命令型プログラミング言語におけるプログラム可逆化

M2019SE009 渡邉将匡
指導教員：横山哲郎

1　はじめに

PDES と呼ばれるシミュレーション技術がある．PDES は並列にシミュレーションを実行することによって実行の高速化を行っている．PDES は並列に動作するため，各プロセス間で同期を行う必要がある．PDES が楽観的に同期を行う場合，計算が正しく行われないことがあり，何らかの手段で計算を正しかった状態までさかのぼる必要がある．そのための手法として，可逆計算を用いた手法が注目されている [1].

従来の手法では，シミュレーションプログラムの状態を保存していた．この手法では状態の保存に多大なオーバヘッドがかかり，メモリ効率が低下してしまう．対して，可逆計算を用いた手法では，状態の保存と比べてメモリ使用量が小さくなる．したがって，オーバヘッドの減少と共にメモリ効率も上がり，従来の手法に比べ実行速度が高速になる [2].

しかし，PDES に可逆計算を用いるには，シミュレーションプログラムが可逆である必要がある．通常のプログラムはほとんどが非可逆であり，プログラムの可逆化手法は Perumalla の C プログラムの可逆化手法 [3] が知られている．

しかし，Perumalla の手法には正しく値を復元できない可逆化の定義が含まれており，また，必要以上の情報の保存を含む可逆化の定義も含まれている．正しく値を復元できないと，予期せぬエラーやバグが発生する可能性がある．情報が必要以上に保存されると，オーバヘッドが増加し，実行性能に悪影響を及ぼす可能性がある．

C プログラムの値が正しく復元できると，予期せぬエラーの発生やバグの発生を防ぐことができる．また，C プログラムの可逆化を効率的に行えるようになると，メモリ使用量や可逆化後のプログラムのオーバヘッドを減らすことができる．したがって，この論文では Perumalla の可逆化の定義を改善した可逆化手法を提案する．また，提案した手法と Perumalla の手法を比較し，評価を行う．

2　関連研究

この章では，可逆計算を応用した関連研究を紹介する．

2.1　PDES

PDES とは，離散事象シミュレーション (DES) を並列に実行するシミュレーション技術である．DES は工場生産や計算機ネットワーク，交通など，様々なシミュレーションに使用されている．PDES は DES を論理プロセス (LP) と呼ばれる単位に分割し，並列実行することで計算速度を向上させている．しかし，並列実行をしているため，各 LP 間の同期をどのように行うかという問題がある．

問題の解決手法は大きく分けて 2 種類存在し，問題の解決手法によって PDES は保守的と楽観的の 2 種類に分類できる．保守的な PDES では，シミュレーションの適用範囲が静的な通信グラフを持つモデルに限定される．楽観的な PDES では，計算が正しく行われなかった場合，計算が正しかった状態までさかのぼる必要がある．

PDES は計算をさかのぼるためのメモリ操作とメモリに記憶しておく情報量がボトルネックとなる場合があり，メモリ使用量の最適化は高速なシミュレーションのために必要である．

可逆計算を用いた手法では，シミュレーションプログラムが可逆である必要があるため，非可逆なシミュレーションプログラムは可逆なシミュレーションプログラムに変換しなければならない．

シミュレーションプログラムの変換を自動で行うことで，プログラム変換時に変換のミスをなくし，コストを下げることが可能である．ここで，変換規則の最適化を行うことで，シミュレーションの高速化を行うことができる．また，値を正しく復元できなければ実行をさかのぼることができないため，変換規則は正しく可逆化が行える必要がある．

2.2　暗号化アルゴリズム

暗号化は第三者への通信内容の秘匿化など，セキュリティに使用されている．暗号化した情報は必ず復号する必要があるため，暗号化アルゴリズムには対応する復号アルゴリズムが必須である．また，通信セキュリティに使用されるため，暗号化，復号化は高速であることが求められる．

暗号化アルゴリズムを実装したプログラムが可逆であれば，プログラムを逆実行すれば復号ができるようになるため，復号アルゴリズムを別のコードで実装する必要がなくなる．また，暗号化アルゴリズムは 1 つの平文に対し，1 つの暗号文が対応するため可逆である．したがって，暗号化アルゴリズムは効率的な可逆プログラムが作成可能である．

