GPUによる相互相関PIVの高速化に関する研究

2011年1月27日提出

人間環境学専攻

田良島 周平

岡本孝司　教授

目 次

図目次

表 目 次

第1章　序論

1.1　研究背景

1.1.1　PIV

流体の流れ場の中には，肉眼で直接観察することが不可能，もしくは困難であるものが多い．そのような複雑な流れを観察するため，流れを人間が認知しやすい視覚的情報 (例えば色や線，マーカー等) を用いて表示することが一般的におこなわれ，これらは流れの可視化 (Flow Visualization) と呼ばれる．効果的に流れの可視化をおこなうには，流れからその速度や温度に関する定量データの計測が必要となるが，速度計や温度計を用いた計測では，可視化のために必要な平面または空間内の多点における情報を取得することは難しく，またコストも高い．

Particle Image Velocimetry(粒子画像流速測定法，PIV) は，このような観察困難な流れの，ある範囲における瞬時・多点の速度情報の計測および可視化をおこなう手法のひとつである．PIV において流れの速度場は，流れ内に散布されたトレーサー粒子をカメラで撮影し，得られた画像をコンピュータで解析することによって導出される．PIV のシステムは，観察する流体の他にカメラ，レーザー等の照明装置，トレーサー粒子，データ解析用途のコンピュータから構成されているが，いずれの機器の性能向上も著しく，その結果高精度，高時空間分可能を持つさまざまな PIV システムが提案，実現されている．

ここでは一例として PIV 実験に用いられる撮影機器に着目する．近年 PIV への高速度カメラの応用が進み，時間分解能の非常に高い PIV(例えば，[30]) が実現されている．高速度カメラの時間分解能は現在最高で数十 KHz にもおよび，これを用いることでマイクロ秒オーダーで変化する現象の解析および可視化も実現されている．一般的に，これらの撮像結果は画像データとして HDD 内に蓄えられ，実験後に画像解析がおこなわれるが，このとき蓄積されるデータは非常に膨大であり，現象 1 秒あたりのデータサイズが数 GB にのぼることも決して珍しくない．解析対象である画像データの大幅な増大に伴い，PIV 実験で得られた画像の処理にかかる時間コストが問題視されるようになってきており．PIV 処理を高速化，効率化する必要性が顕在化

してきているといえる．

一方でこの PIV 画像処理の高性能化を目指した研究は，PIV の研究が始まった当初から活発におこなわれてきた．現在までに様々な手法，さまざまなアルゴリズムが提案されているが，その多くは，90 年代初頭におこなわれた Utami et al. [39]，Willert and Gharib [50]，Adrian [4]，Westerweel [47] らの研究により構築された相互相関 PIV(Cross-correlation PIV，相互相関法) の理論を基礎としている．このアルゴリズムが，Jambunathan [15]，Hart [14]，Scarano [36] らの手によって改良が加えられ，現在では 4×4[pixel] あたりに 1 ベクトルを導出する画像内ベクトル解像度および $1:100$ のダイナミックレンジをもつ PIV 画像処理アルゴリズムが既に開発されている．PIV の次元拡張も進められており，流れ場の 3 次元的挙動を計測する Stereo PIV [32] や Tomographic PIV [9] といった手法が開発されているが，これらの手法で用いられる画像処理アルゴリズムの多くは，相互相関 PIV を応用・発展させたものである．相互相関法は 2 次元 PIV だけでなく，高次元 PIV の基礎もなしているといえる．

相互相関 PIV を基礎に発展を続けてきた PIV アルゴリズムであるが，その精度およびベクトル解像度の向上に伴い，その計算負荷は増加傾向にある．現在の比較的新しい CPU を用いて一般的な PIV 処理を行った場合，その処理には通常数百ミリ秒から数分の時間がかかり，これはたとえば高速度カメラの数 KHz〜数十 KHz の時間解像度に比べると圧倒的に遅い．そしてこの PIV 画像処理の速度で高速度カメラから得られた画像を全て解析することを考えると，その処理には数時間〜数十時間を必要とする．実験の実施からそのデータ解析の一連の流れの効率化を考えたとき，PIV 画像処理がひとつのネックとなっていることは否定できず，この理由からも PIV 画像処理の高速化について検討する必要がある．

PIV 画像処理を高速化することは，PIV 全体のワークフローを効率化を実現するだけでなく，PIV の適用分野を更に拡げる可能性を持っている．その可能性を示す研究の一例として，リアルタイム PIV([53]，[51]) が挙げられる．名前が示すとおり，リアルタイム PIV はカメラから得られた撮像データに直接 PIV 処理をおこない，その結果を可視化する技術であり，その技術を応用した研究がいくつかおこなわれている．中でも Willert ら [51] はリアルタイム処理が可能な PIV システムをセンサーとして利用し，低レイノルズ数流れ場中の翼の姿勢制御を試みている．この技術は研究段階でありいまだ確立されたものでないが，PIV の新たな可能性の一端を示す研究として注目に値する．このリアルタイム PIV は，実験システムの処理速度に画像データ処理が追従できなければ実現およびシステムの性能向上は難しく，よってシステム構築に PIV 処理速度の向上は不可欠である．このように PIV 画像処理の高速化は，単に現存する問題を解決するための手段にとどまらず，今後の PIV の更なる発展を担う要素のひとつであるといえるだ

ろう.

ここまでで PIV 処理の高速化の必要性について述べてきたが，ここでその方針について考えたい．一般に処理を高速化する方針として，その計算処理を何らかのかたちで省略し，精度を犠牲にする代わりに処理速度を上げるという手段はひとつの選択肢として考えられる．しかし少なくとも対象とする流れ場に応じた設定およびチューニングが必要な PIV 画像処理に関しては，計算処理を省略することによる処理速度の向上は個々の対象ごとに議論されるべきことであると筆者は考える．現在の主流である相互相関 PIV 処理の高速化について，極力一般的なかたちで検討し評価する必要があるといえる．そのためには少なくとも，精度とのトレードオフなしに処理の高速化を実現する方法を検討する必要がある．

1.1.2 GPGPU による汎用計算の高速処理

並列計算 (Parallel computing) は，精度を犠牲にすることなく計算処理速度を向上させる手法として一般的に用いられる有効な手段である．並列計算をおこなう手段にはいくつかの選択肢が考えられるが，なかでも数年前から新たなプラットフォームとして GPGPU(General Purpose computing on GPU) と呼ばれる，Graphics Processing Unit(GPU) を用いた汎用計算の並列化手法が注目を集めている．この方法は，本来グラフィックス処理専用の演算器であった GPU が持つ高い並列処理能力を，一般の数値計算や画像処理，シミュレーション等に活用しようとするものである．GPU は非常に多くのプロセッサから構成されており，それらが並列処理をおこなうことで非常に高い性能を出す．GPGPU が可能な GPU を製造している主な企業には NVIDIA 社や AMD 社などが挙げられるが，NVIDIA 社製の最新 GPU はその内部に 500 以上のプロセッサコアを搭載 [2] し，AMD 社製の GPU ならば単体の GPU あたり 1500 以上ものコアを有している [1]．その理論性能は現在 1TFLOPS を超えており，メモリ帯域幅も 150GB/s を上回っている．同規模の理論性能をもつ CPU ベースのスーパーコンピュータと比較しても，その消費電力やコストに関しては GPU が優位である．

GPGPU は今までにもさまざまな分野での応用がすすんでいる．現在もなお GPU の性能は年々向上しており，今後もその応用分野は拡大していくものと考えられる．GPGPU がさまざまな分野で応用されるようになった原因はさまざま考えられるが，大きな要因のひとつとして，ストリームプロセッシング可能なプログラミング環境が大きく発展した点が挙げられる [44]．現行の比較的高性能な GPU は数百の計算コアを有するが，このひとつひとつのコアを意識しながらプログラミングをおこなうことは非常に困難である．しかしながらストリームプロセッシングに

おいて，プログラマは個々のコアを意識する必要はなく，SPMD(Single Program Multiple Data) 型のプログラムを記述すればそれ以降のスケジューリングはコンパイラにより自動で最適化される．なお GPU のプロセッサコアは複数コアでひとつの ALU(Arithmetic Logic Unit) を共有しており，したがってそれらのコアは命令レベルで同じ処理をおこなう SIMD(Single Instruction Multiple Data) 型演算器であるが，性能低下が妥協できる状況であれば実質それすらも意識する必要はない．現在ストリームプロセッシング可能な言語として，NVIDIA 社製 GPU で動作する CUDA，スタンフォード大学で開発された Brook GPU，更には GPU を含む様々なマルチコア計算環境での並列計算を可能にする OpenCL などが存在する．すべての GPU 環境に対応可能であるという点で OpenCL は今後更なる普及が見込まれるが，ストリームプロセッサとしての GPU を用いるためのプログラミング環境であるという点でこれらの言語に本質的な差はない．OpenCL の普及は発展途上の段階にあり，少なくとも NVIDIA 社製の GPU を用いるのであれば CUDA を用いたほうが開発は容易であり，GPU のパフォーマンスをより引き出すことができるというのが現状である．

このように非常に高い性能を誇る GPU であるが，GPU 特有の制約も存在する．GPU を構成するプロセッサコアは，複数プロセッサで ALU のような論理演算ユニットを共有しているため，一般に条件分岐命令などの論理演算が多く含まれる処理ではその性能が大きく低下する．また ALU を共有するプロセッサコア同士は命令レベルで同一の処理のみしか同時に処理できず，それらのコア内で分岐命令が発生するような処理ではスループットが大きく低下する．また本来 GPU は単精度の浮動小数点を用いた画像処理をおこなう演算器であるため，倍精度以上の浮動小数点の演算コストが非常に高い．新しい GPU が発表されるごとにその計算処理性能は上がっているが，現状いまだ単精度浮動小数点演算の処理速度に比べるとその差は大きい．更に，現行の GPU は単独で動作するハードウェアではなく，その処理は CPU を介しておこなわれるため，命令やデータ通信のオーバーヘッドを回避することができない．現在の高性能 GPU は PCI Express によってマザーボードと接続されデータ通信がおこなわれているが，既にこのデータ通信が GPU を利用した処理全体のボトルネックになっている例が報告されている．GPGPU では複数 GPU を用いたマルチ GPU 並列計算をおこなうことが可能であるが，現行のマルチ GPU 並列計算では GPU 間のデータ通信を直接的におこなうことが不可能であり，データ通信の問題がより顕著になることが多い．なお 2011 年 1 月に，Intel 社が内部に GPU のチップセット (現在 iGPU と呼称される) を搭載する CPU の発売を開始しており，AMD 社も APU(Accelerated Processing Unit) と呼称される GPU 統合型 CPU の発売を開始している．これらの CPU と一体化した GPU を用いれば，データ通信の問題は基本的に解決されることになる．しかし CPU 内部に搭載され

る GPU チップセットの性能は現行の高性能 GPU に比べ低く，GPU 統合型 CPU が現状の GPU と同等の性能を獲得できるか否かは現状では未知数である．

ここで相互相関 PIV の GPU 並列計算への適用性について考察を試みる．PIV 実験で得られた画像データの解析処理は本質的には画像処理であり，次章で述べる PIV 画像処理の詳細が示すように並列性も高い．GPU が本来グラフィックス専用のハードウェアであったことも考慮すれば，PIV 処理が GPU によって効果的に高速化される可能性は比較的高いと考えられる．PIV 処理は特殊な状況を除けば単精度浮動小数点の精度で十分であり，このことも PIV 処理が GPU により高速化される可能性の高さを示唆している．しかし，PIV 処理のたびに PIV 粒子画像データをバスを介した転送する必要が発生するため，これが GPU で PIV 処理をおこなう際のネックとなる．また，現行の GPU の仕様を用いた場合画像データの転送は不可避である以上，GPU 並列計算による PIV 画像処理では画像データ転送に必要な時間以上の処理速度を得ることはできず，その処理高速化にはある限界が存在すると考えられる．

このように，GPU 並列計算は相互相関 PIV の高速化の手段として有効である可能性が高いが，その効果を具体的に検証するには実装をふまえる必要がある．ストリームプログラミング環境である CUDA や OpenCL を用いて相互相関 PIV を実装する手法が確立されてはおらず，考えられる手法の中からもっとも高いパフォーマンスを得られる実装方法を明らかにする必要がある．この実装手法の検討により得られた PIV のパフォーマンスを，他手法・他手段による高速化手法と正確に比較するためにも，GPU による PIV 画像処理のパフォーマンスを定量的に評価する必要がある．

本研究は，現在 PIV 画像処理の手法として実質標準的に用いられる相互相関 PIV の高速処理システムの開発を目指している．その手段として GPU 並列計算に着目し，相互相関 PIV に対する GPU 並列計算に有用性を具体的に明らかにするものである．

1.2　関連研究

本節では PIV 画像処理の高速化に関するいくつかの先行研究について取り上げる．PIV 画像処理の高速化については今までに数多くの研究がなされているが，ここではその中から典型的なものを取り上げることで，本研究の位置付けおよび目的を明確にする．

計算負荷を軽減させることによる高速化

相互相関 PIV は画像中に設定された小領域内における相互相関の値を評価する方法であるが，この相互相関をコンピュータで計算する方法はいくつかの手法が提案されている．たとえば，(1.1) 式に示す相互相関係数 (Cross-correlation coefficient) の分子の計算は，高速フーリエ変換 (Fast Fourier Transform，FFT) によって高速に処理できることが知られているものの，その分母の計算には FFT のような手法が存在せず，計算量の高い式として知られている．式 (1.1) で N はデータサイズ，R_{fg} は異なる 2 つの信号 f，g の相互相関係数であり，f_m，g_m は各信号の平均を表している．

$$R_{fg}(\Delta X) = \frac{\sum_{i=1}^{N}\{f(X_i) - f_m\}\{g(X_i + \Delta X) - g_m\}}{\sqrt{\sum_{i=1}^{N}\{f(X_i) - f_m\}^2 \sum_{i=1}^{N}\{g(X_i + \Delta X) - g_m\}^2}} \tag{1.1}$$

Lewis [20] は，相互相関係数を評価式として画像内のテンプレートマッチングをおこなう際に，その分母の計算量を削減する方法を提案している．通常テンプレートマッチングは対象画像中におけるテンプレートの位置を変えながら複数回 (1.1) 式が計算されるが，分母の計算時に必要となる $\{f(X_i)\}^2$ および $\{g(X_i)\}^2$ の計算値はテンプレートの位置によらず一定である．そこで Lewis の方法では，対象画像およびテンプレートの各画素値の自乗値をあらかじめ計算して配列に保持しておくことで，相互相関係数の分母の計算に必要な乗算の回数を最低回数に抑えている．この方法は通常の 2 倍のメモリ空間を必要とするものの，精度を落とすことなく計算量を削減することを可能にする手法であるといえる．

一方で，Yoo [53] らは相互相関係数全体の計算に着目し，相互相関係数を用いたテンプレートマッチングを FFT すら用いることなくを高速化する方法を提案している．彼らの手法は，相互相関ピークの位置におけるテンプレートと対象画像の信号波形が完全に一致するという仮定のもと式 (1.1) の変形をおこない，規格化された信号の値を 1 または -1 の整数値で近似するこ

とで，実質乗算をおこなうことなく相互相関係数の計算が可能であるとしている．当然 Yoo らの方法で得られた相互相関係数の値は近似値であり (1.1) の直接計算から得られる値と異なるが，その処理速度に関しては Lewis の方法より更に大幅な計算量削減を実現している．

本研究は相互相関係数の計算に関するこれらの研究の比較をおこなうものではないが，その立場としては前者の Lewis の研究に近い立場をとる．すなわち，計算精度が低下する可能性をはらむ計算の省略や単純化といった手法については検討をおこなわない．上で示した例のほかに，相互相関係数の分子を整数演算の FFT により更に高速に計算するといった比較的有名な方法も存在するが，PIV にその手法を適用した際の精度については評価が必要であり，それは本研究の範囲外とする．本研究はあくまで，精度を落とすことなく相互相関 PIV の高速化を実現する手法について検討をおこなう．

ハードウェアを活用した高速化

本研究では GPU を用いて相互相関 PIV 画像処理の高速化を検討するが，GPU を含め専用のハードウェアを利用した PIV 高速処理システム開発に関する研究は既にいくつかおこなわれている．

たとえば Maruyama ら ([23]，[10]) は，FPGA(Field-Programmable Gate Array) に相互相関 PIV 処理専用のチューニングをほどこし，1008×1008[pixel] の PIV 画像ペアを 20Hz で解析するシステムを構築している．また Leeser ら [19] は，FPGA を内蔵したスマートカメラを構築することで，1024×1024[pixel] の PIV 画像の処理をデータ転送も含めて 160[ms] で実現している．更に，Schiwiets ら [38] はプログラマブルシェーダ (Programmable shader) を用いた GPU 並列計算によって FFT 相互相関 PIV の処理を実装しており，1024×1024[pixel] の PIV 画像ペアを最高 13Hz で処理することに成功している．いずれの研究も，相互相関 PIV の中で最も計算コストの高い相互相関の計算に焦点を当てて高速化を試みたものである．

このように専用のプロセッサをチューニングすることで PIV 処理を高速化する試みは既におこなわれており，一定の成果が得られている．したがって，GPU による相互相関 PIV の高速化を目指している本研究のアプローチは決して新しいものではない．しかし，ここで挙げた PIV 高速化の研究はハードウェアの有用性を PIV アルゴリズムを用いて主張している傾向が比較的強く，PIV 処理の詳細についてはあまり触れられていない．次章で詳説するように，現在用いられている PIV 画像処理アルゴリズムは複数の処理要素から構成されている．最も計算コストの高い相互相関計算を高速化することが非常に重要であることは先行研究から明らかであるが，

PIV 画像処理の精度やベクトル解像度に大きく影響するその他の処理について並列化および高速化を検討することもまた必要である．本研究では，PIV 画像処理を構成する各要素について GPU 並列計算への適用方法を検討し，その効果を評価する．現在広く用いられている高精度・高ベクトル解像度の PIV 画像処理システムの実装を目指しているという点で，本研究はここで挙げた既存研究とは立場が異なる．

直接相互相関法と FFT 相互相関法

一般に，相関の計算手法には定義式に従って直接計算をおこなう方法と，Wiener-Khintchine の定理に基づく FFT を利用した方法が存在する．相互相関 PIV にもこれに対応するかたちで二種類の計算方法が存在し，相互相関を直接計算する手法は直接相互相関 PIV(Direct cross-correlation PIV，直接法)，FFT を利用して計算をおこなう手法は FFT 相互相関 PIV(FFT-based cross-correlation PIV，FFT 法) と呼ばれる．共に相互相関の計算をおこなうという点で両手法に差異はないが，実際の計算方法は大きく異なっており，処理速度や結果導出されるベクトルの精度も異なる．当然，GPU 並列計算への適用性も手法によって異なるはずであり，その実装方法については個々に検討する必要がある．

直接法と FFT 法の比較をおこなった研究としては，Pust [33] の研究が挙げられる．Pust は，PIV 処理の結果得られるベクトルが整数近辺の値に偏ってしまうピークロッキング現象 ([34] [54]) に焦点を当て，直接法と FFT 法の精度について比較をおこなっており，直接法の方がピークロッキング現象の発生が抑えられていることを示している．しかしその処理速度については，簡潔な議論はおこなっているものの定量的な評価まではなされていない．

一般に，直接法による相互相関計算の負荷は FFT 法のそれよりも高く，データサイズが増すごとに 1 相互相関あたりの直接法の計算コストの差は大きくなる．しかしこと PIV に関しては，直接法で指定されるパラメータのとりようによっては必ずしもその事実は当てはまらず，直接相互相関法の処理速度が FFT 法のそれを上回るケースが存在しうる．直接法は一般に FFT 法よりも結果の精度が高くかつよりロバストであるため，直接法が処理速度で上回るようなケースが存在する場合，そのケースは直接法が精度だけでなく処理速度の観点からも FFT 法よりも有効なケースであるということになる．

本研究では，まず直接法と FFT 法各々について GPU 並列計算への実装方法を検討したのち，両手法の処理速度について比較をおこなう．その結果をもとに，手法を横断した高速 PIV 画像処理手法開発の検討をおこなう．

1.3 研究の目的

以上の研究背景および先行研究のもと，本研究は高速かつ実用的な PIV 画像処理システムの開発を目的とする．その手段として GPU 並列計算に着目し，その性能を可能な限り活用できる実装手法を検討しながら実用的高速 PIV 画像処理システムの開発をおこなう．

本研究の目的を達成するため，本研究は具体的に以下の 3 項目について研究をおこなう．

1. GPU を用いた相互相関 PIV アルゴリズム高速化手法の検討と評価

本研究はまず，相互相関 PIV のアルゴリズムを GPU 並列計算に実装する手法について検討する．GPU の詳細な仕様は様々存在するが，本研究では GPU を階層構造をもったメニーコアストリーミングプロセッサととらえ，極力実装方法について抽象的な議論をおこなったうえで，実際のハードウェアへの実装方法をおこなう．なお，本研究では NVIDIA 社製 GPU に実装をおこない，開発言語としては CUDA を用いる．

相互相関 PIV の計算手法としては直接相互相関法と FFT 相互相関法が考えられる．共に相互相関の計算手法であるがその仕組みは異なっており，並列計算への実装は独立に検討されるべきである．本研究ではこの 2 手法について個々に実装方法を検討評価する．

2. マルチ GPU を用いた相互相関 PIV の高速化手法の検討と評価

マルチ GPU を用いることで，シングル CPU の更に数倍のコアプロセッサを利用した並列計算が可能となるが，その並列化にあたってはマルチ GPU 特有の制約に対処する必要があり，シングル GPU で採用した計算方法をそのまま使用できるとは限らない．

本研究では，マルチ GPU を用いた相互相関 PIV の更なる高速化手法についても検討評価をおこなう．

3. 直接法と FFT 法のハイブリッド相互相関 PIV アルゴリズムの提案と評価

直接相互相関法と FFT 相互相関法は精度および速度について対照的な関係にある．すなわち，直接法は解析結果が高精度かつロバストである代わりに処理時間コストが高く，一方で FFT 法は高速に処理可能であるものの精度やロバスト性に関して直接法に劣る．しかし先行研究の節で述べたように，直接法には FFT 法よりも精度だけでなく処理速度も優位となりうるケースが

存在し得る．このケースを活用することで，直接法および FFT 法の両者のメリットを活かした PIV アルゴリズムが実現する可能性がある．

本研究では，開発した直接相互相関 PIV と FFT 相互相関 PIV との処理速度の比較をおこない，その結果に基づいて両手法を融合した手法をハイブリッド相互相関 PIV と名付けその提案および評価をおこなう．

1.4 本論文の流れ

本論文は，本章および結論を述べた計 6 つの章で構成されている．

2 章では，本研究で高速化の対象となる相互相関 PIV のアルゴリズムについて概説する．相互相関 PIV のアルゴリズムはさまざまなものが考えられ，本研究ではその全てをカバーすることはできていない．本章で，本研究で高速化の対象とするアルゴリズムについても具体的に定義する．

3 章では，シングル GPU の高速化手法の検討とその評価をおこなう．2 章で示される高速化の対象となる各処理を GPU 並計算に実装する手法の提案し，本研究の計算環境における実装結果をふまえた議論をおこなう．

4 章では，前章でのシングル GPU を用いた検討に続き，マルチ GPU を用いた相互相関 PIV の高速化手法について検討する．

5 章では，まず 3 章で共に GPU 並列計算に実装された直接法と FFT 法についてその処理時間の観点から比較をおこなう．この比較をふまえ，両手法を融合したハイブリッド相互相関 PIV の提案をおこなう．提案した手法の評価をおこなうため，人工画像および実験で得られた実データを用いた評価をおこなう．

第2章　相互相関PIV

本章では，高速化の対象となる相互相関 PIV について概説する．まず 2.1 で，現在一般的に用いられる相互相関 PIV を構成する各処理について述べる．その中から，本研究が高速化の対象とするアルゴリズムを 2.2 で定義する．

2.1　相互相関法

本節では相互相関 PIV の最も基本的な処理について述べる．相互相関 PIV により導出されるベクトルは，画像間における局所的輝度値パターンの移動量として定義され，その評価に相互相関が用いられる．画像上に複数設定された各速度計測点に対して，各計測点を中心とした検査領域 (Interrogation Window) が設定され，この検査領域ごとに相互相関の計算がおこなわれる．検査領域のサイズおよび形状は本来任意であるが，後述する FFT 相互相関法の使用を念頭におくとそのサイズは $2^N \times 2^N$[pixel] と設定するのが計算上高都合である．特別な状況を除けば 16×16，32×32，64×64，128×128[pixel] といった矩形の検査領域が設定されるのが一般的である．同様に検査領域の位置も本来自由に設定可能であるが，相互相関 PIV では簡単のため格子上に配置されることが多い．検査領域同士の間隔はさまざま考えられるが，あまり近づけすぎると領域内画素値のほとんどを共有することになり計算の意味がない．隣りあう検査領域がその領域の半分ずつを共有し合うケースが比較的よく用いられるが，このような状況は 50%オーバーラップと呼ばれる．これらの検査領域サイズおよび検査領域の配置方法によって，PIV 画像内に検出されるベクトルの数が変わる．通常検査領域サイズが大きく設置されるほどベクトル解像度は下がり，小さく設定されるほど高解像度になる．しかし小さな検査領域でな領域内部の粒子密度が低下してしまい，画像間で正しい対応づけが失敗してしまう可能性がある．このとき検出される大きな誤差をもつベクトルは誤ベクトル (Spurous vector) と呼ばれ，この誤ベクトルを除去することに焦点を当てた研究も存在する [46]．このように，検査領域のサイズと，ベクトル解像度および誤ベクトルの発生頻度は一般的に負の相関がある．

PIV における相互相関の計算方法には直接相互相関法 (Direct cross-correlation PIV，直接法)

と FFT 相互相関法 (FFT-based cross-correlation PIV，FFT 法) の二種類が存在する．どちらも相互相関の計算手法であることに変わりはないが，計算精度や速度の点でこれらの手法は性質が異なる．以下では各手法について述べる．

