修 士 論 文

勾配法を用いた
オプティカルフロー
高速検出ハードウェアシステムの構築

北陸先端科学技術大学院大学
情報科学研究科　情報システム学専攻

戸田 吉伸

2004 年 3 月

修 士 論 文

勾配法を用いた
オプティカルフロー
高速検出ハードウェアシステムの構築

指導教官　堀口 進 教授

審査委員主査　堀口 進 教授
審査委員　Shen Hong 教授
審査委員　日比野 靖 教授

北陸先端科学技術大学院大学
情報科学研究科　情報システム学専攻

210062 戸田 吉伸

提出年月: 2004 年 2 月

概 要

近年のコンピュータ技術の発展はめざましく，動画像を解析的に処理することが可能になってきた．同様に，撮像素子や画像センサ等の技術も進歩し，解像度の高い動画像が取得できるようになってきており，そのような高精細な動画像をリアルタイムに処理する要求が高まっている．例えば，動画像中の移動物体の検出処理や，奥行き情報に基づく 3 次元位置の認識処理等が挙げられる．これらの処理はロボットの視覚認識システム等で重要なものであり，高精度で高速な処理が求められている．画像処理においてフレームレートの高い動画像を用いることは，連続するフレーム間で移動物体の動きが小さくなるという利点があり，物体追跡などにおいて有効性が示されている．

時間的に連続する画像フレームから画像中の移動物体を解析する手法として，画像上の画素ごとにおける動きベクトルを求める，オプティカルフロー推定法が有効である．オプティカルフローの推定は，動画像全体から見かけ上の速度ベクトル場を抽出する処理であり，これにより非常に詳細な動きの情報を取得できる．そのため，移動物体の検出，動きを基にした領域分割，3 次元的な動き・構造の解析など，ダイナミックシーン解析における様々な問題に利用されている．しかし，その反面，計算量が非常に膨大であるため，リアルタイムで処理を行うためにはハードウェアによる高速化が必要である．特に小さな動きや動いている対象物の形状を検出可能にするためには，密度の高いオプティカルフロー生成が高速で行えるシステムが重要となる．

これまでにも，オプティカルフローの専用処理ハードウェアはいくつか研究されてきた．例えば，256 プロセッシングエレメントからなる一次元アレイで探索範囲を制限しながらマッチングを行うことにより，マッチング法を高速化するハードウェアが開発されている．しかし，テンプレートマッチングでブロックの対応付けを行うこの方法は，拡大・縮小・回転などの複雑な運動に対して弱いという問題がある．信頼性の高いフローベクトルを計算可能な手法である空間的局所最適化法については，パイプラインイメージプロセッサを用いて，高速に処理するハードウェアが研究されている．しかし，252×316 pixels の画像を処理するのに 47.8ms 必要で，高精細な画像を高速に処理しているとはいえない．

本研究では，空間的局所最適化法を高速に計算する方法を提案し，そのハードウェアシステムを設計する．具体的には，空間的局所最適化法で行う局所領域内の重み付き局所和の計算を，列毎の和に分解することで効率の良い計算結果の再利用を可能にする方法を提案し，オプティカルフローの演算がパイプライン的に処理できることを示す．そして，提案方法に基づいたハードウェア回路を設計し，このハードウェアの有効性について検証する．

目 次

第1章　はじめに

1.1　研究の背景と目的

近年のコンピュータ技術の発展はめざましく，計算能力の高い処理装置が非常に安価になってきており，動画像処理といった高い計算能力を必要とする処理ですら，個人が所有するレベルのコンピュータで実行することが可能になってきている．また，デジタルカメラの急速な普及により，CCD に代表される撮像素子などの画像センサ技術も大幅に進歩し，解像度の高い動画像を容易に取得できるようになってきている．このような状況において，動画像処理に対する高解像度化，リアルタイム処理化の要求が高まっている．動画像処理の例としては，動画像中の移動物体の検出処理や，奥行き情報に基づく 3 次元位置の認識処理等が挙げられる．これらの処理はロボットの視覚認識システム等で重要なものであり，高精度で高速な処理が求められている．また，動画像処理においてフレームレートの高い動画像を用いることは，連続するフレーム間で移動物体の動きが小さくなるという利点があり，特に物体追跡などにおいて有効性が示されている．

動画像中から移動物体を解析する手法として，画像上の画素ごとにおける動きベクトルを求めるオプティカルフロー推定法が広く使用されている．オプティカルフローとは，動画像中の見かけの速度ベクトル場のことであり，非常に詳細な動きの情報を取得できる．そのため，移動物体の検出，動きを基にした領域分割，3 次元的な動き・構造の解析など，ダイナミックシーン解析における様々な問題に利用されている．しかし，その反面，オプティカルフローを求める処理は計算量が非常に膨大であるため，ソフトウェアによるリアルタイム処理は難しい．従って，高解像度・高フレームレートの動画像を扱うには，ハードウェアによる高速化が必須である．特に物体の小さな動きの検出や，動いている対象物の形状を検出可能にするためには，密度の高いオプティカルフロー生成を高速に行えるシステムが重要になってくる．

これまでにも，オプティカルフロー推定法を高速に処理する専用のハードウェアはいくつか開発されている．文献 [4] のシステムは，オプティカルフロー推定法の一手法であるブロックマッチング法に基づく手法をハードウェア化したものである．256PE からなる一次元アレイで探索範囲を制限しながらマッチングを行うことにより，時間的に隣接するフレーム間でブロックの対応付けを行う．しかし，テンプレートマッチングでブロックの対応付けを行うこの方法は，拡大・縮小・回転などの複雑な運動に対して，フローの推定精度が低くなる問題がある．信頼性の高いフローベクトルを計算可能な手法である [3] 空間的局所最適化法 [2] については，パイプラインイメージプロセッサを用いて高速に計算す

るハードウェアが研究されている [5]．しかし，252×316 pixels の画像からオプティカルフローを計算するのに 47.8ms 必要で，高解像度・高フレームレートの画像をリアルタイムに処理することはできない．

本研究では，最も信頼性の高いフローベクトルを計算できる手法である，空間的局所最適化法を用いて高速にオプティカルフローを検出するハードウェアシステムを設計し，そのシステムの評価を行う．さらに，局所領域はフレーム全体から求められるので，膨大な量の繰り返し演算になる．そこで，まず複数の局所領域間で領域が重なり合うことに着目し，重み付き局所和を列毎の和に分解することで，重なり合う局所領域間で重み付き局所和の一部を再利用する手法を提案し，その手法を用いることでオプティカルフローの演算がパイプライン的に処理できることを示す．さらに，提案方法に基づいたハードウェア回路を設計し，このハードウェアの有効性について示す．

1.2 本論文の構成

本論文の構成は次の通りである．

第 2 章では，動画像中の移動物体検出における代表的手法について説明し，オプティカルフローの有効性を示す．次にオプティカルフローの計算手法について説明し，空間的局所最適化法を詳しく説明する．

第 3 章では，空間的局所最適化法を計算するプロトタイプハードウェアシステムを設計し，空間的局所最適化法のハードウェア化による計算速度の評価を行う．

第 4 章では，局所領域の重み付き局所和計算において，列和を用いた計算結果再利用法を提案し，ハードウェアによる実装法について述べる．

第 5 章では，提案した手法に基づいて設計したハードウェアの評価を行う．

第 6 章で，本研究のまとめを述べる．

第2章　移動物体の検出法

2.1　はじめに

従来，動画像中から移動物体を検出する研究は様々行われてきた．移動物体の領域検出の代表的な手法として，背景差分法やフレーム間差分法などが挙げられるが，これらの手法を用いることで，各フレーム内での移動物体の存在領域をしることができる．しかし，時間的に隣接するフレーム間においてその移動物体が同一であることを知るためには，移動物体のトラッキングが必要となる．移動物体のトラッキングができれば，その移動物体の動きの軌跡がわかるばかりか，差分を算出することにより速度も得られ，次の瞬間にその物体がどの位置に移動するかを予測することも可能である．移動物体のトラッキングに用いる代表的な手法としては，オプティカルフロー推定法が挙げられる．

本章では，まず移動物体の代表的な移動物体検出法である背景差分法とフレーム間差分法について説明する．次に，本研究で取り扱う空間的局所最適化法を詳しく説明し，その長所を述べる．

2.2　背景差分法

背景差分を用いた手法は，撮影系の移動物体が存在しない画像と背景画像として用意し，それと入力画像との差分をとることで移動物体の存在する領域を検出する手法である．

$$|B(t) - i(t)| < th(t) \tag{2.1}$$

式 (2.1) で，$B(t)$，$i(t)$，$th(t)$ はそれぞれ時刻 t における背景輝度値，入力輝度値，閾値である．このような比較を各ピクセルに対して適用することで，移動物体の存在領域を判断する．

閾値は動的に更新されたり，画素ごとに変化させて，より精度良く移動物体を検出する研究もなされている．背景差分法を用いる場合，正確な背景画像を獲得することができれば，ほぼ的確に移動物体の存在する領域を検出することができる．しかし，実世界環境下においては，照度の変化などやカメラの移動により常に背景が変動し続けるため，それに追従することが必要となる．

2.3　フレーム間差分法

フレーム間差分法は，連続するフレーム間で対応する各画素について差分を求めるものである．時刻 t における着目画素の輝度値を $i(t)$，1 フレーム前の同位置の画素値を $i(t-1)$ とし，閾値を $th(t)$ とすれば，次式で表される．

$$|i(t) - i(t-1)| > th(t) \tag{2.2}$$

このフレーム間差分法を移動物体検出に用いる際には，フレーム間差分の絶対値が大きい部分を以下のいずれかであるとして検出する．

- 移動物体が着目画素に侵入した
- 移動物体が着目画素から出た
- 照度が変化した

フレーム間差分は計算コストが低く，比較的高速に動く物体などを検出する際には有効であが，移動物体が上記の三つの内のどの状態であるかは，フレーム間差分だけでは知ることができない．

2.4　オプティカルフロー

動画像からの速度検出に関する研究は 1970 年ころより取り組まれている．その多くは動画像中の濃淡パターンの対応付けの考えをもとに，速度場を計算する手法が一般的である．こうした対応付けの考えに基づき，動画像より検出される見かけの速度場 (図 2.1 参照) を，狭い意味のオプティカルフローと呼び，この速度場を動画像から得る手法のことをオプティカルフロー推定法と呼ぶ．従来より提案されている手法は，ブロックマッチング法と勾配 (グラディエント) 法に大別される．連続するパターンの直接的な対応付けを実行し，得られる変位ベクトルからオプティカルフローを決定する手法をブロックマッチング法，または単にマッチング法と呼ぶ．一方，パターンの特徴を表す画像関数 (濃淡分布)$f(x,y,t)$ が，運動に際し不変に保たれるとの仮定 $(f(x,y,t) = f(x+\delta x, y+\delta y, t+\delta t))$ より，ある点 (x,y) におけるオプティカルフローの速度と，動画像の濃淡分布の空間勾配，及び時間勾配とを関係づける式 (拘束条件) をもとにした解析手法を，勾配法と呼ぶ．しかし，勾配法で得られる拘束条件だけでは，フローを求めることができないので，他に付加条件が必要となる．ここで使われる条件によって，空間的大域最適化法と空間的局所最適化法の 2 つの手法に大別される．