暗号化アルゴリズムの実装は，個々のステップが非可逆である場合がある．したがって，汎用的な可逆化手法を単に適用しただけでは最適なプログラムとならない．しかし，暗号化アルゴリズムは可逆であるため，暗号化アルゴリズムのプログラムの変換に特化した可逆化手法を開発することで，従来の暗号化プログラムから効率の良い可逆暗号化プログラムを自動生成することができる．

2.3 量子計算

量子計算とは，古典的な計算システムでは実現できない計算速度を実現できる計算システムである．量子計算には，古典的な計算システムで用いる通常のビットとは違う，量子ビットと呼ばれるビットを使用する．量子ビットの演算はすべて可逆であるため，量子ビットを用いた計算は可逆でなければならない．

プログラムの可逆化手法を用いることで，現在のプログラム資産を量子計算へ適用する際のコストを下げることができる．また，最適化された変換規則を用いることで，効率の良い量子計算プログラム作成の手助けになる．

3 準備

この章では，プログラムの可逆化のために必要な用語を定義，説明する．

3.1 プログラムの可逆性

可逆性をもつプログラムとは，1 つ前の状態がたかだか 1 つであるプログラムである．したがって，可逆性をもつプログラムは状態を実行の逆方向に辿ることができる．

3.2 プログラムの可逆化

プログラムに可逆性をもたせることが，プログラムの可逆化である．プログラムに可逆性をもたせるためには，プログラムの実行後にプログラムの逆実行が行えなければならない．プログラムの逆実行とは，プログラムの実行で変更された内部状態を実行前の状態に戻すような実行である．

C 言語のプログラムは逆実行ができないため，可逆化を行うために元のコードを逆実行が可能なコードに変換する必要がある．逆実行を可能にするためには，通常の実行を行うコードと逆実行を行うコードの 2 つのコードが必要となり，通常の実行の最中に破棄されてしまう制御情報などの情報を記憶する必要がある．したがって，逆実行に必要な情報を記憶し通常の実行を行うコードと逆実行を行うコードを元のコードから生成することにより，プログラムの逆実行が可能になる．

ここでは，通常の実行と情報の記憶を行うコードを生成することを順方向変換，記憶された情報を使用し逆実行を行うコードを生成することを逆方向変換とする．また，$\mathcal{T}[\![]\!]$ を順方向変換，$\mathcal{R}[\![]\!]$ を逆方向変換とする．

3.3 情報の記憶

プログラムの可逆化のためには，通常なら破棄されてしまう情報を記憶する必要がある．また，逆実行の際に記憶された情報を読み出す必要がある．したがって，情報を記憶するためのデータ構造と，記憶と読み出しを行う `SAVE()` マクロ，`RESTORE()` マクロを定義する．

まず，情報を記憶するためのデータ構造はスタックを使用する．なぜなら逆実行を行うためには，記憶された情報を逆順に取り出す必要があるからである．次に，`SAVE(x)` マクロは，変数 `x` の値をメモリに記憶するマクロである．記憶するビット数は変数自身の型のサイズとなる．最後に，`RESTORE(x)` は変数 `x` に記憶された情報を読み出すマクロである．読み出すビット数は変数自身の型のサイズとなる．また，`SAVE()` マクロと `RESTORE()` マクロはメモリを用いて情報を管理する．

4 以前の手法の修正

Perumalla の手法では，副作用をもつ演算は表 1 のように可逆化が定義されている．

表 1 Perumalla の可逆化の定義 [3]

元のコード	順方向変換	逆方向変換
`++`（前置）	`++`（前置）	`--`（後置）
`++`（後置）	`++`（後置）	`--`（前置）
`--`（前置）	`--`（前置）	`++`（後置）
`--`（後置）	`--`（後置）	`++`（前置）
`+=`	`+=`	`-=`
`-=`	`-=`	`+=`
`*=`	`*=`	`/=`
`/=`	`/=`	`*=`
`^=`	`^=`	`^=`

