴリズムによる推定結果など, 行列 $\mathbf{X}$ またはベクトル $\mathbf{x}$ に任意の値を設定し, 式 (2) や式 (10) で対角成分を計算すればよい. 本稿では, 初期値として, Beam Forming アルゴリズムによる低分解能な DoA 推定結果であるベクトル $\mathbf{x}$ を使用した. なお, 収束判定には, 閾値 $\delta_{\mathrm{threshold}}$ を設定し, 以下の判定式を用いる.

$$\delta_{\mathrm{threshold}} > \frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|_2}{\|\mathbf{x}^{(k)}\|_2} \qquad (12)$$

以上のように構成された最適化計算は, Algorithm 1 に示す手順で処理され, 波源分布に対応する疎ベクトル $\mathbf{x}$ を推定結果として出力する.

3. 高速化手法

圧縮センシングに基づく DoA 推定では, 角度点数 N に対して計算量 $\mathcal{O}(N^3)$ を伴う連立方程式の計算時間が, 全体の処理時間の大半を占める. このため, この連立方程式計算の高速化が重要な課題となる. 本章では, この課題に対処するため, GPU を用いて **2.** で示した方式の並列処理実装を示すと共に, この実装を対象に, 疎行列を最適解とする特徴を利用した演算量削減手法について説明する.

3.1 GPU の利用法

本研究では, グラフィックス用途のプロセッサである GPU を, 電波の DoA を推定するための並列演算装置として利用する. GPU は, 同一の演算命令を処理する低性能な SIMD (Single Instruction Multiple Data) コア集団を多数並べた構成のため, グラフィックス用途を始め, 単純かつ多量な演算の高速処理に適している.

一方, DoA 推定問題は, 行列積や行列分解等の行列演算を主体に構成されている. このため, この実装の多くは, Intel MKL (math kernel library) 等, BLAS (basic linear algebra subprograms) [10] や LAPACK (linear algebra package)[1] のような行列演算 API を実装したライブラリを利用するだけで済む. このような行列演算に対する GPU の利用実績は多数報告されており, NVIDIA 社の GPU 向けプログラム実装環境として CUDA (compute unified device architecture) を用いる場合は, cuBLAS [13] や MAGMA (matrix algebra on GPU and multicore architectures) [14] といった上記の API を GPU 向けに並列実装したライブラリを利用できる. 我々は, GPU 向けプログラムの実装環境として, この CUDA と CUDA 用の行列演算ライブラリを用いた.

3.2 並列処理の実装

連立方程式 (11) を解く手段としては, 係数行列 $\mathbf{H}$ を分解する直接計算法や, 残差 $\mathbf{B} - \mathbf{AX}$ を最小化する反復計算法等が挙げられる. この計算手段の選定は, 性能設計において重要な点であるが, まずは, LU 分解に基づく直接計算法を用いた場合の処理で試作を行った. この計算法では, 係数行列 $\mathbf{H}$ を, 下三角行列 $\mathbf{L}$ と上三角行列 $\mathbf{U}$ との積 $\mathbf{LU}$ に分解し, 分解後の行列から代入操作で解を得る.

Algorithm 1 と対応する形で, 実装した処理内容を説明する. 最適化ループ外に配置した 2 つの行列積計算 (1–2 行目) は, 結果を $\mathbf{H}_0$, $\mathbf{B}$ に出力するように, BLAS の行列積 API である `cgemm`/`zgemm` で実装した. 最適化ループ内では, まず, ヘッセ近似行列 $\mathbf{H}$ の対角成分を更新する (5–6 行目). これは, 最適化ループ外で計算した行列 $\mathbf{H}_0$ を行列 $\mathbf{H}$ へコピーした後に, 式 (10) を並列数 N で計算し, 計算結果を行列 $\mathbf{H}$ の対角成分に出力するように実装した. 連立方程式の計算 (9 行目) は, LAPACK の LU 分解 API で

ある cgetrf/zgetrf で, 行列 $\mathbf{H}$ に分解後の行列を出力する. その後, 右辺行列 $\mathbf{B}$ をコピーし, 代入操作 API である cgetrs/zgetrs を用いて, このコピー先の領域に $\mathbf{X}^{(k+1)}$ を出力するよう実装した. この計算結果 $\mathbf{X}^{(k+1)}$ をベクトル $\mathbf{x}^{(k+1)}$ へと縮約する処理 (12 行目) は, 並列数 $N \times M$ で, M 次元の行ベクトルに対する ℓ_2-ノルムの並列計算を N 行分まとめて処理するように実装した. 最後に, 収束判定 (15–17 行目) は, ベクトル $\mathbf{x}^{(k+1)}$ と $\mathbf{x}^{(k)}$ との距離 $\mathbf{x}_\Delta$ を並列数 N で計算し, この距離 $\mathbf{x}_\Delta$ に対する ℓ_2-ノルムと, $\mathbf{x}^{(k)}$ に対する ℓ_2-ノルムを, それぞれ, LAPACK の ℓ_2-ノルム API である scnrm2/dznrm2 で計算するように実装した. これらの演算結果は CPU メモリ側に返却されるため, CPU 側が, 2 つの ℓ_2-ノルム値を除算し, 除算結果を閾値 $\delta_{\mathrm{threshold}}$ と比較して反復制御を行うように実装した.