次節以降では，それらの解析手法について述べるとともに，本研究で用いる空間的局所最適化法の利点について説明する．

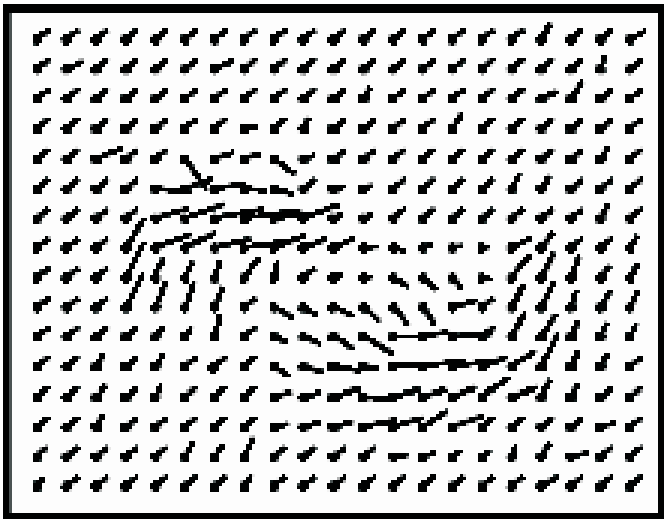

図 2.1: オプティカルフローの様子

2.4.1 ブロックマッチング法

ブロックマッチング法では，画像を一定の大きさの小領域 (ブロック) に分割し，それぞれのブロックの動きベクトルを求める．これは各ブロックが前のフレームのどこに対応するかを探し，対応するブロックの位置の差を動きベクトルとするものである．

ブロックマッチング法では以下の計算を行う．いま第 t フレームの $N \times N$ の大きさのブロックの動きベクトルを求めるものとする．ブロックの左上端の画素の位置を (i_0, j_0) とする．このブロックの画像と直前のフレームの対応する位置から (p, q) だけずらした位置のブロックの画像との差を評価する．マッチングの評価には以下の式に示すような，画素値の差の絶対値の和 (SAD: Sum of Abusolute Difference) などが用いられる．

$$SAD(p,q) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} |f_t(i_0+i, j_0+j) - f_{t-1}(i_0+i+p, j_0+j+q)| \tag{2.3}$$

対応候補のブロックの位置 (p, q) を変えて $SAD(p, q)$ を計算し，最小となる (p, q) を動きベクトル (p_m, q_m) とする探索範囲 R_s は，$-l \leq q$，$q \leq h$ の正方形であることが多い (l, h は正数)．以上を式で表せば，

$$(p_m, q_m) = \arg \min_{(p,q) \in R_s} SAM(p,q) \tag{2.4}$$

となる．なお $\arg\min_{(p,q)\in R_s} SAM(p,q)$ は，$SAM(p, q)$ を最小とする (p, q) を意味する．

ブロックマッチングによって求めなければならない動きベクトルの数は，ブロックサイズ N が大きいほど少なくなるので，なるべく N を大きくした方が計算量が少なくなるが，N が大きくなるとブロックの中の動きが一様でなくなるため，予測誤差が大きくなる．通常ブロックの大きさは 8×8 から 16×16 程度である．また探索範囲は 32×32 程度であ

る．ブロックマッチング法を用いた手法は，輝度値の急激に変化するところでもフローの誤差が少なく雑音にも強いが，探索範囲によっては計算時間が膨大となる．また拡大・縮小，回転等，複雑に運動する物体の検出に弱いという欠点がある．

2.4.2 勾配法

オプティカルフロー推定のもう一つの手法は，勾配法と呼ばれる手法である．勾配法は動画像の濃淡分布の空間勾配および時間勾配とを関係づける式をもとにして解析的に求める．その基本方程式は，Horn ら [1] によって次のように導かれた．

動画像中の時刻 t におけるフレーム上の点 (x,y) の輝度値を $I(x,y,t)$ とし，この点を含むある物体が微少時間 δt の間に x 軸方向に δx，y 軸方向に δy 移動したとする．移動の前後で物体上の点の輝度が不変であると仮定すると，次式が成立する．

$$I(x,y,t) = I(x+\delta x, y+\delta y, t+\delta t) \tag{2.5}$$

この (2.5) 式の右辺を (x,y,t) のまわりでテーラー展開すると次式が得られる．

$$I = I + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + \varepsilon \tag{2.6}$$

ただし，ε は $\delta x, \delta y, \delta t$ の 2 次以上の項である．式 (2.6) の I は $I(x,y,t)$ を指すが，本論文では，x,y,t が自明であり，記述する必要がない場合は適宜 x,y,t を省略した表現を用いる．

式 (2.6) の両辺を δt で割り，$\delta t \to 0$ とすると次式が得られる．

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \tag{2.7}$$

式 (2.7) を下記のように表すと，式 (2.8) のようになる．

$$\begin{cases} dx/dt = u & : \text{点 } (x,y) \text{ 上の速度ベクトルの } x \text{ 成分} \\ dy/dt = v & : \text{点 } (x,y) \text{ 上の速度ベクトルの } y \text{ 成分} \\ \partial I/\partial x = I_x & : I(x,y,t) \text{ の } x \text{ 方向の勾配} \\ \partial I/\partial y = I_y & : I(x,y,t) \text{ の } y \text{ 方向の勾配} \\ \partial I/\partial t = I_t & : I(x,y,t) \text{ の時間方向の勾配} \end{cases}$$

$$I_x u + I_y v + I_t = 0 \tag{2.8}$$

式 (2.8) は勾配条件と呼ばれており，動画像の濃淡値の時間と空間に関する偏微分 (勾配) とオプティカルフロー速度 $(u,v) = (\delta x/\delta t, \delta y/\delta t)$ とを関係づける式である．

勾配法で，ある画素 (x,y) の速度ベクトルを決定するには，動画像より得られるオプティカルフロー拘束条件の他に，もう一つ以上の拘束条件式が必要となる．これについては後述するが，付加する拘束条件によって 2 つの手法に大別される．一つは，オプティカルフローの空間的変化を最小にする条件を加えた空間的大域最適化法 (グローバル法) であり，もう一つは，同一物体の濃淡パターン上の局所領域では，オプティカルフローは一定であると仮定した空間的局所最適化法である．

2.4.3 空間的大域最適化法

空間的大域最適最適化法とは，オプティカルフロー拘束条件とフローを一意に決定する条件として，画像中のオプティカルフローの全変化量を最小にする条件を用いて，各画素のフローを推定する手法である．ここでは，フローを一意に決定する条件として「画像空間での速度分布は滑らかに変化する」という仮定を用いる．この条件を式で表現すると，オプティカルフローの空間変化である次式を最小化することになる．

$$e_s = \int\int_{\Omega} \{(\frac{\partial u}{\partial x})^2 + (\frac{\partial v}{\partial y})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial u}{\partial y})^2\} dxdy \tag{2.9}$$

ここで，積分領域は画像の全領域を表す．また，画像中のオプティカルフローの拘束方程式の誤差の総和の式 (2.10) も同時に最小化することになる．

$$e_c = \int\int_{\Omega} (E_x u + E_y v + E_t)^2 dxdy \tag{2.10}$$

これら式 (2.9)，(2.10) の総和

$$e = e_c + \alpha e_s \tag{2.11}$$

の最小化問題は変分法により解くことができる．

この手法は画像全体の変化量を考慮しているために，エラーフローが画像全体に影響を与え，推定精度が低下してしまうという問題点がある．また，物体の動き境界などの，動きが急激に変化する領域においては，フロー推定精度が低下する．

2.4.4 空間的局所最適化法

空間的局所最適化法では，同一物体の濃淡パターン上の局所領域 R において，オプティカルフローは一定であると仮定する．すなわち，局所領域 R 中の各画素で求まる勾配条件の式は全て同じ解をもつと仮定する．この仮定のもとで，次式で表される二乗誤差 E を最小にすることにより，(u, v) を求める．

$$E = \sum_{x,y \in R} w(x,y)(I_x(x,y,t)u + I_y(x,y,t)v + I_t(x,y,t))^2 \tag{2.12}$$

ただし，$w(x,y)$ は重み関数であり，R の中心により大きな重みをかけるものである．この重みの値は局所領域 R の中心を原点とする (x,y) の座標によって決定される．最小二乗法により，式 (2.12) を最小にする u, v は次式の解で与えられる．

$$A^T W^2 A\boldsymbol{v} = A^T W^2 b \tag{2.13}$$

局所領域 R 内の画素数を p とすると次式で表せる．

$$\begin{aligned} \nabla I(x,y) &= (I_x(x,y), I_y(x,y))^T \\ A &= [\nabla I(x_1,y_1), \cdots, \nabla I(x_p,y_p)] \\ W &= diag[w(x_1,y_1), \cdots, w(x_p,y_p)] \\ b &= -(I_t(x_1,y_1), \cdots, I_t(x_p,y_p))^T \end{aligned}$$

ここで,

$$A^T W^2 A = \begin{bmatrix} \sum wI_x^2 & \sum wI_x I_y \\ \sum wI_y I_x & \sum wI_y^2 \end{bmatrix}$$

$$A^T W^2 b = \begin{bmatrix} \sum wI_x I_t \\ \sum wI_y I_t \end{bmatrix}$$

であるため，式 (2.13) を解くと，フローベクトル $\mathbf{v}=(u,v)$ は次のように求まる．

$$u = -\frac{\sum wI_y^2 \cdot \sum wI_x I_t - \sum wI_x I_y \cdot \sum wI_y I_t}{\sum wI_x^2 \cdot \sum wI_y^2 - (\sum wI_x I_y)^2} \tag{2.14}$$

$$v = -\frac{\sum wI_x^2 \cdot \sum wI_y I_t - \sum wI_x I_y \cdot \sum wI_x I_t}{\sum wI_x^2 \cdot \sum wI_y^2 - (\sum wI_x I_y)^2} \tag{2.15}$$

式 (2.14) と式 (2.15) に示されているように，時刻 t における画像上の点 (x,y) においてフローベクトルを求めるためには，(x,y) を中心とする局所領域 R 内で，$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_x I_y$，$\sum wI_x I_t$，$\sum wI_y I_t$ の各値を計算する必要がある．

空間的局所最適化法は推定誤差が周囲の画素に伝播することはないものの，雑音等によって発生する誤差が 2 乗の重みで影響を及ぼす．また，動き境界においては，近傍画素の大きさに応じて異なる運動領域の拘束直線が含まれてしまい，フロー精度が低下してしまう．しかし，この手法は Barron らが行った比較研究 [3] において，最も良好な結果を得ている．よって本研究では，空間的局所最適化法をハードウェアに実装する方法について検討することにする．

