SS研
ポストペタアプリ性能WG資料

FUJITSU
shaping tomorrow with you

3.5 圧縮流体解析「UPACS-L」

レジスタの不足、大きなループボディに対する最適化の適用

2015年12月3日
富士通株式会社
千葉 修一

調査事項

FUJITSU

- 課題
  - レジスタの不足によりSIMD化/SWPLが適応されない
  - 現象の説明と回避方法の明確化
- 対象コード
  - 圧縮流体解析「UPACS-L」
  - cellFaceVariables（粘性項）
- コードの特徴
  - ステンシル系のアクセス、大きなループボディ
  - 配列記述を手で展開
  - 一時配列のスカラー化

対象コード

FUJITSU

■粘性項コード

blk_rhsViscous.f90：199行目～466行目

```
199   do k = isrt(3),iend(3)
200     do j = isrt(2),iend(2)
201       do i = isrt(1),iend(1)
202
203         i1 = i  + idelta(1) ; j1 = j  + idelta(2) ; k1 = k  + idelta(3)
204
205         !     xi derivative
206
207         !u_0(1:3) = blk%q(i ,j ,k ,2:4) / blk%q(i ,j ,k ,1)
208         !u_1(1:3) = blk%q(i1,j1,k1,2:4) / blk%q(i1,j1,k1,1)
209         u0_1 = blk%q(2,i ,j ,k ) / blk%q(1,i ,j ,k )
210         u0_2 = blk%q(3,i ,j ,k ) / blk%q(1,i ,j ,k )
211         u0_3 = blk%q(4,i ,j ,k ) / blk%q(1,i ,j ,k )
212         u1_1 = blk%q(2,i1,j1,k1) / blk%q(1,i1,j1,k1)
213         u1_2 = blk%q(3,i1,j1,k1) / blk%q(1,i1,j1,k1)
214         u1_3 = blk%q(4,i1,j1,k1) / blk%q(1,i1,j1,k1)
～
```

「宇宙航空研究開発機構) 高木様 圧縮性流体解析プログラム UPACS-Lコード」より

調査結果

FUJITSU

■回答

■レジスタ不足の原因となる二つの要因

1. 配列のアドレス計算を実施するために多くのレジスタを利用
2. 最内ループに対する「Invariant Code Motion」最適化の適用
   a. ループ内不変となる計算式をループ外へ移動し、ループ中の演算を削減
   b. 最内ループが動作中は、移動した演算が利用するレジスタを占有
   c. 最内ループで利用できるレジスタが少なくなる

■対処

- ループ分割することで最適化を促進することが可能
- ただし、データのローカリティ（局所性）を低下させない考慮が必要

詳細情報 (1)

FUJITSU

■FX100におけるレジスタ

- ■HPC-ACEにより整数レジスタが拡張
- ■整数レジスタは、グローバルレジスタを32→64
- ■浮動小数点レジスタは、SIMD時に128本

	FX1	京/FX10/FX100
浮動小数点レジスタ	32本	256本
整数レジスタ	160本	192本

詳細情報 (2)

FUJITSU

■アドレス計算

- ■配列次元が増えるほど多くのレジスタが必要

```
do i = isrt(1),iend(1)
  ...
  blk%q(2,i ,j ,k ) / blk%q(1,i ,j ,k )
```

アドレス計算式

```
blk%q(2,i ,j ,k )アドレス
 = blkアドレス
 + qのオフセット
 + ( 2
     + 1次サイズ * i
     + 2次サイズ * j
     + 3次サイズ * k
   ) * 配列の型サイズ
```

命令イメージ

```
load blk -> r1
add r1 + qのオフセット -> r2
mult 1次サイズ * i -> r3
mult 2次サイズ * j -> r4
mult 3次サイズ * k -> r5
add  2  + r3 -> r6
add  r4 + r5 -> r7
add  r6 + r7 -> r8
mult r8 * 配列の型サイズ -> r9
add  r2 + r9 -> アドレス
```