2.1.1　直接相互相関法

直接相互相関法は，相互相関の定義式に基づいてその計算を直接的におこなう手法である．このとき計算される式はいくつか考えられるが，局所的な輝度値の変化や画像間の明るさの差にロバストな相互相関係数 (Cross-correlaton coefficient) は広く用いられる式であり，本研究でもこの式を用いる．式 (2.1) に相互相関係数の式を示す．

$$R_{fg}(\Delta X, \Delta Y) = \frac{\sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} \{f(X_i, Y_j) - f_m\}\{g(X_i + \Delta X, Y_j + \Delta Y) - g_m\}}{\sqrt{\sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} \{f(X_i, Y_j) - f_m\}^2 \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} \{g(X_i + \Delta X, Y_j + \Delta Y) - g_m\}^2}} \tag{2.1}$$

式 (2.1) について，$f(X, Y)$，$g(X, Y)$ は第一および第二画像中の位置 (X,Y) における画素値，N_{IW} は検査領域の一辺の長さ，f_m および g_m は各画像における検査領域内の画素値平均である．

各検査領域について相互相関のピーク検出するには，複数の相互相関を計算する必要がある．直接相互相関法において，検査領域あたりに計算する相互相関の数は任意であり，これは検査領域と同じ中心をもつ探査領域 (Search region) のサイズをユーザーが指定することで決まる．たとえば探査領域を 20×20[pixel] と指定した場合，計算される相互相関の数はひとつの検査領域あたり $20 \times 20 = 400$ となる．第二画像中に探査領域内の各点を中心とする候補の領域が設定され，第一画像の検査領域との相互相関が計算される．候補領域のサイズは検査領域と等しく設定される．各 PIV 処理システムが解析可能なベクトルサイズの幅を評価する指標として，PIV で計測した最大，最小のベクトルの長さの差で定義されるダイナミックレンジ (Dynamic range)[3] が広く用いられるが，この探査領域を設定したケースではダイナミックレンジが $10 \times \sqrt{2} = 14.1$ より高くなることはあり得ない．探査領域サイズを大きく設定するほどにそのダイナミックレンジは向上する可能性が高いが，一方で計算をおこなう相互相関の数が増し，処理時間は伸びるという特徴がある．これは後述する FFT 相互相関法に対する直接法のひとつの特徴である．

2.1.2 FFT 相互相関法

FFT 相互相関法は直接法と同様，検査領域で定義される局所的輝度値パターンの類似度を相互相関で評価する手法であるが，相互相関の計算に高速フーリエ変換 (FFT) を用いている点で異なっている．いま，原関数 $f(x)$，$g(x)$ のフーリエ変換を $\mathcal{F}[f(x)]$，$\mathcal{F}[g(x)]$ と表すと，相互相関関数 $C_{fg}(\Delta x)$ は，クロススペクトル $\mathcal{F}^*[f(x)]\mathcal{F}[g(x)]$ の逆フーリエ変換によって求めることができる．これは Wiener-Khintchine の定理によって保証される．FFT 相互相関法で用いられる FFT は二次元 FFT であるが，基本的には全く同様の考え方で相互相関を求めることができる．

FFT 相互相関法では，検査領域を第一，第二画像の同位置に設定し，それらの領域内で相互相関が求められる．一見直接相互相関法と違いがないようにも思えるが，直接法における探査領域およびそれに伴う候補領域の概念は FFT 法には存在しないという点が大きく異なる．FFT 法で一検査領域あたりに求められる相互相関の数は，検査領域の一辺を N_{IW}[pixel] とすると N_{IW}^2 要素であり，すなわち検査領域サイズのみに依存する．したがって，FFT 法がもつダイナミックレンジの限界も設定された検査領域に依存し，アルゴリズムを変更しない限りユーザーがその数を指定することはできない．

直接法同様，FFT 相互相関法でも相互相関の具体的な評価式はいくつか選択肢が考えられるが，本研究では式 (2.2) に示す規格化なし相互相関関数を用いる．FFT 法では第二画像中でも検査領域が移動することはないため，式 (2.1) にあるような規格化は実質精度に影響を与えない．平均値の減算をおこなわないことで画像間の明るさの変化に対するロバスト性は低下することが予想されるが，これも PIV 画像の質が高い状況ではその影響は小さい．本研究では FFT 相互相関法の高速に計算が可能であるという特徴に着目している．領域内平均画素値を考慮した相互相関の使用は選択肢として考えられるものの計算量が増えるため，本研究ではその検討はおこなっていない．

$$R_{fg}(\Delta X, \Delta Y) = \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} f(X_i, Y_j) g(X_i + \Delta X, Y_j + \Delta Y) \tag{2.2}$$

2.1.3 サブピクセル解析

直接法・FFT 法どちらの手法を用いるにせよ，現在の PIV が標本化されたデジタル画像を用いた手法である以上，相互相関の評価による輝度値パターンの追跡のみではピクセル単位の移動量しか得ることはできない．PIV におけるサブピクセル解析 ([34]，[54]) は，ピークおよびそ

の周辺の相互相関の値を利用することで，もう一桁小さなオーダーの変位まで取得する方法である．[34]・[54] には，ガウス分布または二次曲線を仮定したフィッテングによりその極大を求める方法と，それらの重心を計算する方法が取り上げられているが，本研究では，その中で最も一般的に用いられるガウス分布を仮定したサブピクセル移動量の計算方法を使用する．このときサブピクセル移動量 P_{sub} は，式 (2.3) で計算できる．

$$P_{sub} = i - \frac{1}{2}\frac{\ln C_{i+1} - \ln C_{i-1}}{\ln C_{i+1} - 2\ln C_i + \ln C_{i-1}} \tag{2.3}$$

式 (2.3) について，C_i はピーク位置における相互相関の値であり，サブピクセルはピークを中心とする隣接 3 点 (C_{i-1}，C_i，C_{i+1}) の相関値を用いて計算される．本研究で扱う PIV は二次元平面を対象としたものであり，したがって式 (2.3) は X 方向および Y 方向についてそれぞれ独立に計算される．

2.2　誤ベクトル除去

相互相関による評価で誤ったピーク検出がおこなわれることに起因する誤ベクトル (Spurious vector) の発生は，PIV の精度や可視化結果の質を低下させる大きな要因となる．誤ベクトル除去の方法としては大きく 2 つのアプローチが提案されており，ひとつは Westerweel が提案する統計的後処理を用いた誤ベクトル除去 [46]，もうひとつは Hart らが提案する相互相関の質自体を改善することによる誤ベクトル除去 [13] がある．

統計的後処理により誤ベクトルを除去する方法は，導出された全ベクトルのうち誤ベクトルが正解ベクトルに比べ十分少ないという仮定のもとおこなわれる．この手法は比較的容易に実装することができ，その効果も比較的高いが，相互相関法によるピーク検出が全体として失敗してしまっている場合この方法では効果は期待できない．その点相互相関自体の質自体の改善をおこなう手法は本質的であり，適用範囲も広い．Hart は具体的には，隣合う検査領域に属する相互相関同士の乗算によって，相互相関の質を高める方法を提案している．しかしこの手法では，計算量が大きくなってしまうといったデメリットも存在する．これらの手法間の優劣はつけがたく，手法の選択は対象画像ごとに議論されるべきである．

本研究は PIV の処理速度について議論したものであり，その観点からは Hart らの方法は相互相関法の実装によってある程度結果が推測できる．そこで本研究では，統計的手法による誤ベクトル除去処理に絞って，高速化を検討する．

2.3 再帰的手法による高解像度化とベクトル補間

2.1 冒頭で述べたように，検査領域サイズと PIV 画像中のベクトル解像度および誤ベクトル発生頻度には負の相関がある．大きな検査領域を設定すれば誤ベクトルの発生は抑えられるがベクトル空間解像度は下がり，小さな検査領域を設定すれば解像度は向上するものの誤ベクトルの発生頻度は増す．これらのトレードオフを解決する手段として，再帰的相互相関法 (Recursive cross-correlation method) がある．この手法では，まず比較的大きな検査領域で PIV をおこない，各点における変位を大まかに絞る．そしてその点を中心に今度は小さな検査領域を設定し，PIV を再度実行することで，誤ベクトルの発生を抑えつつベクトルの解像度を上げる．この際，再帰の各ステップごとに導出されるベクトルの数は異なるため，ある段における PIV の結果は，次の段のベクトル解像度と一致するよう補間されて使用される．補間の手法は Uncer ら [43] が示すように様々なものがあるが，PIV における再帰のベクトル補間の手法としてはその中からバイリニア法が使用されることが多い．

2.4 並列化の方針

以上で述べてきた処理要素を組み合わせることで，現在広く用いられている PIV 画像処理を構築することができる．たとえば単純な相互相関は 2.1.1〜2.1.3 で述べた処理要素から構築することが可能であり，これに 2.2 で述べた誤ベクトル除去処理を付加することもできる．ベクトル補間をおこない検査領域サイズを小さくしながら PIV 処理を繰り返すことで，再帰的 PIV の実装も可能となる．

したがって，本章で示した相互相関 PIV を構成する要素ごとに GPU 並列計算への適用方法を開発すれば，それらを組み合わせることで GPU 上で動作する高速 PIV 処理システムが開発できるということになる．次章では，本章で示した各処理要素について GPU 並列計算への適用方法を検討評価し，それらを組み合わせることで PIV システムを構築していく．

第3章　シングルGPUを用いた相互相関PIVの高速化

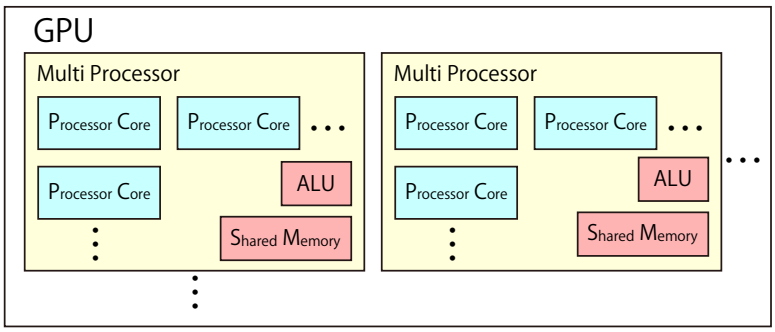
本章ではGPU並列計算により相互相関PIVを高速化する手法について検討および評価をおこなう．前章で，現在用いられる高性能なPIV処理アルゴリズムの多くがいくつかの要素の組み合わせで構成されていることを示した．本章ではまずGPUのプログラミングモデルについて概説し，このモデルに基づいて各要素ごとにGPUへの実装方法を検討する．最後に，その結果開発した相互相関PIVについて主に処理速度の観点から評価をおこなう．

3.1　GPUのプログラミングモデル

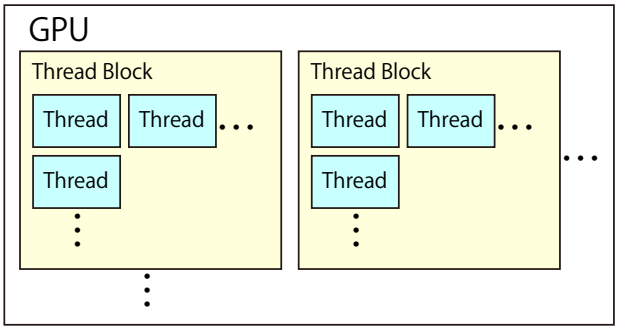
1.1でも述べたように，GPUはその内部に多数のプロセッサコアを有するが，ストリームプロセッシング可能な開発環境を用いることで個々のコアを意識することなくGPU並列計算の開発が可能である．fig. 3.1にハードウェアとしてのGPUのアーキテクチャとGPUのプログラミングモデルを簡単に示す．GPU内のプロセッサコアは複数コアで1つのマルチプロセッサを形成しており，その中で1つのALUを共有する (fig. 3.1 (a))．マルチプロセッサ内では複数コアによる協調処理が可能であり，コア間のデータ転送はGPUのチップ外メモリやshared memoryと呼ばれるオンチップメモリを介しておこなわれる．このマルチプロセッサが複数集まることで1つのGPUを形成しており，したがってGPUの構造は一種の階層構造をなしているといえる．一方GPUのプログラミングモデルでは，開発者は処理単位であるスレッドおよび複数スレッドにより構成されるスレッドブロックの処理を記述することになる．スレッドはプロセッサコアに，スレッドブロックはマルチプロセッサに対応しており，アーキテクチャと同様の階層構造を形成している．各スレッドの処理をまとめたスレッドブロック全体の処理は一連の処理 (カーネルと呼ばれる) としてGPUで実行される．カーネルの処理を記述すれば，それ以降のスケジューリングはコンパイラにより自動で最適化される．

したがってGPU並列計算の適用方法を検討することは，処理単位であるスレッドの処理・ス

レッドブロックの処理・GPU の処理単位であるカーネルの処理のデザインをもっともパフォーマンスが高くなるかたちで最適化することに等しい．以降では，2 章で示した相互相関 PIV を構成する各処理要素について GPU 並列計算の適用方法を検討する．すなわち，各要素を実行する際のスレッド・スレッドブロック・カーネルの各処理について考えていく．



(a) GPU Architecture



(b) GPU Programming Model

Fig 3.1: Schematic diagram of hardware architecture and programming model of GPU: In GPU architecture, GPU consists of a number of multi processors and each multi processor consists of a number of processor cores. In each multi processor, processor cores can work together transferring data with each other using off-chip memory or on-chip shared memory. In GPU programming model, only processing of threads and thread block is designed by developers and the following schedulings are optimized by the compiler.

3.2 相互相関 PIV を構成する各要素の並列化

3.2.1 画像データの格納

PIV 画像処理は，粒子画像データをもとにベクトル場の導出をおこなう方法である．GPU 並列計算で PIV 処理をおこなうためには，画像データを GPU のチップ外メモリに格納する必要があるが，格納できるメモリはグローバルメモリやテクスチャメモリなど複数候補が考えられ，どのメモリを利用するかが問題となる．

本研究ではキャッシュサイズが最大のメモリを利用するという基準を設けて，画像データの格納先を決定している．相互相関の計算において画素データは複数回参照される可能性が高く，キャッシュサイズが大きいほどメモリアクセスが効率化されると考えられる．たとえば NVIDIA 社製の GPU である Geforce GTX285 (240 Cores) のアーキテクチャでは，グローバルメモリにはキャッシュが利かないもののテクスチャメモリではスレッドブロックあたり 6KB〜8KB のキャッシュを利用することができる．そのためこの GPU を用いる際はテクスチャメモリに画像データを格納する．一方で同様に NVIDIA 社製の Geforce GTX480(480 Cores) のアーキテクチャではグローバルメモリにキャッシュが利き，そのサイズは最大で 48KB とテクスチャメモリのキャッシュサイズよりも大きい．そのため GTX480 を用いる際にはグローバルメモリに画像データを格納する．

3.2.2 直接相互相関法

相互相関法では画像中に複数の検査領域が設定され，各検査領域ごとにベクトルが導出される．誤ベクトル除去処理などをおこなう前の段階では各ベクトルは独立に計算されるため，各検査領域における相互相関の計算は手法によらず並列に実行することが可能である．

直接相互相関法における相互相関は，第一画像中の検査領域と第二画像中の候補領域とから式 (2.1) に従って直接的に計算される．このとき各候補領域の相互相関計算もまた独立であり，並列に実行することが可能である．したがって直接法で計算される全ての相互相関は独立であり，並列に計算が可能であるということになる．これは FFT 法と比較したときの直接法の大きな特徴のひとつである．

本研究では，GPU のプログラミングモデルで直接相互相関法を並列計算する 2 通りの方法を検討する．ひとつは GPU の最小処理単位であるスレッドに相互相関の計算を割りあてる並列化

手法であり，もうひとつは1スレッドブロックに1つの相互相関の計算を割り当てる手法である．GPUがもつ階層構造のどの階層で相互相関単位の計算をおこなうかがこの両手法の最大の違いである．

1スレッドに1相互相関の計算を割り当てた並列化は，直接法における全て相互相関が計算上独立であるという特徴を活用した方法だといえる．全ての相互相関は独立であるため，1つの相互相関に対応した各スレッドもまた独立に実行可能であり，各スレッド間のデータ通信も発生しない．したがってスレッドの同期も考慮する必要がなく，待機状態が発生するスレッドも存在しない．しかしこの並列化手法では各スレッドが相互相関計算のすべてを担うため，各スレッドがfor文などの条件分岐命令を処理することになり，これが計算速度を下げる要因となる．条件分岐命令の総数は検査領域が大きくなるにつれて増加するため，大きな検査領域ほどこの影響が強くなると考えられる．

それに対し1スレッドブロックに1相互相関の計算を割り当てた並列化手法では，複数スレッドの協調動作により1つの相互相関の計算をおこなう．この際発生するデータ転送には，複数スレッドが高速にアクセス可能なマルチプロセッサ内メモリを活用する．これにより基本的には条件分岐処理をおこなうことなく相互相関の計算を実行することができる．本研究では[58]で示されている方法を参考に，複数スレッドを用いて1つの相互相関の計算をおこなう．その方法をfig. 3.2に示す．各相互相関の計算は領域内データの総和計算から計算される．そこでスレッド間でデータ交換が可能であることを活かし，木構造的にその和を求める．まず図中のステップ1で，各スレッドがデータ配列中の異なる2要素にアクセスし，その和を求め異なる要素内に格納する．ステップ2では，配列のうちステップ1で値が格納された部分に対し前と同様な処理をおこなう．このとき，実際の総和計算には前ステップの半数のスレッドが参加しており，残りのスレッドは待ち状態となる．このステップを繰り返すことで，最終的には配列内の1要素に総和計算の結果が格納されることになる．この方法は配列要素数が2の階乗であることを前提としている．一般的なデータサイズに対応させるためには，実際のデータサイズよりも大きくかつ2の階乗で表せる最小の要素配列を確保し，余分な配列にはゼロを格納するといった方法(ゼロパディング)をとる．この並列化手法では条件分岐命令を消去することが可能であるものの，複数スレッドが同期しながら処理をおこなうため待機状態となるスレッドの発生を避けることができない．すなわち1相互相関あたりの計算量が小さくなるにつれて待機スレッドの割合は相対的に増すため，検査領域が小さく設定されるほどに待機スレッドの影響は大きくなると考えられる．

本研究では，1スレッドに1相互相関を割り当てた手法と1スレッドブロックに1相互相関を

割り当てた手法をそれぞれ実装し，その処理速度を比較した．その結果を fig. 3.3 に示す．fig. 3.3 は NVIDIA 社製 GPU の Geforce GTX 480(480 Cores) を用いて得られたものであり，その横軸は検査領域の一辺の長さ [pixel] を表し，縦軸は処理時間を表す．本研究での実装では，いずれの検査領域サイズにおいても 1 スレッドに 1 相互相関の計算を割り当てた並列化手法の方が処理時間は短く優位であることが fig. 3.3 からわかる．1 スレッドブロックに 1 相互相関を割り当てた手法は検査領域サイズが大きくなるにつれて処理速度が向上するものの，最大の検査領域サイズでもその処理速度は他方の約 1/4 でしかなかった．この差の原因としてはキャッシュの効果が考えられる．1 スレッドに 1 相互相関を対応させた手法では領域内画素データを共有する相互相関が同一スレッドブロック内に多く存在するため，スレッドブロックで 1 つの相互相関を計算する手法に比べメモリアクセスが効率化され，結果このような差が生じたと考えられる．また 1 スレッドブロックに 1 相互相関を対応させた手法では検査領域サイズでその処理時間が大きく変化している一方で，1 スレッドに 1 相互相関を割り当てた手法ではその処理時間は検査領域サイズに影響をほとんど受けていないこともわかる．

fig. 3.3 で示した結果はあくまで本研究の計算環境での結果であるが，手法間の処理速度差は大きく，1 スレッドに 1 相互相関の計算を割り当てた手法が他の計算環境でも高いパフォーマンスを示す可能性は高いと考えられる．以降で述べる PIV の処理速度では，1 スレッドに 1 相互相関を割り当てた並列化手法により高速化された結果を GPU 並列計算による直接法の処理速度として採用する．

3.2.3 FFT 相互相関法

3.2.1 で述べたように，画像中の各検査領域ごとに相互相関の計算処理は独立であり容易に並列化が可能である．ここでは各検査領域での相互相関計算をいかに並列化するかについて中心に議論をおこなう．

FFT 相互相関法による相互相関の計算は以下の手順で実行される．

1. 第一画像中の各検査領域について 2 次元 FFT
2. 第二画像中の各検査領域について 2 次元 FFT
3. 2 次元 FFT 結果の要素積
4. 要素積の 2 次元逆 FFT

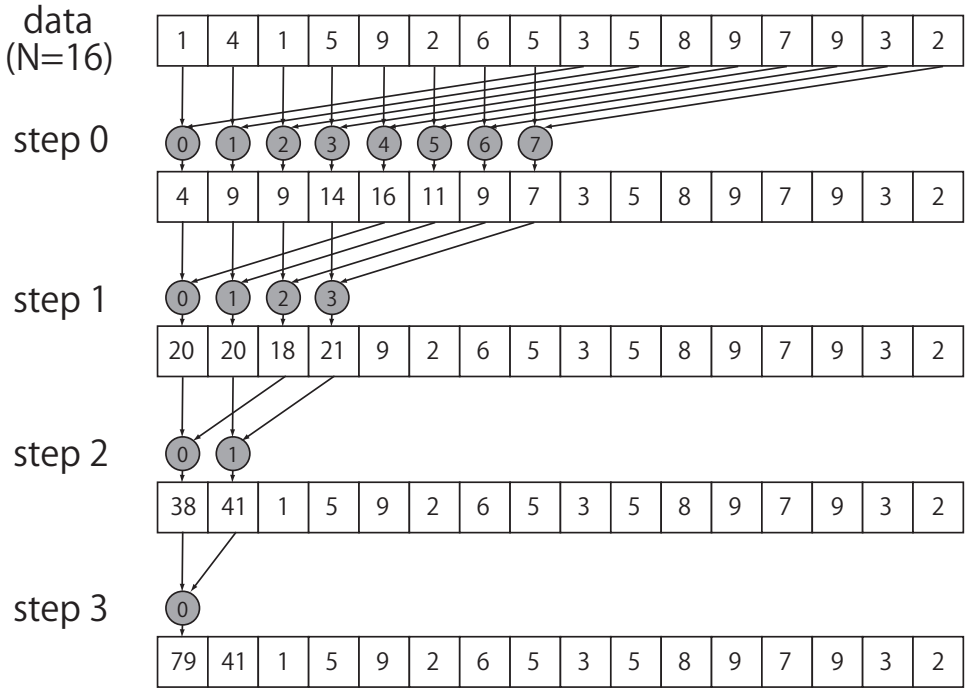


Fig 3.2: Schematic diagram of reduction operation by multiple threads: In step 0, $N/2$ threads are used to compute summation of each two elements. In the following steps, the same operations are continued decreasing the number of threads joined in the operation. Total summation can be obtained in one position on the memory after $\log N$ steps.

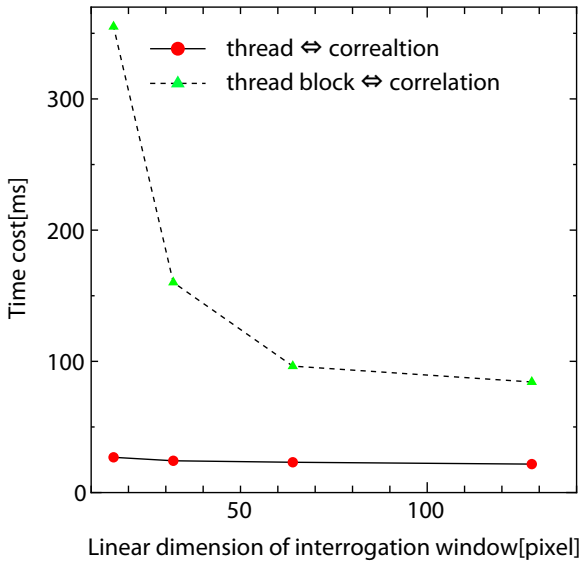


Fig 3.3: Speed comparison between method a) and b) as a fuction of interrogation window: Vertical axis shows time cost per PIV(1024×1024[pixel] image pair，50% overlap) and horizontal axis shows linear dimension of interrogation window[pixel]. In each interrogation window size, search region size is set to 20^2[pixel].