2.4.5 高速化のアプローチ

前節において，本研究で用いるオプティカルフロー推定法は，空間的局所最適化法であると述べた．この手法は，任意の大きさの局所領域 R を定め，その R 内の各点から得られた輝度値を用いて式 (2.14) と式 (2.15) を解き，フローベクトルを求めるが，これらの式に含まれる，$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_x I_y$，$\sum wI_x I_t$，$\sum wI_y I_t$ といった 5 つの重み付き局所和を求める計算が，全体の計算の大半を占めることになる．また，フローベクトルはフレーム内の全ての点の分だけ求める必要があるため，局所領域も解像度分だけ必要になる．つまり，解像度の分だけ式 (2.14) と式 (2.15) の計算を，繰り返し行わなければならない．よって，空間的局所最適化法を高速に処理するためには，重み付き局所和を求める演算を高速化することが重要であると考えられる．

次章では，この空間的局所最適化法の式 (式 (2.14) と式 (2.15)) を単純にハードウェア処理した場合について議論し，そのままでは高解像度・高フレームレートに対応するのが難

しいことを示し，第四章で，重み付き局所和の繰り返し演算を高速に計算する手法について詳しく説明する．

2.5 まとめ

本章では，移動物体検出に用いる手法について説明し，オプティカルフローの有効性を示した．また，オプティカルフローの検出法は，マッチング法と勾配法の 2 つに大別され，それぞれの手法について説明を行った．また，勾配法には付加する拘束条件によって 2 つの手法に大別されることを示し，それぞれの手法について説明を行った．

文献 [3] の比較研究により，最も信頼性の高いフローベクトルを計算できる手法は，空間的局所最適化法であることがわかっている．次章では，空間的局所最適化法を単純にハードウェア処理する方法について述べる．また第四章では，空間的局所最適化法を高速に処理する提案手法について述べる．

第3章　プロトタイプハードウェア

3.1　はじめに

オプティカルフロー推定法にはマッチング法と勾配法があり，勾配法にも空間的大域最適化法と空間的局所最適化法があることはすでに述べた．また，過去の研究において，空間的局所最適化法が最も信頼性の高いフローベクトルを計算できることが明らかになっている．

空間的局所最適化法の式 (2.14)，(2.15) における，重み付き局所和，$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ などの計算は，全ての画素において行われる繰り返し演算であるため，非常に計算量が大きい．つまり，この総和演算を高速に処理できれば，フローベクトルを求める計算全体を高速に行うことが可能なのである．

本章ではオプティカルフローの計算を，空間的局所最適化法を用いて処理するハードウェアが，どの程度の計算時間を必要とするか見積もるために設計したプロトタイプハードウェアについて説明する．このプロトタイプハードウェアは，空間的局所最適化法の式 (2.14) と式 (2.15) において，5 種類の重み付き局所和 $\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ を求める計算が最も計算量の大きいことに着目し，この 5 種類の重み付き局所和を並列に処理し，かつ繰り返し演算をパイプライン的処理することにより，全体の計算を高速化する．

3.1.1　ハードウェアによる計算法

空間的局所最適化法によるフローベクトルの計算は，第 2 章において示した式 (2.14)，(2.15) に，観測的に求まる値を代入して解けば良い．ここでは，実際にハードウェア上でどのような計算が行われるかについて説明する．

デジタル画像を扱う場合，偏微分は次のように空間的，時間的に隣接する輝度値の差分で近似される．

$$\begin{cases} I_t(x,y,t) &= I(x,y,t+1) - I(x,y,t) \\ I_x(x,y,t) &= I(x+1,y,t) - I(x,y,t) \\ I_y(x,y,t) &= I(x,y+1,t) - I(x,y,t) \end{cases}$$

局所領域を一辺の大きさが r の正方形とするならば，$I_x(x,y,t)$，$I_y(x,y,t)$ を計算するためには，局所領域 $r \times r$ 画素の他に，隣接する 1 行分 ($r \times 1$ 画素) と 1 列分 ($1 \times r$ 画素) のデータが必要になる．また，$I_t(x,y,t)$ を計算するためには，時間的に隣接する画素のデー

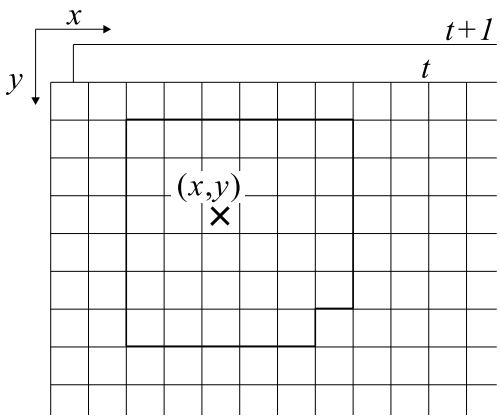

図 3.1: 計算に必要なデータ

タが必要になる．例えば，局所領域を一辺の大きさが $r = 5$ の正方形とするならば，I_x，I_y を計算するためには，図 3.1 に示すように，局所領域 5×5 の他に，隣接する 1 行分 (5×1 画素) と 1 列分 (1×5 画素) のデータが必要になる．

3.1.2　重み付き局所和のパイプライン的処理

オプティカルフローを空間的局所最適化法を用いて算出する場合，$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ などの重み付き局所和を求める計算は，ある時刻のフレーム内に含まれる全ての局所領域において行われる繰り返し演算であり，全体の計算の中で最も計算量が大きい．この部分を並列計算及びパイプライン的に処理することが，全体の高速化に大きく関わってくる．

プロトタイプハードウェアでは，総和演算以前の計算をパイプライン的に行う．各ステージは偏微分計算ステージ，乗算ステージ，重み計算ステージ，総和計算ステージであり，各点における重み付き局所和を，並列して行える分の演算器を用意する．各ステージでは，クロックに同期して計算結果を次のステージに送る．

重み付き局所和を逐次処理で計算するには 100 ステップ必要だが，パイプライン的に処理を行うと 25 ステップ (ただし，オーバーヘッドが 3 ステップ必要である) で終了することができ，逐次処理に比べて非常に高速である．

3.2　プロセッサの構成

式 (2.14)，(2.15) に基づいて，オプティカルフローを計算するプロセッサを設計した．設計環境は以下の通りである．

- 回路設計ツール：Xilinx 社製 ISE 6.1

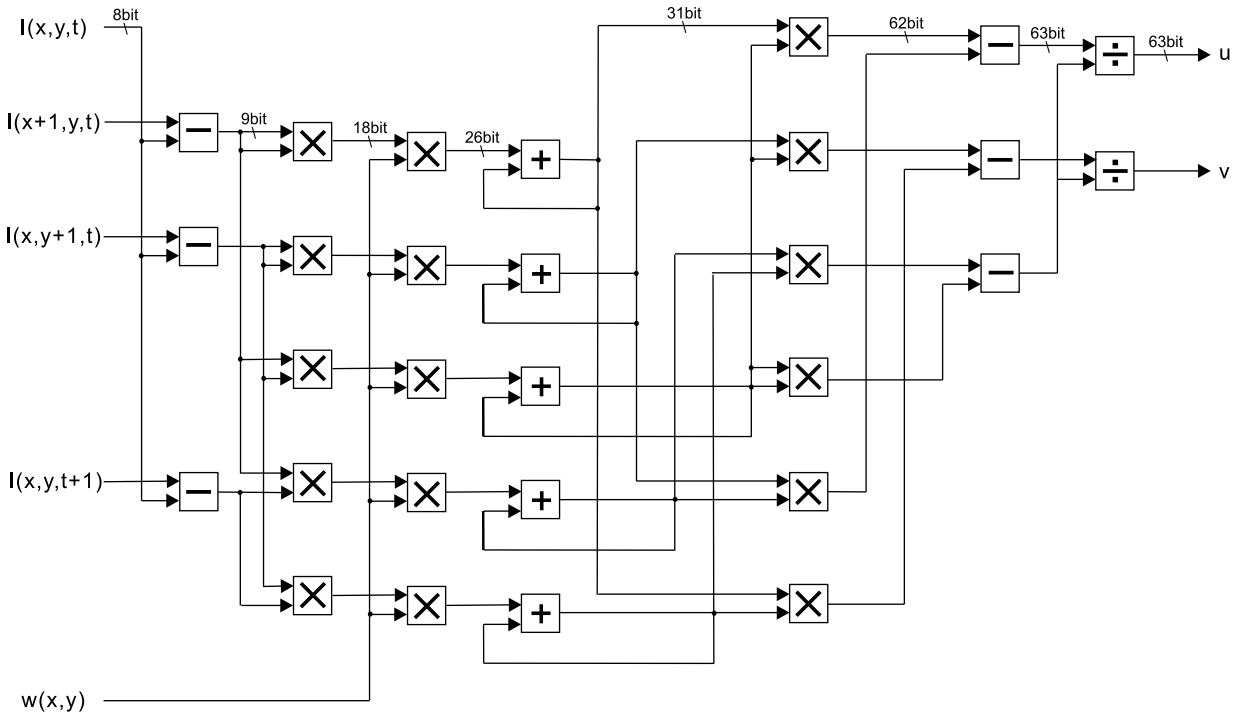

図 3.2: プロトタイプハードウェア

- 波形シミュレータ：Model Technology 社製 ModelSim XE 5.6
- FPGA ボード：NALLATECH 社製 BenOne
- FPGA デバイス：Xilinx 社製 XC2V4000

全体の構成を図 3.2 に示す.

前節で示したとおり，各ステージは並列して計算が可能である．また，重み付き局所和計算以降も，同様にパイプライン的構成になっているが，重み付き局所和の計算に 25 ステップ必要で，待ち時間が長いため効率がよいとはいえない.

3.3 シミュレーション波形

図 3.3 に，プロトタイプハードウェアのシミュレーション波形を示す．図 3.3 にて明らかなように，1 ステップ目で入力されたデータが 28 ステップかかって重み付き局所和が出力された．全体が 31 ステップなので，重み付き局所和の計算が大部分を占めている.

