可逆 C コンパイラの設計と実装

2016SE082 外川 淳平
指導教員: 横山 哲郎

1 はじめに

あいうえおかきくけこさしすせそたちつてとはひふへほま

1.1 研究の背景

あいうえおかきくけこさしすせそたちつてとはひふへほま

可逆計算とは，計算過程において直前と直後の状態が一意に決まる計算である．可逆なプログラムは可逆計算のみで実行されるプログラムであり、逆実行が可能である．プログラムの逆実行は，投機的実行でのロールバックの代用やデバックの効率化に使用することができる．特に並列離散事象シミュレーションにおいて効果的なことがわかっており，レンセラー大学の投機的実行を含むシミュレーションシステム（ROSS）との並列実行が行われている．また，可逆な論理演算においては，熱放出の下限が存在しないことがわかっている．一般的なプログラムの多くは破壊的代入などを行うため非可逆である．可逆計算を取り入れる方法として，計算機に可逆性を与える方法，言語そのものに可逆性を与える方法，そしてプログラムに対して可逆性を与える方法が知られている．それぞれ，可逆計算機，可逆言語，プログラム可逆変換器とよばれ可逆チューリングマシン，janus，Reverse C Compiler（RCC）などが存在する．RCC は一般的な言語である C 言語を対象としたプログラム可逆変換器であり，C 言語の言語機能全体を使用できる．しかし，可逆化を行うにあたって多くのオーバヘッドが必要となる．

1.2 研究の目的

今回提案するのは自動変換で生成されるプログラムの効率化である．プログラムの自動変換では，Perumalla らが開発した可逆 C コンパイラ（RCC）が有名です．RCC での変換に必要なオーバヘッドをより減らすことを目標とし，これにより自動生成されたプログラム，元のプログラム，提案する方法での比較を行い証明する．

2 関連研究

2.1 Reverse C Compiler（RCC）

RCC は C 言語を対象とする可逆変換器であり，C 言語で書かれたプログラムを入力とし，前進実行用プログラムと逆実行用プログラムを出力とする．この二つのプログラムを C コンパイラに入力することで可逆プログラムを得ることができる．RCC は C++言語の言語機能全体が使用でき，レンセラー大学の投機的シミュレーションシステム（ROSS）との並列実行が可能である．しかし，実行時のオーバーヘッドが大きくなってしまう．

2.2 可逆論理演算

Landauer は非可逆な論理演算は必ず熱放出を伴い，可逆な論理演算には熱放出の下限がないことを指摘している．また，非可逆な論理演算に可逆性を与える方法として Landauer 法が知られている．Landauer 法とは非可逆な論理演算を行う際，追加の情報を記録することで情報消失をなくす方法である．

2.3 可逆チューリングマシン

可逆チューリングマシンとは，常に状態の直後が唯一であるチューリングマシンである．チューリング完全であることが分かっており，チューリングマシンの計算のすべてを模倣することができる．Bennertt 法を利用することで，計算過程で発生する追加情報を削除し結果と入力のみを出力する可逆チューリングマシンを実現できる．しかし実行時間は通常の倍程度になる．

2.4 可逆プログラミング言語

可逆プログラミングによって書かれたプログラムは常に可逆である．例として，janus や R などが存在する．

Janus では，破壊的な代入などを認めず，一般的な言語では非可逆な，繰り返し文や分岐文も可逆であり，簡単に可逆プログラムを作成することが可能である．しかし，Janus でプログラムを作成する場合，非可逆な言語で発展してきた技術を使用することができない場合がある．

2.5 プログラム逆実行の応用

プログラム逆実行は，ロールバックやデバックへの応用が知られている．Perumalla らの研究 [3] によって投機的実行を含む並列シミュレーションにおいて，逆実行を用いたロールバックを使用することで実行速度の改善が可能であることが分かった．従来の状態保存と比べ逆実行を取り入れたほうが，前進実行におけるオーバーヘッドを小さくすることが可能である．デバックへの応用では，バグの発生個所を逆実行で探すことで，実行回数の削減が期待されている．

3 RCC の設計

3.1 RCC の設計準備

本章では Haskell を用いて設計した RCC の開発の解説を行う．RCC は，C 言語で書かれたプログラムの不可逆部分を可逆にするものである．今回は繰り返し文，分岐文のみを対象として Perumalla らの RCC[4]を基にして可逆化するコンパイラを設計する．また，開発環境としては BNFC を使用した．

3.2 逆実行用プログラム

逆実行の時に使用するプログラムを逆実行用プログラムという．RCC ではこれを自動生成する．建設的な演算子とその対となる演算子を指定し自動生成を行う．例え

ば，C 言語での演算子*=の対になるの演算子/=である．この自動生成による逆計算では丸目誤差などが原因で実行時に必ずしも正しく計算されるとは限らない．

3.3 RCC の設計

RCCを用い図1に示すよう可逆プログラムを生成する．元プログラムを RCC へ入力し，前進実行用プログラムと逆実行用プログラムを生成する．その後，C コンパイラに二つを入力することで可逆プラグラムを得る．