FFT 法で計算される FFT は 2 次元 FFT である．2 次元 FFT を計算する手法はいくつか提案されているが，1 次元 FFT の直積として多次元 FFT を計算する Row-Column 法 [**?**, loan1992cff] もっとも広く用いられる手法であり，本研究でもこの方法を用いる．Row-column 法による 2 次元 FFT および逆 FFT の計算は以下の手順で実行される．

1. 各行について 1 次元 FFT
2. 転置
3. 各列について 1 次元 FFT
4. 転置

Row-column 法において転置をせず 1 次元 FFT 処理をおこなうことも可能であるが，転置をすることで FFT 時のメモリアクセスを効率化できる．FFT 処理結果をソートされた状態で得るためには 2 回の転置が必要になるが，相互相関の計算結果を正しく得るだけならば Row-column 法の最後の転置をおこなう必要はない．したがって Row-column 法を用いた FFT 相互相関法の処理は 1 次元 FFT・転置・要素積を最小処理単位とする計 10 の処理要素から構成される．その処理の流れを以下に示す．

1. 第一画像中の各検査領域について2次元FFT
   (1) 各行について1次元FFT
   (2) 転置
   (3) 各列について1次元FFT
2. 第二画像中の各検査領域について2次元FFT
   (4) 各行について1次元FFT
   (5) 転置
   (6) 各列について1次元FFT
3. (7)2次元FFT結果の要素積
4. 第一画像中の各検査領域について2次元FFT
   (8) 各行について1次元FFT
   (9) 転置
   (10) 各列について1次元FFT

本研究では，上に示す処理単位から構成されるFFT法の並列化手法を2通り検討する．ひとつは上に示した各処理単位ごとにGPUの処理単位であるカーネルを構成する方法，もうひとつは2次元FFTの処理をひとつのカーネルにまとめる方法である．手法の違いはFFT法を構成するカーネルの数であり，処理要素ごとにカーネルを構成する方法では計10カーネル，2次元FFTをカーネルにまとめる方法では計4カーネルからFFT法が構成される．

処理要素ごとにカーネルを構成する方法のメリットは，個々の処理について最適な並列化を検討できる点にある．上に示した手順からFFT法の大部分は1次元FFTと転置から構成されていることがわかるが，これらの処理が全く同一の並列化で最適なパフォーマンスが得られる可能性は低い．処理要素ごとにカーネルを構成する手法は，処理ごとに使用するスレッドの数などを指定できるという点で複数処理をまとめる手法よりも優位である．しかし処理要素ごとにカーネルを構成した場合，各カーネルの処理結果はチップ外のメモリを介してロード/ストアがおこなわれるため，これがFFT法全体のパフォーマンスを下げる可能性がある．一方で複数処理要素をひとつのカーネルにまとめる手法ではデータ転送にチップ内の高速なメモリを使用す

ることが可能であるため，データ転送に起因する処理全体のパフォーマンス低下の影響が小さいと考えられる．また，GPU 処理の呼び出しにはオーバーヘッドが伴うが，複数処理をカーネルにまとめた方法はオーバーヘッドの影響も受けにくいことが予想される．

これら 2 手法を GPU に実装した結果を fig. 3.4，fig. 3.5 および fig. 3.6 に示す．fig. 3.4，3.5，3.6 はそれぞれ 1024×1024[pixel]，512×512[pixel]，256×256[pixel] の粒子画像ペアを解析したときの処理時間を示す．検査領域はいずれのサイズでも 50%オーバーラップで設定されている．いずれの画像サイズでも大きな検査領域サイズでも処理要素ごとのカーネルを構成した方法 (10 カーネル) の処理速度が他方を上回っているが，サイズが小さくなるごとにその差は縮まり，あるサイズで複数処理をひとつのカーネルにまとめた方法 (4 カーネル) の処理速度が他方を上回っていることがわかる．これは検査領域サイズが小さくなるにつれデータ転送コストの割合が増し，データ転送の点で相対的に有利な処理をまとめた手法の優位性が高くなっていることが原因だと考えられる．また GPU 処理のオーバーヘッドについて，画像サイズが大きいケースでは手法によらず無視できるほど小さいコストであるが，画像サイズが小さくなるごとにオーバーヘッドの占める割合は増加していることがわかる．要素ごとにカーネルを構成した手法の方が全体に占めるオーバーヘッドの割合は高く，256×256[pixel] の画像サイズではオーバーヘッドは処理全体の約 2 割を占めていることがわかる．現行の GPU の仕様ではこのオーバーヘッドは避けることができないため，小さな画像サイズでは GPU を用いた処理高速化の限界が近づきつつあるということもできる．

これらの結果から，検査領域サイズによらず FFT 法で高いパフォーマンスを得るためには，検査領域サイズによって 1 つのカーネルが担う処理を調整する必要があるということがわかる．大きな検査領域サイズでは処理要素ごとにカーネルを構築し，小さな検査領域サイズでは複数処理要素をまとめてカーネルを構築することで高い処理速度が得られると考えられる．また画像サイズが大きい場合は GPU 呼び出しに伴うオーバーヘッドは無視できるほど小さいが，小さな画像サイズではその影響も考慮にいれる必要があることもわかる．今後 GPU の性能が向上し，処理速度が更に向上した際には，大きな画像サイズでの処理でもオーバーヘッドの影響が無視できなくなってくると予想できる．

3.2.4 ピーク検出とサブピクセル解析

いずれの手法で導出された相互相関も，検査領域ごとにその中でピークが検出され，ピーク近傍の相関値と合わせてサブピクセルの計算がおこなわれる．ピーク検出において同一の検査

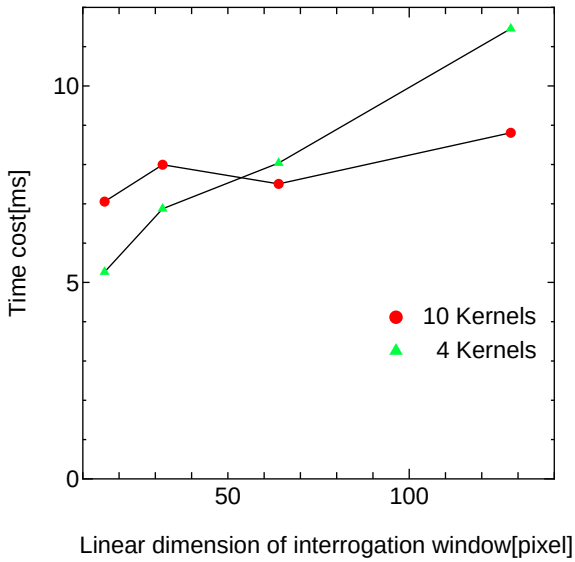


Fig 3.4: Time cost of 10-kernels method and 4-kernel method as a fuction of interrogation window size in case of 1024×1024 size image: In in larger interrogation window size, 10-Kernel method is faster than 4-kernel method. However as the interrogation window size becomes smaller, the gap also becomes smaller and in 16×16 interrogation widnow, time cost of 4-kernel method is reversed. Time cost of overhead following GPU processing is so small that it can be neglected.

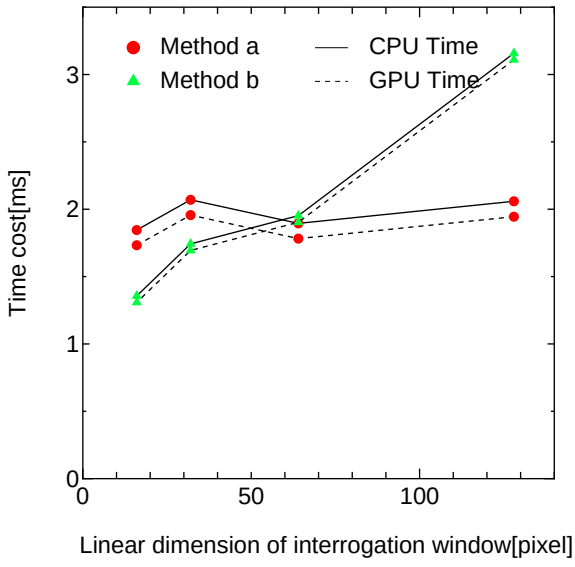


Fig 3.5: Time cost of 10-kernels method and 4-kernel method as a fuction of interrogation window size in case of 512×512 size image: The tendency of processing speed is the same with the case of 1024×1024[pixel] image size. In this case, the overhead of GPU processing is relatively larger than the case than 1024×1024[pixel] image size.

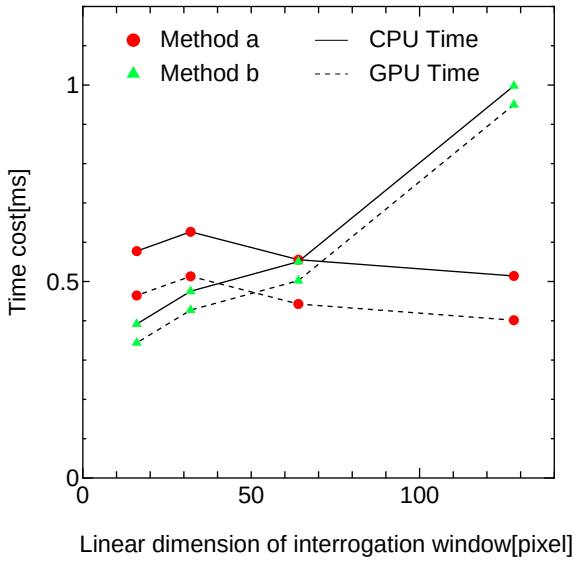


Fig 3.6: Time cost of 10-kernels method and 4-kernel method as a fuction of interrogation window size in case of 256×256 size image: The tendency of processing speed is the same with the case of 1024×1024[pixel] image size. The overhead of GPU processing is relatively larger than the case than 512×512[pixel] image size and in the case of 10-kernel method, about 20% of total processing time is consumed by the overhead.

領域に属する複数の相互相関値はその大きさを比較する必要がある．本研究ではひとつのスレッドブロックにひとつの検査領域のピーク検出を割り当て，複数スレッドの協調動作によりピーク検出をおこなう方法で並列化をおこなう．その処理の概要を fig. 3.7 に示す．

fig. 3.7 は 1 次元データ配列 (データサイズ N) のピーク位置を検出するアルゴリズムを示している．配列中の 2 要素のに対する処理を木構造的に繰り返す手法であり，これはスレッドブロック内の複数スレッドを用いた総和計算の手法とよく似ている．異なる点は，ピーク検出処理ではピークの値および場所を検出するためにその位置を要素にもつ配列 (インデクス配列と呼ぶ) を用意している点である．値の比較は当然相関の値を参照しておこなわれるが，その際のデータの交換は fig. 3.7 が示すようにインデクス配列においておこなわれる．$\log N$ ステップの処理後に，配列内のある要素に相互相関ピークの位置が格納され，その位置を参照することでピークの値を検出することができる．

実際にピーク検出がおこなわれるのは 2 次元配列上のデータである．本研究では，まず 2 次元配列の各行についてピーク検出をおこない，各行のピークおよび位置から構成されるデータ配列に更に 1 次元データのピーク検出処理をおこなっている．

また上で述べた手法はデータサイズが 2 の階乗で表せるものにしか対応しておらず，一般的な

Table 3.1: Time cost of peak detection and sub-pixel: These results are obtained from 1024×1024[pixel] size image and 50 % overlap of interrogation window.

interrogation window size[pixel]	128×128	64×64	32×32	16×16
Peak detection and Sub-pixel[ms]	0.78	0.89	0.83	0.92

データサイズの配列に対するピーク検出をおこなうためには総和計算のときと同様ゼロパディングをおこなう必要がある．本研究で検討する検査領域は一辺が 2 の階乗で表せる矩形領域であるため，相互相関の数が検査領域サイズのみに依存する FFT 相互相関法ではゼロパディングをおこなう必要は生じないが，相互相関の数が探査領域サイズに依存する直接相互相関法ではゼロパディングが必要な可能性が高い．本研究では，直接法と FFT 法とでゼロパディング処理の有無のみが異なるピーク検出処理を実装している．

なお本研究の実装では，ピーク検出処理とサブピクセルとをひとつの GPU 処理にまとめたが，これはサブピクセル計算の負荷とデータ転送とのトレードオフを検討したうえでの判断である．この実装では 1 つのスレッドブロック内で一度しかサブピクセル計算が実行されず，サブピクセル計算中は多くのスレッドがアイドリング状態となっている．したがって本実装はサブピクセル計算として最適ではない．しかしサブピクセル計算の負荷は，その処理を独立した GPU 処理とすることにより発生する低速メモリとのデータ通信負荷に比べ小さくまた処理の記述も簡単であるため，ここでは処理をまとめている．

各検査領域サイズにおけるピーク検出およびサブピクセル計算の処理時間を Table 3.1 に示す．画像サイズは 1024×1024[pixel] であり，検査領域は 50%オーバーラップで設置している．いずれの検査領域サイズでも，1[ms] 弱で処理できていることがわかる．

3.2.5　誤ベクトル除去

2.2 で，誤ベクトル除去の手法は大きく 2 通りの手法があることを述べた．このうち Hart[13] が提案する隣接検査領域間の相互相関の積を考える手法の並列化は比較的容易であると考えられる．各検査領域での相互相関が得られている状況では各相関同士の積は独立であるため，各スレッドに 1 乗算を割り当てることで並列化できる．

Hart の方法に比べると，Westerweel[48] が提案する統計的処理による誤ベクトル除去を並列計算に適用することは難しい．統計処理による誤ベクトル除去は，対象の近傍ベクトルのデー

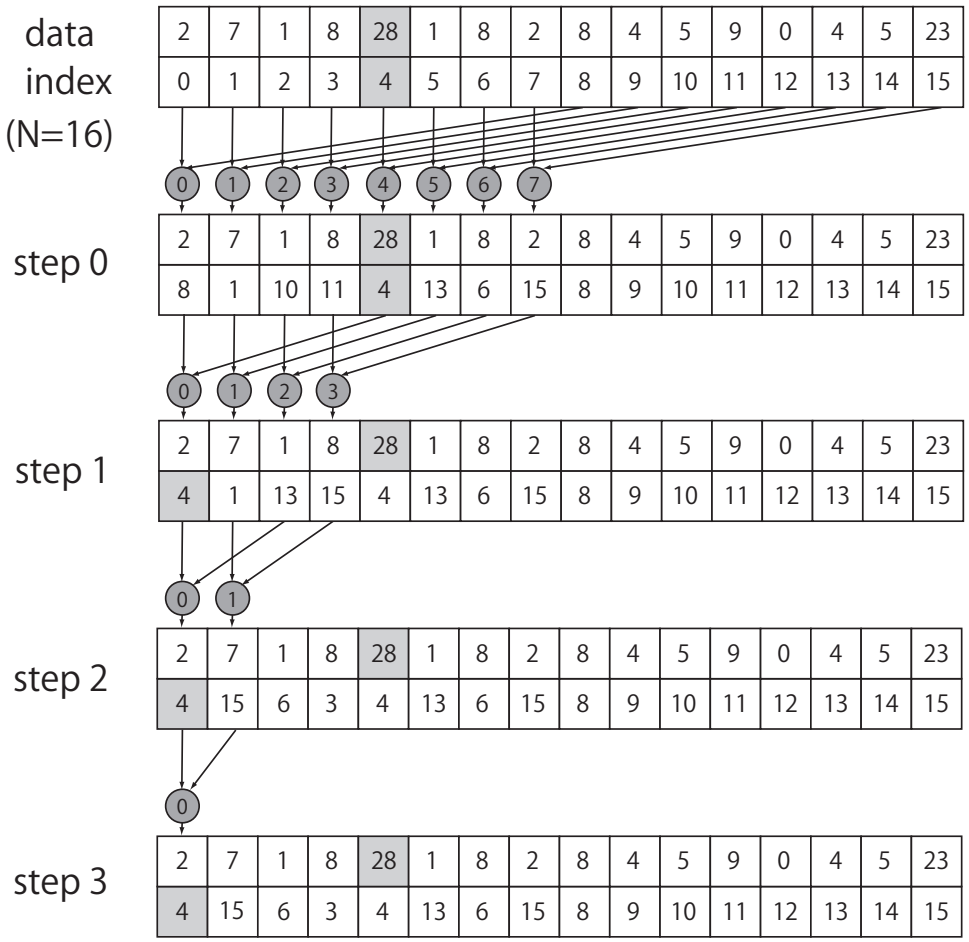


Fig 3.7: Schematic diagram of peak detection by muitiple threads: In step 0, $N/2$ threads are used to compare correlation values. if the correlation at the position of the thread number is smaller than the correlation compared at the time, index value at the position of thre thread number (not correlation value) is replaced by the index at the postion of larger correlation. In the following steps, the same operation is executed halving the number of threads used by the operation. The index of the peak is obtained in one position of the index memory after $\log N$ steps.

タを参照することでそのベクトルが誤ベクトルか否かを判定する方法である．そのため隣り合うベクトル同士は互いに依存関係にあり，並列計算への適用が難しい．

本研究では 1 つのスレッドブロックにすべての誤ベクトル処理を割り当て，行ごとに誤ベクトル除去をおこなうことで統計的誤ベクトル除去処理をおこなっている．すなわち任意の行の 1 要素に 1 スレッドを割り当てて行単位で並列に誤ベクトル検出をおこない，結果を更新して次の行の処理に移る．この処理をすべての行についておこなうことで，ベクトル全体の誤ベクトル除去処理をおこなっている．

fig. 3.8 に，各検査領域サイズにおける誤ベクトル処理に要する時間を示す．この結果は 1024×1024[pixel] の画像を用いて得られたものであり，検査領域は 50%オーバーラップで設定されている．いずれの検査領域サイズでも 1[ms] 以下の処理時間ではあるが，検査領域が小さくなりベクトル解像度が増すにつれてその処理時間が大きく増加していることがわかる．

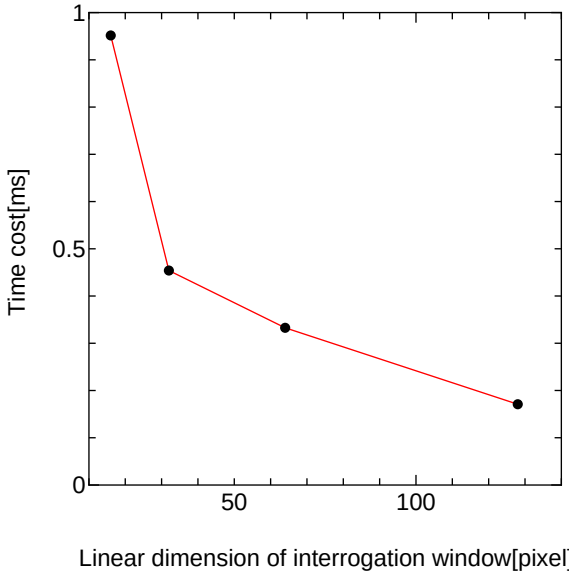


Fig 3.8: Time cost of detecting and removing spurious vectors as a function of interrogation window size: This result is obtained by processing 1024×1024 [pixel] image size. The time cost drastically increase as the interrogatin window size becomes smaller and the number of vector becomes larger because only one thread block can be used to detect spurious vector correctly.

3.2.6　再帰的手法に伴うベクトル補間

再帰的手法では，1 ステップ前の結果をふまえ小さな検査領域でより高解像度な PIV が実行される．その際に 1 ステップ前のベクトルデータは次ステップの解像度に合わせ補間する必要

があるが，この処理は全てのベクトルについて計算上独立である．本研究では，補間される 1 ベクトルに 1 スレッドを割り当てて計算をおこなっている．なお各ベクトルはバイリニアで補間される．

画像内ベクトル解像度が上がるほど補間の処理時間は増加すると考えられるが，本研究の計算環境で実装したところ 1024×1024[pixel] の画像を用いたときでも処理に要した時間は高々 0.1ms 未満であった．これは相互相関の計算に比べると無視できるほど短い時間であり，補間計算が処理速度全体に及ぼす影響は非常に小さいことがわかる．

3.2.7 CPU ⇔ GPU 間のデータ転送

GPU で解析される画像データは CPU のメモリから GPU のメモリへ転送する必要があり，GPU 並列処理の結果導出されるベクトルデータは，GPU のメモリから CPU のメモリへ転送する必要がある．しかしこの CPU ⇔ GPU 間のデータ転送は一般に非常に遅くいため，転送量および転送回数は可能な限り減らす必要がある．本研究では，今までに述べてきた並列化手法を用いた相互相関 PIV を構成する全要素を GPU で処理することで，CPU ⇔ GPU 間のデータ転送回数を最低回数の 2 回に抑える．

3.2.8 画像データ転送と PIV 画像処理の並列化

PIV 画像処理は時系列画像データに対して適用されることが一般的であり，その際 PIV 画像処理は複数回連続で実行される．このような場合には，データ転送と処理の並列化が検討できる．すなわち，fig. 3.9 に示すように依存関係のない画像データおよび PIV 処理結果の転送と PIV 処理を同時に実行することで，データ転送の時間を実質遮蔽することができる．

3.3 再帰的相互相関 PIV の処理速度評価

以上の各要素の並列化手法の検討評価をもとに，本研究では再帰的直接相互相関 PIV および再帰的 FFT 相互相関 PIV の処理システムを開発した．ここでは，それらの総合的な処理速度について評価をおこなう．

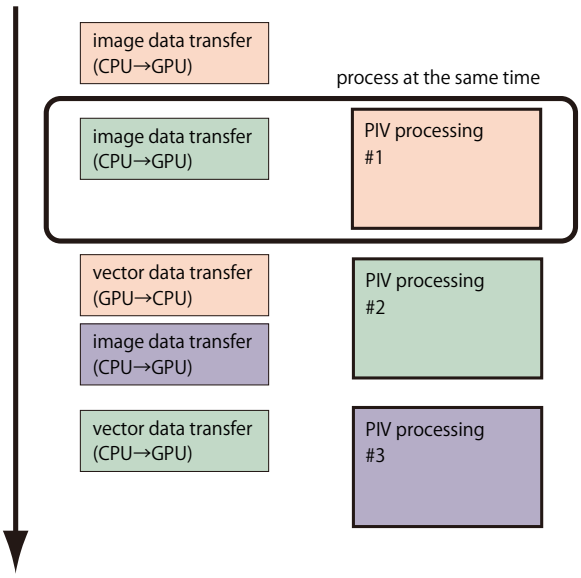


Fig 3.9: Schematic diagram of parallel execution of GPU prcessing and data transfer: In case of sequential PIV processing, time cost of image data transger from CPU to GPU and vector data transfer from GPU to CPU can be shielded by executing PIV processing and data transfer in parallel.

シングル CPU との処理速度の比較

ここでは堀田ら [59] が実験で取得した画像を用いて，直接相互相関 PIV についてシングル CPU とシングル GPU との速度比較をおこなう．用いる画像サイズは 1024×512[pixel] であり，再帰的相互相関法を用いる．検査領域は [128×128，64×64，32×32][pixel] と変化させ，探査領域は [20×20，10×10，6×6] と変化させた．このときの処理速度に関する結果を Table 3.2 に示す.

Table 3.2: Speed comparison about direct cross-corrlation PIV between CPU and GPU: This table shows about 800 times faster processing than single CPU processing is achieved by using single GPU.

	single GPU	single CPU
Time cost[ms]	19.8	15530
CPU/GPU[times]	784.3	/

Table 3.2 から明らかなように，PIV の処理は大幅に高速化されていることがわかり，その高速化は 800 倍弱にまでのぼることが確認された.

なおここで解析対象とした堀田の実験画像については 5.3.2 で改めて議論をおこなっている．実験画像や可視化結果については 5.3.2 を参照されたい.

1 ベクトルあたりの導出にかかる時間コスト

ここでは，画像中の 1 ベクトルの導出にかかる処理時間について考察をおこなう．以下の fig. 3.10 は，okamoto ら [31] が作成した標準画像と呼ばれる人工画像に対し，直接相互相関 PIV 処理をおこない，その結果を可視化し，第一画像にオーバーラップして表示したものである．この標準画像は伸長渦を表している．検査領域は 32×32[pixel]，50%オーバーラップで設定し，探査領域は 8×8[pixel] と設定した．再帰はおこなっていない．その結果，fig. 3.10 の画像は 8.13[ms] で解析され，3969 のベクトルが導出された．したがって，この画像において 1 ベクトルの導出に必要な処理時間は $8.13 \div 3969 = 2.05[\mu s]$ であるということになる．

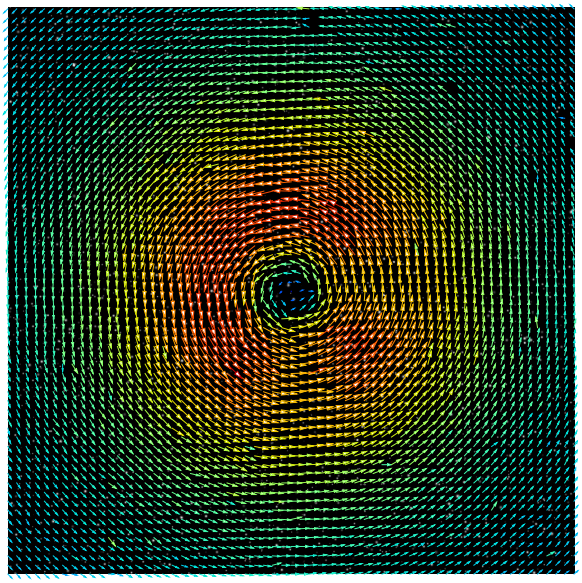


Fig 3.10: Visualization result of stretched vortex overlapped on the synthetic image:

GPU 並列計算による高速化の限界

3.2 において，連続 PIV 画像処理においては粒子画像データの転送時間を GPU 処理と並列におこない，その時間で遮蔽するよう実装をおこなったと述べた．しかし，PIV 処理が画像データ転送にかかる時間を上回るケースでは，逆に PIV 処理の時間がデータ転送時間に遮蔽されることになる．すなわち，GPU 並列計算では CPU $\Leftrightarrow$ GPU 間のデータ転送に必要な時間以上の高速化は原理的に不可能であり，CPU から GPU への画像データの転送と GPU から CPU へのベクトルデータの転送にかかる合計時間が GPU を用いた PIV 処理の限界処理速度ということに

なる．ベクトルデータのサイズは画像データにくらべれば圧倒的に小さいため，実質画像データサイズに PIV 処理は律速される．

現在，単精度扶浮動小数点型の変数で確保された 1024×512[pixel] の画像データペアの転送には約 1ms の時間がかかる．しかし先の結果に示したように，同サイズの画像の PIV 解析には数ミリ秒〜数十ミリ秒の処理時間が必要であり，GPU による PIV の限界処理速度にはまだ差がある．したがって，GPU によって PIV が高速化される余地はいまだ残っており，今後更なる高性能 GPU が発表されれば，それを用いることで PIV は更に高速化される可能性があることになる．

3.4　本章のまとめ

本章では，相互相関 PIV を GPU 並列計算に適用する手法について検討および評価をおこなった．2 章で示した相互相関 PIV を構成する各要素ごとにプログラミングモデルにもとづく並列化および高速化を検討し，特に相互相関 PIV の処理時間の多くを占める相互相関の計算部分については，直接法と FFT 法それぞれに対し考えられる複数の手法を検討し，実装をふまえた比較をおこなった．

直接相互相関法では，1 スレッドに 1 つの相互相関の計算を割り当てた手法と，1 スレッドブロックに 1 つの相互相関の計算を割り当てた手法を検討した．両手法ともメリット・デメリットをもつものの，本研究の計算環境に実装した結果，1GPU スレッドに相互相関の計算を割り当てた手法が圧倒的に高い性能を示した．その差は最小で約 4 倍，小さな検査領域では数十倍と大きく開きがある．この結果から，1 スレッドに 1 相互相関を割り当てる手法は直接法を効果的に並列化する手法であることがわかり，また他の GPU 計算環境でもその有効は高いことが推測される．

FFT 相互相関法では GPU 処理単位であるカーネルの数を変えて実装をおこない，それらの処理速度の比較をおこなった．本研究の計算環境で実装をおこなった結果，手法の有効性は検査領域サイズに依存し，大きな検査領域サイズでは FFT 法を構成する 1 次元 FFT などの要素ごとにカーネルを構成した方が効果的であり，検査領域が小さい場合には複数処理要素をひとつのカーネルにまとめて構成した方が，低速なメモリを介するデータ転送量が削減され高速であるということを確認した．また 1024×1024 [pixel] のような画像サイズでは GPU 呼び出しにともなうオーバーヘッドは無視できるほど小さいものの，画像サイズが小さくなり計算コストが低下するにつれ，オーバーヘッドの影響が無視できないほど大きくなることを確認した．これ