3.4 まとめ

本章ではオプティカルフローの計算を，空間的局所最適化法を用いて処理するハードウェアが，どの程度の計算時間を必要とするか見積もるために設計したプロトタイプハードウェアについて説明した．空間的局所最適化法の式(2.14)，(2.15)における，重み付き局所和，$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$などの計算は，全ての画素において行われる繰り返し演算であるため，非常に計算量が大きいので，総和演算を高速に処理することが重要であるそのため，このプロトタイプハードウェアは，式(2.14)と式(2.15)の5種類の重み付き局所和を並列に処理し，かつ繰り返し演算をパイプライン的処理することにより，全体の計算を高速化する．この方法では重み付き局所和を求めるのにステップ数がかかり，その後の，式(2.14)，(2.15)に基づいて実際にフローベクトル(u, v)を求める計算に，大きな待ち時間が発生することが分かった．この結果をふまえて，次章では計算結果を再利用することで，各局所領域における重み付き局所和の計算時間を短縮し，空間的局所最適化法を高速に処理する計算方法を提案する．

図 3.3: プロトタイプハードウェアのシミュレーション波形

第4章　部分列和再利用法によるオプティカルフロー高速計算ハードウェアシステム

4.1　はじめに

オプティカルフローを空間的局所最適化法を用いて計算する場合，最も計算に時間のかかる部分は，式(2.14)，および式(2.15)中の$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$などの重み付き局所和を，全ての画素において計算を行う部分である．

第3章では，この重み付き局所和を求める計算を，並列かつパイプライン的に処理する方法を提案したが，この方法では重み付き局所和を求めるのにステップ数がかかり，その後の，式(2.14)，(2.15)に基づいて実際にフローベクトル(u, v)を求める計算に，大きな待ち時間が発生することがわかった．

そこで本章では，重み付き局所和を行，列方向に分解することで，局所和の計算結果を再利用し，局所和の計算もパイプライン的に行える手法を提案する．まず，重みが二次元ガウス関数であることに着目し，一次元ガウス関数同士の乗算に置き換えることで，重みを乗ずる処理を行と列に分解して行うことができることを示す．次に，一次元ガウス関数を用いた重み付き列和が，重なり合う局所領域内で共有できることを示し，効率の良い計算結果再利用法を提案する．さらに，この部分列和再利用法を用いることで，オプティカルフロー計算全体のパイプライン的処理が可能になることを示し，その計算方法について詳述する．

最後に，その計算方法に基づいたプロセッサの構成法について説明する．

4.2　計算結果再利用法

4.2.1　重み計算の分割

オプティカルフローを空間的局所最適化法を用いて算出する場合，$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$などの重み付き局所和を求める計算は，ある時刻のフレーム内に含まれる全ての局所領域において行われる繰り返し演算であり，全体の計算の中で最も計算量が大きい．

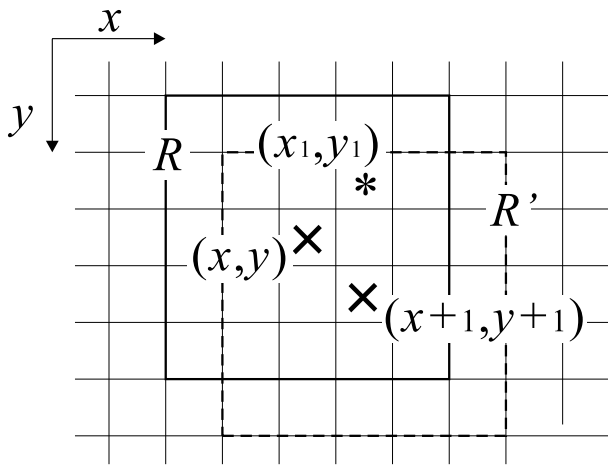

図 4.1: 隣接する局所領域

図 4.1 のように，画素 (x, y) を中心とした局所領域と，隣接する画素 $(x+1, y+1)$ を中心とした局所領域が存在するとする．この 2 つの局所領域は大部分の領域のデータを共有しているため，異なる局所領域間で同じ座標の点における計算結果を再利用することは容易にできそうである．確かに各点における偏微分やそれらを乗算した値を再利用することは可能である．しかし，重み $w(x, y)$ の値は局所領域内の中心点を原点とした相対座標によって一意に決められているため，重みを乗じた後の値を再利用することはできない．すなわち，図 4.1 のように同じフレームの同じ座標 (x_1, y_1) から得られたデータであっても，異なる局所領域では計算に用いられる重みの値が異なるため，次式に示される通り，重みを乗じた後の計算結果を再利用することはできない．

$$w(x_1 - x, y_1 - y)I_x^2(x_1, y_1, t_1) \neq w(x_1 - x + 1, y_1 - y + 1)I_x^2(x_1, y_1, t_1) \tag{4.1}$$

よって，再利用できる値は I_x^2 などの偏微分値に限定されるが，重み付き局所和を求めるときの繰り返し演算の回数はほとんど減らないため，効率的ではない．

そこで，重み計算後の値を再利用する方法を考える．まず，重み $w(x, y)$ が 2 次元ガウス関数であることに着目する．$g(x)$ を 1 次元ガウス関数とすると，重み $w(x, y)$ は $w(x, y) = g(x)g(y)$ と変形可能である．図 4.1 のように，$\sum wI_x^2$ を (x', y') を原点とする局所領域の座標系 (i, j) で表す．ただし，$r \times r$ の局所領域において，$-\lfloor r/2 \rfloor \leq i \leq \lfloor rn/2 \rfloor$，$-\lfloor r/2 \rfloor \leq j \leq \lfloor r/2 \rfloor$ であり，$x = x' + i$，$y = y' + j$ である．よって重み付き局所和 $\sum w(x, y)I_x^2$ は次のように変形できる．

$$\sum_{x,y \in R} w(x, y)I_x^2(x, y) = \sum_{-\lfloor r/2 \rfloor \leq i \leq \lfloor r/2 \rfloor} g(i) \sum_{-\lfloor r/2 \rfloor \leq j \leq \lfloor r/2 \rfloor} g(j)I_x^2(x' + i, y' + j) \tag{4.2}$$

式 (4.2) より，重み関数は行方向，列方向に分解して適用可能であるということがわかる．

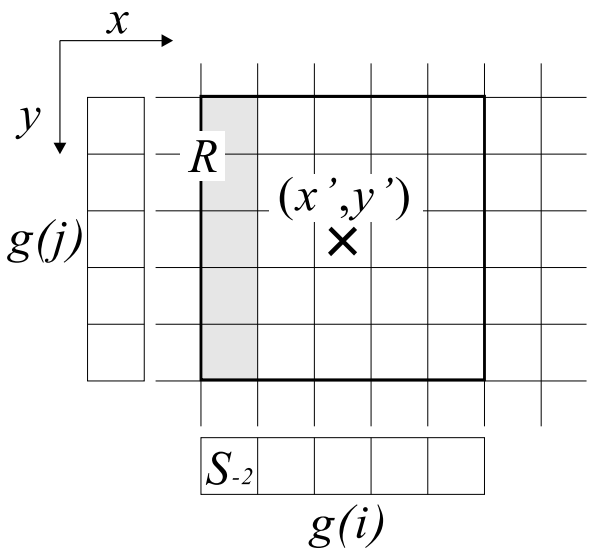

図 4.2: 行と列に分けて重みを計算する様子

4.2.2 列和を利用した計算結果の再利用法

次に局所領域 R と，R から x 方向に一画素だけずれた局所領域 R' を考える．図 4.3 のように局所領域 R の左端の列から 0，1，$\cdots$，n というように列番号を付加すると，第 k 列における I_x^2 の重み付き列和 S_k は次のように求まる．

$$S_k = \sum_{-\lfloor r/2 \rfloor \le j \le \lfloor r/2 \rfloor} g(j) I_x^2(x' + k - \lfloor \frac{r}{2} \rfloor, y' + j) \tag{4.3}$$

よって，R の重み付き局所和は，式 (4.2)，(4.3) を用いて，次のように表される．

$$\begin{aligned} \sum_{x,y \in R} w I_x^2 &= \sum_{-\lfloor r/2 \rfloor \le i \le \lfloor r/2 \rfloor} g(i) S_{i+\lfloor r/2 \rfloor} \\ &= g(-\lfloor r/2 \rfloor) S_0 + g(-\lfloor r/2 \rfloor + 1) S_1 + \cdots + g(\lfloor r/2 \rfloor) S_r \end{aligned} \tag{4.4}$$

また，同様に局所領域 R' の左端の列から 1，2，$\cdots$，$r+1$ と列番号を付加すると，第 l 列における I_x^2 の重み付き列和 S_l は次のように求まる．

$$S_l = \sum_{-\lfloor r/2 \rfloor \le j \le \lfloor r/2 \rfloor} g(j) I_x^2(x' + l - \lfloor \frac{r}{2} \rfloor, y' + j) \tag{4.5}$$

よって，R の重み付き局所和は，式 (4.2)，(4.5) を用いて，次のように表される．

$$\begin{aligned} \sum_{x,y \in R} w I_x^2 &= \sum_{-\lfloor r/2 \rfloor \le i \le \lfloor r/2 \rfloor} g(i) S_{i+\lfloor r/2 \rfloor + 1} \\ &= g(-\lfloor r/2 \rfloor) S_1 + g(-\lfloor r/2 \rfloor + 1) S_2 + \cdots + g(\lfloor n/2 \rfloor) S_{r+1} \end{aligned} \tag{4.6}$$

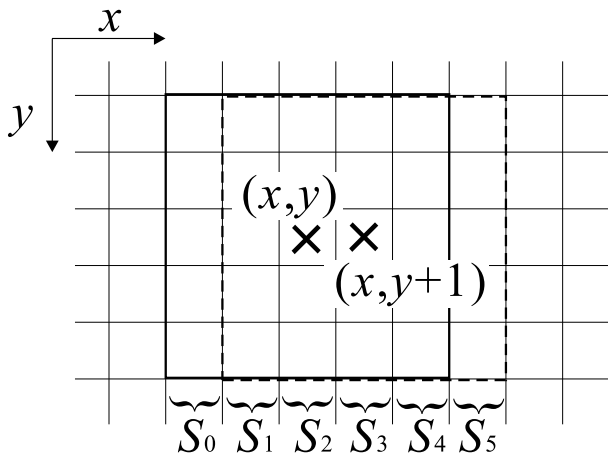

図 4.3: 5×5 の場合の列の局所和

式 (4.4)，式 (4.6) より，2 つの局所領域の計算結果において列方向の重み付き局所和 S_1，S_2，$\cdots$，S_r を共有していることは明らかである．つまり，局所領域 R' は重なり合う直前の局所領域 R の計算結果を用いることで，大部分の計算を削減できる．この方法により，最初に一つの局所領域における重み付き局所和さえ計算してしまえば，次の局所領域からは i 列分の和を計算するだけで，重み付き局所和を得ることが可能である．

フローベクトルを計算する点が x 軸方向に移動していく場合，この方法は有効である．

4.2.3 計算方法

前節の考察より，本論文での計算法を以下に示す．

- フェーズ 1：偏微分とそれらの計算

  局所領域 R 内の任意の点 (x, y, t) の輝度値 $I(x, y, t)$，及びその点における偏微分計算に必要な点 $(x+1, y, t)$，$(x, y+1, t)$，$(x, y, t+1)$ の輝度値 $I(x+1, y, t)$，$I(x, y+1, t)$，$I(x, y, t+1)$ より，偏微分 I_x，I_y，I_t を計算する．また，求めた偏微分の値同士を乗算することにより I_x^2，I_y^2，I_xI_y，I_xI_t，I_yI_t を得る．

  これらの処理は，局所領域 R 中の各行毎に独立して行う．

- フェーズ 2：重み付き列和の計算

  局所領域 R 内の任意の点における I_x^2，I_y^2，I_xI_y，I_xI_t，I_yI_t の各値に対し，行方向の重み $g(j)$ を乗算する．この処理を，局所領域 R 内の同じ列の点から得られた値の全てに施す．そして，得られた一列分の各値から，重み付き列和 $S_i(I_x)$ を求める．