3.3 演算量の削減

連立方程式 $\mathbf{HX} = \mathbf{B}$ の直接計算法では, LU 分解やコレスキー分解といった行列分解アルゴリズムを用いて係数行列 $\mathbf{H}$ を分解し, 分解後の行列から代入操作で解 $\mathbf{X}$ を求める. 演算量を削減するための提案法を説明するため, まずは, 従来の行列分解手順について述べる.

図 4 に, 行列分解操作の基本手順を示す. 行列 $\mathbf{H}$ の分解は, 対角成分 ($\mathrm{diag}\{[H_{1,1}, H_{2,2}, \ldots, H_{N,N}]\}$) を左上隅から右下隅へと順に選択し, 選択した対角成分毎に, 係数行列 $\mathbf{H}$ の更新を行う. まず, 左上隅の対角成分 $H_{1,1}$ を選択して分解操作を開始する. 次に, 対角成分と同じ列ベクトル ($[H_{2,1}, H_{3,1}, \ldots, H_{N,1}]^T$) を更新する. この更新処理は, LU 分解を用いる場合, 各成分を, 選択した対角成分で除算する操作となる. その後, 選択した対角成分と同じ列 ($[H_{2,1}, H_{3,1}, \ldots, H_{N,1}]^T$) と同じ行 ($H_{1,2}, H_{1,3}, \ldots, H_{1,N}$) の積を計算し, 残る部分行列を更新する. この更新処理は LU 分解を用いる場合, 残る部分行列から上記の行列積の計算結果との差を求める操作となる. 以上の操作で, 選択した対角成分 $H_{1,1}$ と, 対角成分 $H_{1,1}$ と同じ行 ($[H_{1,2}, H_{1,3}, \ldots, H_{1,N}]$) と列 ($[H_{2,1}, H_{3,1}, \ldots, H_{N,1}]]^T$) に対する参照と更新が完了し, これらの行列成分が, 分解後の行列成分として確定する. 以降, 残る部分行列に対して, 同様の手順を繰り替えすことで, 行列 $\mathbf{H}$ の分解が完了する.

一方, 圧縮センシングに基づく高分解能 DoA 推定アルゴリズムでは, 連立方程式 $\mathbf{HX} = \mathbf{B}$ の係数行列 $\mathbf{H}$ の非対角成分は定数値であり, 対角成分も, 最適化の進行に伴い, 反復中に定数となる領域が増えていく特徴を持つ. すなわち, 図 5 に示すように, 係数行列 $\mathbf{H}$ の対角成分は, ベクトル $\mathbf{x}$ の各成分を式 (10) で修整した値となるため, 反復 (k) 周目においてベクトル $\mathbf{x}$ の零成分を, 反復 ($k+1$) 周目以降も零のまま不変な成分として扱えば, この零成分に対応する対角成分は, 修整量が一定となるため, 反復 (k) 周目以降は定数値となる.