らの結果から，FFT 法を効果的に GPU 並列計算に実装するには，使用される検査領域サイズおよび画像サイズをふまえ並列化する手法を選択すべきである，ということが明らかになった．

その他の要素についても GPU 並列計算への実装を検討したうえで，GPU による再帰的直接相互相関 PIV と，再帰的 FFT 相互相関 PIV を開発し，その評価をおこなった．シングル CPU による処理と比較した結果，直接法においてその処理は約 800 倍高速化されていることが確認された．また一般的に用いられる 1024×1024[pixel] 程度の画像サイズならば，数〜数十ミリ秒で PIV 処理が実行可能であり，1 ベクトルあたりの処理速度は最高で約 2 マイクロセカンドに達していることを確認した．

現行の GPU で PIV 処理をおこなう場合 CPU ⇔ GPU 間のデータ転送は避けられず，よって GPU による高速処理にはある一定の限界が存在するが，本研究で開発した PIV はその限界には達していない．したがって，GPU による処理は更なる高速化の余地があると考えられる．本章は，現行の高性能 GPU によって少なくともミリセカンドオーダーでの実用的 PIV 処理が可能であること，および GPU を用いることで更なる高速化実現可能であること明らかにしたといえる．

第4章　マルチGPUを用いた相互相関PIVの高速化

3章では，シングルGPUによる並列計算で数十〜数百HzのPIV処理が可能なことを示した．しかしGPU並列計算が本質的にもつ限界は3.3に示されたとおりであり，3章の結果はこの処理速度を獲得するまでに至っていない．したがって，GPUの性能が向上することに伴うPIV処理速度向上の余地はいまだ残されているといえる．

GPUの性能自体を向上させることは本研究の範囲を逸脱してしまうが，疑似的に更なる高性能GPUを使用できる手段として，マルチGPUを用いた並列計算が考えられる．GPUを複数個使用することによってシステムの並列性は更に向上し，処理速度の更なる向上が期待できるが，一方でマルチGPUシステムではGPU間のデータ通信に大きなコストがかかるといった制約があり，シングルGPUで検討した高速化手法をそのまま適用しても高い性能が得られるとは限らない．

本章では相互相関PIVをマルチGPU並列計算に適用する方法について検討をおこない，3章同様本研究の計算環境で実装をおこなった結果と合わせて考察をおこなう．

なお本章で用いる計算環境は，付録Aに示す計算環境-Ⅱである．

4.1　マルチGPU並列計算

冒頭で述べたように，マルチGPUを用いる最大のメリットはその高い並列性である．マルチGPUを用いることで，1000を超えるプロセッサコア数を有する計算機をデスクトップPCの規模で構築するができる．

ここではCUDAの仕様に限定して議論をおこなうと，現在マルチGPUはCPUのマルチスレッド処理を介して使用することができる．スレッドはマルチGPUを構成するGPUの数だけ生成され，各スレッドがひとつのGPUを制御することになる．CPUのマルチスレッド計算プラットフォームはpthreadやOpenMP，MPI(Message Passing Interface)などが考えられるが，本

研究では実装が比較的容易な windows thread を用いている．

マルチ GPU はその並列性でシングル GPU を大きく上回るが，データ転送に関してはシングル GPU にはない制約が存在する．現行のマルチ GPU 計算環境においては GPU 間の直接データ転送は不可能であり，GPU 間でデータを交換するためには必ず CPU のメモリを介する必要がある．CPU ⇔ GPU 間のデータ転送は GPU 並列計算で発生するデータ転送の中で最も低速なものであり，全体のパフォーマンスを下げる大きな要因となる．

したがって，マルチ GPU 並列計算により効果的な処理高速化を実現するためには，極力 GPU 間のデータ転送を発生させないかたちで複数 GPU に処理を分散する必要がある．以下ではこのような問題意識のもと，マルチ GPU 並列計算により相互相関 PIV を更に高速化する手法について検討していく．

4.2　マルチ GPU を用いた並列化手法

PIV 画像処理は，単一画像ペアを対象として一度だけ実行されるケースと時系列画像ペアに対して連続で実行されるケースとが考えられる．したがってマルチ GPU 並列計算により PIV 画像処理を高速化する手法としては，(1)PIV の処理単位をマルチ GPU で更に並列処理し高速化する方法と (2) シングル GPU による PIV 処理を複数 GPU で並列に処理する方法とが考えられる．

(1)PIV の処理単位をマルチ GPU で更に並列処理し高速化する方法 は，複数 GPU で PIV 処理単位の計算負荷を分散させる方法である．PIV 処理単位が高速化されるため，連続処理される PIV の回数によらず一定の性能が得られる．一方で (2) シングル GPU による PIV 処理を複数 GPU で並列に処理する方法 は，PIV 画像処理が時系列の複数画像ペアに対し連続で実行されることを前提とした手法である．PIV 処理単位あたりにかかる時間はシングル GPU 使用時と変わらないが，これらが並列に処理されることで処理全体としての高速化が実現できると考えられる．

以下では各手法について，より詳細な議論をおこなう．

4.2.1　PIV の処理単位をマルチ GPU で更に並列処理し高速化する方法

PIV の処理単位をマルチ GPU で高速化する手法はさまざまなものが考えられる．相互相関 PIV において相関計算は処理時間の多くの割合を占めるが，これに着目すれば fig. 4.1 に示すような手法が考えられる．

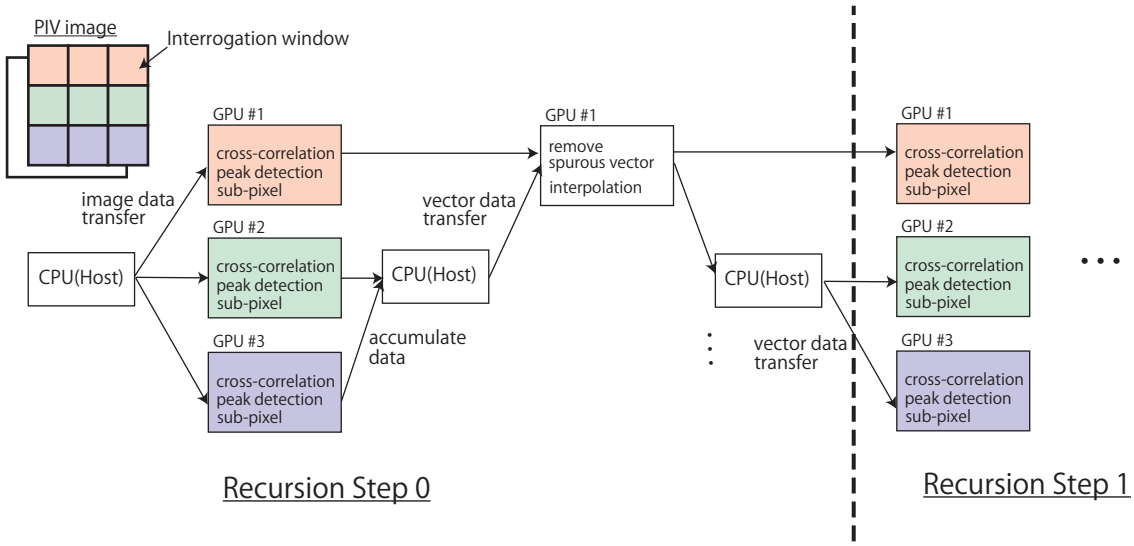


Fig 4.1: Schematic diagram of distributing cross correlation computing by multi-GPU: In this method, cross-correlation computing is distributed to each GPU. However in case of recursive or iterative method, GPUs must be synchronized to transfer vector data, which causes significant slowdown of total PIV processing. So in this study, this method is not implemented to multi-GPU.

Fig. 4.1 に示す手法は，GPU あたりの相互相関の計算負荷軽減をねらいとした手法である．図に示すように，各 GPU が担当する検査領域をあらかじめ割り当てておき，GPU は担当する検査領域に関する計算のみをおこなう．相関の計算が各 GPU で終了したのちにそのデータをひとつの GPU に集計し誤ベクトル除去や補間をおこなうことで，PIV 処理単位の計算負荷を各 GPU に分散させることができる．

しかし再帰的相互相関 PIV の実装を考慮すると，この手法では再帰のステップが増えるたびに Fig. 4.1 に示した GPU 間データ転送および同期が発生することになり，これが処理の大きなネックとなる．現行のマルチ GPU 計算環境では GPU 間のデータ通信に CPU のメモリを介する必要があり，その通信速度は GPU のビデオメモリにもまして低速である．より一般的な相互相関 PIV を高速化するためには，GPU 間データ通信をより回避できるような処理分散手法を検討する必要がある．

そこで本研究では，画像を分割し各 GPU に割り当てるという手法をとる．すなわち本手法は，分割した画像を小さな画像とみなし，その画像に対する PIV 処理を各 GPU に割り当てることで，GPU あたりの負荷を分散させる手法であるといえる．並列に動作している各 GPU は実質独立した画像に対する PIV 処理をおこなっているため，GPU 間のデータ通信は発生せず，最後にデータを CPU メモリで集計する際以外は GPU 間の同期をとる必要もない．この点で本

手法は上で述べたものよりも有効な手法であるといえる．しかしこの手法を用いた場合，分割された領域間の端部の扱いには注意を要する．画像を等分して GPU に転送した場合，元の画像の分割部分は分割された画像の端部であるため，本来存在すべき画素が分割された画像には存在せず，元画像とは異なる相関計算がおこなわれてしまう．そこで本研究では，分割部分を相互にオーバーラップした領域を各 GPU に転送することで，元画像における分割された部分のベクトルの計算もおこなっている．分割された部分のベクトルの計算はその領域を含む各 GPU が計算をおこなうため，この方法ではトータルの計算量が増加してしまうものの，正しい結果を得るためにはやむを得ない．

この画像分割手法を CUDA に実装し，その処理時間を測定した結果を Fig. 4.2 に示す．Fig. 4.2 は 256×256，512×512，1024×1024[pixel] の各画像サイズについて，シングル GPU，2 GPU および 4 GPU を使用したときの処理時間を示している．各条件ともに同一の PIV 処理をおこなっており，その解析結果は等しい．なおここで示す処理時間は，各 GPU へのデータ転送を開始した瞬間から，結果が正しい順序で CPU のメモリに確保されるまでの時間と定義している．

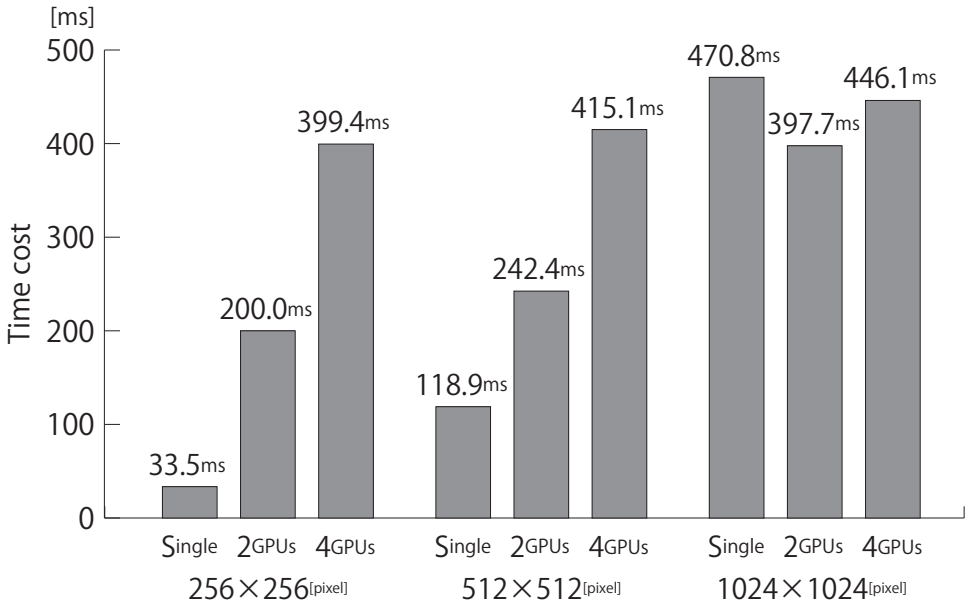


Fig 4.2: Time cost of image-split method in case of 256×256, 512×512 and 1024×1024[pixel] images: In case of 256×256 and 512×512[pixel] images, time costs of image-split method are higher than single-GPU processing because image data transfer to each GPU cannot be done at the same moment. In case of 1024×1024[pixel] image, image-split method is at most 1.1 times faster than single-GPU processing but it's not so reasonable.

Fig. 4.2 から明らかなように，256×256[pixel] および 512×512[pixel] サイズの画像を使用した条件では，画像分割手法を用いることでその処理性能が大きく下がってしまうことが確認された．画像サイズ 1024×1024[pixel] のケースでは一応の高速化は実現できているものの，その性能向上比は最高でも 1.1 倍程度であり，使用している GPU の数と比較すると妥当なものとはいえない．少なくとも本手法を用いた限りでは，PIV 処理単位の高速化手段としてマルチ GPU の有効性を確認することはできていない．

画像分割手法を用いた多くのケースで処理性能が低下してしまう原因として，GPU のメモリへの画像データの転送が同時に実行されていない点が挙げられる．本研究で使用したマルチ GPU 計算環境では，ミリセカンドオーダーでデータ転送の同期をとることができず，数ミリ〜百ミリ秒弱の幅でデータ転送の遅延が生じてしまった．その影響は使用する GPU を増やすほどに増していくため，この影響から Fig. 4.2 に示した結果が得られた可能性が高い．1024×1024[pixel] の画像サイズでは処理時間がデータ遅延にくらべ大きいため，本手法を用いることで多少の高速化が実現したと考えることができる．

PIV の処理単位をマルチ GPU で更に並列処理する手法は，本項で示した方法以外にも様々なものが考えられる．その点でマルチ GPU を用いて PIV 処理単位を高速化する方法には検討の余地があるといえるが，各 GPU への画像データ転送がミリセカンドオーダーで同期させることが難しいという事実は大きな制約となる．シングル GPU を用いた高速化で既に最高数ミリ秒の処理速度が達成されており，最高百ミリ秒ものデータ転送の遅延は決定的なデメリットとなる．数ミリセカンドから数十ミリセカンドオーダーの PIV 画像処理単位を更に高速化する手段としてマルチ GPU を用いることは，少なくとも本研究の結果からは非常に難しいといわざるを得ない．一方で GPU を用いても数百ミリセカンド以上の処理時間がかかるような非常に計算負荷の高い PIV 処理単位を高速化する手段としては，マルチ GPU は検討の余地があるといえる．その際には，GPU 間のデータ転送を極力避けつつデータ転送の遅延にもロバストであるような手法を検討すべきだろう．

4.2.2 シングル GPU による処理を複数 GPU で並列処理する方法

シングル GPU による PIV 画像処理を複数 GPU で並列に処理する方法は，3 章で示したシングル GPU による PIV 処理を複数 GPU で並列に実行する手法である．正しい結果を得るだけの目的であれば，各 GPU は完全に独立して処理をおこなうことが可能であり，GPU 間の同期をとる必要はない．しかしこの場合，処理される PIV の順序は実際に実行されるまで不明であり，

時系列順に処理結果を得ることができない．そのため本研究では，各 GPU と対応する CPU スレッド間で同期をとることで，マルチ GPU を用いながら時系列順に PIV 処理が実行可能なシステムを構築する．

実装および考察

ここでは本手法を CUDA を用いて実装した結果について示す．マルチ GPU 並列計算の妥当性を評価するにあたって，シングル GPU 使用時に得られた性能にたいしてどの程度の性能が得られたのかが重要な指標となる．ここでは，データサイズおよび PIV 処理時間を変化させたときの本手法の性能を，シングル GPU による処理時間と比較しながら議論をおこなう．

Fig. 4.3 に，連続処理される画像ペアの枚数を変化させたときの本手法の性能を示す．本研究で用いた計算環境では最大 4GPU によるマルチ GPU 環境が構築可能である．そのため Fig. 4.3 では，2GPU，3GPU，4GPU 使用時における PIV 連続処理の時間コストを，処理される PIV 画像ペアの数の関数としてプロットしている．PIV 解析に使用されている画像データサイズは 1024×512[pixel] であり，ペアで合計 1Mpixel のデータサイズとなる．PIV 処理単位として実行されている処理は再帰的直接相互相関法であり，シングル GPU で処理した際には約 80ms の処理時間を必要とする計算負荷をかけた．FFT 相互相関法を同様に実装したシステムの時間計測は今回おこなっていないが，本手法では相関計算の手法の違いには本質的な差異はなく，FFT 法を用いた場合でも同様な結果が得られる．なおここで示した処理時間には，各 CPU スレッドにおける GPU の初期化にかかる時間が含まれており，各 CPU スレッドにつき一度だけ GPU は初期化される．

Fig. 4.3 より，いずれのケースにおいてもシングル GPU 使用時に比べ処理性能が向上していることがわかる．連続処理する PIV 画像ペアの数を増やすことでシングル GPU 比の性能が緩やかに向上していくことが読みとれるが，いずれの GPU 数でもその性能向上は理論的限界の約 85%程度にとどまった．これは処理時間に GPU の初期化時間が含まれていること以外に，CPU スレッドで同期をとっていることや，CPU $\Leftrightarrow$ GPU 間のデータ転送を担うバスがボトルネックとなっていることが要因として考えられる．しかし一定の処理高速化は実現されており，たとえば 4GPU を使用したケースでは 600MB の画像データ解析が 15 秒足らずで処理可能であることがわかる．

一方で Fig. 4.4 は，PIV 処理単位の時間コストを変化させたときの本手法の性能の変化を示している．いずれの結果も 1024×512[pixel] サイズの画像 600 ペアを本手法で処理することで

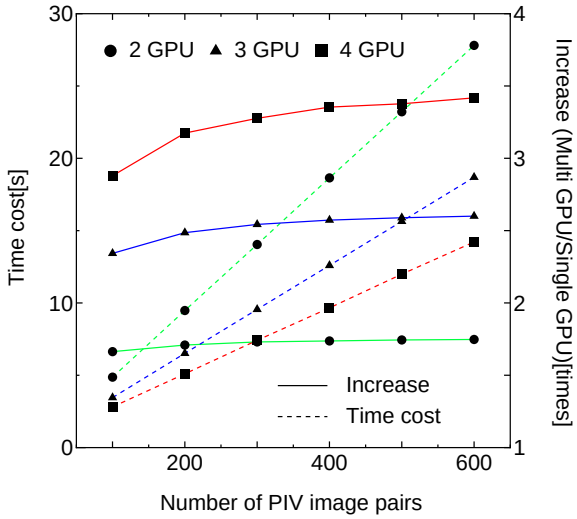


Fig 4.3: Time cost and performance improvement of parallel method of single-GPU processing as a function of total image pairs in case of 2, 3 and 4 GPUs: The time cost of single-CPU PIV processing is about 80ms. In any case, performance of multi-GPU compared to single-GPU gradually improves and about 85% of imaginary performance improvement can be obtained.

得られたものである．Fig. 4.4 から，PIV 処理単位の時間コストが高いほど本手法の性能が向上していることがわかる．逆に処理単位の時間コストが小さいケースではマルチ GPU の性能は総じて低く，2GPU のケースではマルチ GPU を用いると性能が低下してしまうケースすら存在した．これらは PIV 処理単位あたりの処理速度が速くなるにつれて，データ転送にかかる時間および CPU スレッド間での同期処理にかかる時間の占める割合が相対的に高くなることに起因していると考えられる．すなわち，本手法の効果はシングル GPU での処理時間に大きく依存し，シングル GPU である程度の処理時間を必要とする処理に対しては一定の効果を示すものの，シングル GPU による処理で既に非常に高速な処理を本手法で更に高速化することはできないということを示している．これはマルチ GPU を構成する GPU の性能によらず一般的にいえることであり，本章の計算環境で用いている GPU を更に高性能なものに変更しても結果は同様だと考えられる．処理を律速する要素は GPU ではなく，データ転送を担う PCI Express などのバスや GPU のドライバであると考えられ，シングル GPU で既に高速な処理をマルチ GPU で更に高速化するためには，これら GPU 周辺要素の性能向上が不可欠と考えられる．

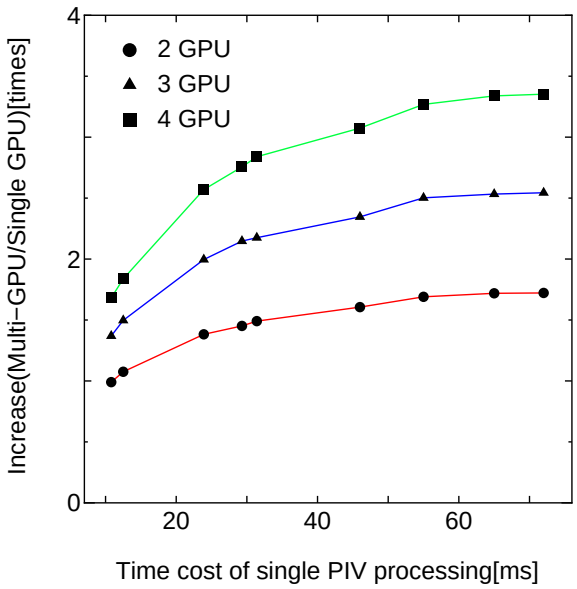


Fig 4.4: Performance improvement of parallel method of single-GPU PIV processing as a function of single-GPU processing time: This graph clearly shows the performance of multi-GPU processing depends on the time cost of single-GPU processing because the effect of data transfer between CPU and GPU and of synchronization of CPU threads becomes higher as single-GPU processing becomes faster.

4.3　本章のまとめ

本章では，シングル GPU に続きマルチ GPU 並列計算を用いた相互相関 PIV の高速化手法の検討評価をおこなった．まず PIV 画像処理単位をマルチ GPU により高速化する手法を検討し，画像を分割して各 GPU で分散処理をおこなう手法の実装評価をおこなったものの，データ転送をミリセカンドオーダーで並列に実行することができず効果的な高速化を実現することができなかった．PIV 処理単位をマルチ GPU により並列化する手法は検討の余地があると考えられるが，少なくとも本研究の結果から，マルチ GPU の効果が得られるのはシングル GPU による PIV 画像処理に比較的高い時間コストがかかるケースであると推測される．

次に PIV 画像処理が時系列粒子画像ペアに対して連続で実行されるケースを前提とし，シングル GPU による PIV 画像処理をマルチ GPU で時系列順に実行する手法を提案・開発した．その性能を評価した結果，限定的ではあるもののシングル GPU での時系列処理に比べ更に処理が高速化可能であることを確認した．シングル GPU での PIV 処理が数十ミリセカンドかかるような状況では，2GPU・3GPU・4GPU いずれのケースについても最高で理論性能の平均約 85%を獲得することができた．しかしシングル GPU と比較した際の性能比は PIV 処理単位の時間コ

ストに強く依存し，シングル GPU のみで既に高い処理速度が実現されているようなケースでは，マルチ GPU による効果は小さいことが確認された．

いずれのアプローチをとった場合でも，マルチ GPU 並列計算の性能はシングル GPU でどの程度処理が高速化されているかに強く影響を受けていることがわかる．シングル GPU で処理コストが比較的高いケースではマルチ GPU 並列計算により一定の効果を得ることができるものの，シングル GPU で既に非常に高速な処理を更に高速化する手段としては，マルチ GPU は有効な手段ではないということが本章から明らかになった．

第5章　直接法とFFT法を融合したハイブリッド相互相関PIVの提案と評価

3章で，直接相互相関法とFFT相互相関法各々について高速の手法を検討し，その結果をもとにPIV処理速度の評価をおこなった．本章ではまず，それら2種類の相互相関PIVについて比較検討をおこなう．そしてその比較結果をもとに，直接相互相関法とFFT相互相関法を融合したハイブリッド相互相関PIVを提案・開発し，検査画像および実画像を用いてその評価をおこなう．

なお，4章で示したマルチGPUの使用について本章では考慮しないが，本章で示す手法のマルチGPUへの適用する場合，その方法は4章で示したものと変わらず，よってその結果も同様である．

5.1　直接相互相関法とFFT相互相関法の比較

本節では，直接相互相関法とFFT相互相関法について比較をおこなう．

直接法もFFT法も，ともに与えられた二つの信号の相互相関を求める手法である．一般的に，FFTを用いた相互相関の計算は直接法に比べて高速であると言われているが，ことPIVへの適用を考えたときには，その2手法の比較には注意が必要である．本節では，直接法とFFT法について，その計算量および実際の処理時間の観点から比較をおこなう．

5.1.1　計算量の比較

一般にFFTを利用した相互相関の計算は直接相互相関を計算する方法に比べて高速であると言われるが，ことPIVに関してはこれは必ずしも正しい主張ではない．なぜなら直接相互相関法には探査領域の概念があり，検査領域あたり計算される相互相関の数は直接相互相関法とFFT相互相関法とで異なるからである．ここでは，各手法で必要となる計算量を比較することで，両者の計算処理速度について考察をおこなう．

ここでは同サイズの検査領域 (サイズ $N_{IW} \times N_{IW}$[pixel]) を対象として，相互相関を計算するために必要な両手法の計算量について考える．なおここでは減算は加算とみなし，加算と乗算のコストも等価であるとする．現在多くの CPU アーキテクチャにおいて加算と乗算のコストは等価であり，GPU においてもまた同様であることが多い．たとえばまた本研究で用いる NVIDIA 社製の GPU(GTX480，GTX285) は，1 クロックに 1 回の乗算と加算が実行可能である．

直接相互相関法

本研究の直接相互相関 PIV で計算される相互相関は，式 (2.1) に示される相互相関係数である．この式ををを変形すると，

$$R_{fg}^2(\Delta X, \Delta Y) = \frac{\left\{ \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} \{f(X_i, Y_j) - f_m\}\{g(X_i + \Delta X, Y_j + \Delta Y) - g_m\} \right\}^2}{\sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} \{f(X_i, Y_j) - f_m\}^2 \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} \{g(X_i + \Delta X, Y_j + \Delta Y) - g_m\}^2} \tag{5.1}$$

$$R_{fg}^2(\Delta X, \Delta Y) = \frac{\{(f,g) - N_{IW}^2 f_m g_m\}^2}{\{(f,f) - N_{IW}^2 f_m^2\}\{(g,g) - N_{IW}^2 g_m^2\}} \tag{5.2}$$