- フェーズ 3：重み付き局所和の計算

局所領域 R 内の重み付き列和に列方向の重み $g(i)$ を乗じる．この処理を，R 内の全ての列和に対して行う．こうして，R 内の重み付き局所和 $\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ を求める．このとき，R と領域を共有する他の 4 つの局所領域における重み付き局所和も並列して計算する．

- フェーズ 4：フローベクトル (u, v) の計算

  式(2.14)，(2.15)に基づいて，局所領域 R の重み付き局所和 $\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ から，フローベクトル (u, v) を求める．

4.3 プロセッサの構成

空間的局所最適化法を列和再利用法を用いてパイプライン化したプロセッサの構成について説明する．構成としては，前節で示した計算法に基づき各フェーズ毎にユニット化した．なお，局所領域は 5×5 の大きさとしている．

偏微分計算ユニットではフェーズ 1 に示されている偏微分計算と得られた偏微分同士の乗算を行う．重み付き列和計算ユニットではフェーズ 2 でで示されている重み付き列和の計算を行う．重み付き局所和計算ユニットではフェーズ 3 で示されている重み付き局所和の計算を行う．フローベクトル計算ユニットではフェーズ 4 で示しされているフローベクトルの計算を行う．

設計環境は，第 3 章で示したものと同様である．

次節から各ユニット毎に詳細な動作を示す．

4.3.1 偏微分計算ユニット

このユニットは輝度値を入力とし，偏微分 I_x，I_y，I_t とそれぞれの積 I_x^2，I_y^2，I_xI_y，I_xI_t，I_yI_t を，各行毎に並列に計算するユニットである．それぞれの列では，I_x^2，I_y^2，I_xI_y，I_xI_t，I_yI_t を，この順番で 1 クロック毎に出力する．

図 4.4 のように，全く構成の等しい回路ブロックが 5 つ縦に並んでいる．一つのブロックが 4 つの入力を持っている．点 (x, y, t) でフローベクトルを計算する場合，4 つの入力にはそれぞれ $I(x, y, t)$，$I(x+1, y, t)$，$I(x, y+1, t)$，$I(x, y, t+1)$ の値が 5 クロックの間入力される．スイッチは内部にカウンタを持っており，表 4.1 のようにカウンタの値に従って，4 つの値を出力する (但し，図 4.4 のスイッチから出ている出力の上から Output0，Output1，Output2，Output3 とする)．このようにして減算器に入力される値を制御することで，最終的にこのユニットは 1 クロック毎に $I_x^2(x, y, t)$，$I_y^2(x, y, t)$，$I_xI_y(x, y, t)$，$I_xI_t(x, y, t)$，$I_yI_t(x, y, t)$ を出力する．

点 (x, y, t) における計算結果が出力されたら，点 (x, y, t) を x 方向に +1 だけずらした点 $(x+1, y, t)$ の計算結果 $I_x^2(x+1, y, t)$，$I_y^2(x+1, y, t)$，$I_xI_y(x+1, y, t)$，$I_xI_t(x+1, y, t)$，$I_yI_t(x+1, y, t)$ が出力される．以下，同様に処理が継続される．

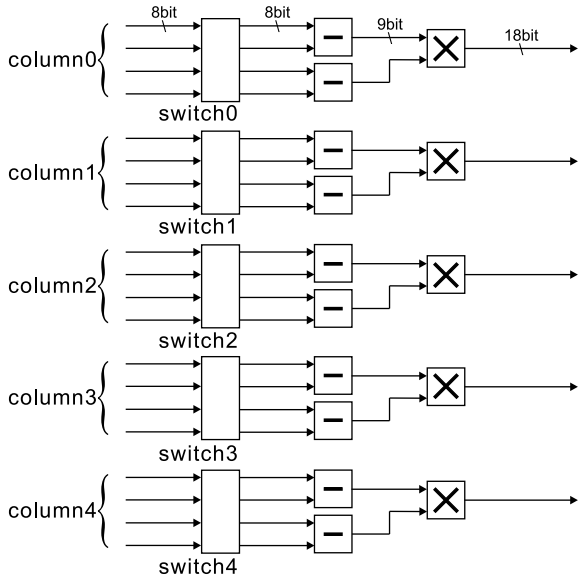

図 4.4: 偏微分計算ユニット

Step	0	1	2	3	4
output0	$I(x+1,y,t)$	$I(x,y+1,t)$	$I(x+1,y,t)$	$I(x+1,y,t)$	$I(x,y+1,t)$
output1	$I(x,y,t)$	$I(x,y,t)$	$I(x,y,t)$	$I(x,y,t)$	$I(x,y,t)$
output2	$I(x+1,y,t)$	$I(x,y+1,t)$	$I(x,+1y,t)$	$I(x,y,t+1)$	$I(x,y,t+1)$
output3	$I(x,y,t)$	$I(x,y,t)$	$I(x,y,t)$	$I(x,y,t)$	$I(x,y,t)$

表 4.1: スイッチの動作

この 5 つのブロックは，それぞれ局所領域の一列分の処理を並列に実行する．ただし，後述する重み付き列和計算部への入力の関係上，各ブロックへの入力は上から 1 クロックずつ遅れ，階段状になる．当然，計算結果も 1 クロックずつ遅れて出力される．

図 4.5 は，実際に設計した偏微分計算ユニットのシミュレーション結果である．列 1 から列 4 までの値を階段状に入力すると，その計算結果も階段状に出力されている様子が見て取れる．

4.3.2 重み付き列和計算ユニット

このユニットは，偏微分計算ユニットから順番に 1 クロックずつ遅れて入力される値に，その行固有の重みを乗じ，次の行の値と足し合わせていく．全ての行の値を足し合わせた

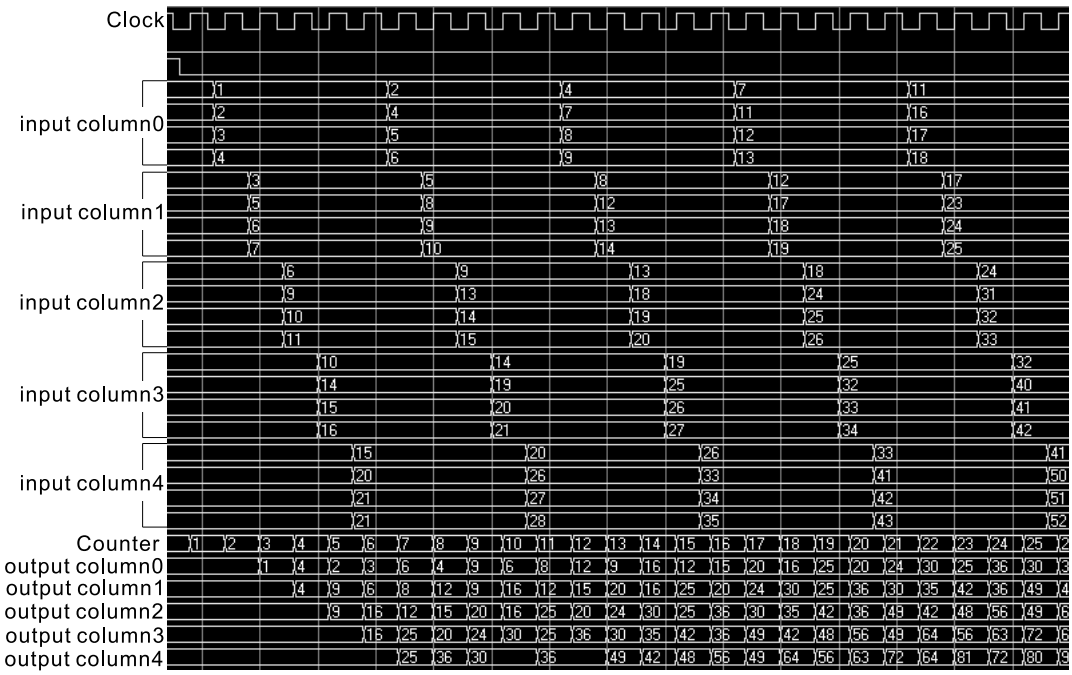

図 4.5: 偏微分計算ユニットの入出力の様子

値が，重み付き列和としてこのユニットから出力される．

図 4.6 のように，全く等しい構成の回路ブロックが 5 つ縦に並んでいる．それぞれの入力には，偏微分計算ユニットの計算結果がそれぞれ入力される．それぞれのブロックでは，入力された値にそのブロックが対応している行の重みを乗じる．その後，上のブロックの計算結果と足し合わされ，下のブロックに送られる．ただし，最上段のブロックには上からの入力が存在しないので，0 を入力する．

例えば，一番上のブロックで計算された値 $g(0)I_x^2(x,y,t)$ は 0 と足し合わされ，次のブロックに送られる．二段目のブロックでは，一段目の計算結果 $g(0)I_x^2(x,y,t)$ が $g(1)I_x^2(x,y+1,t)$ と足し合わされ，$\sum_{j=0}^{1} I_x^2(x,y+j,t)$ となる．こうして，最下段のブロックから出力される値は $\sum_{j=0}^{4} I_x^2(x,y+j,t)$ となる．$I_y^2(x+1,y,t)$，$I_xI_y(x+1,y,t)$，$I_xI_t(x+1,y,t)$，$I_yI_t(x+1,y,t)$ も同様である．

このように，各ブロックでは重みを乗じた値に，上のブロックの計算結果を足し合わせ下のブロックに送るという処理をするので，各ブロックへの入力は上段より 1 クロックずつ遅れることになる．よって，このユニットへの入力は階段状にずれたものになる．

このユニットの計算結果は $\sum_{j=0}^{4} I_x^2(x,y+j,t)$，$\sum_{j=0}^{4} I_y^2(x,y+j,t)$，$\sum_{j=0}^{4} I_xI_y(x,y+j,t)$，$\sum_{j=0}^{4} I_xI_t(x,y+j,t)$，$\sum_{j=0}^{4} I_yI_t(x,y+j,t)$ の順で出力され，その後は x 方向に +1 ずれた列の和，$\sum_{j=0}^{4} I_x^2(x+1,y+j,t)$，$\cdots$ が出力される．以下，同様に処理が継続される．

図 4.7 は，実際に設計した重み付き列和計算ユニットのシミュレーション結果である．偏微分計算ユニットから出力された各列の計算結果が階段状に入力され，その総和が出力

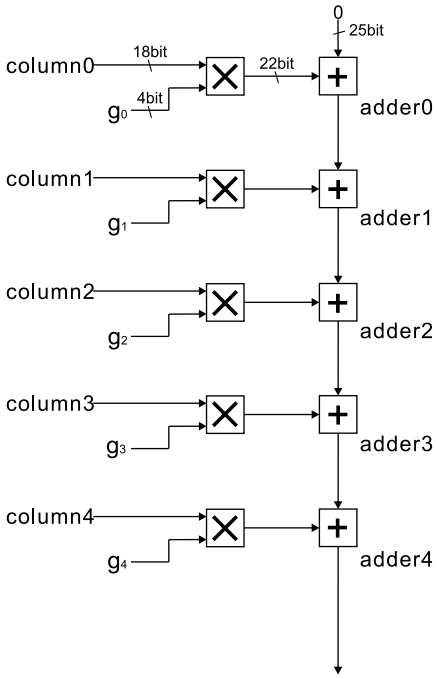

図 4.6: 重み付き列和計算ユニット

されている様子が見て取れる．

4.3.3 重み付き局所和計算ユニット

このユニットでは，入力された重み付き列和に列の重みを乗じ，隣の列の値と足し合わせていくことで，局所領域全体の重み付き局所和を求め，出力する．また，重なり合う5つの局所領域における重み付き局所和を並列して計算することで，計算処理の待ち時間をなくしている．