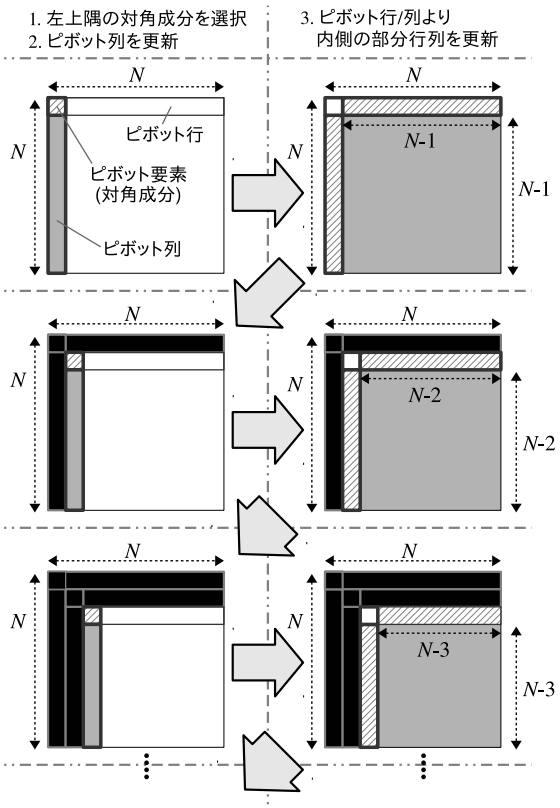

図 4 連立方程式を解く際の行列分解手順

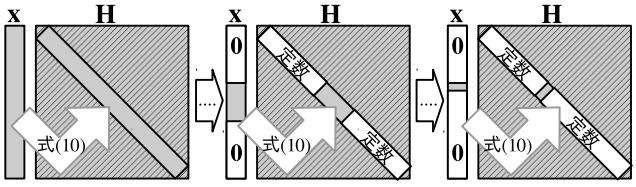

図 5 ベクトル $\mathbf{x}$ の疎性と行列 $\mathbf{H}$ の対角成分との関係

ここで, 連立方程式 $\mathbf{HX} = \mathbf{B}$ の計算が, 対角成分を順に選択して, 行列全体を更新していく点に着目すると, 対角定数成分が左上隅に集まっている場合, 反復 (k) 周目と ($k+1$) 周目以降で, 対角定数成分に関する演算手順が同一になることが分かる. 例えば, 反復 (k) 周目と ($k+1$) 周目で, 行列 $\mathbf{H}$ の左上隅の対角成分 $H_{1,1}$ が同じ値であれば, (k) 周目と ($k+1$) 周目で, 更新後の列ベクトル $[H_{2,1}, H_{3,1}, \ldots, H_{N,1}]^T$ と, 残る部分行列への修整量は同一である.

以上で述べた行列の分解手順と特徴を用いて, 行列 $\mathbf{H}$ の対角定数成分に対応する演算結果を再利用する手法を提案する. この手法では, 行列 $\mathbf{H}$ の分解時に, 対角定数成分を左上隅側に集約する並べ替え処理を行う. この並べ替え処理を用いて, 反復 (k) 周目の連立方程式の計算時に, 対角定数成分を基点とした係数行列 $\mathbf{H}$ に対する更新値を記録しておくことで, 最適化 ($k+1$) 周目以降の連立方程式の計算では, 対角定数成分を基点とした係数行列 $\mathbf{H}$ に対する更新値の計算を代入操作に置き換え, 演算量を削減する.

図 6 に, 提案法を導入した準ニュートン法による最適化の手順を示す. 図 3 に示す従来の手順とは異なり, 行列 $\mathbf{H}$ に対する更新値の計算結果を格納する領域 $\mathbf{P}$ と, ベクトル $\mathbf{x}$ の並べ替え情報 $\mathbf{q}$ を用いて, 連立方程式 $\mathbf{HX} = \mathbf{B}$ の計算量を削減している.

提案法を導入した最適化計算は, 反復毎に以下の手順で処理を行う.

Step 1. ベクトル x の並べ替え

ベクトル $\mathbf{x}$ を, 零成分が上端側に集まるように並べ替える. この際, 零成分であるか否かを判定する閾値を設定し, 零と判定された成分値を, 以降の反復で修整を加えないように扱う. また, この並べ替え順序の情報を $\mathbf{q}$ へ記録する.

Step 2. 行列 $\mathbf{H}_0, \mathbf{B}, \mathbf{P}$ の並べ替え

図 7 に示すように, 順序情報 $\mathbf{q}$ に基づき, ベクトル $\mathbf{x}$ の並べ替え順序に対応するように, 行列 $\mathbf{H}_0, \mathbf{B}, \mathbf{P}$ を行/列単位で並べ替える.

Step 3. 行列 H の対角成分の更新

式 (10) より, ベクトル $\mathbf{x}$ からベクトル $\mathbf{s}$ を計算し, 係数行列 $\mathbf{H}$ の対角成分に設定する. 図 8 に示すように, 対角定数成分が左上隅へと集約される.

Step 4. 連立方程式の計算

図 9 に示すとおり, 行列 $\mathbf{P}^{(k-1)}$ から, 反復 $(k-1)$ 周目までに記録した対角定数成分を基点とする更新値を取得し, これを係数行列 $\mathbf{H}$ に適用する. その後, 対応する更新値の記録を持たない対角成分を選択し, 残る連立方程式の計算を行う.

Step 5. 更新値の記録

図 9 に示すとおり, 新たに定数となった対角成分を基点とする係数行列 $\mathbf{H}$ に対する更新値として, 確定した成分値と部分行列に対する差分値を行列 $\mathbf{P}$ に記録する.

収束後, 順序情報 $\mathbf{q}$ に基づき, 本来の順序へ並べ替えた疎ベクトル $\mathbf{x}$ を, 推定結果として出力する.