ただし，

$$(f,g) = \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} f(X_i, Y_j) g(X_i + \Delta X, Y_j + \Delta Y) \tag{5.3}$$

$$(f,f) = \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} f^2(X_i, Y_j) \tag{5.4}$$

$$(g,g) = \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} g^2(X_i + \Delta X, Y_j + \Delta Y) \tag{5.5}$$

$$N_{IW}^2 f_m = \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} f(X_i, Y_j) \tag{5.6}$$

$$N_{IW}^2 g_m = \sum_{i=1}^{N_{IW}} \sum_{j=1}^{N_{IW}} g(X_i + \Delta X, Y_j + \Delta Y) \tag{5.7}$$

となり，計算機では式 (5.3)〜(5.7) の 5 つの式がループで計算される．この計算には 3 回の乗算と 5 回の加算が含まれるため，探査領域サイズを $N_{SW} \times N_{SW}$[pixel] とすればその計算量は

$$8N_{SW}^2 N_{IW}^2 \tag{5.8}$$

となる．なお，式 (5.3)〜(5.7) から相互相関係数を計算する際にかかる計算量は，これらの式の計算量に比べて十分小さいためここでは無視している．式 (5.8) から明らかなように，直接相互相関法の計算量は，検査領域サイズ N_{IW} と探査領域サイズ N_{SW} に依存していることがわかる．

FFT 相互相関法

次に FFT 相互相関法の計算量について考える．サイズ N の 1 次元 FFT の計算量が $O(N \log N)$ であることは広く知られているが，具体的な計算量は手法により異なっている．ここでは，基数 2 の 1 次元 FFT の計算量としてよく用いられる $5N \log_2 N$ を採用する (参考文献！)．FFT 相互相関法で用いられる FFT は 2 次元 FFT であるが，Row-Column 法 (参考文献！) を用いれば，これはサイズ N の 1 次元 FFT を 2N 回実行することによって計算できる．実際には，f，g に対する 2 次元 FFT と，フーリエ変換された f,g の要素積の 2 次元逆 FFT の計 3 回 2 次元 FFT の計算が必要である．したがって，FFT 相互相関法におけるサイズ N_{IW}^2 の 1 検査領域にかかる計算量は，

$$30N_{IW}^2 \log_2 N_{IW} + N_{IW}^2 \tag{5.9}$$

となる．なお式 (5.9) 第二項は，フーリエ変換された f,g の要素積の計算量である．FFT 相互相関法には探査領域の概念は存在しないため，その計算量は検査領域サイズのみに依存する．

直接相互相関法と FFT 相互相関法との計算量の比較

同一サイズの検査領域 ($N_{IW} \times N_{IW}$[pixel]) における両手法の計算量を比較したとき，その大小は直接相互相関法で用いられる探査領域サイズ N_{SW} で決まる．いま，式 (5.8) と式 (5.9) とを用いて，両手法の計算量が等しくなる状態，すなわち

$$8N_{SW}^2 N_{IW}^2 = 30N_{IW}^2 \log_2 N_{IW} + N_{IW}^2 \tag{5.10}$$

が満たされるときの探査領域サイズを考え, これを臨界探査領域サイズ (linear dimension of critical search window)N_{CSR} と定義する．検査領域と探査領域のサイズ比を表す変数 R を

$$R = \frac{N_{CSR}}{N_{IW}} \tag{5.11}$$

と定義すれば，式 (5.11) は，

$$R = \sqrt{\frac{30 \log N_{IW} + 1}{8 N_{IW}^2}} \tag{5.12}$$

と書ける．また $N_{CSR} = RN_{IW}$ であるため，(5.13) 式は，

$$N_{CSR} = \sqrt{\frac{15}{4}(\log N_{IW} + 1)} \tag{5.13}$$

と変形できる．(5.12) 式と (5.13) 式はそれぞれ各検査領域に対する臨界探査領域サイズの比と臨界探査領域サイズそのものを示している．これらの式を一般的に用いられる検査領域サイズ (8×8，16×16，32×32，64×64，128×128[pixel]) についてプロットしグラフ化したものが Fig. 5.1 である．実線に赤丸の線グラフが R を示し，点線に緑の三角の線グラフが臨界探査領域サイズ N_{CSR} を示す.

Fig. 5.1 から，検査領域サイズが大きくなるにつれ臨界探査領域サイズは緩やかに大きくなるが，検査領域とのサイズ比は急激に下がっていることがわかる．両手法の計算量のみから比較した場合，臨界探査領域サイズはたかだか 3×3〜5×5 [pixel] であり，それ以上の探査領域を設定した場合直接法の計算量は FFT 法を上回ることになる．これらの臨界探査領域は，直接法で設定される探査領域としては非常に小さい．しかし，いずれの検査領域サイズでも直接法の計算量が FFT 法の計算量を上回るケースが存在することはこの図から明らかである．実際の計算機による処理ではデータ転送コストなども加味されるため，実際の臨界探査領域は Fig. 5.1 で示されたサイズとは異なる可能性が高い.

以降では本研究の環境を例に，実際の処理時間から臨界探査領域サイズを見積もる.

5.1.2　実際の処理時間の比較

ここでは 3 章で開発した直接相互相関法および FFT 相互相関法を用いて処理時間の比較をおこない，実環境での臨界探査領域サイズを見積もる．直接相互相関法と FFT 相互相関法の処理の差異は，2.1.1 および 2.1.2 で示した相互相関計算部分のみである．したがってその処理速度を比較するためには，相互相関計算部分にかかる時間の比較をおこなえばよい.

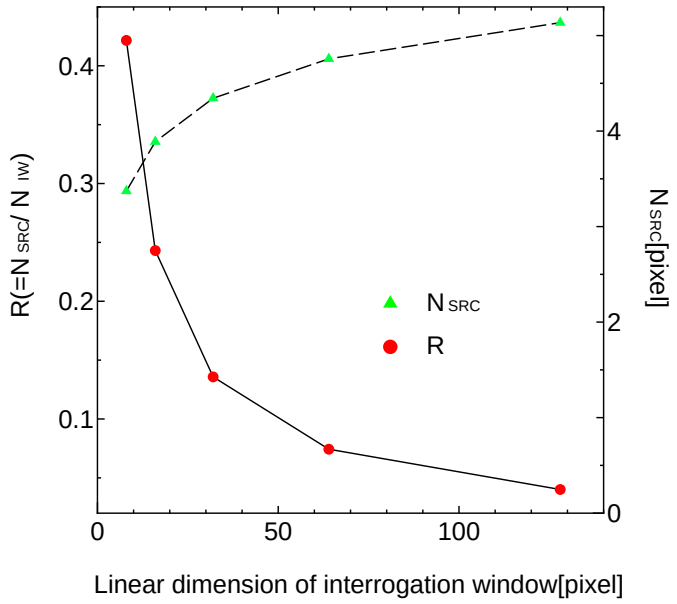


Fig 5.1: $R(=N_{SW}/N_{IW})$ and Linear dimension of critical search window[pixel] as a function of linear dimension of interrogation window[pixel]. The solid line with red squares shows R and the dash line with green triangles shows linear dimension of critical search window.

Fig. 5.2，Fig. 5.3，Fig. 5.4 は，それぞれ画像サイズ 1024×1024，512×512，256×256[pixel] における各手法の相互相関計算時間を，直接法で設定される探査領域サイズごとにプロットしたものである．これらの図は，各検査領域サイズについて本計算環境において最も高いパフォーマンスが出た手法で得られた結果を用いて作成した．FFT 相互相関法には探査領域は存在しないため，その処理時間は探査領域サイズによらず一定である．各図において実線はオーバーヘッドを含めた処理時間，点線は GPU 処理の合計時間を示す．直接法の GPU 処理の合計時間が示されていないが，これはオーバーヘッドをも含めた処理時間とほとんど変わらないためである．1024×1024[pixel] の画像サイズなどではオーバーヘッドに対して処理時間が十分長いため，FFT 法でも直接法同様にオーバーヘッドの影響は無視できるほど小さい．しかし Fig. 5.4 などに示されるように，画像サイズが小さくなるにつれて計算量の総量が減り，結果オーバーヘッドの時間コストが相対的に影響を強めていることがわかる．

Fig. 5.2 〜 Fig. 5.4 で示した多くのケースで，探査領域が大きくなるにつれその処理時間は増大し，ある検査領域サイズで FFT 法による処理時間を超過していることがわかる．すなわちこれらの図を用いることで，各条件について実際の処理時間に基づいた臨界探査領域サイズがわかる．この臨界探査領域サイズ N_{CSR} を検査領域サイズ N_{IW} の関数としてプロットしたものが Fig. 5.5 であり，検査領域に対する臨界探査領域のサイズ比 $R = N_{CSR}/N_{IW}$ を同様にプロッ

トしたものが Fig. 5.6 である．実線は各画像サイズの結果を示しており，点線は Fig. 5.1 で示した各手法の計算量から見積もった臨界探査領域サイズである．本研究の直接法の実装手法では並列度があまり上がらないケース (画像サイズ 256×256，512×512[pixel] における検査領域 128×128[pixel] のケースなど) では例外的な挙動を示しているものの，1024×1024[pixel] の画像サイズにおいては，検査領域が大きくなるにつれ臨界探査領域自体は緩やかに大きくなる一方，検査領域とのサイズ比は急激に低下しており，これは計算量から見積もった結果の傾向とよく一致している．しかし臨界探査領域のサイズは，多くのケースで計算量から見積もったサイズに比べ大きくなることが確認できる．直接法で十分な並列化がなされていないケースを除けば，本研究の環境では 10×10[pixel] 前後の探査領域を設定した直接法の処理時間は FFT 法とほぼ変わらないか，もしくは FFT 法よりも高速であることがわかる．

直接相互相関法のダイナミックレンジを決める要素として探査領域を捉えた場合，10×10[pixel] の探査領域は小さいといわざるを得ない．しかしたとえば再帰的相互相関 PIV における最終ステップや画像変形のために繰り返される相互相関計算ステップにおいては，探査領域を小さく設定することは結果の精度の観点からも正当化される．少なくとも本研究の計算環境においては，直接法による計算が処理速度の観点から FFT 法よりも有効であるケースが存在することが Fig.5.6 および Fig. 5.5 からわかる．

データ転送の観点から見れば FFT 法は直接法に比べて複雑であり，よってそのコストは高い．本研究が示した結果はひとつの計算環境で得られた結果であり完全に立証することは難しいが，データ転送付加も合わせて考えた場合，他の計算環境における臨界探査領域のサイズも，本研究の結果同様計算から見積もったサイズに比べ大きな値が得られる可能性が高いと推測される．

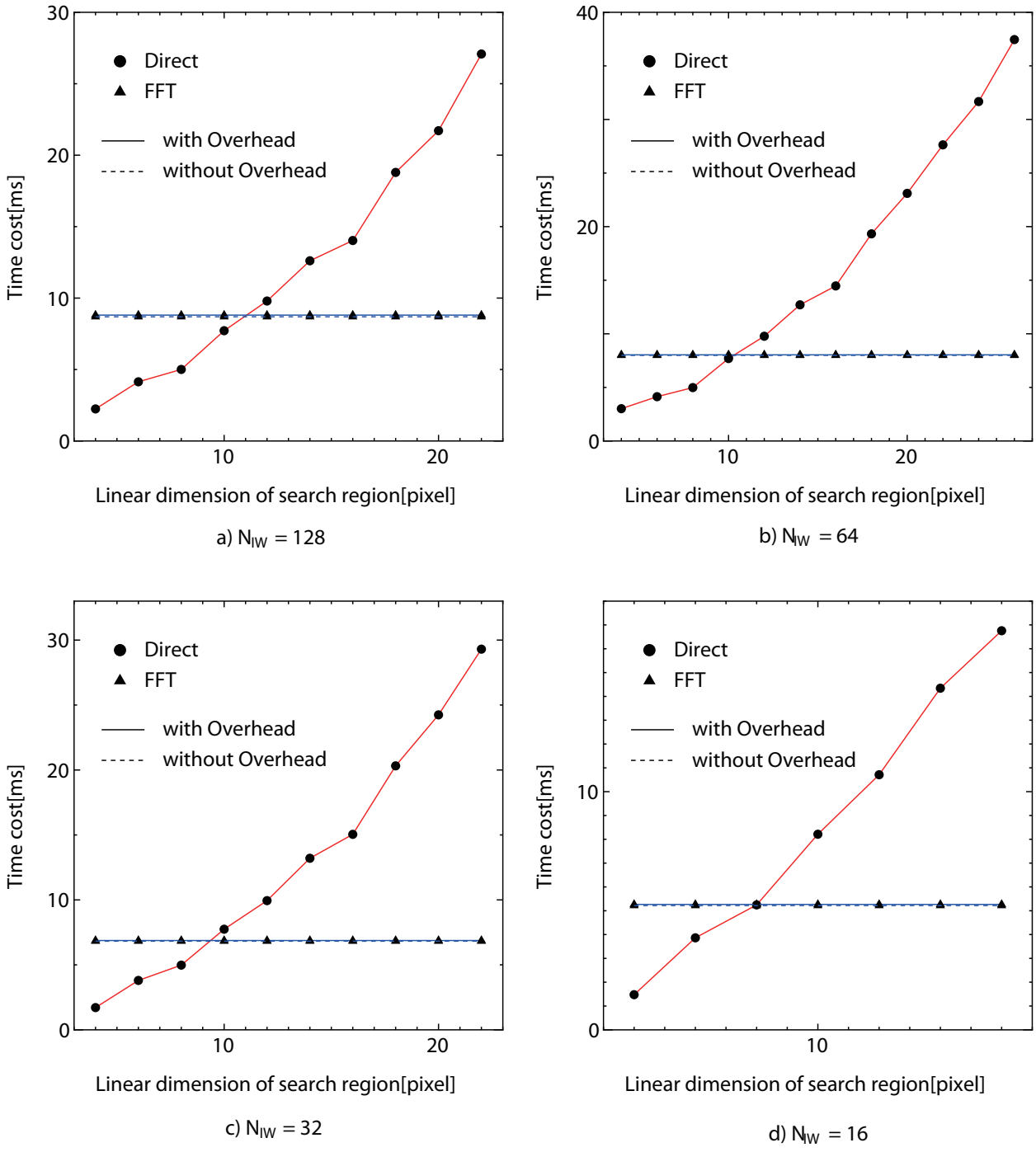


Fig 5.2: Speed comparison between direct method and FFT-based method in case of 1024×1024[pixel] size image: In this case, overhead of GPU processing is relatively so small that it can be neglected.

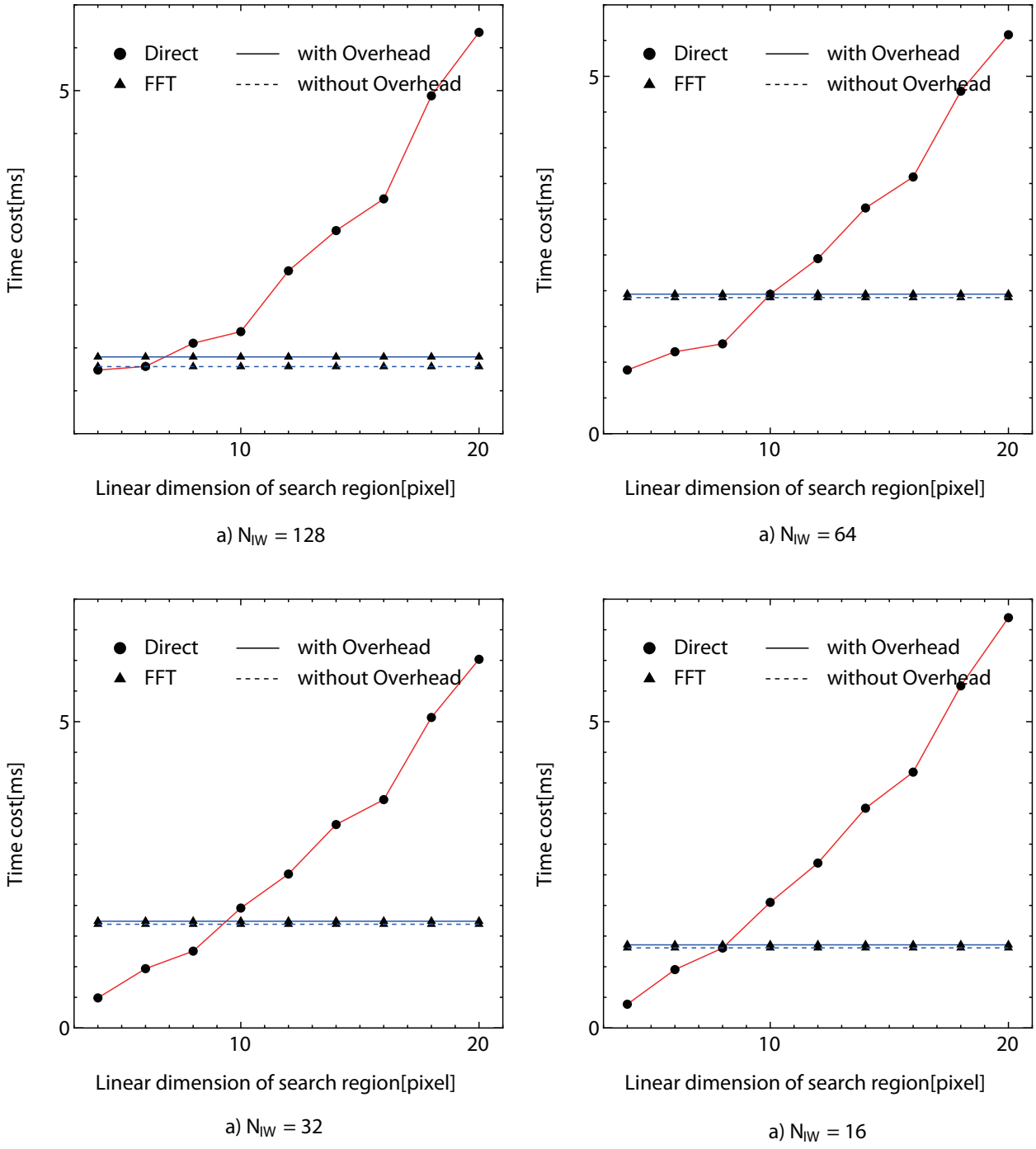


Fig 5.3: Speed comparison between direct method and FFT-based method in case of 512×512[pixel] size image: The tendency is almost the same with the case of 1024×1024[pixel] image size except the effect of overhead becomes relatively larger.

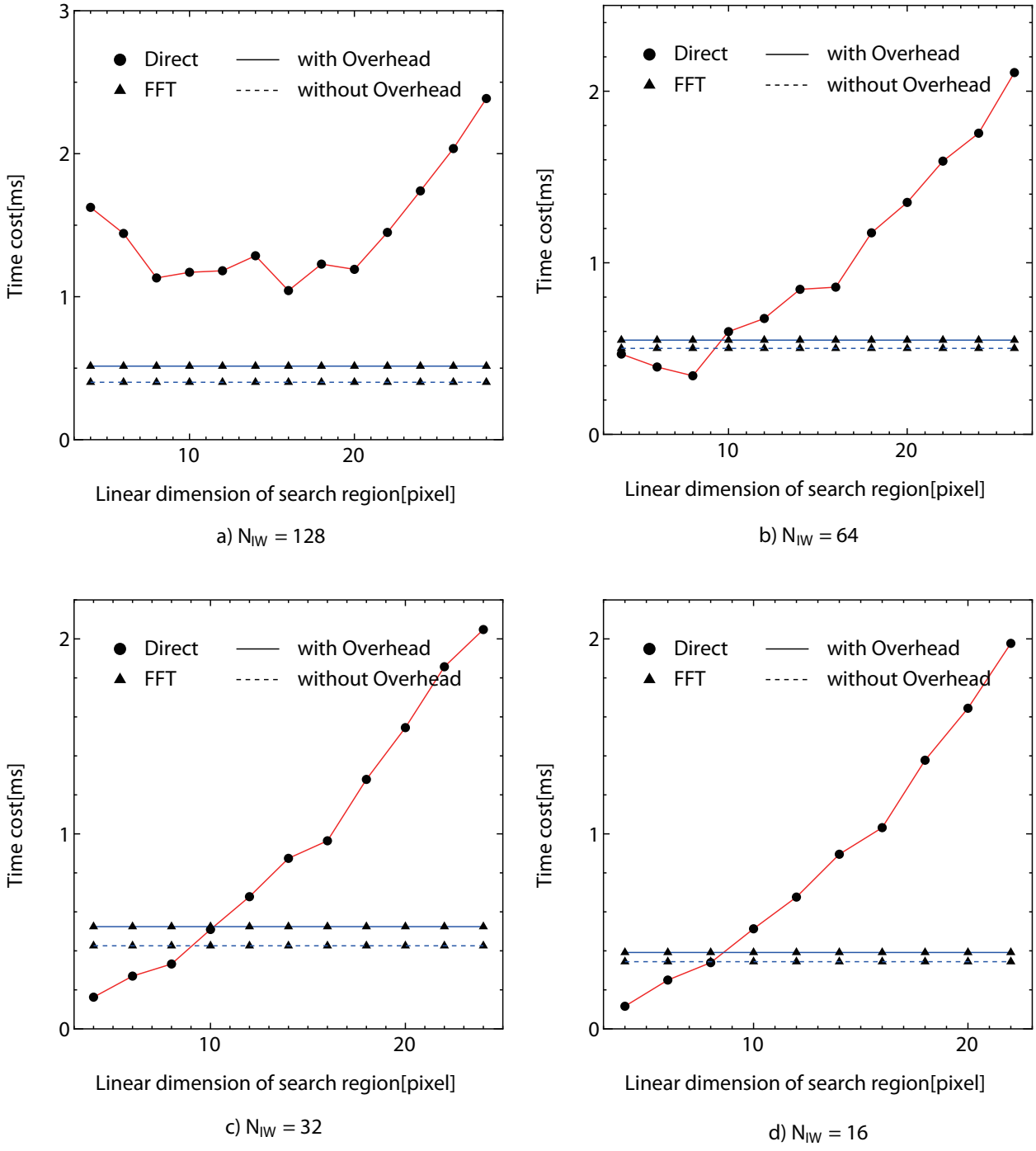


Fig 5.4: Speed comparison between direct method and FFT-base method in case of 256×256[pixel] size image: In case of $N_{IW} = 128$[pixel], there is no cross-section between direct method and FFT-based method because the number of cross-correlations is so small that parallelization cannot be optimized. This means critical search region size is zero(or none) in this case.

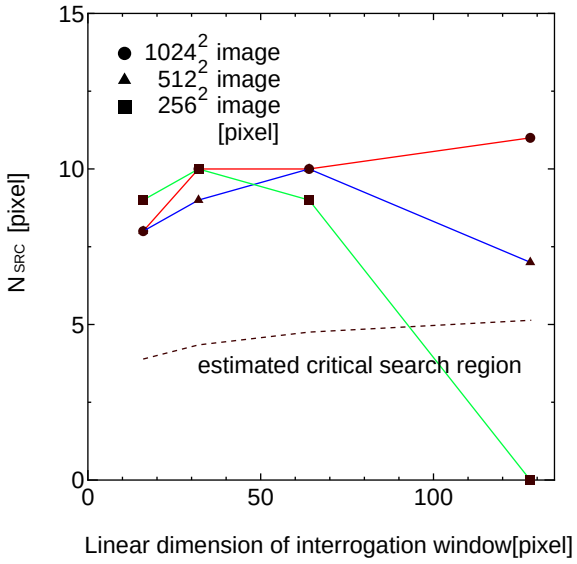


Fig 5.5: Critical search region size as a function of interrogation window size: In this graph, solid line is obtained from speed comparison between direct method and FFT-based method in each image size and dashed line is obtained from the comparison of computation load between these two methods. This graph shows critical search region size obtained from actual speed comparison tends to be larger than estimated critical search region size.