図4.8のように，全く等しい構成の回路ブロックが横に5つ並んでいる．入力された重

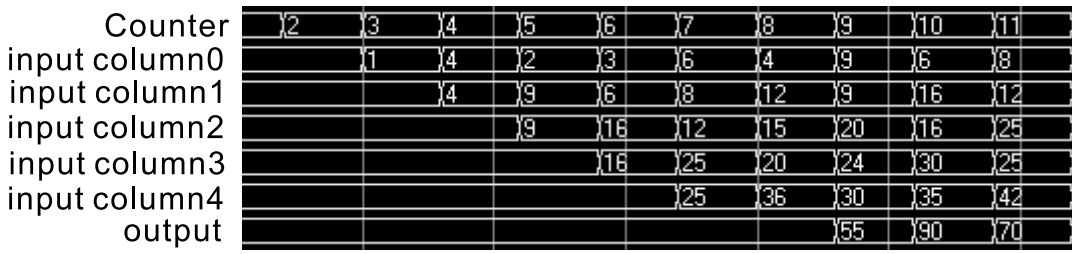

図 4.7: 列和計算ユニットの入出力の様子

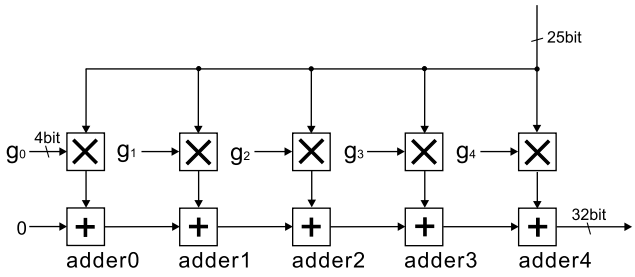

図 4.8: 重み付き局所和計算ユニット

Step	$\cdots$	n	n+1	n+2	n+3	n+4	n+5	$\cdots$
adder0	$\cdots$	g_0S_n	g_0S_{n+1}	g_0S_{n+2}	g_0S_{n+3}	g_0S_{n+4}	g_0S_{n+5}	$\cdots$
adder1	$\cdots$	$\cdots$	$\sum_{i=0}^{1} g_iS_{n+i}$	$\sum_{i=0}^{2} g_iS_{n+i}$	$\sum_{i=0}^{3} g_iS_{n+i}$	$\sum_{i=0}^{4} g_iS_{n+i}$	$\sum_{i=0}^{n+0} g_iS_{n+i}$	$\cdots$
adder2	$\cdots$	$\cdots$	$\cdots$	$\sum_{i=0}^{2} g_iS_{n+i}$	$\sum_{i=0}^{3} g_iS_{n+i}$	$\sum_{i=0}^{4} g_iS_{n+i}$	$\sum_{i=0}^{5} g_iS_{n+i}$	$\cdots$
adder3	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\sum_{i=0}^{3} g_iS_{n+i}$	$\sum_{i=0}^{4} g_iS_{n+i}$	$\sum_{i=0}^{5} g_iS_{n+i}$	$\cdots$
adder4	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\sum_{i=0}^{4} g_iS_{n+i}$	$\sum_{i=0}^{5} g_iS_{n+i}$	$\cdots$

表 4.2: 各加算器の計算結果

み付き列和は，全てのブロックに入力される．それぞれのブロックでは，入力された値にそのブロックが対応している列の重みを乗じる．その後，左のブロックの処理結果と足し合わせて右のブロックに送る．ただし，最も左のブロックには左からの入力が存在しないので，0 を入力する．

表4.2 は，あるステップ n〜n+5 のまでで，各加算器 (左から adder0，adder1，$\cdots$，adder4 とする) がどのような値を出力するかを示している．ただし，簡単化のために I_x^2 に関する重み付き局所和だけを求めることとし，ステップ n の時に入力される I_x^2 の重み付き列和を S_n とする．また，わかりやすいようにステップ n 以前に入力された値を扱った演算を省略した．

表 4.2 において，n ステップ目に adder0 に入力された値 g_0S_n は 0 と足し合わされ，1 ステップ遅れて図 4.8 における右側の adder1 に渡す．そうすることで，重み付き列和 S_n と，S_n から 1 ステップ遅れて入力される S_n の隣の重み付き列和 S_{n+1} との和をとることができる同様にして重み付き列和は加算されていき，右端の加算器 adder4 から出力される値が，局所領域全体の重み付き局所和となる．

実際には，入力は $\sum_{j=0}^{4} I_x^2(x,y+j,t)$，$\sum_{j=0}^{4} I_y^2(x,y+j,t)$，$\sum_{j=0}^{4} I_xI_y(x,y+j,t)$，$\sum_{j=0}^{4} I_xI_t(x,y+$

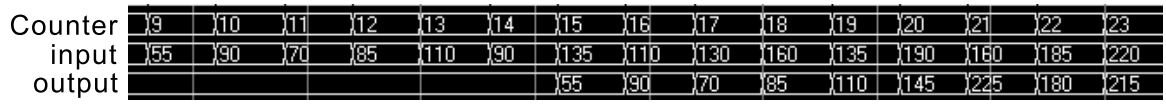

図 4.9: 重み付き局所和計算ユニットの入出力の様子

$j,t)$，$\sum_{j=0}^{4} I_y I_t(x,y+j,t)$，$\sum_{j=0}^{4} I_x^2(x,y+j,t)$，$\cdots$ の順で行われるので，各加算器と加算器を結ぶ間に，5 クロックの遅延装置が組み込まれており，適切な値を計算するようになっている．

よって，このユニットの出力は $\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ の順に出力される．また，局所領域一つ分の重み付き局所我が出力されたら，間をおかずに次の局所領域の重み付き局所我が，同じような順序で出力される．

図 4.9 は，実際に設計した重み付き局所和計算ユニットのシミュレーション結果である．これは，表 4.2 とは異なり，5 種類の重み付き局所和を順番に計算するため，一つにつき 5 個に分割された列和を 5 回にわたって入力し，それぞれ独立に和を取っている様子が見て取れる．

4.3.4 フローベクトル計算ユニット

フローベクトル計算部は，重み付き局所和を入力とし，式 (2.14)，(2.15) に基づいてフローベクトル (u,v) を求め，出力する．

図 4.10 にフローベクトル計算ユニットの回路構成を示す．これを見るとわかる通り，このユニットの構成は，プロトタイプシステムと同様，同時に計算できる処理は並列に行うように演算器が配置されている．ただし，入力が一つで，1 ステップずつ異なる値が入力されるので，図 4.10 の”cache”に示される様に入力直後に一時保存用のメモリを用意し，式 (2.14)，(2.15) を解くのに必要な 5 つの重み付き局所和 $\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ を格納し，これら全ての値がそろったところで計算を開始する．

図 4.11 に，実際に設計したフローベクトル計算ユニットのシミュレーション結果を示す．

4.3.5 全体の構成

前節までに各ユニット毎の構成と，その動作について示した．本節では，全体の構成について述べる．

図 4.12 は全体の回路構成を示している．

図 4.13 は，実際に設計したプロセッサ全体のシミュレーション結果である．

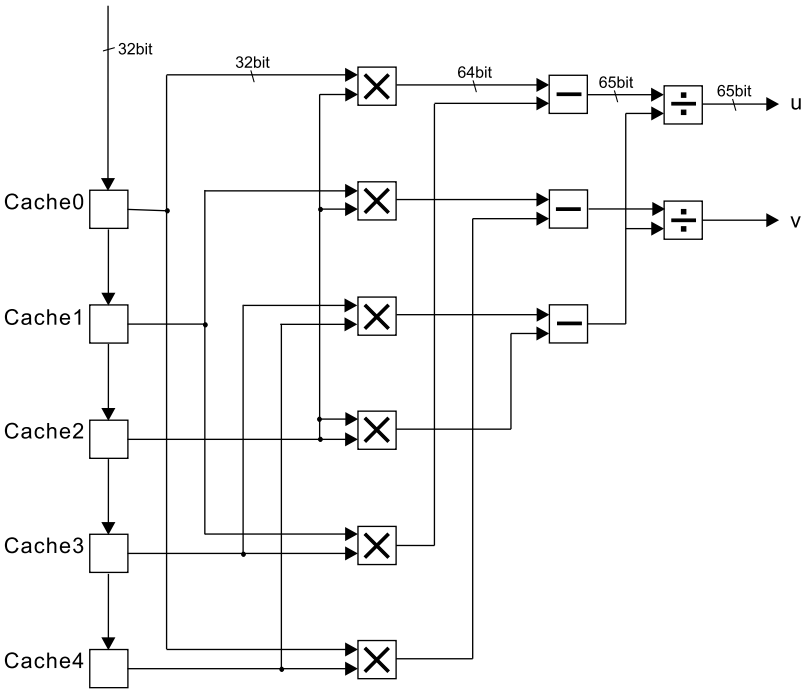

図 4.10: フローベクトル計算ユニット

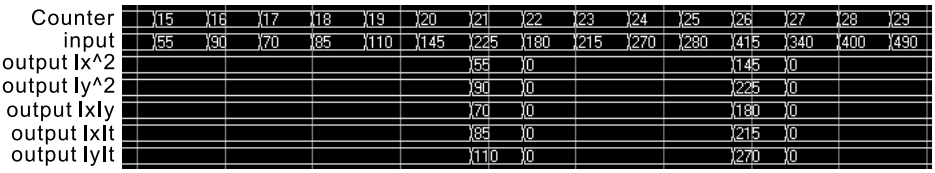

図 4.11: フローベクトル計算ユニットの入出力の様子

4.4 まとめ

オプティカルフローを空間的局所最適化法を用いて計算する場合，最も計算に時間のかかる部分は，式 (2.14)，および式 (2.15) 中の $\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$ などの重み付き局所和を，全ての画素において計算を行う部分である．本章では，重み付き局所和を行，列方向に分解することで，局所和の計算結果を再利用し，局所和の計算もパイプライン的に行える手法を提案した．

オプティカルフローを空間的局所最適化法を用いて計算する場合，フローを求める点を中心とした任意の大きさの局所領域を設定し，領域中の全ての点における x，y，t 方向の偏微分を求め，それらを乗算する．そして得られた値に局所領域内の相対座標に基づく重みをかけ，総和 (重み付き局所和) をとる．局所領域からは 5 種類の重み付き局所和が得られ，これを空間的局所最適化法の式に代入して，フローを計算する．この計算の中で最