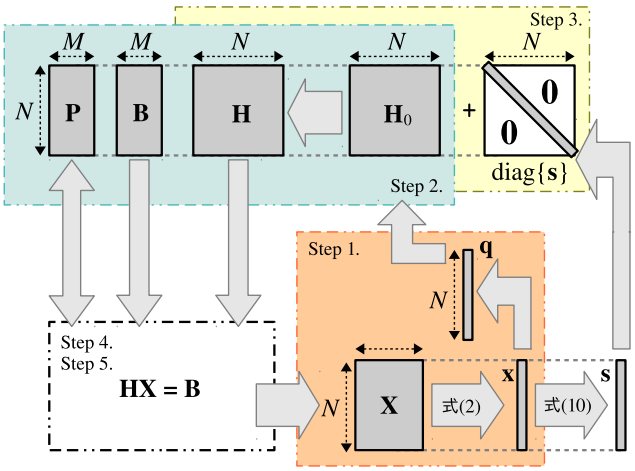

図 6 提案方式による高分解能 DoA 推定の処理手順

この演算結果の再利用手順は, 係数行列 $\mathbf{H}$ を分解した後の代入操作にも適用できる. 格納領域 $\mathbf{P}$ に記録した対角定数成分と, この対角成分と同じ行/列は, 分解後の行列の一部であるため, この部分に対応する代入操作の結果も, 計算結果を記録しておくことで, 同じ結果を再利用できる.

これにより, ベクトル $\mathbf{x}$ の零成分の数を C と置くと, $N \times N$ の行列に対して $\mathcal{O}(N^3)$ を要する反復毎の演算を, $\mathcal{O}\left((N-C)^3\right)$ へ削減した.

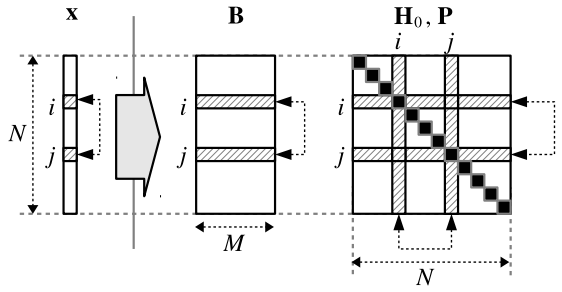

図 7 ベクトル $\mathbf{x}$ の並べ替えに対応する各行列の並べ替え

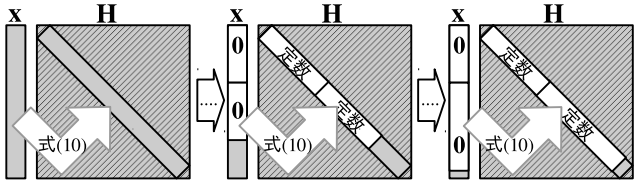

図 8 ベクトル $\mathbf{x}$ の疎性と行列 $\mathbf{H}$ の対角成分との関係 (提案手法)

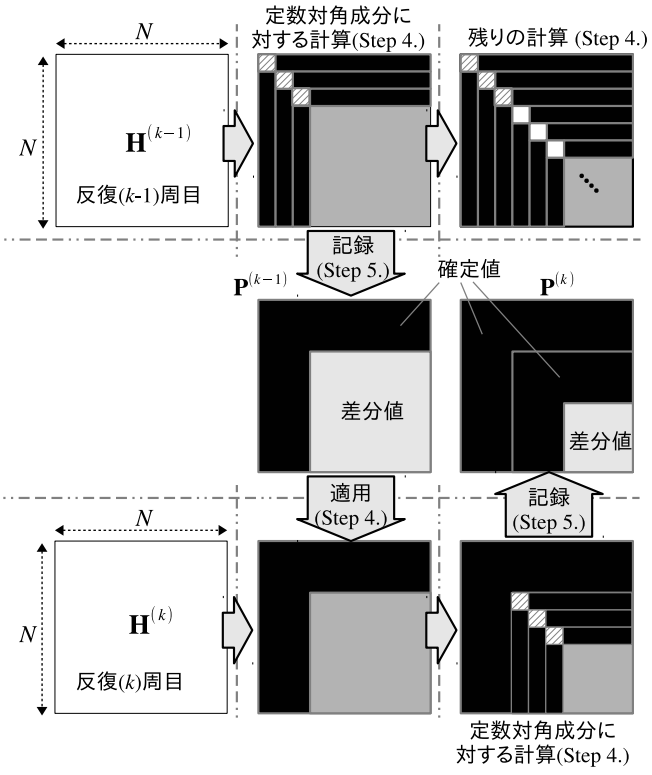

図 9 定数成分に対する演算結果の記録・再利用の手順

3.4 関連研究

圧縮センシングには, 制約項を ℓ_0-ノルムや ℓ_1-ノルムとするなど, 推定精度や演算量の特徴が異なる方式が多様に存在する. この高速化に関する先行事例としては, 低負荷な線形計画法として解を得る近似方式や, 反復毎に, 行列 $\mathbf{X}$ の成分値を零に置き換える方式等が報告されている [3][9][17].