これらはすべて，演算子を変換するだけで値を復元できる演算である．表 1 には記述されていない副作用をもつ演算子は，すべて破壊的な演算であり，順方向変換では演算の直前に，`SAVE()` マクロを用いて左辺の変数の値を記憶するコードに変換される．逆方向変換では，元の演算は `RESTORE()` マクロを用いて左辺の変数の値を復元するコードに変換される．

Perumalla の手法の複合代入演算の可逆化では，右辺に左辺の変数が記述される可能性が考慮されておらず，値が復元できない可能性があった．更に，演算子`*=`では右辺の値が 0 である場合は正しく値を復元できず，演算子`/=`では余りがでるような値の組み合わせである場合は正しく値を復元できない．また，変数の型と演算子の種類によっては桁あふれなど，C 言語では未定義の動作を引き起こすことがあった．したがって，正しく値を復元できるように可逆化の定義を修正した手法が表 2 となる．ここで提案する可逆化では前提条件を追加し，前提条件によって異なる変換を行う．前提条件を満たす場合，元の演算は演算子を変換するだけで値を復元できる演算となるため，Perumalla の可逆化を使用できる．満たさない場合，元の演算は破壊的な演算となるため，破壊的な演算と同様の変換を行う．他の演算子も同様に前提条件を追加し，表 2 のように可逆化手法を提案できる．この可逆化手法の前提条件は，構文解析で確認可能である．

表 2 提案手法

元のコード	前提条件	順方向変換	逆方向変換
x += e	左辺の型が桁あふれを起こさない 左辺と同じ変数が右辺に出現しない	x += e	x -= e
	上記を満たさない	SAVE(x) x += e	RESTORE(x)

5 最適化手法

この章では，Perumalla の手法よりもメモリに記憶する情報を減らすことができる可逆化手法を提案する．

5.1 局所変数を記憶しなくて良い場合

通常，局所変数の値はどこかで記憶されなければならない．なぜなら，局所変数の値は関数の終了時に破棄されてしまうからである．しかし，局所変数の値を記憶しなくても，値を復元できる場合がある．それは，局所変数の初期値が別の変数であり，初期値から変更がない場合である．この場合，局所変数の値は別の変数として記憶されているため，値が復元可能となる．この手法を関数の引数に適用したものが表 3 となる．引数の初期値は別の変数が代入されているため，順方向変換，逆方向変換でコードの変換を行わない．変換を行わないことにより，変数の代入によって値を復元している．この可逆化は，ポインタ演算がないことが前提となる．なぜなら，ポインタ演算を含む場合，変数の依存関係の解析が困難になってしまうからである．ポインタ演算の有無と副作用をもつ演算の有無は構文解析で確認可能である．Perumalla の手法では，すべての局所変数はスコープを抜ける直前に値が記憶されている．提案手法により，記憶しなくても良い局所変数は記憶せずに済むようになった．

また，Perumalla の手法では，ループを逆実行するために追加の変数を用意し，繰り返し回数を記憶している．しかし，この手法では繰り返し回数が変数に記憶されるループも，もともと存在する変数とは別に繰り返し回数を記憶するための変数を用意する必要がある．これは，繰り返し回数を二重に記憶してしまっており，本来記憶しなくても良い変数を用意しない最適化が可能である．この最適化を適用した手法が表 4 になる．ただし，繰り返し回数を記憶する変数は 0 から始まり，1 回の繰り返しごとに値が 1 ずつ増加することが前提となる．

これらの可逆化手法が使用できるコード例は，例えば関数の引数がループの繰り返し回数になっている場合である．このようなコードは行列計算で特に頻出する．

5.2 制御情報の圧縮

Perumalla の手法では，if 文などの分岐を可逆化するために追加の変数を用意し，制御情報として分岐ごとに値が設定され，分岐の合流直後に SAVE() マクロを用いて記憶される．

確率に偏りのある分岐では，同じ分岐が繰り返された際，同じ分岐先を選ぶことが多くなる．したがって，記憶される制御情報は同じ値が長く続くことになる．このように同じ値が長く続く場合，可逆圧縮技術の 1 つである算術符号を使って情報を圧縮することができる．このとき，確率の偏りが大きいほど圧縮率も高くなる．また，情報の圧縮は可逆的に行われるため，プログラムの可逆性を損なうことはない．