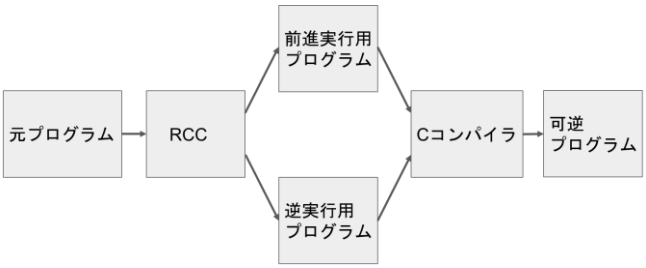

図 1 RCC を用いた可逆コンパイル［4］

可逆性を与えるためにまず，SAVE とRESTORE のマクロを用意する．SAVE が非可逆部分の情報消失を記憶し，RESTORE でそれを呼び出し逆実行を行うことができる．そして下記の図2，図3のように分岐文 switch と繰り返し文 while．元プログラムが C 言語で書かれたプログラム，変換プログラムが可逆プログラム，逆プログラムが逆実行用のプログラムである．

元プログラム	前進実行用 プログラム	逆実行用 プログラム

switch(expr)
{
case v1:
s1
break;
case v2:
s2
/*fall-thru*/
case v3:
s1
break;
default:
s4
break;
}
	{
	char c = 0, d = 0;
	switch(expr)
	{
	case v1: c=1; d++;
	s1
	break;
	case v2: c=2; d++;
	s2
	/*fall-thru*/
	case v3: c=3; d++;
	s1
	break;
	default: c=0; d++;
	s4
	break;
	}
	SAVE(c);
	SAVE(d);
	}
		{
		char c , d ;
		RESTORE(d);
		RESTORE(c);
		switch(c)
		{
		case 0:
		rs4
		if(--d <= 0) break;
		case 3:
		rs3
		if(--d <= 0) break;
		case 2:
		rs2
		if(--d <= 0) break;
		case 1:
		rs1
		}
		}

図 2 switch 文［4］

元プログラム	前進実行用 プログラム	逆実行用 プログラム

while(condition)
s
	{
	int c = 0;
	while(condition)
	{
	c++;
	s
	}
	SAVE(c);
	}
		{
		int c = 0;
		RESTORE(c);
		while(c > 0)
		{
		rs
		--c;
		}
		}

図 3 while 文［4］

while 文の可逆プログラミムでは追加情報として変数cを記憶している．生成された逆プログラムを使用する際，変数cがあることで一意に実行が可能となっている．また switch 文では追加情報として変数 c と変数 d を記憶している．ここで追加情報を記憶するために使用する変数はそれぞれがそれ以前に使用されていない変数である．

4 RCC の効率化

4.1 RCC の実装

現在一部分のみ，実装が可能である．今後，実装範囲を広げる必要がある．そして，元プログラムからのオーバヘッドを確認する．

4.2 効率化の指標

効率には様々な指標が存在する．RCC を用いた投機的実行を含む並列シミュレーションにおいて，スループットの向上が期待されることがわかっている．しかし，ロールバックでは状態保存と比べオーバーヘッドが大きくなることがあることもわかっている．また，建設的な計算には効果的で破壊的計算には効果が薄いことがわかっているが，今回は状態保存と比べ前進実行において効果的というところに焦点をあて，より効率化を目指す．状態保存を含めない場合，元のプログラムに比べ可逆プログラムはオーバーヘッドがかかる．本研究ではこのオーバーヘッドを指標として用いる．

5 おわりに

本研究は第一に RCC を開発と実装を目標とする．第二にの RCC 効率化を目標とする．特に可逆プログムへのオーバヘッドの減少を対象としている．今後の課題として，RCC の実装範囲の拡大，RCC の変換規則の改善によるオーバヘッドの減少がある．

参考文献

[1] Hänninen, I., Lu, H., Blair, P.E., et al.: Reversible and Adiabatic Computing: Energy-Efficiency Maximized, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface, pp.341–356 (2014).

[2] Schordan, M., Oppelstrup, T., Jefferson, D., et al.: Generation of Reversible C++ Code for Optimistic Parallel Discrete Event Simulation, New Generation Computing, Vol.36, No.3, pp.257–280 (2018)

[3] Carothers, D.C., Perumalla, S.K., Fujimoto, M.R.: Efficient Optimistic Parallel Simulations Using Reverse Computation, ACM TOMACS, Vol.9, No.3, pp.224–253 (1999)

[4] Perumalla S.K.,: Introduction to Reversible Computing, CRC (2013)