特に, 文献 [18] では, 反復毎に連立方程式が巨大化する圧縮センシング方式に対し, 提案法と同様に, 反復間で行列分解結果を再利用する手法が示されている. ただし, これは, 特定の圧縮センシング方式に限定した手法であり, 提案法とは異なり, ベクトル $\mathbf{x}$ の零成分に対応した演算結果を再利用しない. なお, 提案法は, 疎ベクトルを最適解とする特徴を利用し, この零成分に対応する演算結果を再利用するように構成したものである. この特徴は, いずれの圧縮センシング方式においても共通するため, 提案法は, 他の圧縮センシング方式に対しても適用できると考えられる.

また, GPU を利用した並列高速化についても, ℓ_0-ノルムを制約項とした手法への適用事例 [2][6] や, ℓ_1-ノルムを制約項とした手法への適用事例 [4][7] が報告されている.

4. 性能評価

表 1 に示すプロセッサとソフトウェアを用いて, **3.2.** で示した並列実装方式の実測値に基づき, **3.3.** で示した提案法による演算量の削減効果の見積り評価を示す.

4.1 評価データ

処理時間を測定するための評価データとして, 到来波数を 8, アンテナ素子数 L を 16, 時間サンプル数 M を 512, そして, 角度点数 N を多様に設定し, 走査情報を表す $L \times M$ の複素行列 $\mathbf{A}$ と受信信号を表す $L \times N$ の複素行列 $\mathbf{Y}$ をシミュレーションによって複数生成した. 角度点数 N は, 251 から 12,001 の範囲で設定し, 計 18 種類の受信信号 $\mathbf{Y}$ を用意した. これらのデータを入力し, 図 10 に示すような最適化後の波源分布を出力するまでの処理時間を時間計測の対象とした.

また, 計測対象の処理は, CPU–GPU 共に倍精度演算で実施し, CPU については 6 コアを使ったマルチスレッド実行を設定した.

4.2 演算時間の見積り方法

提案法は, 角度点数 N に対して, 最適化の各反復におけるベクトル $\mathbf{x}$ の零成分の数 C に応じて, 演算量を $\mathcal{O}(N^3)$ から $\mathcal{O}\left((N-C)^3\right)$ へと削減したものである. まず, 提案法による演算時間の見積りを行うために, 最適化の反復毎に, ベクトル $\mathbf{x}$ の零成分の数を調査した. この際, 零成分を判定する閾値に 10^{-4} を設定し, ベクトル $\mathbf{x}$ の各成分の絶対値と比較するようにして, 零成分の数を集計した.

図 11 に, 最適化の進行に伴うベクトル $\mathbf{x}$ の非零成分の数と零成分の割合の推移を示す. なお, 図 11 の各凡例は, 角度点数 251 から 12,001 の範囲で用意したデータ 18 種を入力とした場合の各最適化計算に対応したものである. 図 11 上の零成分の割合に着目すると, 角度点数 N のサイズに依存せず, 最適化の早期段階で零成分の数が急激に増加し, 反復 15 周目を境に, 90% 以上の成分が零成分で構成された疎ベクトルとなっていることが分かる.

次に, 18 種ある各データの各反復時の非零成分の数 $(N-C)$ から, サイズ $(N-C)^2$ の係数行列 $\mathbf{H}$ に対する連

表 1 性能測定に使用したプロセッサとソフトウェア

	Intel CPU	NVIDIA GPU
型番	Xeon X5680	Tesla C2070
コア周波数	3.33 GHz	1.15 GHz
並列度	6 コア × 128bit SIMD	14 × 32 コア
理論性能 (倍精度)	79.9 GFLOPS	515.2 GFLOPS
コンパイラ	Intel Compiler 13.1.3	CUDA 4.2
ライブラリ	Intel MKL 11.1	cuBLAS 4.2 MAGMA 1.4.0