算術符号は静的に記号の生成確率を求め，符号化を行うが，動的に生成確率を求めて符号化を行う手法もある．そのため，事前に繰り返し回数が分からないループにもこの手法は適用できると予想される．このような分岐が繰り返される例は，再帰関数やループ中の正常処理分岐など多岐にわたる．

6 評価

評価は，Perumalla の手法とこの論文で提案した手法を実際のコードに適用し，SAVE() マクロの呼出し回数を比較する．なぜなら，メモリ操作は CPU 演算と比べて比較的低速であり，オーバヘッドとなり，性能に影響を与える大きな要因となるからである．また，プログラムの可逆性により RESTORE() マクロは SAVE() マクロと同じ実行回数になるため，比較する必要はない．SAVE() マクロは情報の記憶にメモリを使用するため SAVE() マクロの実行回数の解析により，メモリの使用量を解析できる．

比較に使用するプログラムは，確率に大きな偏りがある swicth 文をループ中に実行するプログラムと文献 [1] の三重ループのプログラムである．swicth 文のプログラムは制御情報の圧縮を比較することに適している．また，三重ループのプログラムは実際にベンチマークに用いられた行列積のプログラムである．これらのプログラムを手動で作成し，それぞれの手法を適用してメモリ使用量を手動で解析することで評価を行う．

7 むすび

今回，Perumalla の可逆化手法を修正し，最適化された手法を提案した．また，新たな制御情報の圧縮手法の提案をした．これによって，特定の場合でも正しく値を復元できるようになった．また，特定の場合で不要になる制御情報の記憶をなくし，オーバヘッドを減らすことができた．さらに，確率に偏りがあるループ中の分岐の制御情報を圧縮することができれば，メモリ使用量を減らすことができ

表 3 関数の引数を記憶しない可逆化

	元のコード	順方向変換	逆方向変換

Perumalla の手法	void f(int x)
	{
	s
	}
		void f(int x)
		{
		$\mathcal{T}[\![s]\!]$
		SAVE(x)
		}
			void f(int x)
			{
			RESTORE(x)
			$\mathcal{R}[\![s]\!]$
			}

提案手法		void f(int x)
		{
		$\mathcal{T}[\![s]\!]$
		}
			void f(int x)
			{
			$\mathcal{R}[\![s]\!]$
			}

表 4 ループ回数が変数に保存される可逆化

	元のコード	順方向変換	逆方向変換

Perumalla の手法	{
	int i;
	i=0;
	while(i<n)
	{
	s
	i++;
	}
	}
		{
		int i;
		int c=0;
		i=0;
		while(i<n)
		{
		c++;
		$\mathcal{T}[\![s]\!]$
		i++;
		}
		SAVE(c);
		SAVE(i);
		}
			{
			int i;
			int c;
			RESTORE(i)
			RESTORE(c)
			while(c)
			{
			--i;
			$\mathcal{R}[\![s]\!]$
			--c;
			}
			}

提案手法		{
		int i;
		i=0;
		while(i<n)
		{
		$\mathcal{T}[\![s]\!]$
		i++;
		}
		SAVE(i);
		}
			{
			int i;
			RESTORE(i)
			while(i)
			{
			--i;
			$\mathcal{R}[\![s]\!]$
			}
			}

る．今後は提案した手法をプログラムに適用し，以前の手法と比較して評価を行う．

参考文献

[1] Schordan, M., Oppelstrup, T., Thomsen, M.K. and Glück, R.: *Reversible Languages and Incremental State Saving in Optimistic Parallel Discrete Event Simulation*, pp.187–207, Springer-Verlag (2020).

[2] Carothers, C.D., Perumalla, K.S. and Fujimoto, R.M.: Efficient Optimistic Parallel Simulations Using Reverse Computation, *ACM Trans. Model. Comput. Simul.*, Vol.9, No.3, p.224–253 (1999).

[3] Perumalla, K.S.: *Introduction to Reversible Computing*, pp.147–176 (2013).