1E−8 並列リダクションにおける引数評価時の決定法

干場美佳子
日本アイ・ビー・エム株式会社 東京基礎研究所

はじめに

関数型プログラミング言語は、副作用がないこと、部分式の独立性から、引数評価を同時に行なうことによって、容易に並列実行できる。また、完全遅延評価を行なう場合には、strictness anlysis [1] によって評価を行なう引数を決定する。しかし、strictness anlysis だけでは効果的な並列実行には不十分である。

ここでは実行方法としてコンビネータによるリダクションを用い、引数の並列処理のための解析方法を提案する。

コンビネータとリダクション

コンビネータとしてスーパーコンビネータ [2]、リダクションの方法としてグラフリダクション [3] を用いる。スーパーコンビネータは多引数関数を複数のコンビネータに分解し、並列度を増加させる。

スーパーコンビネータは関数定義中から自由変数を含まない最大の部分式 (mfe) をパラメータとしてとり出すことにで得られ、完全遅延評価が可能である。パラメータとしてリダクション可能な mfe(mrfe) を取り出しても、この効果は変わらない。

$$f = \lambda x.\lambda y.\mathit{if}\ (=\ x\ y)\ (1+\ x)\ y$$

mfe
$$\mathrm{F}\ \ x \qquad = \mathrm{F0}\ (=\ x)\ (1+\ x)$$
$$\mathrm{F0}\ \ eqx\ px\ y\ = \mathrm{IF}\ (eqx\ y)\ px\ y$$

mrfe
$$\mathrm{F}\ \ x \qquad = \mathrm{F0}\ x\ (1+\ x)$$
$$\mathrm{F0}\ \ x\ px\ y\ = \mathrm{IF}\ (=\ x\ y)\ px\ y$$

式の評価は、二進木で表された式をコンビネータの書き換え規則にしたがって変形することで実行される。最外最左のコンビネータの規則を適用し、その部分式のルートセルを書き換えることで完全遅延評価を行なう。

並列リダクション

数値演算子のように引数評価が必要な場合、引数がリダクション実行後に必ず評価される場合について、引数の評価をを並列に実行する。つまり、複数の strict な引数をリダクション実行前に評価し並列性を得る。前述のコンビネータ F1(mrfe) では、リダクション実行前に x,y の2つの引数が評価可能である。

しかし、単に strict な引数をすべて事前評価したのでは最大の並列度は得られない。多引数関数は複数のコンビネータに分解されるために引数の評価が分散し、並列実行が起こらない場合がある。

分割征服法による和は、＋の2つの引数の同時評価を期待している。

$$sum0 = \lambda xy.\mathit{if}\ (=\ x\ y)\ x\ (sum2\ x\ y\ (\mathit{floor}\ (+\ x\ y)\ 2))$$
$$sum2 = \lambda xym.\ +\ (sum0\ x\ (1+\ m))\ (sum0\ m\ y)$$

この定義から生成されるコンビネータは、(1)

$$\mathrm{SUM0}\ a_{01} \qquad = \mathrm{SUM1}\ (\mathrm{SUM2}\ \ a_{01})\ a_{01}$$
$$\mathrm{SUM1}\ a_{11}\ a_{12}\ a_{13}$$
$$\quad = \mathrm{IF}\ (=\ a_{12}\ a_{13})\ a_{12}\ (a_{11}\ a_{13}\ (\mathit{floor}\ (+a_{12}\ a_{13})\ 2))$$
$$\mathrm{SUM2}\ a_{21} \qquad = \mathrm{SUM3}\ (\mathrm{SUM0}\ a_{21})$$
$$\mathrm{SUM3}\ a_{31}\ a_{32}\ a_{33} = +(a_{31}\ (1+\ a_{33})\ (\mathrm{SUM0}\ a_{33}\ a_{32})$$