も時間のかかる部分は，重み付き局所和を全ての画素において行うところである．この処理を高速化するために，提案する部分列和再利用法では，重なり合う局所領域間で計算結果の一部を再利用し，計算量を大幅に削減する．部分列和再利用法では，まず二次元ガウス関数である重みを，一次元ガウス関数に分解する．そうすることにより，重みの計算を行と列とで別々に行うことができるので，重み付き局所和が行数分の重み付き列和に分解される．よって，y 軸方向の変位がなく重なり合う局所領域間では，一部の重み付き列和を共有することが可能となる．つまり，局所領域が x 軸方向にのみ変位する限り，重み付き局所和は一列分の重み付き列和を求めるだけで，得ることができる．

この手法を用いると，従来非常に計算時間がかかっていた重み付き局所和を求める繰り返し演算を，約 $1/3$ に削減できることが分かった．また，計算結果を再利用することにより，画像データの読み込み回数も減るため，データ転送量も少なくなり，処理の高速化が可能になることを示した．

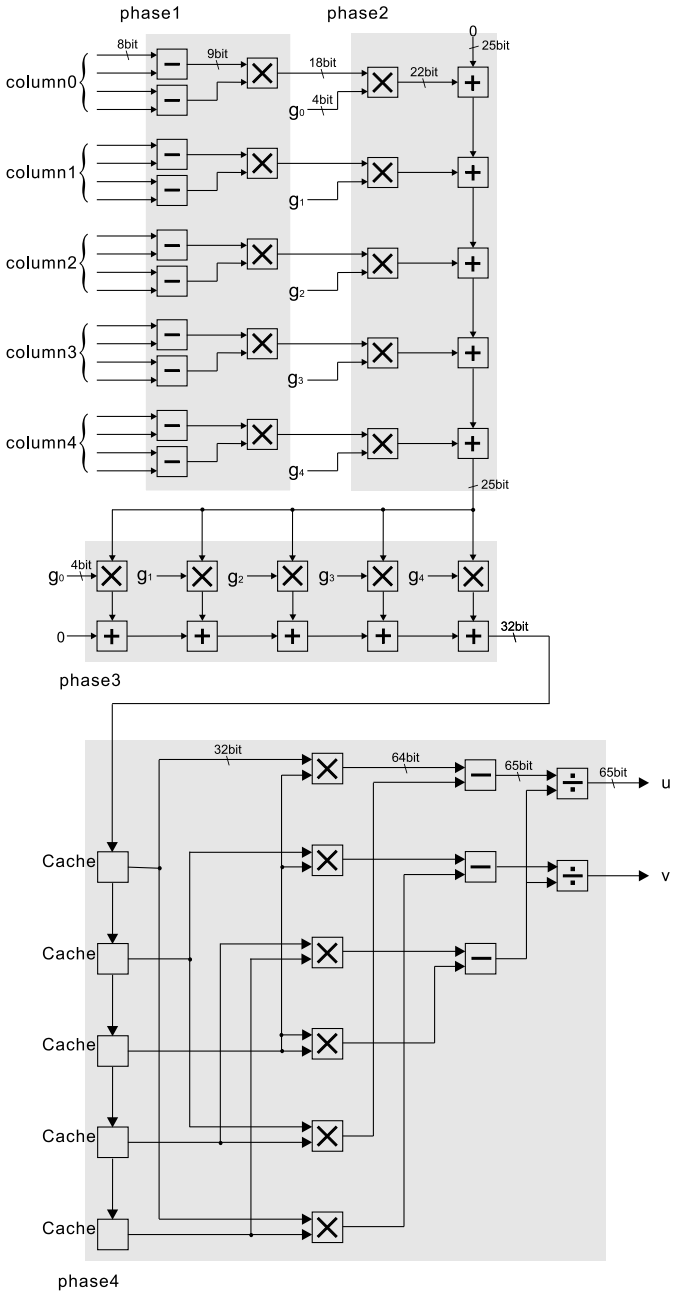
phase1
phase2
8bit
9bit
18bit
22bit
0
25bit
4bit
column0
column1
column2
column3
column4
g0
g1
g2
g3
g4
25bit
4bit
0
32bit
phase3
32bit
64bit
65bit
65bit
u
v
Cache
Cache
Cache
Cache
Cache
phase4

図 4.12: 全体の構成

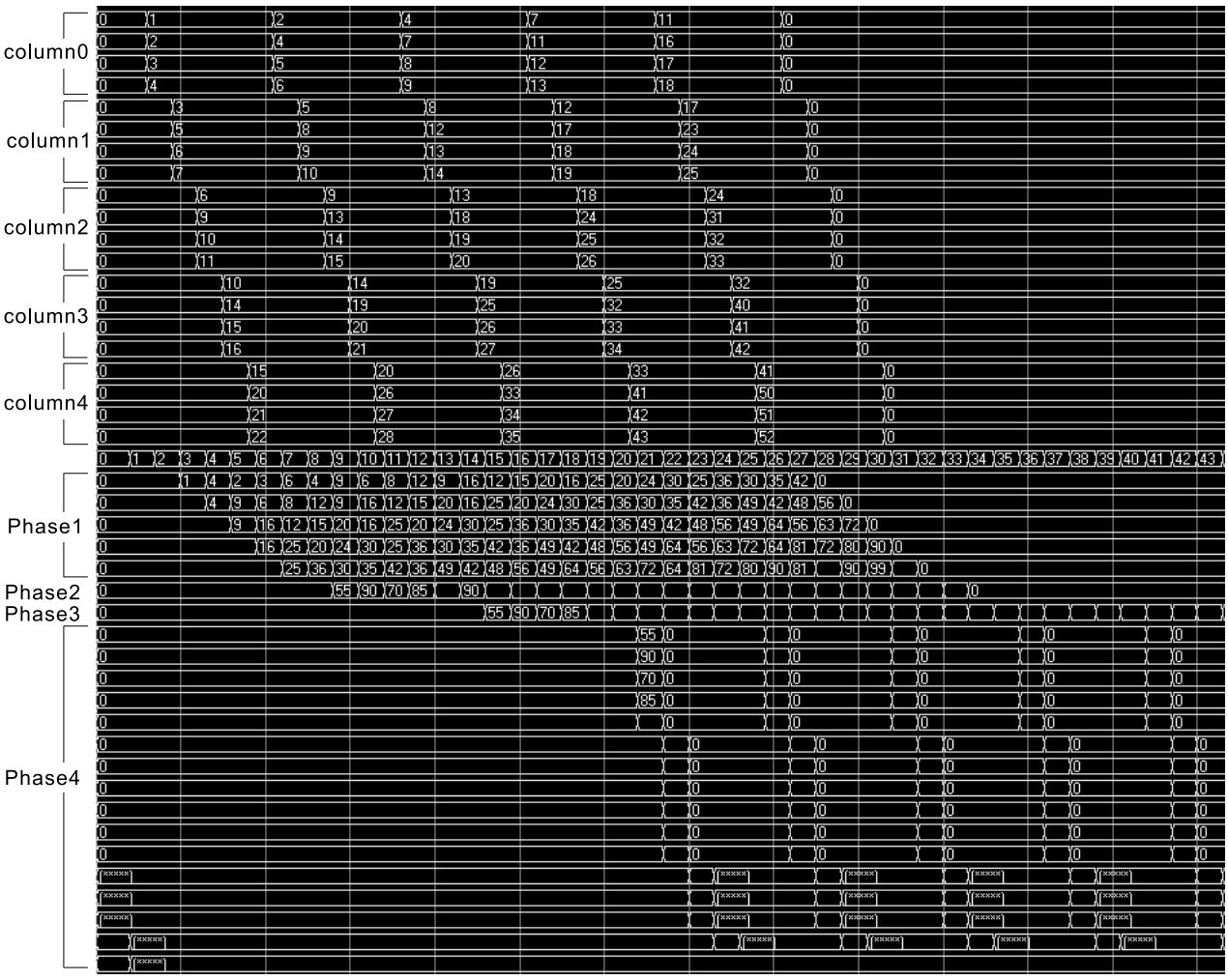

図 4.13: 全体の動作の様子

第5章　オプティカルフロー高速計算プロセッサの評価

5.1　はじめに

オプティカルフローを空間的局所最適化法を用いて計算する場合，最も計算に時間のかかる部分は，式(2.14)，および式(2.15)中の$\sum wI_x^2$，$\sum wI_y^2$，$\sum wI_xI_y$，$\sum wI_xI_t$，$\sum wI_yI_t$などの重み付き局所和を，全ての画素において計算を行う部分であった．第4章では，この重み付き局所和を求める計算を列和に分解して処理することにより，重なり合う局所領域間で重み付き局所和計算の一部を共有できることを明らかにし，重み付き局所和を求める計算に必要な計算量を減らすことで，高速化を行う手法を提案した．また，そのハードウェア実装法について詳細に述べた．

そこで，本章ではまず提案した手法に基づくプロセッサの計算時間の評価を行う．同様にして，空間的局所最適化法を逐次計算で行った場合と第3章で設計したプロトタイプハードウェアで計算した場合の計算時間を見積もり，提案手法のプロセッサとの比較を行う．さらに，従来手法との比較を行い，本手法の有効性を明らかにする．最後に，回路量について述べる．

5.2　計算時間の評価

本節では，まず今回設計したプロセッサについて処理に必要なクロック数を明らかにする．次に，逐次処理とプロトタイプハードウェアの場合の処理時間を示し，提案手法を用いることでどの程度処理時間が短縮されるかを明らかにする．最後に，従来手法であるパイプラインイメージプロセッサを用いた手法との比較を行う．こちらは，ステップ数による評価を行うことができないので，単純に処理時間だけで評価を行う．

5.2.1　計算時間の見積もり

今回，設計したプロセッサにおいて，加算器，減算器，乗算器が1クロック，除算器が5クロックで動作すると仮定すると，各ブロックはパイプライン的に動作しているので，各画素におけるフローベクトルは5クロック毎に出力される．ただし，前列の最後尾の局所領域におけるフローベクトルが出力されてから，次列の先頭の局所領域におけるフロー

ベクトルが出力されるまで，25 クロックの待ち時間が発生する．また，一番初めに計算する局所領域に限り，最初の値が入力されてから計算結果が出力されるまで 46 クロック必要であるので，21 クロックがパイプライン処理のオーバーヘッドになる．

従って，設計したプロセッサは，$X \times Y$ pixels のフレームの処理に必要なクロック数 N_1 は次式で表される．

$$N_1 = 21 + (25 + 5 \times X) \times Y \tag{5.1}$$

同様にして，式 (2.14)，式 (2.15) で示される空間的局所最適化法の式を，完全に逐次処理で計算した場合を考えると，1 画素あたり 344 クロック必要である．よって，$X \times Y$ pixels の画像を処理するのに，必要なクロック数 N_2 は次式で表される．

$$N_2 = 344 \times X \times Y \tag{5.2}$$

また，プロトタイプハードウェアで計算した場合，局所領域におけるフローベクトルが出力されてから次の局所領域におけるフローベクトルが出力されるまで 25 クロック必要である．ただし，最初に計算される局所領域に限り，パイプライン処理のオーバーヘッドとして 10 クロック必要となる．従って，$X \times Y$ pixels の画像を処理するのに，必要なクロック数 N_3 は次式で表される．

$$N_3 = 10 + 25 \times X \times Y \tag{5.3}$$

よって，動作周波数を F[Hz] とした場合，1 フレームの処理にかかる時間 T[sec] は，クロック数 N を用いて次式で表される．

$$T = \frac{N}{F} \tag{5.4}$$

式 (5.4) に式 (5.1)，式 (5.2)，式 (5.3) で求まったクロック数を代入することにより，それぞれの手法における処理時間を見積もることができる．