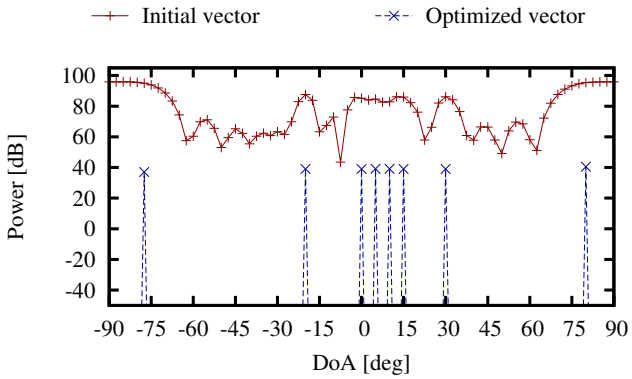

図 10 設定した到来波データに対する最適化前後の波源分布

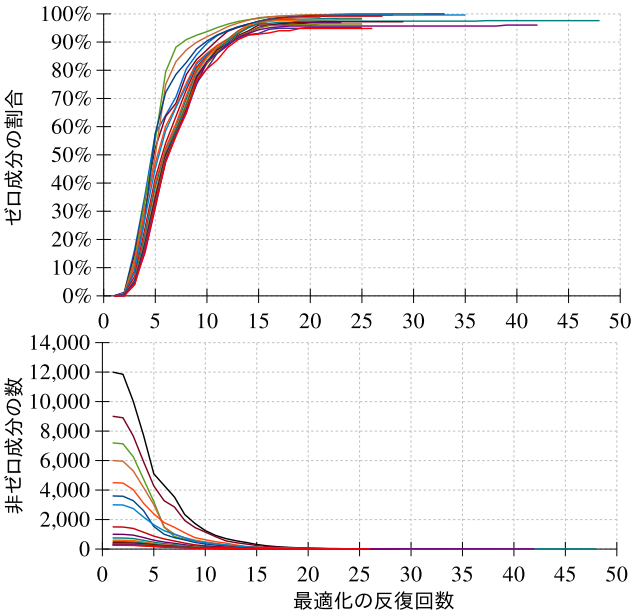

図 11 角度点数 N に応じた 18 種の最適化計算におけるベクトル $\mathbf{x}$ の非零成分の数 (下) と零成分の割合 (上) の推移

立方程式 $\mathbf{HX} = \mathbf{B}$ の計算時間を実測した．ここで, 零成分に対応した演算結果を再利用するためのオーバヘッドは考慮せず, 反復毎に, これらの実測した計算時間を, サイズ N^2 の係数行列 $\mathbf{H}$ に対する連立方程式 $\mathbf{HX} = \mathbf{B}$ の計算時間と置き換えることで, 提案手法による最適化時間の見積りを行った．

4.3 評価結果

表 2 に, 従来法と提案法のそれぞれに対して, CPU と GPU を利用した際の処理時間を示す．また, 図 12 に, 角度点数 N を 3,601 とした場合の, 最適化の進行に伴う処理時間の推移を示す．このうち, 従来手法と CPU/GPU の組み合わせについては実測値であり, 提案手法に関しては見積り値となる．また, 表 3 に, GPU による高速化の効果と, 提案手法と GPU を組み合わせた場合の高速化の効果を示す．

表 2 各手法および各問題サイズにおける処理時間 (秒)

	従来法		提案法	
N	CPU	GPU	CPU	GPU
251	0.18	0.24	0.05	0.13
273	0.21	0.25	0.05	0.13
301	0.41	0.44	0.07	0.22
334	0.29	0.31	0.07	0.15
376	0.34	0.32	0.09	0.15
429	0.57	0.44	0.12	0.19
501	1.29	0.95	0.19	0.31
601	1.00	0.62	0.22	0.21
751	1.55	0.84	0.34	0.26
1,001	**2.64**	1.05	0.58	0.33
1,501	7.37	**2.76**	1.41	0.63
3,001	52.55	16.25	6.83	2.11
3,601	91.12	19.98	9.65	**2.74**
4,501	132.74	34.47	17.44	4.70
6,001	310.54	77.34	35.20	8.89
7,201	529.19	129.62	55.17	13.73
9,001	1,025.81	250.97	105.08	26.41
12,001	2,349.31	554.85	228.44	59.84

表 3 GPU および提案法による高速化率

	GPU		提案法 +GPU	
	(従来法同士)	(提案法同士)	従来法 +	従来法 +
N	CPU 比	CPU 比	GPU 比	CPU 比
251	0.77	0.34	1.79	1.38
273	0.85	0.39	1.90	1.61
301	0.93	0.34	1.97	1.83
334	0.94	0.49	2.13	1.99
376	1.06	0.59	2.19	2.33
429	1.28	0.62	2.33	2.99
501	1.36	0.62	3.12	4.23
601	1.62	1.01	2.89	4.68
751	1.86	1.29	3.21	5.96
1,001	2.52	1.74	3.16	7.94
1,501	2.67	2.25	4.41	11.76
3,001	3.23	3.24	7.70	24.90
3,601	4.56	3.52	7.29	33.26
4,501	3.85	3.71	7.33	28.22
6,001	4.02	3.96	8.70	34.94
7,201	4.08	4.02	9.44	38.55
9,001	4.09	3.98	9.50	38.85
12,001	4.23	3.82	9.27	39.26