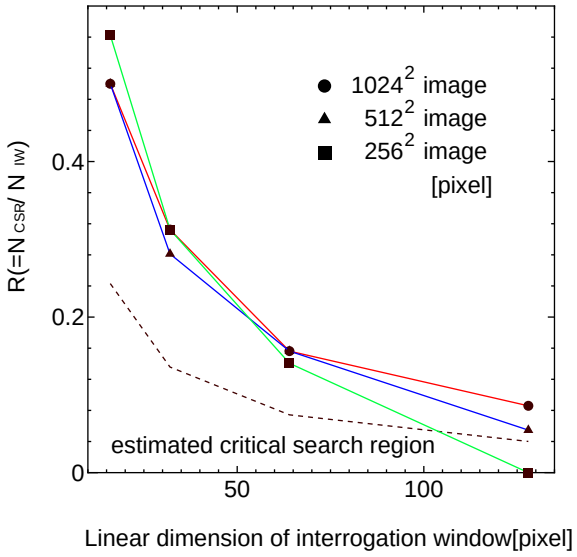


Fig 5.6: $R = N_{CSR}/N_{IW}$ as a function of interrogation window size: This graph shows R drastically decreases as interrogation window size increases. and R obtained from actual speed comparison tends to be larger than estimated critical search region size, which corresponds to the result of Fig. 5.5

5.2 ハイブリッド相互相関 PIV

直接相互相関法において，前節で議論した臨界探査領域サイズより小さな探査領域を設定した場合，直接法の処理速度は FFT 法を上回る．FFT 法に対する直接法の精度およびロバスト性の高さは，評価式の違いおよび FFT 法における信号の周期性の仮定を考慮すれば理論的に明らかである．探査領域を小さく設定することは直接相互相関 PIV のダイナミックレンジを低下させることにつながりうるが，たとえば再帰的手法の初回以降のステップではピーク位置が探査領域中央近傍に存在していることが前提であり，これらのステップにおいて小さな検査領域を設定することはダイナミックレンジの低下にはほぼ影響を与えない．これらの考察をふまえると，直接法と FFT 法とを融合したハイブリッド相互相関 PIV(Hybrid cross-correlation PIV，ハイブリッド法) の有効性が推察できる．すなわち再帰的相互相関 PIV において，初期のステップではダイナミックレンジが広くかつ高速な FFT 法を利用して PIV 処理をおこない，以降のステップでは臨界探査領域より小さな探査領域を設定した直接法による PIV をおこなうことで，直接法よりも高ダイナミックレンジで，FFT 法よりもロバスト性が高く，両手法よりも更に高速な PIV が実現する可能性がある．本章ではこのハイブリッド相互相関 PIV の提案をおこなう．以降でそのアルゴリズムの概要を述べる

5.2.1 ハイブリッド相互相関 PIV のアルゴリズム

ハイブリッド相互相関 PIV は，相互相関の計算に直接法と FFT 法を併用した PIV である．本研究では再帰的相互相関 PIV に対してハイブリッド法の適用を検討する．すなわち，再帰的手法の各ステップで FFT 法と直接法を切り替えながら PIV 処理をおこなっていく．直接相互相関 PIV と FFT 相互相関 PIV の両者が開発できている状況ならば，上記アルゴリズムに基づいてハイブリッド相互相関 PIV システムを構築することは比較的容易である．

前節で述べた直接法と FFT 法の各々のメリットを活かすことを考えたとき，再帰の初期のステップでは FFT 法を利用し，途中で直接法に切り替えるという方針は一意に定まる．アルゴリズムとして考察が必要なのは，FFT 法から直接法へと切り替える判定基準である．方針としては大きくふたつ考えられ，ひとつはステップごとに使用する相互相関計算をあらかじめ決めておく方法，もうひとつは動的な判断基準を設けそれに基づいて処理を切り替える方法が考えられる．

本研究で実装したハイブリッド相互相関 PIV は前者の方針をとった．すなわち，再帰処理の

最初のステップには FFT 相互相関法を用い，続く全てのステップには直接相互相関法を用いることとした．このとき初回以外のステップでは探査領域サイズを指定する必要があるが，開発したシステムではこれらのサイズはユーザーによって指定される．臨界探査領域より小さな探査領域が設定されることで，このハイブリッド法は既存手法よりも高速な処理が可能になると考えられる．

本研究では以上のようなアプローチでシステムの実装をおこなったが，特に相互相関計算手法の切り替え処理等に関しては，他手法との比較を踏まえた更なる議論が必要である．相関計算の切り替えによってハイブリッド法の性能は大きく変化する可能性が高い．精度・速度の両面から，ハイブリッド法の性能を最大限引き出すことができる手法の開発が望まれる．

5.3　ハイブリッド相互相関 PIV の評価

ここでは，前節で示したハイブリッド相互相関 PIV の評価をおこなう．人工画像および実画像を含む 3 パターンの時系列画像データに対し，開発したハイブリッド相互相関 PIV を適用し，その精度と処理速度について議論をおこなう．

5.3.1　標準画像による評価

ここでは人工画像を用いたハイブリッド法の評価をおこなう．Fig. 5.7 に本研究で作成した人工画像の例を示す．この PIV 粒子画像はハーゲン・ポアズイユ流れを表しており，Okamoto ら [31] がおこなった標準画像作成に関する研究成果に基づいて作成した．ハーゲン・ポアズイユ流れは導管内での層流の厳密解として広く知られており，その速度は導管と接する部分でゼロ，導管中心で最大となる．Fig. 5.7 に示した画像の縦方向のサイズは導管の直径に対応しており，したがってその速度分布は上下端でゼロ，垂直方向の中心部分で最大となる．

ここでは，ハーゲン・ポアズイユ流れの最高流速 D が異なる複数の人工画像を作成し，それぞれに対して再帰的直接相互相関 PIV，再帰的 FFT 相互相関 PIV，再帰的ハイブリッド相互相関 PIV を適用したときの処理時間および最小二乗誤差 (Root mean square, RMS エラー) を計測した．人工画像からは速度場の理論値が得られるため，RMS は各手法の解析結果と理論値とから得ている．処理速度の結果を Fig. 5.8 に，RMS エラーの結果を Fig. 5.9 に示す．人工画像のサイズは 1024×1024[pixel] であり，いずれの条件においても最終的な検査領域サイズは 32×32[pixel]，粒子密度は $180000/1$Mpixel，画像中に導出されるベクトルは 3969 である．ハー

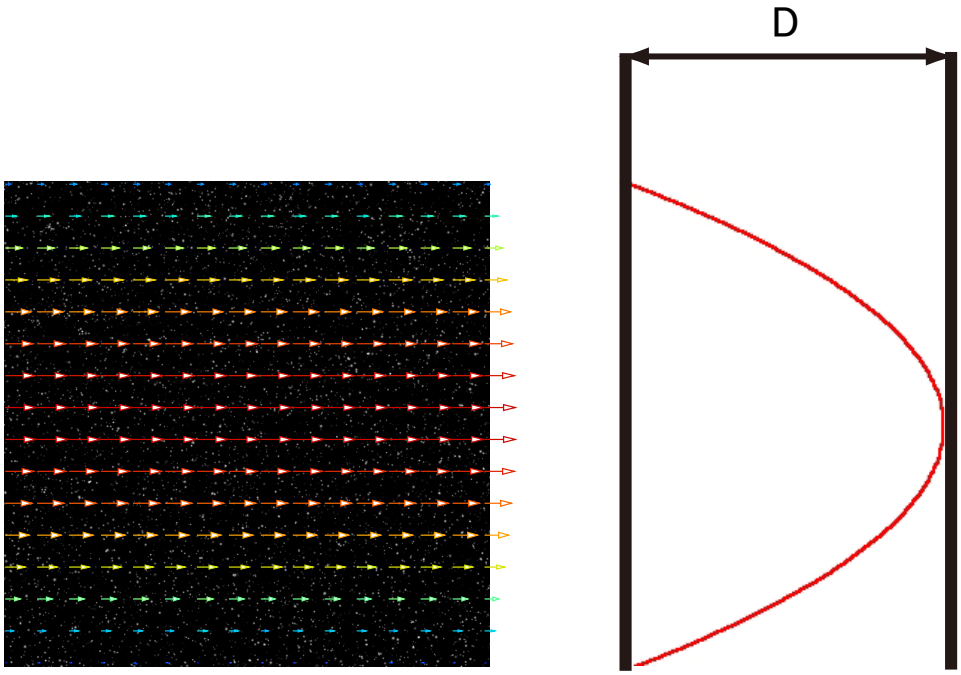


Fig 5.7: Synthetic image of Hagen-Poiseuille flow and the definition of D: On the left graph, color vector map is obtained from the result of PIV processing. On the right graph, D means maximum displacement of Hagen-Poiseuille flow. In this study, various synthetic images, which have different D are prepared to validate the effect of hybrid method.

ゲン・ポアズイユ流れの最大流速 D は 4,8,16,24,32[pixel] の計 5 パターンの画像を作成し解析に用いた．なお再帰ステップの回数や各ステップにおける探査領域サイズの設定については Table 5.1 に示す．これらはいずれも，ベクトルを正しく解像するために必要な最低限の計算条件である．ハイブリッド法は 5.2 で述べたように，再帰の初回ステップを FFT 法で，以降のステップを直接法で計算している．

Fig. 5.8 から，比較的小さな D の値においてはどの手法の計算時間にもあまり違いがない．しかし D の値が大きくなるにつれて，直接法による計算時間は爆発的に増加しているのに対し，FFT 法およびハイブリッド法では再帰ステップが増加しない限り計算時間は増加していない．数十ピクセルの変位を検出するためには大きな探査領域を設定する必要があり，これが直接法の計算時間を増加させる原因となっている．一方で Fig. 5.9 に示される各手法の精度を比較すると，FFT 法も直接法も最大流速 D が大きくなるにつれ RMS エラーも増大しているが，その誤差の大きさには最大で約 2 倍の開きがあることがわかる．ハイブリッド法も同様に D のサイズに依存してエラーも増大しているが，その誤差は FFT 法ではなく直接法に依存していることがわかる．これは，再帰の最終ステップが直接法により計算されていることに起因する．なお，

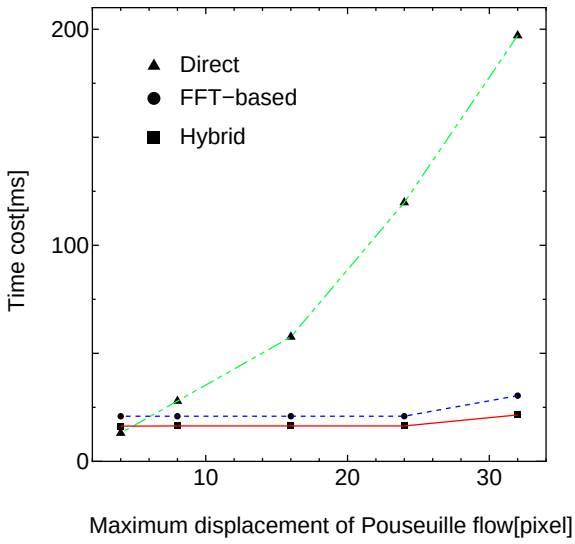


Fig 5.8: Time cost of each PIV processing as a function of maximum displacement of Hagen-Pouseuille flow: This graph shows hybrid method is the fastest in almost all cases. The time cost of direct method drastically increases as D increases because larger search region must be set to detect the peak in the distance.

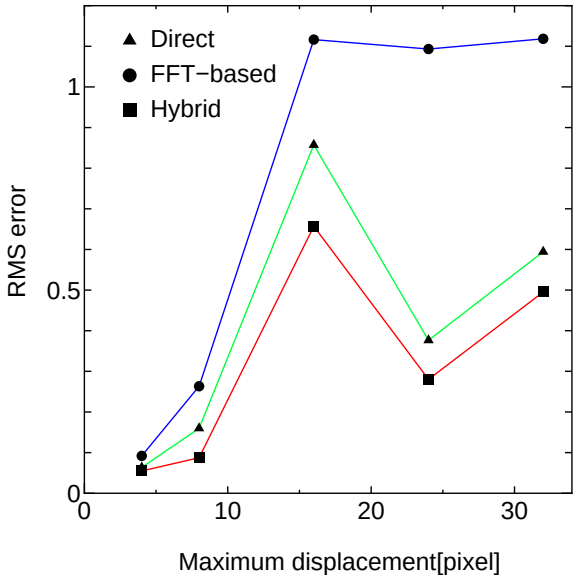


Fig 5.9: RMS error of each PIV processing as a funcition of maximum displacement of Hagen-Pouiseuille flow: This graph shows RMS error increases as D becomes larger but the error of direct and hybrid method is about the half of FFT-based method. Error of hybrid method follows the one of direct method because the latter steps of hybrid method is processed by direct method.

Table 5.1: Parameter setting for each PIV method: In this table, interrogation window size is shown like X^2 and search window size is shown like [X^2]

.

D[pixel]	Direct	FFT-based	Hybrid
4	$32^2[12^2]$	$64^2 \rightarrow 32^2$	$64^2 \rightarrow 32^2$
8	$32^2[20^2]$	$64^2 \rightarrow 32^2$	$64^2 \rightarrow 32^2[6^2]$
16	$64^2[32^2] \rightarrow 32^2[6^2]$	$64^2 \rightarrow 32^2$	$64^2 \rightarrow 32^2[6^2]$
24	$128^2[48^2] \rightarrow 64^2[8^2] \rightarrow 32^2[6^2]$	$64^2 \rightarrow 32^2$	$64^2 \rightarrow 32^2[6^2]$
32	$128^2[64^2] \rightarrow 64^2[8^2] \rightarrow 32^2[6^2]$	$128^2 \rightarrow 64^2 \rightarrow 32^2$	$128^2 \rightarrow 64^2[8^2] \rightarrow 32^2[6^2]$

本結果に限っていえば，ハイブリッド法の誤差は直接法および FFT 法単独での結果に比べ総じて低かった．この原因は明らかではなく，異なったテストケースでの精度評価の結果と比較したうえでの議論が必要である．

この評価実験結果で最も注目すべきなのは，ハイブリッド法の処理速度である．Fig. 5.8 に示されるようにその処理速度は FFT 法よりも更に高速あることが確認できる．これはハイブリッド法中の直接法で設定される探査領域サイズが，臨界探査領域よりも小さく設定されていることの効果である．したがってこの評価結果から，ハイブリッド法は，FFT 法と同等のピーク検出のレンジをもちかつ直接法と同等またはそれ以上の精度で PIV 処理を，どちらの手法よりも更に高速に処理可能な手法であることを確認することができた．またハイブリッド法は特に大きな変位を検出する必要があるケースにおいてその効果を発揮する手法であるということも，本評価実験から明らかになった．

以降では実験により得られた実画像を用いることで，ハイブリッド法をより詳細に評価していく．

5.3.2 堀田ら (2011) の実験画像を用いた評価

堀田ら [59] は現在，直列 2 円柱構造物の流体励起振動について研究をおこなっており，円柱の振動特性や相互作用について PIV 解析による解明を試みている．本項では堀田らがおこなった実験で得られた画像を用いてハイブリッド相互相関 PIV の評価をおこなう．なお現象の分析および解明は本研究の範囲外であり，よってここではそれらには言及しない．

堀田らの実験画像データは合計で数 GB にものぼり，ここでその全てを紹介することはできない．本項ではその時系列画像の中の 1 ペアを例としてとりあげる．解析対象となる画像ペアを Fig. 5.10 に示す．この画像において流れは左 (上流) から右 (下流) に向かって流れており，この流れに励起されることで画像中央部の 2 つの円柱構造物が振動する．画像サイズは 1024×512[pixel] である．

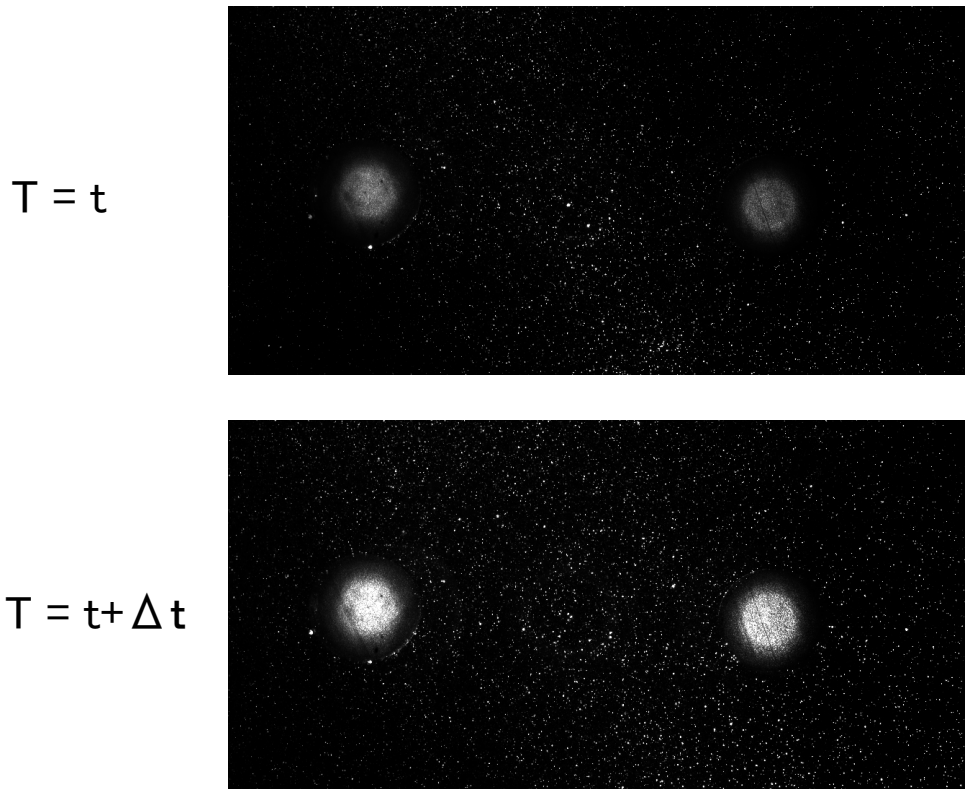


Fig 5.10: Sample image pair of Hotta’s experiment:

この画像ペアに対して，再帰的直接相互相関 PIV，再帰的 FFT 相互相関 PIV および再帰的ハイブリッド相互相関 PIV を適用した結果を Fig. 5.11，Fig. 5.12 および Fig. 5.13 にそれぞれ示す．いずれの処理も再帰のステップは 2 回設定し，検査領域は [128×128，64×64，32×32[pixel]] と遷移する．各図にはステップごとの PIV 解析結果が示されており，上のステップでの結果を反映して下の段の PIV 解析がおこなわれる．導出されるベクトルの数は，検査領域 128×128[pixel] で 105，64×64[pixel] で 465，32×32[pixel] で 1953 である．最終的に導出されたベクトルを第一画像上にオーバーラップして表示した結果を Fig. 5.14 に示す．

Fig.5.11〜Fig. 5.13 から，1 ステップ進んだ検査領域サイズ 64×64[pixel] の段階までは，多少の誤ベクトルの発生が確認できるものの総じてどの手法でも比較的良好な可視化結果が得られ

ている．いずれの結果でも，下流側円柱前部を交差するかたちで流れ場が形成されている様子を観察することができる．しかし更に 1 ステップすすめた検査領域サイズ 32×32[pixel] の段階になると手法間で結果に違いが生じ，FFT 法による解析では誤ベクトルが多く発生してしまっている一方，直接法とハイブリッド法からは依然として良好な可視化結果が得られている．

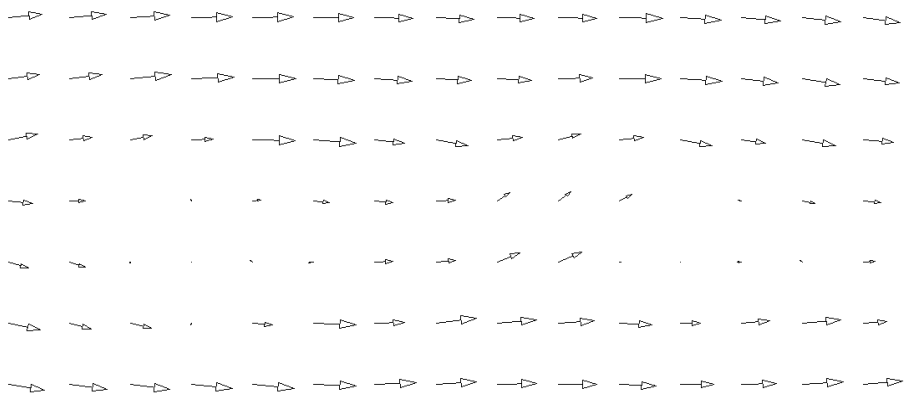
本画像ペアの処理時間について Table 5.2 に示す．臨界探査領域よりも小さな探査領域を設定しているため，ここでもハイブリッド法の処理速度は最も速い．ほぼ同精度と考えれられる直接法と比較すると，約 1.6 倍高速化が達成できており，これは CPU による直接相互相関 PIV の処理速度と比較すれば約 1300 倍にのぼる．

このように，堀田らの実験画像を用いた解析においても，ハイブリッド法は FFT 法以上に高精度かつロバストであり，直接法と比較してその処理が高速であることが確認できた．これらの結果から，実画像の解析においても全項と同様にハイブリッド法の有用性を確認することができたといえる．

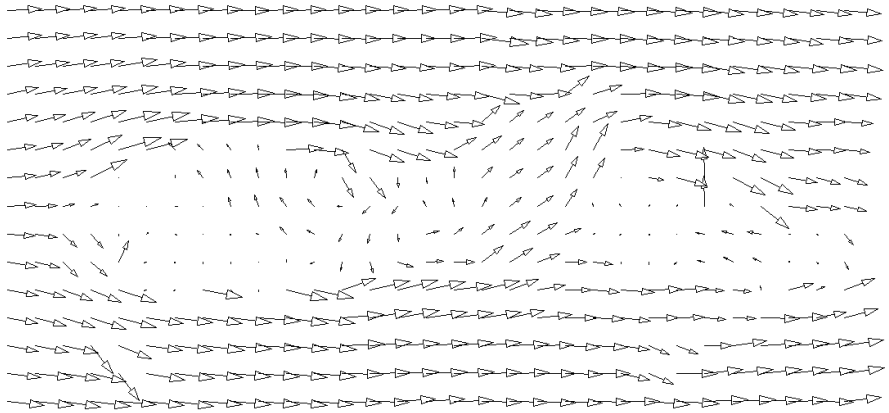
堀田らの実験画像間の平均変位は比較的小さく，実画像解析におけるハイブリッド法のダイナミックレンジについてはこの評価から議論することはできない．続く実画像を用いた解析では，比較的高いダイナミックレンジを必要とするケースにおけるハイブリッド相互相関 PIV の性能について議論をおこなう．

Table 5.2: Speed comparison between each PIV methods: This table shows hybrid method is the fastest of these three methods and about 1300 times faster processing compared to single-CPU processing is achieved.

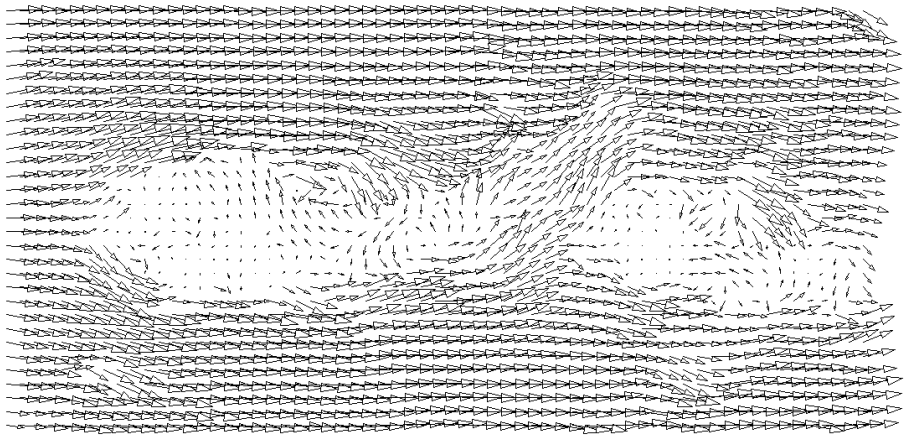
	Time cost[ms]	CPU/GPU[times]
Direct(GPU)	19.8	784.3
FFT-based(GPU)	14.2	1093.7
Hybrid(GPU)	12.2	1273
Direct(CPU)	15530	/



Recursion step 0
Interrogation window: 128^2 [pixel]

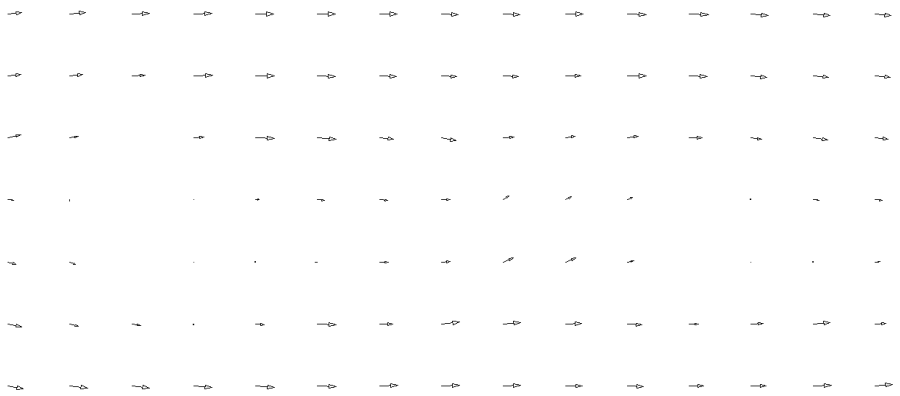


Recursion step 1
Interrogation window: 64^2 [pixel]

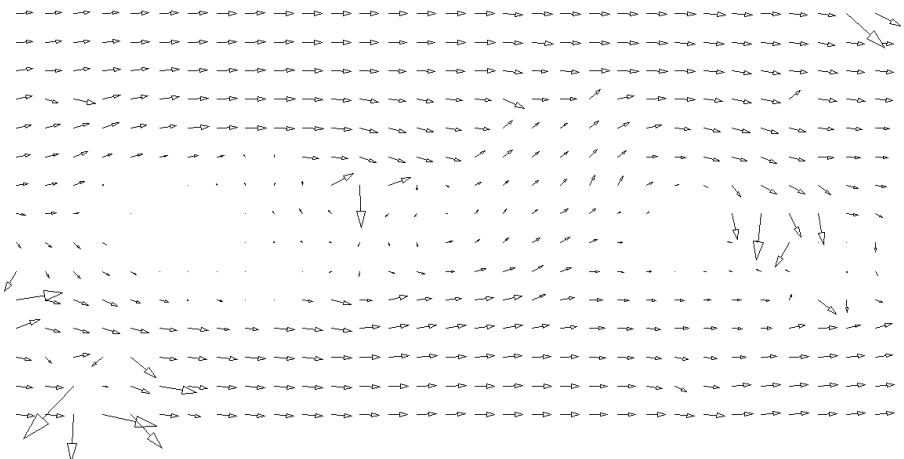


Recursion step 2
Interrogation window: 32^2 [pixel]

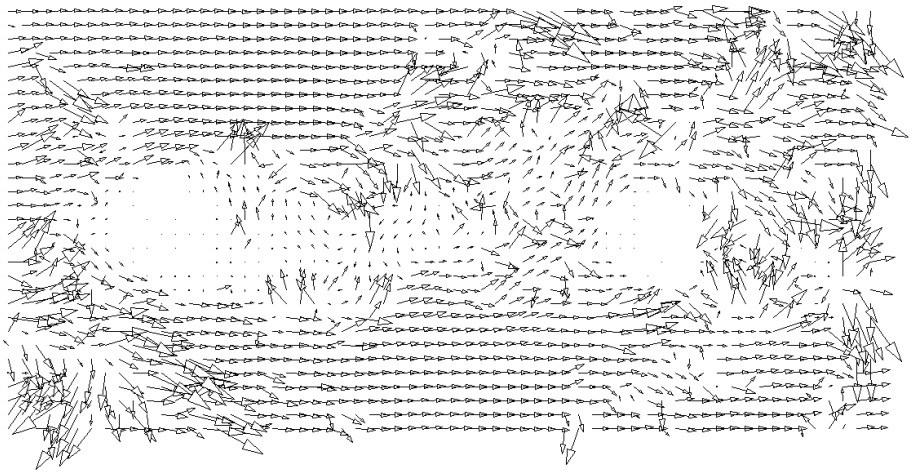
Fig 5.11: PIV result of Hotta's experiment in case of recursive direct cross-correlation PIV



Recursion step 0
Interrogation window: 128^2 [pixel]