式 (5.1) から明らかなように，本研究で提案した手法は横方向に長い画像に対して非常に有効である．しかし，縦方向に長い画像では，パイプライン処理のオーバーヘッドが大きくなるため，性能が低下することが予想される．これは，局所領域の移動方向を，横方向に固定せずに，長辺方向に動的に対応できるようにすれば問題ない．

5.2.2 計算時間の比較

空間的局所最適化法を計算する各手法について，その処理時間の導出はすでに行った．ここでは，具体的な数値を用いて，各手法について処理時間の比較を行う．

動作周波数 $F = 26 \times 10^6$ Hz，$X = Y$ として，それぞれの手法の処理時間を比較すると，図 5.1 のようになる．ただし，動作周波数に用いられている値は，今回設計したプロセッサで使われている除算器が，5 クロック以内に計算を終えることができる最大の周波数である．この図では，逐次処理の計算時間が提案手法の計算時間に対して非常に大き

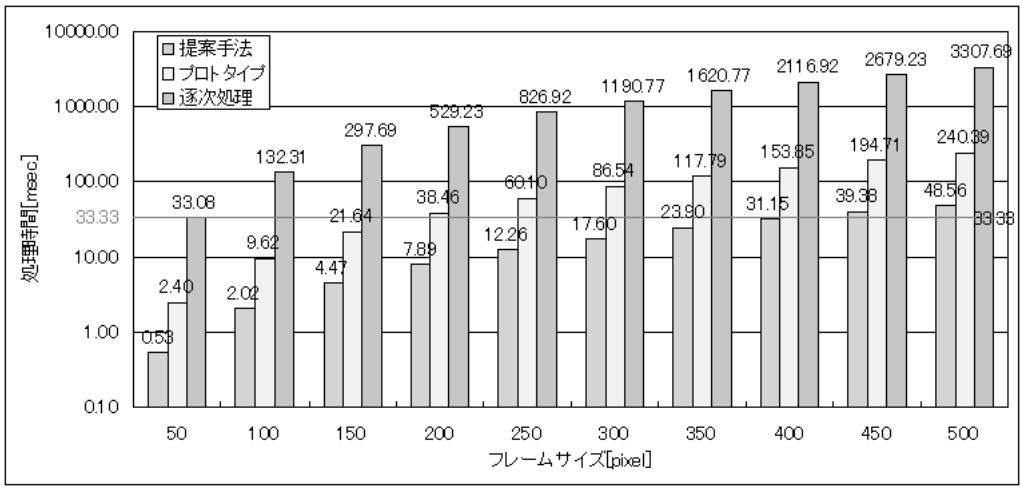

図 5.1: 1 フレームあたりの処理時間の比較

くなるため，縦軸は対数で表している．この結果から分かるとおり，逐次処理に対して約 1/60，プロトタイプハードウェアに対して約 1/5，処理時間が短縮されている．また，ビデオレート (秒間 30 フレーム) での処理を実現するためには，処理時間は最高でも 33MHz 以内に納める必要があるが，今回設計したプロセッサでは約 400 × 400pixels の画像までの処理が可能であると考えられる．

5.2.3 従来手法との比較

今回設計したプロセッサを用いて 252 × 316 pixels の画像を処理する場合，約 41 万ステップかかることになる．仮に，このプロセッサを動作周波数 26MHz で動作させた場合，その処理速度は約 15.6msec と見積もれる．パイプラインイメージプロセッサを用いた手法 [5] では，動作周波数 40MHz に対して処理時間が 47.8msec であった．仮に，今回設計したプロセッサを 40MHz で動作させた場合，その処理時間は約 10.2msec であり，データ入出力のオーバーヘッドを考慮に入れたとしても，今回設計したプロセッサの方が十分に高速である．

5.3 回路量

今回設計したプロセッサ内の各ユニットの使用 LUT 数とチップ全体に対する LUT 使用率，及び，使用乗算器数は，回路設計ツールより表 5.1 のようになった．全体の回路量が 400 万ゲート規模なので，今回設計したプロセッサの回路量は約 276 万ゲート規模である．特にフローベクトル計算ユニットの回路量の多さが目立つが，フローベクトル計算ユニットは 65bit の除算器を 2 個内蔵しており，この除算器の回路量が 90 万ゲート規模と巨

	LUT 使用数	LUT 使用率	18×18 乗算器使用数
偏微分計算ユニット	315	0.7%	5
列和計算ユニット	332	0.7%	5
局所和計算ユニット	1118	2.4%	10
フローベクトル計算ユニット	26512	57.5%	24
プロセッサ全体	28259	61.3%	44

表 5.1: 各ユニットの回路量

大だからである．

今回設計したプロセッサは，内部演算の bit 数を削減していないので，最終的に 65bit という非常に大きな bit 数となり，それが全体の回路量を膨大なものにしていると思われる．この bit 数を削減することにより，プロセッサ全体の回路量を削減できる．また，bit 数の削減は除算の高速化にもつながり，動作周波数の向上にも寄与すると考えられる．さらに，今回設計したプロセッサでは，5 つの重み付き局所和を計算するのに 5 クロック必要という制約があるため，除算はせいぜい 5 クロック以内に計算を終えられれば良いため，それほど高速な除算器を用意する必要がなくなり，さらなる回路量の削減を見込むことができる．

5.4 まとめ

本研究では，空間的局所最適化法でオプティカルフローを計算する際に，重み付き列和を利用することにより，空間的局所最適化法を高速に処理する手法を提案し，その手法に基づいたプロセッサを設計することで，パイプライン的に高速に処理可能なことを示した．

本章では，計算時間の見積もりを行い，逐次処理の場合に比べて処理時間を 1/60 短縮することができた．また，従来のシステムよりも高速に動作することを明らかにした．さらに，回路量の評価により，今回設計したプロセッサは 276 万ゲート規模であることがわかった．

第6章　まとめ

6.1　まとめ

本論文では，空間的局所最適化法でオプティカルフローを計算する際に，その最も計算量の大きい部分である重み付き局所和の繰り返し演算を，重み付き列和を利用して重み付き局所和の一部を共有することにより，高速に処理する手法を提案した．また，その手法に基づいたプロセッサ回路を設計し，パイプライン的に高速に処理可能なことを示した．さらに計算時間の評価により，従来のシステムよりも高速に動作することを明らかにした．

第 2 章では，移動物体の検出法について従来から提案されてきた手法を説明し，オプティカルフロー推定法の有効性を示した．また，数あるオプティカルフロー推定法の手法の中でも，空間的局所最適化法が有効であることを示した．

第 3 章では，空間的局所最適化法のハードウェア処理と，今回設計したプロトタイプハードウェアシステムについて説明した．

第 4 章では，2 次元ガウス関数である重みを，1 次元ガウス関数に展開することにより，重み付き局所和を重み付き列和に分解できることを明らかにし，重なり合う別の局所領域間で重み付き局所和の一部を共有し，計算結果を再利用する部分列和再利用法を提案した．また，提案した手法のハードウェア実装法について明らかにした．

第 5 章では，提案した手法に基づき設計したプロセッサの計算時間の見積もりを行い，その見積もりに従って計算時間の評価を行った．そして，今回設計したプロセッサが，従来のハードウェアより高速に動作することを示した．

6.2　今後の課題

本節では，今後の課題について，さらなる高速化と再構成可能性の面から説明する．

6.2.1　さらなる高速化

今回設計したプロセッサは内部演算の bit 数の削減を行っていないので，最終的に 64bit という巨大な bit 数の演算を余儀なくされている．これは回路量を増やすだけでなく，計算に要する時間も増大させるので，精度とのトレードオフを検討しながら削減することを

考えたい．

他のアプローチとしては，プロセッサの数を増やして並列同時実行する方法が挙げられる．本研究で提案した手法は1行毎に計算を行うので，複数のプロセッサで複数行同時実行するという方法は容易に可能である．このとき，前述した内部演算の削減により除算器が十分に高速化されていたならば，除算器を共有することが可能になり，回路量の増加を抑制可能であると見込まれる．

6.2.2 再構成可能性について

本研究で設計したプロセッサでは，局所領域の大きさを 5×5pixels に固定したものであった．しかし，それ以下の大きさの局所領域に対しては，局所領域の大きさに合わせて，重みの値とタイミングの取り方を対応させ，列和計算ユニットと局所和計算ユニットの一部の演算器を使わないようにプロセッサを再構成するだけで，容易に対処が可能である．また，本プロセッサの構成から，演算器の数を増やすことは容易であり，5×5pixels よりも大きな局所領域に対しても，予備の演算器を用意しておき，局所領域の大きさに合わせてプロセッサを再構成することによって，十分対処可能である．さらに大きな局所領域に対しては，複数回に分けて計算を行うことでも対処可能である．まず，計算可能な列の分だけ先に計算してしまい，その途中結果を増設したキャッシュに保持しておく．次に残りの列の分と保持しておいたキャッシュを使用する，というような処理になる．

謝辞

本研究を行うに当たり御指導御鞭撻を戴いた北陸先端科学技術大学院大学 堀口 進 教授に深く感謝致します.
また，サブテーマで御指導を戴いた小谷 一孔 助教授に厚く御礼申し上げます.
林亮子助手には，貴重なコメントを戴き深く感謝申し上げます.
多くの有益なご指摘を戴きました福士助手に深く感謝いたします.
最後に，日頃よりお世話になった堀口研究室の皆様に厚く御礼申し上げます.

参考文献

[1] B. K. P. Horn and B. G. Schunck "Determining Optical Flow" Artificial Intelligence, Vol. 17, pp.185-203, 1981.

[2] B. Lucas and T. Kanade, "An Intensice Image Registration Technique with An Applicaton to Stereo Vision" Proc. of DARPA Image Understanding Workshop, pp.121-130, 1981.

[3] J. L. Barron, D. J. Fleet and S. S. Beauchemin, "System and Experiment Performance of Optical Flow Techniques" Int'l Jounal of Computer Vision, 12:1, pp.43-77, 1994.

[4] "http://www.incx.nec.co.jp/imap-vision/" 2000.

[5] Miguel V. Correia and Aurelio C. Campilho, "Real-Time Implementation of an Optical Flow Algorithm" Proc. of ICPR'02 1051-4651/02, 2002.

[6] 三池 秀敏，古賀 和利他，"パソコンによる動画像処理" 森北出版株式会社，1993.

[7] 浅田 稔 "ダイナミックシーンの理解" 電子情報通信学会編

研究業績