詳細情報 (3)

FUJITSU

■Invariant Code Motion

- ■ループの制御変数に依存しない演算をループ外へ移動
- ■演算は削減、ただし占有できるレジスタも減少

```
do i = isrt(1),iend(1)
  i1 = i  + idelta(1)
  j1 = j  + idelta(2)
  k1 = k  + idelta(3)
```

命令イメージ（最適化前）

```
LOOP_i:
  load idelta(1) -> r1
  load idelta(2) -> r2
  load idelta(3) -> r3
  add  i + r1 -> i1
  add  j + r2 -> k1
  add  k + r3 -> k1
  ...
  ++i
  条件中 LOOP_Iへ
```

命令イメージ（最適化後）

```
load idelta(1) -> r1
load idelta(2) -> r2
load idelta(3) -> r3
add  j + r2 -> k1
add  k + r3 -> k1
LOOP_i:
  add  i + r1 -> i1
  ...
  ++i
  条件中 LOOP_Iへ
```


詳細情報 (4)

FUJITSU

■アドレス計算に対する最適化

- ■Strength reduction

命令イメージ(最適化前)

```
LOOP_i:
  load blk -> r1
  add r1 + qのオフセット -> r2
  mult 1次サイズ * i -> r3
  mult 2次サイズ * j -> r4
  mult 3次サイズ * k -> r5
  add  2  + r3 -> r6
  add  r4 + r5 -> r7
  add  r6 + r7 -> r8
  mult r8 * 配列の型サイズ -> r9
  add  r2 + r9 -> アドレス
  ...
  ++i
  条件中 LOOP_Iへ
```

命令イメージ（最適化後）

```
LOOP_i:
  load blk -> r1
  add r1 + qのオフセット -> r2
  mult 1次サイズ * i -> r3
  mult 2次サイズ * j -> r4
  mult 3次サイズ * k -> r5
  add  2  + r3 -> r6
  add  r4 + r5 -> r7
  add  r6 + r7 -> r8
  ~~mult r8 * 配列の型サイズ -> r9~~
  add  r2 + r8 -> アドレス
  ...
  i+=配列の型サイズ
  条件中 LOOP_Iへ
```

詳細情報 (4)

FUJITSU

■アドレス計算に対する最適化

■Invariant Code Motion

命令イメージ(最適化前)

```
LOOP_i:
  load blk -> r1
  add r1 + qのオフセット -> r2
  mult 1次サイズ * i -> r3
  mult 2次サイズ * j -> r4
  mult 3次サイズ * k -> r5
  add  2  + r3 -> r6
  add  r4 + r5 -> r7
  add  r6 + r7 -> r8
  mult r8 * 配列の型サイズ -> r9
  add  r2 + r9 -> アドレス
  ...
  ++i
  条件中 LOOP_Iへ
```

命令イメージ（最適化後）

```
  load blk -> r1
  add r1 + qのオフセット -> r2
  mult 2次サイズ * j -> r4
  mult 3次サイズ * k -> r5
  add  r4 + r5 -> r7
  add  2 + r7 -> r6
LOOP_i:
  mult 1次サイズ * i -> r3
  add  r6 + r3 -> r8
  mult r8 * 配列の型サイズ -> r9
  add  r8 + r9 -> アドレス
  ...
  i+=配列の型サイズ
  条件中 LOOP_Iへ
```

※ 移動した演算は共通で利用可能

詳細情報 (5)

FUJITSU

■最適化とレジスタの関係

■ソフトウェアパイプラインニングはさらにレジスタが必要

■演算とレジスタのバランスが重要

■コンパイラへのフィードバック

■演算／レジスタの見積もり精度を向上

- 大局的なレジスタ見積もりを検討
- オプション、指示行などによる選択を検討
- アドレッシング方式の選択を検討

FUJITSU

shaping tomorrow with you