実行時コード最適化の自動化 *

7B－8

杉田祐也† 原田賢一‡
慶応義塾大学理工学部§

1 はじめに

実行時のコンテキストについてプログラムを特化することによって、プログラムの性能を飛躍的に向上できる場合がある。実行時に値が決定される定数や、実行中の限られた間だけ有効な値を用いたプログラムでは、それらに依存した計算や制御構造を除去することができる。近年では、**部分評価** (partial evaluation)[1] の技法を利用するなどしてプログラムを自動的に特化し、**実行時コード生成** (run-time code generation)[2] の技法を使い、そのようなコードを生成することによって実行効率を改善する処理系 [3, 4, 5] が提案されている。

これらの処理系では、コードを特化するための解析の多くを静的に行うことによって、実行時のコストを小さくしているが、コード生成のコストは実行時間に比べて小さくないので、全体として性能を向上するためには生成したコードが繰り返し実行されなくてはならない。したがって、ある実行時コード生成が性能を改善できるかどうかを判定するためには、プログラムの特化による実行時間の短縮と、そのプログラムが使用される回数が分からなくてはならない。実行時間の短縮は、**速度向上解析** (speedup analysis)[1, 5] によって得ることができる。しかし、プログラム中のある部分が実行される回数を解析することは、一般的には不可能である。そのために、プログラマが特化すべき部分を指定しなくてはならなかったり、コード生成のコストが償却できないようなコード生成をしてしまい、全体としての性能が低下してしてしまったりする。

本研究では、インクリメンタルに実行時コード生成を行なうことによって、複雑な解析を必要としないで、従来よりも安全に実行時コード最適化の方法を提案する。はじめにコストの小さい方法を用いて実行時間の短縮をし、さらに何回も実行された場合には、実行時コード生成を用いて、高度に最適化されたコードを生成する。はじめの最適化によって得られる実行時間の短縮を利用してコード生成をするために、従来よりも全体として効率が低下する場合を少なくすることができる。

2 インクリメンタルなコード生成

対象言語には、手続き型言語のサブセットを選んだ。このサブセットは配列変数を含むが、ポインタは含まないものとする。

本研究では、特にループ内にあるループに対する特化を扱う。すなわち、繰り返し使用される可能性のあるループに対してループアンローリングを行う。特化の対象となるループは、ループ条件がそのループを含むループにおけるループ不変値だけに依存しているループとする。このようなループは、それを含むループ内に制御がある間は、同じ回数だけ繰り返すのでアンロールした結果は、その間は使用できる。この場合、外側のループのループ不変値だけでなく、特化の対象となるループ変数の値についても特化されたコードを生成することができ、高度な最適化が可能である。しかし、ループの繰返し数が大きくなるに従って、ループをアンローリングしたコードを生成することのコストは、大きなものとなる。このため、アンロールしたコードの使用回数が少ない場合には、大幅な性能の低下となる。

そこで、一度にループ全体のコード生成を行なわずに、そのループが実行される回数に応じて、何回かに分けて展開をしていく方法を提案する。ここで、ループが実行される回数とは、特化の対象となるループが使用される回数、すなわち、外側のループの繰り返される回数のことを示している。

はじめに、少ない繰返し数の分のコードだけを生成する。このコード実行から得られる実行時間の短縮によって、コード生成に要したコストを償却することができた場合に、残りのコードを生成する。このような方式を用いることによって、ループが実行されるほど、多くの繰り返しがアンロールされる。よく実行される部分はコー

---

*An Approach to Automatic Run-time Code Optimization

†SUGITA, Yuuya
‡HARADA, Ken'ichi
§Dept. of Computer Science, Keio University, 3-14-1 Hiyoshi, Kouhoku-ku, Yokohama 223, JAPAN

ド生成がすすみ、あまり実行されない部分は、ほとんどコード生成が行われない。また、ループの停止性についても複雑な解析を行う必要はない。停止しないループは、はじめの何回かの繰返しがアンロールされても、その後にコード生成されることはないので、コード生成器が停止しないことはない。

コードを生成するタイミングと、各アンローリングで生成されるコードの量は速度向上解析を応用することによって、実行時間の短縮と、コード生成にかかる時間から静的に決定できるので、実行中にプロファイリングをする必要はない。

3 初期の最適化

以上のような方法によって、意味のないコード生成を減らすことが可能であるが、問題点がある。まず、一番はじめに行なうコード生成のコストが償却できない状況が多発すると、大幅な性能の低下が起こる可能性がある。これに対処するために、はじめにアンロールする繰返し数を減らすと、それを実行しても実行時間の短縮があまり得られないので、ループのアンロールが完了するまでにループが何回も実行されなくてはならない。この場合、性能の向上はあまり望めない。

ここで、その効果がループの全ての繰返しに及ぶような最適化があると都合がよい。そのような最適化は、アンロールしなくてはならない量に比例して効果がでるので、アンロールの完了を早めることができる。そこで、**メモイゼーション** (memoization) を使い、アンローリングする場合に除去できる計算の結果をそれぞれの繰返しについてキャッシュし、利用する。この場合には実行時コード生成に比べてわずかなコストで、実行時間の短縮が可能である。また、実行時コード生成によるアンローリングで効果がでる場合と、メモイゼーションで効果がでる場合は似ているので、アンロールの前段階としての性質はよい。

メモイゼーションによって得られる実行時間の短縮が、実行時コード生成を行うのに十分になるだけ、プログラムが繰り返し使用されれば、実行時コード生成を開始する。こうすることによって、実行時最適化の開始時にかかるコストを低く抑えることができるので、安全性が高まり、またコード生成を素早く行えるので、性能改善の機会を逃す可能性が減少する。

4 結論

実行時コード生成の処理系を自動化することを目的として、比較的安全に性能の改善が可能となる方法を提案した。現在、処理系を実装し、評価中である。

5 謝辞

本研究に貴重な助言をくださった、東京大学大学院総合文化研究科増原英彦氏と慶応義塾大学大学院理工学研究科滝本宗宏氏に感謝します。

参考文献

[1] N. Jones, C. G. and P.Sestoft: *Partial Evaluation and Automatic Program Generation*, Prentice Hall (1993).

[2] David Keppel, S. J. E. and Henry, R. R.: A case for runtime code generation, Technical Report 91-11-04, Department of Computer Science and Engineering, University of Washington (1991).

[3] Joel Auslander, M. P.: Fast, Effective Dynamic Compilation, *ACM Conference on Programming Language Design and Implementation* (1996).

[4] Leone, M. and Lee, P.: Optimizing ML with runtime code generation, *ACM Conference on Programming Language Design and Implementation* (1996).

[5] 藤波順久: オブジェクト指向言語の実行時最適化, 日本ソフトウエア科学会第 *12* 回大会論文集, pp. 245–248 (1995).

[6] Todd B. Knoblock and Erik Ruf: Data specialization, *ACM Conference on Programming Language Design and Implementation* (1996).