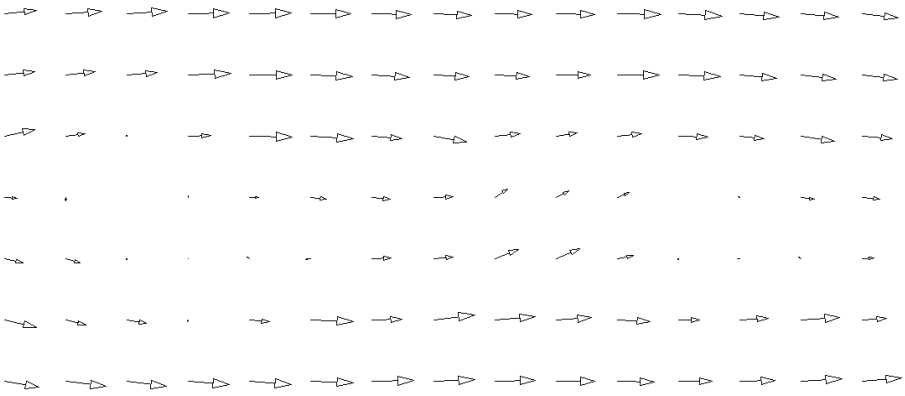


Recursion step 1
Interrogation window: 64^2 [pixel]

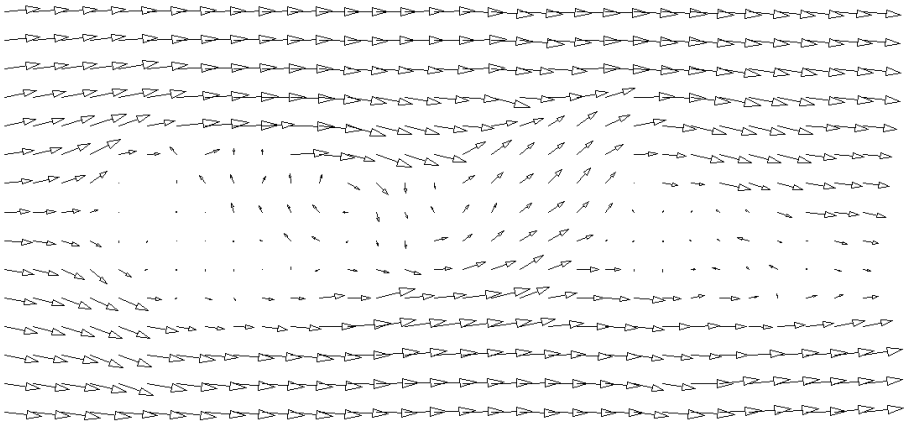


Recursion step 2
Interrogation window: 32^2 [pixel]

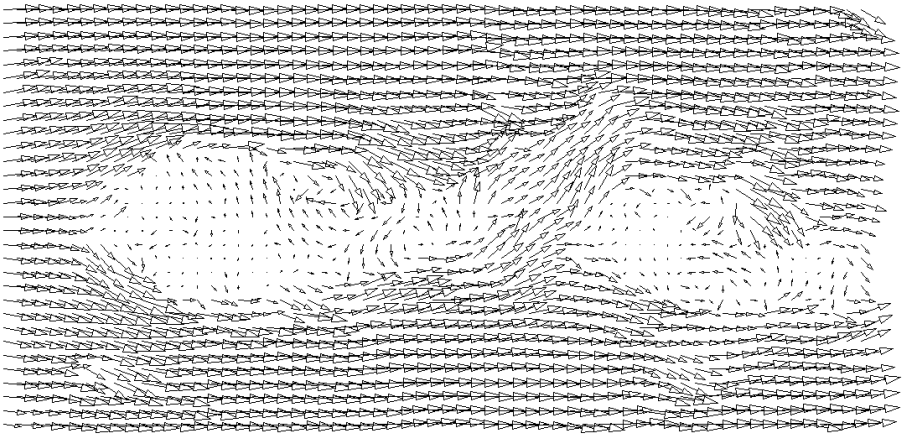
Fig 5.12: PIV result of Hotta's experiment in case of recursive FFT-based cross-correlation PIV



Recursion step 0
Interrogation window: 128^2 [pixel]

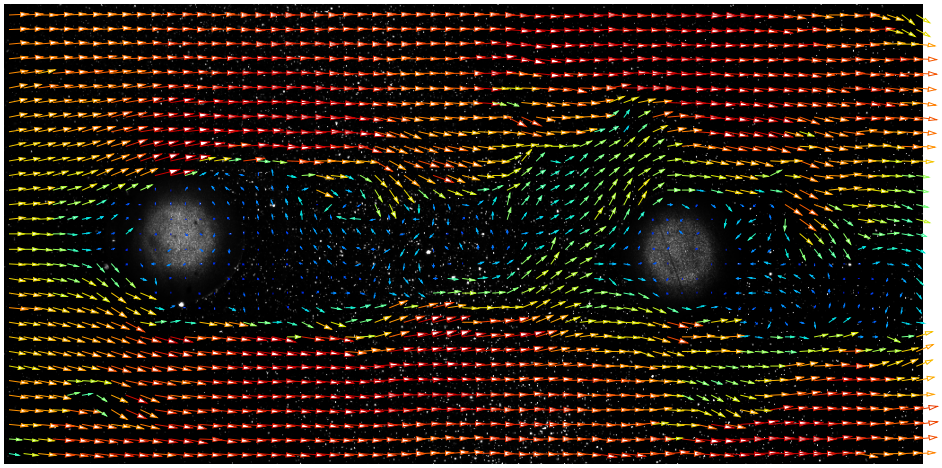


Recursion step 1
Interrogation window: 64^2 [pixel]

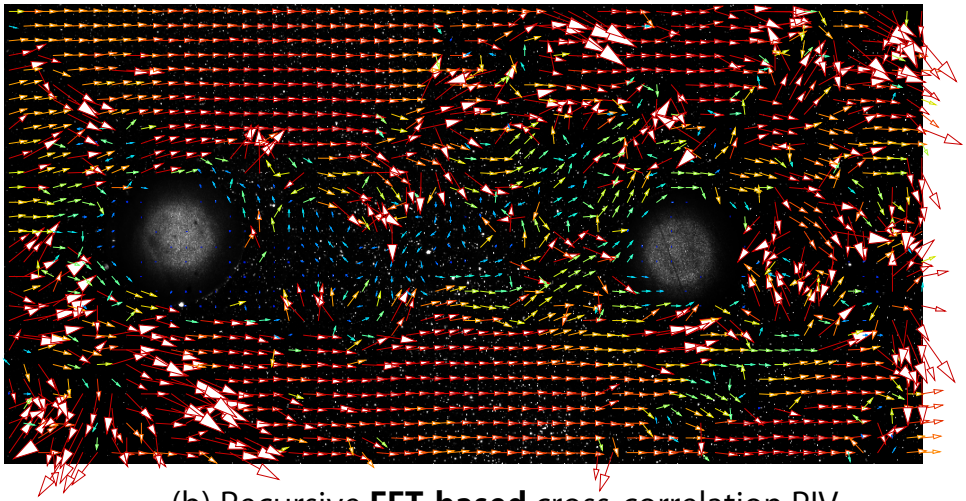


Recursion step 2
Interrogation window: 32^2 [pixel]

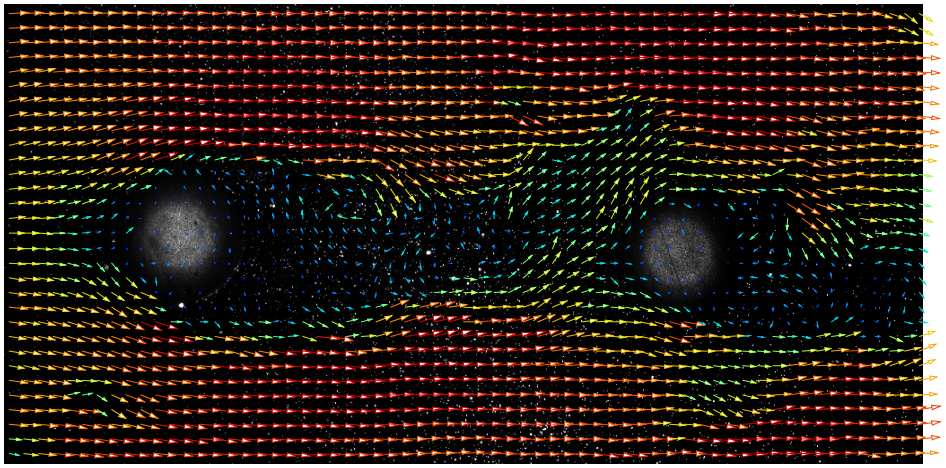
Fig 5.13: PIV result of Hotta's experiment in case of recursive hybrid cross-correlation PIV: Step 0 is resolved using FFT-based method and the following steps is done by direct method.



(a) Recursive **direct** cross-correlation PIV



(b) Recursive **FFT-based** cross-correlation PIV



(c) Recursive **Hybrid** cross-correlation PIV

Fig 5.14: Real image data overlapped by PIV result: These vectors are the same with the one of step 2 in Fig. 5.11, Fig. 5.12 and Fig. 5.13. The last interrogation window size is 32×32[pixel] and 1953 vectors are finally resolved.

5.3.3　照沼 (2011) の実験画像を用いた評価

照沼ら [57] は，微小流れ場における温度速度同時計測手法の開発の一環として，マイクロスケールの流れにも追従するような新たな粒子の開発および評価をおこなっている．その際照沼らは開発した粒子の流れに対する追従性を評価するため，PIV 処理を用いている．

Fig. 5.15 に，照沼らの実験で得られた画像の一例を示す．この画像は，照沼らによって開発された重合式感温粒子 [57] の流動性評価をおこなう過程で得られた画像である．Y 字チャネル下部の 2 つの分岐管から粒子を含んだ流体が流れ込み，これらが合流して右上部へと流れ出ている．第一画像と第二画像とで平均輝度値が大きく変化していることがわかるが，これは温度計測を同時におこなうため照明に燐光を使用していることに起因する．画像間の平均輝度値が異なっているという点で，速度検出用の PIV 画像としては比較的厳しいものであるといえる．

照沼らは様々な流速環境下で実験をおこなっている．本研究ではその中から，平均流速が中程度のもの (ケース a)) と，平均流速が高いもの (ケース b)) を用いてハイブリッド法の評価をおこなう．

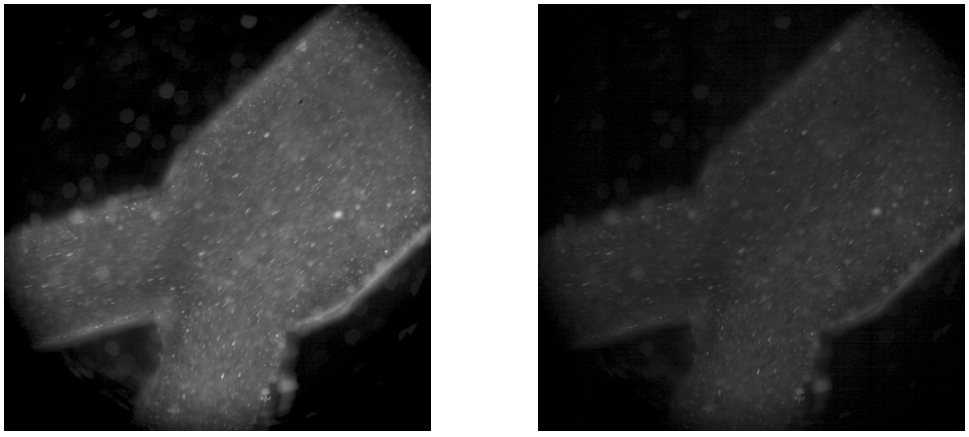


Fig 5.15: Sample image pair of Terunuma’s experiment

これらの画像ペアに対して今まで同様各 PIV アルゴリズムを適用したステップごとの結果を，ケース a) については Fig. 5.16〜Fig. 5.18 に，ケース b) については Fig. 5.19〜Fig. 5.21 にそれぞれ示す．また各ケースの処理時間はそれぞれ Table 5.3 および Table 5.4 に示されている．

ケース a) について，Fig. 5.16 の直接法による解析結果を見ると，上下の分岐管から流入してくる流れの速度は実際には異なっていることがわかる．そのため分岐がまとまった部分におい

ても，その上下で流速に違いがあることがわかる．FFT 法でも同様な考察は可能ではあるが，全体的に直接法にくらべ流速が低く算出されており，直接法に比べあいまいな可視化結果になっている．特に上側分岐管の端の部分では本来高い流速が検出されていない．ハイブリッド法を用いることで，最終的な可視化結果はより直接法に近づき，上下での速度差が FFT 法よりもはっきりと可視化できていることがわかる．また Table 5.3 から，ハイブリッド法の処理速度は他の両手法よりも高速であることも確認できる．しかし FFT 法で大きく結果が異なってしまっていた上側分岐管の端部分の速度は，ハイブリッド法でも検出できていないことがわかる．

ハイブリッド法においてこのような現象が生じるのは，初期ステップの FFT 法による PIV でピークが正しく検出できていないことに起因する．これは初期ステップで正しいピーク検出がおこなわれていることを仮定する再帰的手法全般に見られる現象であり，ハイブリッド法そのものに起因する現象ではない．完全ではないものの，FFT 法に比べより現象を明確に可視化する手段としてハイブリッド法は一定の効果があることがこの結果からわかる．

一方流速の大きいケース b) を直接法により解析した Fig. 5.19 からは，このケースにおける上下分岐管の速度差はあまりなく，分岐がまとまった管内中央部で流速が高くなっていることが見てとれるが，FFT 法ではこの流速が速い部分をほとんど検出できていない．そしてこのケース b) では，FFT 法の結果に基づくハイブリッド法でも流速の高い部分を検出することができなかった．これは画像間流速が高いため，初期ステップの FFT 法で検出が失敗した場合ハイブリッド法中の直接法で設定される小さな検査領域ではピーク位置をカバーできないためだと考えられる．流速があまり高くないケースではハイブリッド法による精度改善が見込めるが，流速が大きいケースではハイブリッド法であっても精度向上に寄与できないケースが存在することがこの結果からわかる．Table 5.4 を見ると本ケースでもハイブリッド法が他手法より高速であることがわかるが，少なくとも本ケースについては，解析結果を改善するために付加的な処理が必要であると考えられる．

Table 5.3: Speed comparison of each PIV method in case a)(moderate flow)

Interrogation widnow size[pixel]	128×128	$128 \times 128 \rightarrow 64 \times 64$	$128 \times 128 \rightarrow 32 \times 32$
Direct[ms]	51.3	56.8	61.2
FFT-based[ms]	11.9	20.9	30.4
Hybrid[ms]	11.9	16.4	21.5

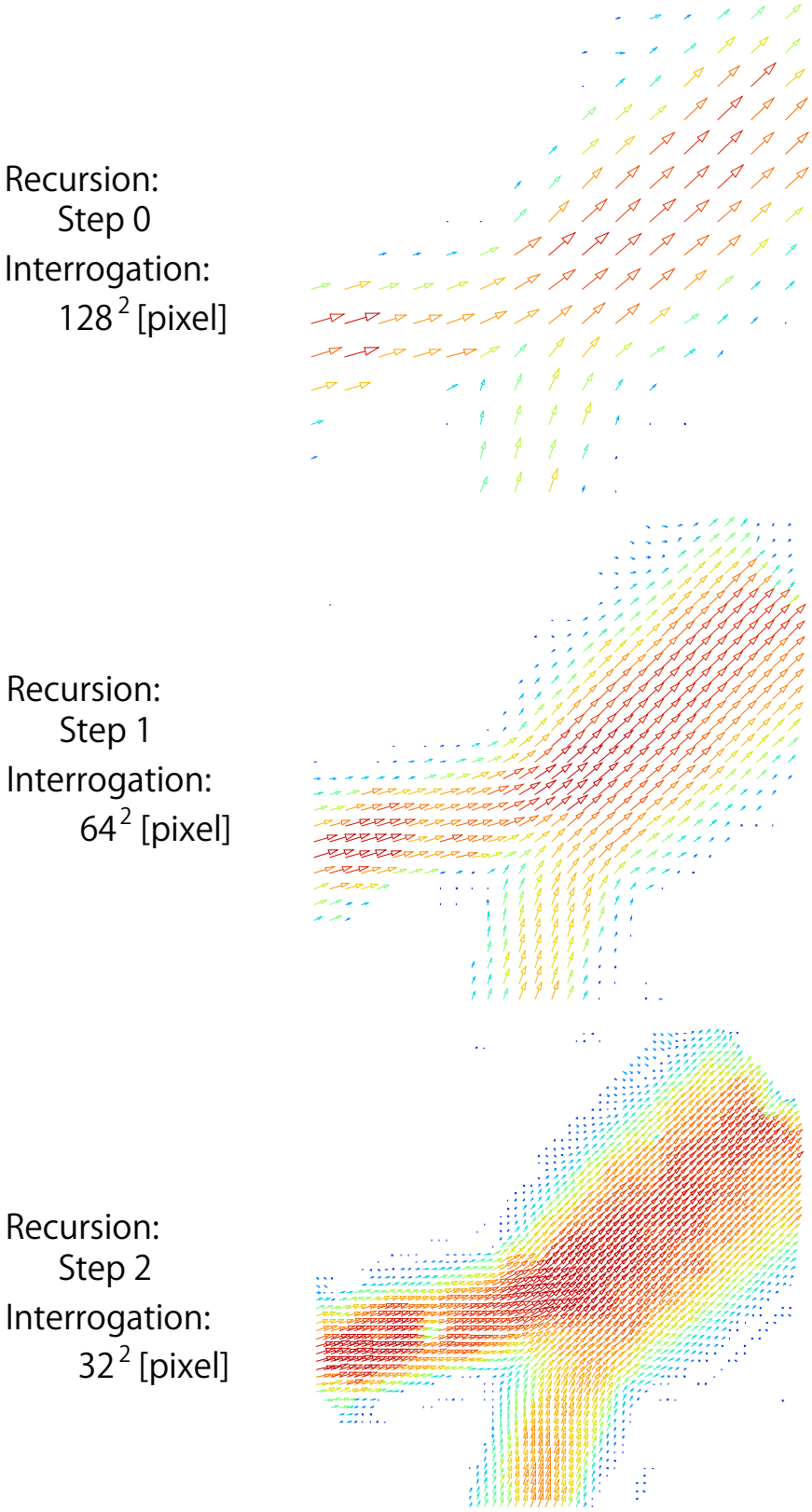


Fig 5.16: PIV result of case a)(moderate flow) in case of direct cross-correlation PIV

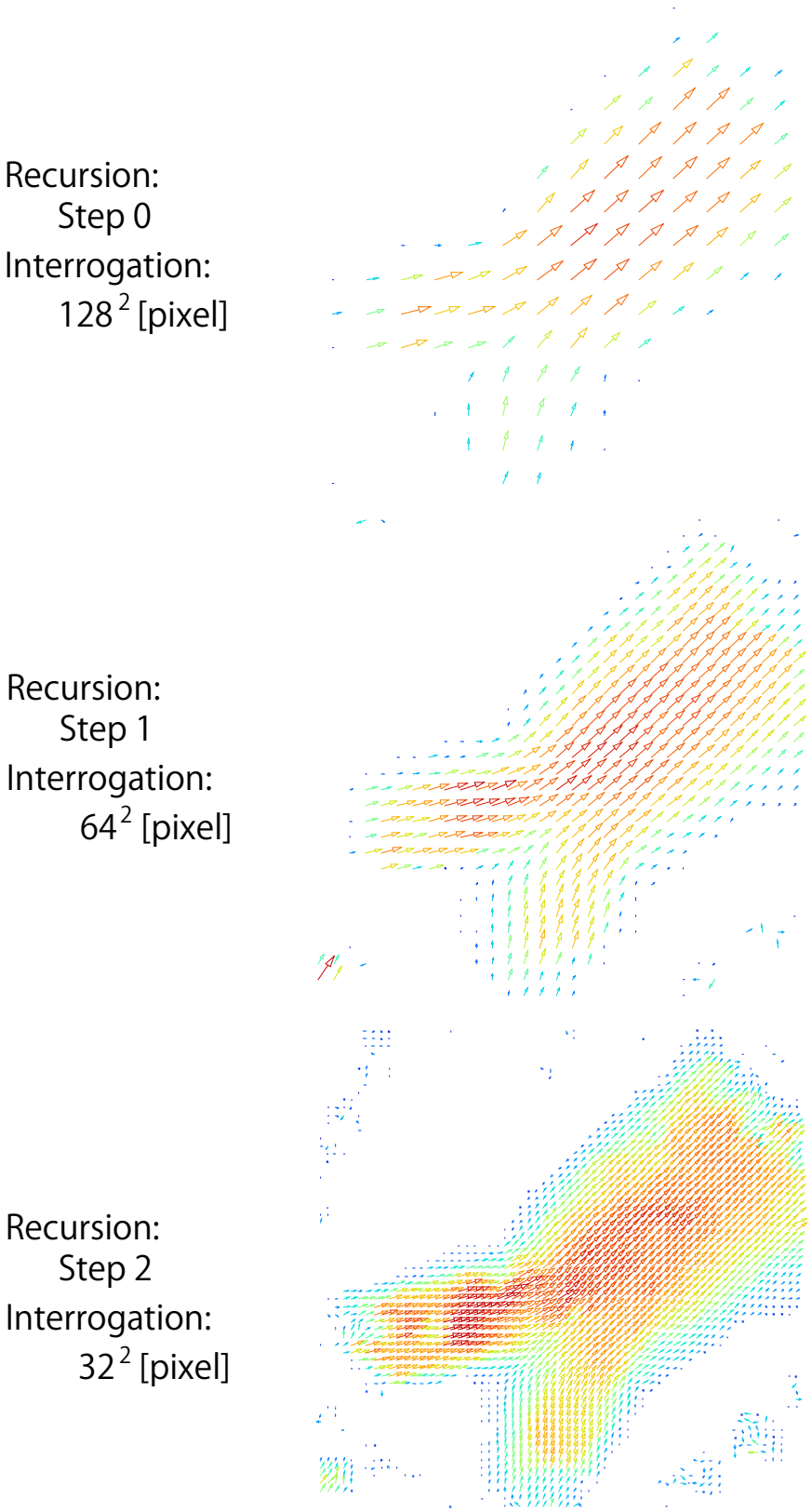


Fig 5.17: PIV result of case a)(moderate flow) in case of FFT-based cross-correlation PIV

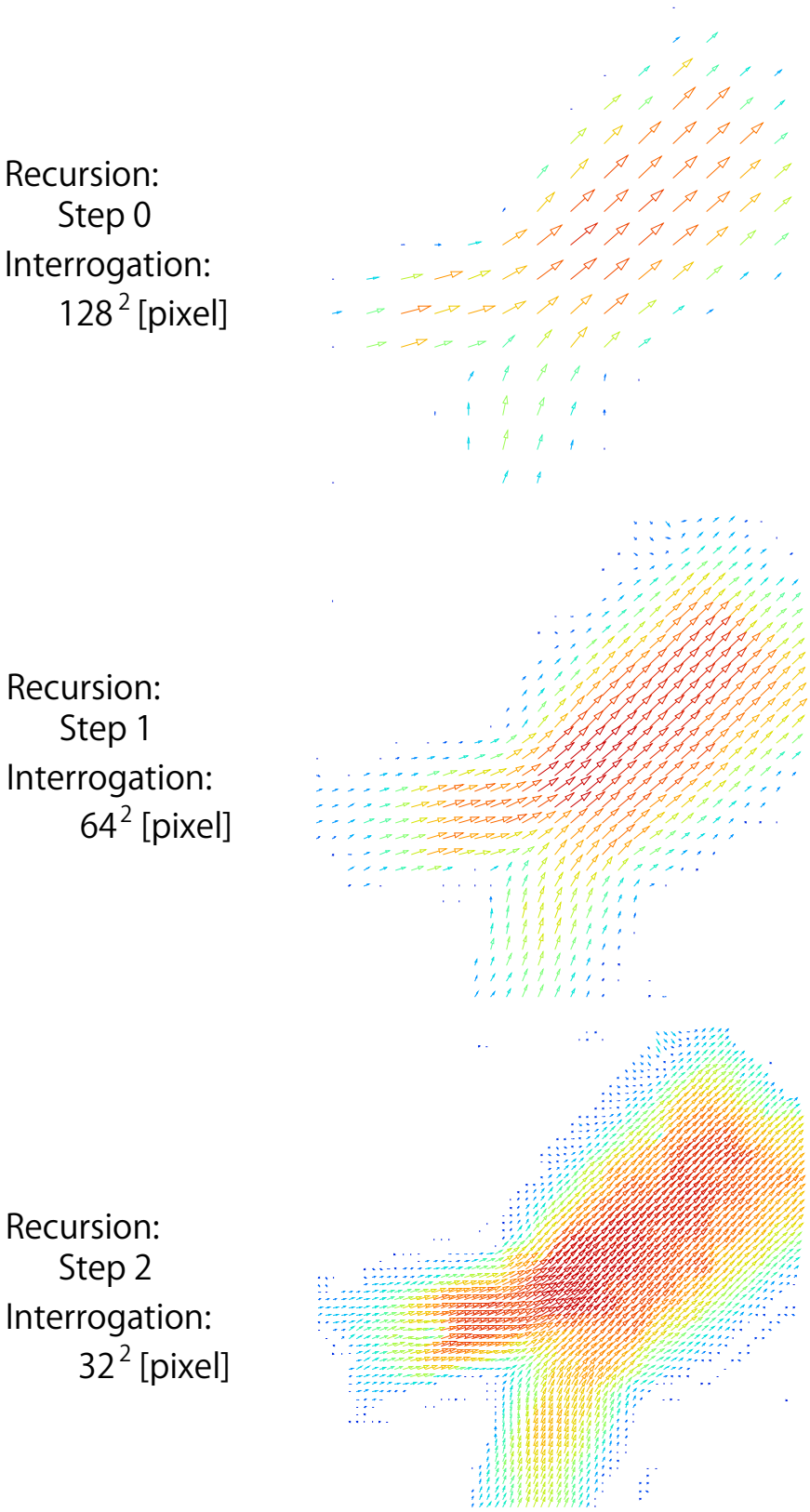


Fig 5.18: PIV result of case a)(moderate flow) in case of hybrid cross-correlation PIV