まず, CPU に対する GPU の高速化効果は, 最大で 4 倍程度となった．また, 従来法に対する提案法の高速化効果は, 1.8 倍から 9.5 倍程度となった．この GPU と提案法を組み合わせた場合, 従来手法と CPU の組み合わせを基準とすると, 高速化効果は, 角度点数 N の値が大きいほど高く, 最大で角度点数 N が 12,001 の場合に 39.3 倍程度の効果となることが分かった．これは, 従来法が, 角度点数 N に大して $\mathcal{O}(N^3)$ の演算量を要するところを, 提案法では, ベクトル $\mathbf{x}$ の零成分の数 C に応じて演算量を $\mathcal{O}\left((N-C)^3\right)$ へ改善したためである．

次に, 角度点数 N を 1,001 に設定した推定では, 従来法を CPU で実行した場合は約 2.6 秒を要していたが, 従来法を GPU で実行した場合は, これと同等の時間で, 角度点数 N を 1,501 に設定した推定を実施できることが分かった．同様に, 提案法を GPU で実行した場合は, これと同等の時間で, 角度点数 N を 3,601 に設定した推定を実施できることが分かった．すなわち, GPU による並列計算と, 演算量を削減する提案法の両者を用いることで, 従来法を CPU で並列計算した場合と同等の処理時間で, 約 3.6 倍の角度分解能を要する DoA 推定を実施できる見込みが得られた．

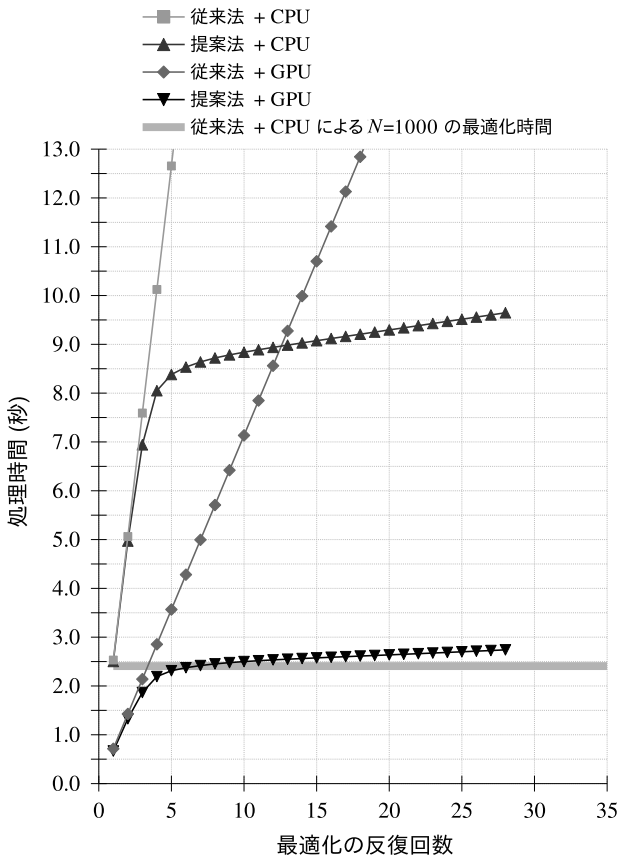

図 12 各方式における反復毎の処理時間の推移 ($N = 3,601$)

5. まとめと今後の課題

本稿では, 圧縮センシングに基づく高分解能な DoA 推定アルゴリズムの高速化を目的に, GPU 向け並列プログラムの実装方式を示し, 未知行列の零成分に対応する演算結果を再利用する演算量削減法を提案した.

この高速化効果として, 角度点数 $N = 1,001$ の波源分布を, CPU による並列計算および従来法で推定する場合と, 角度点数 $N = 3,601$ の波源分布を, GPU による並列計算および提案法で推定する場合が, 同等の約 2.6 秒で完了する見積り結果を得た. すなわち, GPU による並列計算と提案法を用いれば, 演算時間の増加を招くことなく, 従来に比べて 3.6 倍程度の角度分解能の向上を見込めることが分かった.

今後は, 本稿で示した提案法を実装し, 実測性能の評価に取り組む予定である. また, 連立方程式の計算アルゴリズム, 最適化アルゴリズム, そして, 圧縮センシングの方式が異なる場合に対する提案法の適用方式と高速化効果の調査が今後の課題である.