この場合には＋の2つの引数は並列に評価される。
$sum2$ の定義中の引数の順序を xmy とすると、(2)

$$\mathrm{SUM2}\ a_{21} \qquad = \mathrm{SUM3}\ (\mathrm{SUM0}\ a_{21})$$
$$\mathrm{SUM3}\ a_{31}\ a_{32} \qquad = \mathrm{SUM4}\ (\mathrm{SUM0}\ a_{32})\ (a_{31}\ (1+\ a_{32}))$$
$$\mathrm{SUM4}\ a_{41}\ a_{42}\ a_{43} = +\,a_{42}\ (a_{41}\ a_{43})$$

ここで、a_{42} は strict であり、SUM4 の実行前に評価される。つまり、期待していた＋の引数の並列評価は、SUM4 の前後に分けらる。

こうした損失は、引数の順序だけでは解決できない。strict な式の評価は引数が揃った時点で行なわれるため、並列評価を期待される式が共通の引数を含まないとき並列性は失われる。

引数評価時を決める要素

並列性を失わずに事前評価を行なうコンビネータの引数を決定し、さらに、無駄な事前評価を無くすための解析が必要である。その要素として次の3つを考える。

- どのコンビネータをトップレベルとみなすか
- コンビネータ間の依存関係
- リダクション実行後の共有

トップレベルの設定は、そのコンビネータがトップレベルから呼ばれるのことがあるのか、他のリダクションの結果としてだけ現れるのか、を決定する。トップレベルから呼ばれないコンビネータの引数は、コンビネータ間で渡されるだけで、評価済みの場合がある。こうした引数の事前評価は並列度の増加に寄与しない。

コンビネータの依存関係をどこまで考慮するかは、strictness analysis の強さを決定する。2つ以上のコンビネータを越えて strict である引数、high order function の引数となることで strict になる引数が存在する。(1) の a_{32} は SUM0 の引数ではないが、SUM1 の引数となり、strict

---

Arguments Evaluation Stratagy of Parallel Reduction

Mikako HOSHIBA

IBM Research, Tokyou Research Laboratory, IBM Japan, Ltd.

である。(2) の a_{32} は a_{31} が (SUM0 a_{21}) であることから、(1+ a_{32}) が SUM1 の第3引数になり strict である。

共有の情報は引数をいつ評価するかを決定する。共有を起こす引数は、その後の式の評価中に同時評価され、タスクの中断を引き起こす可能性がある。つまり、strict な共有引数の事前評価は、その後の並列度の増加につながる。

実験結果

次の条件のもとで分割征服法(2)について事前評価を行なう引数を決定し、シミュレータによってステップ数を計測した。シミュレータはメモリアクセスなどの各手続きに、適切な重みづけを行なったものである。

トップレベルコンビネータ (top)
- 0 すべてのコンビネータ
- 1 関数として定義されたコンビネータ
- 2 実際のプログラム中に現れる関数に対応するコンビネータ

コンビネータ依存関係 (dep)
- 0 考慮しない
- 1 次のコンビネータへの渡される引数
- 2 high order function に対する引数の strictness

いつ引数を評価するか (tim)
- 0 すべての strict な引数を事前評価
- 1 つぎのコンビネータに渡されるだけの引数は、できるかぎり他にも strict な引数があるときに評価
- 2 共有を起こす場合に事前評価
- 3 後に同時評価を起こす可能性のある共有引数、これと並列に評価できる strict な引数を事前評価

(*sum*0 4 1) / 2 agent

a_{01}	a_{11} a_{12} a_{13}	a_{21}	a_{31} a_{32}	a_{41} a_{42} a_{43}	count	top,dep,tim
○	○ × ○	○	○ ○	○ ○ ×	10922	000 010
○	○ × ○	×	○ ○	× × ×	6917	003 013
○	○ × ○	×	× ○	× × ×	6569	001 002 012
×	○ × ○	○	○ ○	○ ○ ×	11338	101 111
×	○ × ○	○	○ ○	○ ○ ○	11377	121
○	△ × ○	×	× ○	× × ×	6569	102 112 122
○	△ × ○	×	○ ○	× × ×	6917	103 113 123
×	○ × ○	△	○ ○	○ ○ ×	12818	201 211 221
○	△ × ○	△	× ○	× × △	6519	202 212 222
○	△ × ○	△	○ ○	× × △	6892	203 213 223