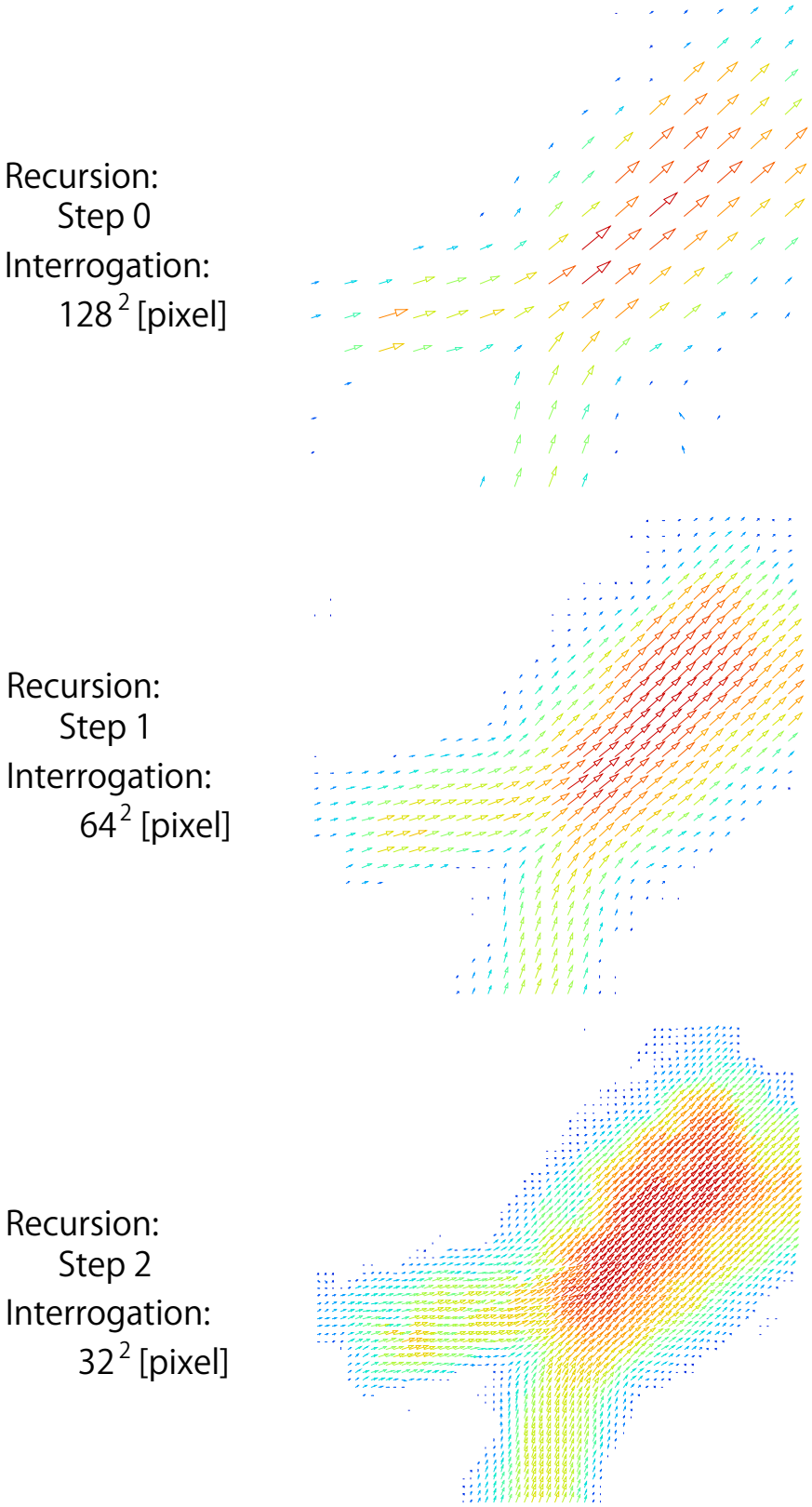


Fig 5.19: PIV result of case b)(fast flow) in case of direct cross-correlation PIV

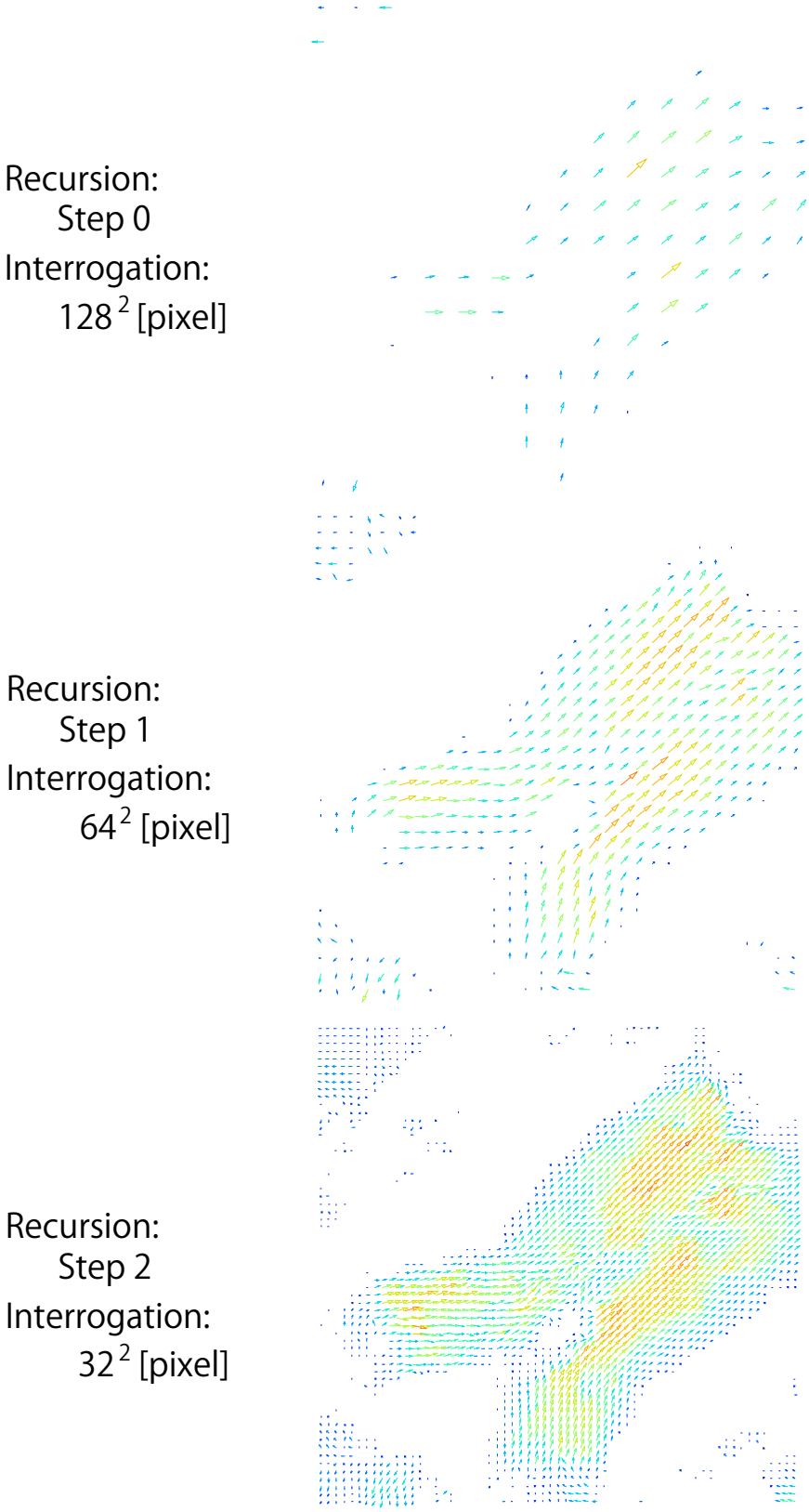


Fig 5.20: PIV result of case b)(fast flow) in case of FFT-based cross-correlation PIV

Fig 5.21: PIV result of case a)(fast flow) in case of hybrid cross-correlation PIV

Table 5.4: Speed comparison of each PIV method in case of b)(fast flow): in case of FFT-based and hybrid method, results are almost the same with Table. 5.3. In diredt method, time cost becomes bigger because the displacement is so large that search region also must be set large.

Interrogation widnow size[pixel]	128×128	$128 \times 128 \rightarrow 64 \times 64$	$128 \times 128 \rightarrow 32 \times 32$
Direct[ms]	188.0	193.4	198.0
FFT-based[ms]	12.2	21.8	31.0
Hybrid[ms]	12.2	17.5	22.7

5.4　本章のまとめ

本章ではまず，相互相関 PIV の計算手法として用いられる直接相互相関法と FFT 相互相関法の差異に着目した．一般に FFT による相互相関の計算は直接計算する手法に比べ高速であるとされるが，こと PIV に関しては必ずしもそれは正しくなく，本研究で定義した臨界探査領域を下回るサイズでは直接法が優位である，ということを両手法の計算量の比較から示した．実際の計算環境における臨界探査領域を本研究で開発した直接法と FFT 法の処理時間から見積もったところ，実際にそのような検査領域サイズが存在し，そのサイズは多くのケースで計算量のみの比較から得られた値より大きいことを確認した．実際に臨界探査領域サイズは，再帰的直接相互相関 PIV の最終ステップ等で利用するには十分なサイズであった．

この比較をふまえ，本研究はハイブリッド相互相関 PIV の提案をおこなった．この PIV は，ダイナミックレンジに大きく影響する再帰の初期段階では単位相互相関あたりの処理速度が速い FFT 法を用い，ダイナミックレンジにほぼ影響しない終盤の再帰ステップでは高精度かつロバストな直接法を用いるアルゴリズムである．人工画像および実画像を用いた評価をおこなった結果，このハイブリッド法が FFT 法と同等の処理速度で広いレンジのベクトルを検出可能であること，直接法と同水準の精度およびロバスト性をもっていること，そして両手法よりも更に高速な処理が可能であることを確認した．ハイブリッド法は必ずしも両手法よりも高速なわけではなく，その処理速度は内部の直接法で設定される探査領域サイズに影響する．本研究で定義した臨界探査領域よりも小さな探査領域を設定することで，ハイブリッド法の処理速度は両手法よりも更に高速であることが保証されることを明らかにした．しかし初期の FFT 法によるステップでベクトル検出に失敗してしまうようなケースでは，ハイブリッド法を用いてもその精度は向上しないことが確認された．ハイブリッド法にはたとえば FFT 法と直接法との切り替えのタイミングなど，いくつかの点で検討の余地があり，本評価で確認されたハイブリッド

法の問題点と合わせて，今後更に検討をおこなう必要がある．

第6章　結論

本研究の目的は，実用的かつ高速処理可能な PIV 処理システムの開発すること，そしてその高速化手段としての GPU の有用性を検証することであった．本研究から得られた知見は，以下の 3 点にまとめられる．

(1) 本研究では相互相関 PIV 処理の高速化手段として GPU に着目し，GPU の性能を極力活かすかたちで PIV 処理の実装をおこなった結果，PIV 画像処理はミリセカンドオーダーで実行可能となり，シングル CPU による処理と比較すると約 800 倍の高速化を実現することができた．GPU を用いた高速化にはデータ転送等の観点から限界があるものの，本研究で開発した PIV はいまだその処理速度に達してはおらず，よって更なる高性能 GPU が使用可能となれば，GPU によって PIV は今以上に高速に処理が実現する可能性が高い．

したがって本研究から，GPU は相互相関 PIV をミリセカンドオーダー高速処理するための有効な手段であること，限界は存在しうるものの今後も PIV の更なる高速化に寄与しうるものであることが明らかになった．

(2) 一方で，相互相関 PIV を高速化する手段としてのマルチ GPU の有用性は限定的であった．シングル GPU による PIV 画像処理の時間コストがある程度高いケースではその有用性が確認できたものの，シングル GPU 並列計算で既に非常に高速な処理を更に高速化する手段としてのマルチ GPU の有用性を確認することはできなかった．これはマルチ GPU が扱う問題として PIV 画像処理が非常に小さく，マルチ GPU がもつデータ転送のネックが顕在化したためだと考えられる．

よって本研究から，シングル GPU での時間コストが比較的高い PIV 画像処理を高速化する手段としてマルチ GPU は有用である一方，時間コストが小さい PIV を更に高速化する手段としてはマルチ GPU はあまり有効ではないということが明らかになった．

(3) 本研究で提案したハイブリッド相互相関法は，FFT 相互相関法と同等のダイナミックレンジを高速に実現し，直接相互相関法と同等の精度とロバスト性を有し，臨界探査領域よりも小さ

い探査領域を設定した場合両手法より更に高速な処理を可能にする手法であることが，人工画像および実画像の解析結果から示された．ハイブリッド法を用いることで 1024×1024[pixel] の画像ペアを約 50Hz の速度で実行することが可能となり，シングル CPU による PIV 画像処理の時間と比較するとその高速化は約 1300 倍にものぼることが確認された．本研究では再帰的 PIV への応用のみにとどまったが，ハイブリッド法は FFT 法の精度を向上させる画像変形ステップ等への利用など，その適用分野は本研究が示した以上に広範であることが予想できる．

本研究は，ハイブリッド相互相関法が精度・ピーク検出レンジ・処理速度いずれの観点からも有効な手法であり，その応用分野は今後更に拡がる可能性があることを明らかにした．

付録A　本研究で用いた計算環境

本研究で用いた計算環境について以下に示す.

Table 6.1: Computational environment used in this study

	計算環境-I	計算環境-Ⅱ
CPU	Intel Core i7 920(2.66GHz)	Core i7 920(2.66GHz)
GPU	Geforce GTX480	Geforce GTX 285 × 4
bus	PCI express x16	PCI express 2.0 x16 ×4

また，使用した GPU のスペックを以下に示す.

Table 6.2: Spec of GPU used in this study

	GTX 480	GTX 285
Stream processor	480	240
CUDA Core	15	30
Core clock	1.4GHz	1.73GHz
Memory bandwidth[GB/s]	177.4GB/s	159.0GB/s
Peak performance	1345.0GFLOPS	708.5GFLOPS
video memory	1.5GB	1.0GB
Shared memory / multiprocessor	48MB/16MB	16MB
L1 cache for global memory / multiprocessor	16MB/48MB	None
Texture cache for global memory / multiprocessor	4MB～6MB	4MB～8MB
Registers / multiprocessor		8MB
Shared memory banks	32	16
Warp size	32	32
Maximum number of threads per block	512	1024
Maximum number of resident threads per multiprocessor	1024	1536

参考文献

[1] AMD: AMD Radeon HD 6970. http://www.amd.com/jp/products/desktop/graphics/amd-radeon-hd-6000/hd-6970/Pages/amd-radeon-hd-6970-overview.aspx.

[2] NVIDIA Corporation: Geforce GTX 580. http://www.nvidia.co.jp/object/product-geforce-gtx-580-jp.html.

[3] R J Adrian. Dynamic ranges of velocity and spatial resolution of particle image velocimetry. *Meas. Sci. Technol.*, Vol. 8, pp. 1393–1398, 1997.

[4] Ronald J. Adrian. Particle-imaging techniques for experimental fluid mechanics. *Annu. Rev. Fluid Mech*, Vol. 23, pp. 261–304, 1991.

[5] T. Astarita. Analysis of interpolation schemes for image deformation methods in piv: effect of noise on the accuracy and spatial resolution. *Experiments in Fluids*, Vol. 40, pp. 977–987, 2006.

[6] T. Astarita and G. Cardone. Analysis of interpolation schemes for image deformation methods in piv. *Experiments in Fluids*, Vol. 38, pp. 233–243, 2005.

[7] F. Champagnat, A. Plyer, G. Le Besnerais, B. Leclaire, and Y. Le Sant. How to calculate dense piv vector fields at video rate. In *PIV09- 8th Int. Symp. on Particle Image Velocimetry*, Melbourne, Victoria, Australia, August 2009.

[8] Kelly Cohen, Thomas E. McLaughlin, James Myatt, and Stefan Siegel. Real-time particle image velocimetry for closed-loop flow control studies. In *41st Aerospace Sciences Meeting and Exhibit*, No. 2003-0920 in AIAA, Reno, Nevada, January 2003.

[9] G. E. Elsinga, F. Scarano, B. Wieneke, and B. W. van Oudheusden. Tomographic particle image velocimetry. *Exp Fluids*, Vol. 41, pp. 933–947, 2006.

[10] T Fujiwara, K Fujimoto, and T Maruyama. A real-time visualization system for piv. In P.Y.K. Cheung et al., editor, *FPL 2003, LNCS 2778,*, pp. pp. 437–447, 2003.

[11] Naga K. Govindaraju, Brandon Lloyd, Yuri Dotsenko, Burton Smith, and John Manferdelli. High performance discrete fourier transforms on graphics processors. In *Proc. ACM/IEEE Conf. on Supercomputing, pp. 1-12, IEEE Press, Piscataway, NJ*, 2008.

[12] D. P. Hart. High-speed piv analysis using compressed image correlation. *Journal of Fluids Engineering*, Vol. 120, pp. 463–470, 1998.

[13] D. P. Hart. Piv error correction. *Experiments in Fluids*, Vol. 29, pp. pp. 13–22, 2000.

[14] Douglas P. Hart. Super-resolution piv by recursive local-correlation. *Journal of Visualization*, Vol. 10, , 1999.

[15] K Jambunathan, X Y Ju, B N Dobbins, and S Ashforth-Frost. An improved cross correlation technique for particle image velocimetry. *Meas. Sci. Technol.*, Vol. 6, pp. 507–514, 1995.

[16] P. Kondratieva, J. Georgii, R. Westermann, H. Petermeier, W. Kowalczyk, and A. Delgado. A real-time model-based approach for the reconstruction of fluid flows induced by microorganisms. *Exp Fluids*, Vol. 45, pp. pp. 203–222, 2008.

[17] B. Leclaire, B. Jaubert, F. Champagnat, G. Le Besnerais, and Y. Le Sant. Folki-3c: a simple, fast and direct algorithm for stereo piv. In *PIV09- 8th Int. Symp. on Particle Image Velocimetry*, Melbourne, Victoria, Australia, August 2009.

[18] V W Lee and et al. Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu. ISCA ’10, Saint-Malo, France., June 2010.

[19] M Leeser, S Miller, and H Yu. Smart camera based on reconfigurable hardware enables diverse real-time applications. the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM ’04), 2004.

[20] J. P. Lewis. Fast normalized cross-correlation. In *Vision Interface*, pp. 120–123, 1995.

[21] D. Brandon Lloyd, Chas Boyd, and Naga Govindaraju. Fast computation of general fourier transforms on gpus. In *2008 IEEE International Conference on Multimedia and Expo (ICME2008)*, pp. 5–8, 2008.

[22] C. V. Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial Mathematics, 1992.

[23] T Maruyama, Y Yamaguchi, and A Kawase. An approach to real-time visualization of piv method with fpga. In G. Brebner and R. Woods, editors, *FPL2001, LNCS 2147*,, pp. 601–606, 2001.

[24] Kenneth Moreland and Edward Angel. The fft on a gpu. In *Graphics Hardware*, 2003.

[25] H. Nobach and C. Tropea. Improvements to piv image analysis by recognizing the velocity gradients. *Experiments in Fluids*, Vol. 39, pp. 612–620, 2005.

[26] Nvidia Corporation. *CUDA CUFFT Library Version 3.1*, May 2010.

[27] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide Version 3.1*, May 2010.

[28] NVIDIA Corporation. *Tuning CUDA Applications for Fermi*, 2010.

[29] Yasuhiko Ogata, Toshio Endo, and Satoshi Matsuoka. Cpu および gpu を併用する fft ライブラリの提案と評価. 情報処理学会研究報告 ハイパフォーマンスコンピューティング, Vol. 80, pp. 13–18, 2007.

[30] K. Okamoto. Dynamic piv: a strong tool to resolve the unsteady phenomena. In *Particle Image Velocimetry: Recent Improvements*. Springer, 2003.

[31] K Okamoto, S Nishio, T Saga, and T Kobayashi. Standard images for particle image velocimetry. *Meas. Sci. Technol.*, Vol. 11, pp. 685–691, 2000.

[32] A K Prasad. Stereoscopic particle image velocimetry. *Exp Fluids*, Vol. 29, pp. 103–116, 2000.

[33] Oliver Pust. Piv: Direct cross-correlation compared with fft-based cross-correlation. In *the 10th International Symposium on Applications of Laser Techniques to Fluid Mechanics*, 2000.

[34] M Raffel, C Willert, and J Kompenhans. *Prarticle Image Velocimetry -A Practical Guide*. Springer, 1998.

[35] P. Ruhnau, T. Kohlberger, C. Schnorr, and H. Nobach. Variational optical flow estimation for particle image velocimetry. *Experiments in Fluids*, Vol. 38, pp. 21–32, 2005.

[36] F Scarano. Iterative image deformation methods in piv. *Meas. Sci. Technol.*, Vol. 13, pp. R1–R19, 2002.

[37] F. Scarano and M. L. Riethmuller. Iterative multigrid approach in piv image processing with discrete window offset. *Experiments in Fluids*, Vol. 26, pp. 513–523, 1999.

[38] Thomas Schiwietz and Rudiger Westermann. Gpu-piv. In *Vision, Modeling and Visualization 2004*, 2004.

[39] Utami T, Blackwelder RF, and Ueno T. A cross-correlation technique for velocity field extraction from particulate visualization. *Experiments in Fluids*, Vol. 10, pp. 213–223, 1991.

[40] S. Tarashima, S. Someya, and K. Okamoto. Acceleration of recursive cross-correlation piv using multiple gpus. In *AJTEC 2011- 8th ASME-JSME Thermal Engineering Joint Conference*, 2011.

[41] S. Tarashima, M. Tange, S. Someya, and K. Okamoto. Gpu accelerated direct cross-correlation piv with window deformation. In *15th Int Symp on Applications of Laser Techniques to Fluid Mechanics*, Lisbon, Portugal, July 2010.

[42] R Theunissen, F Scarano, and M L Riethmuller. An adaptive sampling and windowing interrogation method in piv. *Meas. Sci. Technol.*, Vol. 18, pp. 275–287, 2007.

[43] P. Thevenaz, T. Blu, and M. Unser. Image interpolation and resampling. In I.N. Bankman, editor, *Handbook of Medical Imaging, Processing and Analysis*, pp. 393–420. Academic Press, San Diego Ca, USA, 2000.

[44] S Venkatasubramanian. The Graphics Card as a Stream Computer. In *SIGMOD-DIMACS Workshop on Management and Processing of Data Streams*, 2003.

[45] Vivek Venugopal, Cameron D. Patterson, and Kevin Shinpaugh. Accelerating particle image velocimetry using hybrid architectures. In *Symposium on Application Accelerators in High Performance Computing*, 2009.

[46] J. Westerweel. Efficient detection of spurious vectors in particle image velocimetry data. *Experiments in Fluids*, Vol. 16, pp. 236–247, 1994.

[47] J Westerweel. Fundamentals of digital particle image velocimetry. *Meas. Sci. Technol.*, Vol. 8, pp. 1379–1392, 1997.

[48] J. Westerweel, D. Dabiri, and M. Gharib. The effect of a discrete window offset on the accuracy of cross-correlation analysis of digital piv recordings. *Experiments in Fluids*, Vol. 23, pp. 20–28, 1997.

[49] C E. Willert. Adaptive piv processing based on ensemble correlation. In *14th Int Symp on Applications of Laser Techniques to Fluid Mechanics*, Lisbon, Portugal, July 2008.

[50] C.E. Willert and M. Gharib. Digital particle image velocimetry. *Experiments in Fluids*, Vol. 10, pp. 181–193, 1991.

[51] Christian E. Willert, Matthew J. Munson, and Morteza Gharib. Real-time particle image velocimetry for closed-loop flow control applications. In *15th Int Symp on Applications of Laser Techniques to Fluid Mechanics*, 2010.

[52] Jae-Chern Yoo and Tae Hee Han. Fast normalized cross-correlation. *Circuits, Systems and Signal Processing*, Vol. 28, pp. 819–843, 2009.

[53] Haiqian Yu, Miriam Leeser, Gilead Tadmor, and Stefan Siegel. Real-time particle image velocimetry for feedback loops using fpga implementation. *Journal of Aerospace Computing, Information, and Communication*, Vol. 3, pp. 52–62, 2006.

[54] 可視化情報学会（編）. PIV ハンドブック. 森北出版株式会社, 2002.

[55] 額田彰, 松岡聡. Cuda gpu 向けの自動最適化 fft ライブラリ. 情報処理学会論文誌コンピューティングシステム (ACS). Vol. 2, No. 3, pp. 107–115. 情報処理学会. Sep. 2009., Vol. 2, No. 3, pp. 107–115, September 2009.

[56] 高橋大介. Fft におけるキャッシュ最適化方式. スーパーコンピューティングニュース, Vol. 9, pp. 123–134, 2008.

[57] 照沼勇人. 微小流れ場における温度ならびに速度同時計測手法の開発. Master’s thesis, 東京大学大学院, 2011.

[58] 青木尊之, 額田彰. はじめての CUDA プログラミング. 工学社, 2009.

[59] 堀田周作.　類似の振動特性を持つ直列 2 円柱構造物の流体関連振動現象に関する研究. Master’s thesis, 東京大学大学院, 2011.

謝辞

本研究をすすめるにあたり，本当に多くの方々にご協力いただきました．この場を借りて厚くお礼申し上げます．まことにありがとうございました．

岡本先生には研究テーマの設定から方向性の指導まで，研究に関するほぼ全てのことで本当に御世話になりました．学部 4 年生を含めた計 3 年間この研究室に所属させていただき，いろいろなことに取り組みまた学ぶことができたのは，ひとえに岡本先生がその環境を提供してくださったからです．本当にありがとうございました．

染矢先生には研究の進め方などで様々なアドバイスやフォローをしていただきました．どんな些細なことでも相談に乗ってくださり，真摯にアドバイスをしていただけてとても心強かったです．先生と二人旅だったリスボンの学会も僕個人としては大変楽しく，また勉強になりました．本当にありがとうございました．

また研究室の同期の皆，秘書の方々，先輩方，後輩の皆さん，にも大変お世話になりました．研究を修士論文としてまとめられたのはもちろんのこと，この 3 年間を楽しく有意義に過ごすことができたのは皆さまのおかげです．まことにありがとうございました．

そして最後にいつも影で支えてれた家族にも感謝の意を表したいと思います．本当にありがとうございました．

平成 23 年 2 月 14 日

田良島 周平