参考文献

[1] Angerson, E., Bai, Z., Dongarra, J., Greenbaum, A., Mckenney, A., Du Croz, J., Hammarling, S., Dammel, J., Bischof, C., and Sorensen, D.: LAPACK: A Portrable Linear Algebra Library for High-Performance Computers, in *Proc. Supercomputing '90*, pp. 2–11, 1990.

[2] Blanchard, J. D., and Tanner, J.: GPU accelerated greedy algorithms for compressed sensing, *Mathematical Programming Computation*, Vol. 5, No. 3, pp. 267–304, Springer, 2013.

[3] Blumensath, T., and Davies, M. E.: Iterative hard thresholding for compressed sensing, *Applied and Computational Harmonic Analysis*, Vol. 27, No. 3, pp. 265–274, 2009.

[4] Borghi, A., Darbon, J., Peyronnet, S., Chan, T. F., and, Osher, S.: A Simple Compressive Sensing Algorithm for Parallel Many-Core Architectures, *Journal of Signal Processing Systems*, Vol. 71, No. 1, pp. 1–20, Springer, 2013.

[5] Dohono, D. L.: Compressed sensing, *IEEE Trans. Information Theory*, Vol. 52, No. 4, pp. 1289–1306, 2006.

[6] Fang, Y., Chen, L., Wu, J., and Huang, B.: GPU Implementation of Orthogonal Matching Pursuit for Compressive Sensing, in *Proc. IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS 2011)*, pp. 1044–1047, 2011.

[7] Fiandrotti, A., Fosson, S. M., Ravazzi, C., and Magli, E.: PISTA: Parallel Iterative Soft Thresholding Algorithm for Sparse Image Recovery, in *Proc. 2013 Picture Coding Symposium (PCS)*, pp. 325–328, 2013.

[8] Ge, D., Jiang, X., Ye, Y.: A note on the complexity of L_p minimization, *Mathematical Programming*, Vol. 129, No. 2, pp. 285–299, Springer, 2011.

[9] Hale, E. T., Yin, W., and Zhang, Y.: Fixed-point continuation for ℓ_1-minimization: Methodology and convergence, *SIAM Journal on Optimization*, Vol. 19, No. 3, pp. 1107–1130, 2008.

[10] Lawson, C. L., Hanson, R. J., Kincaid, D.R., and Krogh, F. T.: Basic Linear Algebra Subprograms for Fortran Usage, *ACM Trans. Mathematical Software*, Vol. 5, No. 3, pp. 308–323, 1979.

[11] Malioutov, D. M., Çetin, M., and Willsky, A. S.: A Sparse Signal Reconstruction Perspective for Source Localization with Sensor Arrays, *IEEE Trans. Signal Processing*, Vol. 53, No, 8, pp. 3010–3022, 2005.

[12] Miller, M. I., and Fuhrmann, D. R.: Maximum-likelihood narrow-band direction finding and the EM algorithm, *IEEE Trans. Acoustics, Speech and Signal Processing*, Vol. 38, No. 9, 1990.

[13] NVIDIA: cuBLAS, [Online], Available: `https://developer.nvidia.com/cublas` [Accessed: 21 Oct. 2014].

[14] Tomov, S., Nath, R., Ltaief, H., and Dongarra, J.: Dense Linear Algebra Solvers for Multicore with GPU Accelerators, in *Proc. 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–8, 2010.

[15] Veen, B. D. V., and Buckle, K. M.: Beamforming: a versatile approach to spatial filtering, *IEEE ASSP Magazine*, Vol. 5, No. 2, pp. 4–24, 1988.

[16] Wang, H., and Kaveh, M.: Coherent Signal-Subspace Processing for the Detection and Estimation of Angles of Arrival of Multiple Wideband Sources, *IEEE Trans. Acoustic, Speech and Signal Processing*, Vol. 33, No. 4, pp. 823–831, 1985.

[17] Xu, Z., Chang, X., Xu, F., and Zhang, H.: $L_{1/2}$ Regularization: A Tresholding Representation Theory and a Fast Solver, *IEEE Trans. Neural Networks and Learning Systems*, Vol. 23, No. 7, pp. 1013–1027, 2012.

[18] Yang, D., Peterson G. D., and Li, H.: Compressed sensing and Cholesky decomposition on FPGAs and GPUs, *Parallel Computing*, Vol. 38, No. 8, pp. 421–437, Elsevier, 2012.

[19] 高橋善樹, 鈴木信弘, 若山俊夫, 網嶋武, 原六蔵: DOA 推定のための到来波情報を用いたスパース信号処理, 信学技報, A·P, Vol. 113, No. 384, pp. 105–110, 2014.