○：事前評価 △：評価済み ×：評価しない

＋の2つの引数の並列評価は、a_{42} が事前評価されない場合、つまり tim=2,3 で行なわれる。

コンビネータの依存関係とトップレベルコンビネータは、評価済みの引数に関する情報を与えている。今回の例では、並列度に影響を与えていないが、引数の状態についての有効な情報となっている。

コンビネータ間の依存関係は、high order function まで解析を行なった方が並列度は増すが、strictness analysis のコストが大きくなる。したがって、解析のためには high order function を作りやすい mfe よりも mrfe が、コンビネータ生成のパラメータとして望ましい。

引数を評価するかどうかの決定は、共有と並列評価可能な引数の数による方法が効果的であることがわかる。

その他の要素

今回はコンビネータの書き換え規則に対し、引数の事前評価を決定したが、この他に並列度に影響を与える要因として次のようなものが考えられる。

書き換え規則中には含まれないが、その後のコンビネータの引数となるもの、たとえば(SUM0 e_1 e_2) の e_2 は SUM0 の引数ではないが、SUM1 の第3引数として SUM0 の実行前に評価可能である。これを SUM0 の第1引数と同時評価すれば並列度が増加する。こうした引数予備軍は多引数関数のほとんどの場合に現れる。これについても解析を行ない、strict ならば本来のコンビネータが現れる前に事前評価を行なうことで並列度の増加が期待できる。

また、コンビネータの生成も並列度を考えて行なう必要がある。多引数関数の共有されない部分式に対応するコンビネータは並列度と、リダクションのコストの比較によって生成を決定するべきである。

3引数関数 f が第1引数を持つ形で共有されないとき、第1、第2引数は同時にパラメータ化できる。

$$g = \lambda x.\lambda y.\ h\,(f\ x\ 1\ y)\,(f\ y\ 1\ y)\,(f\ y\ 1\ x)$$
$$f = \lambda x.\lambda y.\lambda z.\ exp$$

F1 $a_1 \ldots a_m\ z$ = EXP

(1) 別々にパラメータ化した場合

F x = F0 $e_1 \ldots e_i\ ex_{i+1} \ldots ex_n$

F0 $a_1 \ldots a_n\ y$ = F1 $a_1 \ldots a_i\ ey_{i+1} \ldots ey_m$

(2) 同時にパラメータ化した場合

F $x\ y$ = F1 $e_1 \ldots e_i\ exy_{i+1} \ldots exy_m$

(1) では $ex_{i+1} \ldots ex_n$ からも並列性を得られるが、(2) よりもリダクションの回数が増える。したがって、得られる並列度とリダクションのコストを比較した上で生成するコンビネータを決定すべきである。

おわりに

コンビネータリダクションで事前評価を行なう引数を決定する方法を提案した。並列処理の可能な部分をどうやって見つけし、それをどのように実行するかは、コンビネータリダクション、関数型言語だけでなく、並列処理における一般的な問題である。本稿はこの問題に対する一つのアプローチとして有効であると考える。

References

[1] Clack,C., Peyton Jones,S.L *Strict analysis - a practical approach, Functional Programming and Computer Architecture,LNCS201,Springer-Verlag,1985*

[2] Peyton Jones,S.L. *An introduction to fully-lazy super combinators, Conbinator and Functional Programming Language,LNCS242 Springer-Verlag,1985*

[3] Turner,D.A. *New Implimentation Technique for Applicative Languages, Software Practice and Experience vol.9,1979*