Cray XC40 における自動ベクトル化機能の評価

疋田 淳一 [1)], 島袋 友里 [1)], 當山 達也 [1)]

1) 京都大学 情報部

computing@kudpc.kyoto-u.ac.jp

Evaluation of automatic vectorization function in Cray XC40

Junichi Hikita [1)], Yuri Shimabukuro [1)], Tatsuya Tohyama [1)]

1) Information Department, Kyoto Univ.

概要

近年のプロセッサでは SIMD 命令の強化による性能向上が行われており、これを利用するためにはコンパイラの自動ベクトル化機能が重要である。本稿では、京都大学学術情報メディアセンターのスーパーコンピュータシステムである Cray XC40 および Netlib/Vectord ベンチマークを用いて自動ベクトル化機能の評価を行った結果について報告する。

1 はじめに

現在のプロセッサの演算性能は SIMD（Single Instruction, Multiple Data）型の命令セットによる性能向上が行われてきており、ユーザプログラムの性能向上にはプロセッサの SIMD 命令の活用が必要となる。

学術情報メディアセンター（以下「本センター」と言う）が導入しているスーパーコンピュータシステムには、Intel Xeon Phi KNL や Intel Xeon Broadwell プロセッサが搭載されており、AVX512 や AVX2 といった SIMD 命令の実装されている。これらの命令セットは、特別にチューニングすることで最大限活用することもできるが、多くのユーザの利用形態としては、コンパイラの自動ベクトル化機能により活用されるため、コンパイラの自動ベクトル化機能が優れているかどうかが、プロセッサの性能を発揮する上で重要な鍵となる。

そこで、スーパーコンピュータ関連のソフトウェア資産が集められている Netlib [1]サイトで公開されている Vectord ベンチマークを利用し、本センターのスーパーコンピュータに導入しているプロセッサおよびコンパイラの組み合わせにおける自動ベクトル化機能の評価を行った。

本稿では、検証に用いた計算機およびソフトウェア環境について紹介した後、Netlib/Vectord ベンチマークを用いた評価結果について報告する。

2 対象システムとコンパイラ

今回の評価対象とするシステムは Intel Xeon Phi KNL を搭載した Cray XC40 である。全国共同利用の設備として全国の学術研究者に対して提供しているスーパーコンピュータシステムであり、HPCI（革新的ハイパフォーマンス・コンピューティング・インフラ）や JHPCN（学際大規模情報基盤共同利用・共同研究拠点）の提供資源としても活用されている。

評価するコンパイラとして、Cray 社製の Cray Compiling Environment（以下、「Cray コンパイラ」と言う）、Intel 社製の Intel Parallel Studio XE（以下、「Intel コンパイラ」と言う）、オープンソースである GNU Compiler Collection（以下、「GNU コンパイラ」と言う）の 3 種を用いた。システムの諸元とコンパイラのバージョンについて表 1 に示す。いずれのコンパイラも Xeon Phi KNL で利用可能な AVX512 命令に対応しているバージョンである。

表 1　Cray XC40

システム	Cray XC40
プロセッサ	Intel Xeon Phi KNL 7250 1.4GHz 68Cores (Quadrant mode) 3.0464 TFlops
メモリ	DDR 4-2133 96GB MCDRAM 16GB (Cache mode)
OS	Cray Linux Environment 6.0 UP07
コンパイラ	Cray Compiling Environment 8.7.9 Intel Parallel Studio XE 19.1.2.254 GNU Compiler Collection 9.2.0

3 Netlib/Vectord ベンチマーク

3.1 ベンチマークの概要

Netlib/Vectord は、ベクトル計算機のベクトル化機能を評価するためのテストスイートとして開発されたベンチマークプログラムである。プログラムは、FORTRAN77 でコーディングされており、ベンチマークのためのドライバルーチン群から構成される maind.f と、ベクトル化機能の評価のために集められた 135 種類のループを含む loopd.f の二つのファイルで構成される。Vectord ベンチマークは、コンパイルリストやメッセージからだけのベクトル機能の評価だけではなく、ループ長（配列サイズ）10，100，1000 での実行結果および性能も検証することができる。

3.2 ベンチマークの改修

Vectord ベンチマークは 1991 年に開発されたものであるため、今回使用するにあたって評価環境に適応させるために以下 3 点の改修を行った。

１）Fortran90 への変換

プログラムを扱いやすくするために、github で公開されている FORTRAN77 から Fortran90 に変換が可能な f2f.pl プログラムを用いて自動変換を行った。一部変換に不具合がある点については手作業で修正を行った。

２）ループ長の拡張

自動ベクトル化の効果を測定するには十分なループ長も必要となるため、配列サイズを 1,000 から 10,000 に拡張を行った。

３）時間計測ルーチンの修正

Vectord ベンチマークは組み込み関数による CPU 時間の計測処理が組み込まれているが、コンパイラに依存しない経過時間の計測のために、gettimeofday ルーチンを使って経過時間を計時するよう修正した。

3.3 コンパイルオプション

各コンパイラの自動ベクトル化機能検証のために、Vectord ベンチマークの loopd.f90 に対しては、コンパイラの自動ベクトル化および最適化オプションを有効にするとともに、自動ベクトル化の対象となったループを見分けるためのコンパイルリスト、メッセージ出力のためのオプションを指定した。なお、高度な最適化の副作用を避けるためにソースコードを跨る最適化は行っていない。コンパイル時のオプションを表 2 に示す。

表 2　コンパイルオプション

コンパイラ	オプション
Intel コンパイラ	-O3 -xmic-avx512 -ip -no-prec-div -list -qopt-report=5 -qopt-report-phase=all -mcmodel=medium
Cray コンパイラ	-O3 -hpic –h msgs -ra
GNU コンパイラ	-Ofast -march=knl -ffast-math -ftree-vectorize -ftree-vectorizer-verbose=5 -fopt-info-optall-all -mcmodel=large -fpic

4 計測結果

4.1 ベクトル化されたループ数と実行時間

loopd.f90 に含まれる 135 種類のループを表 2 に示すオプションでコンパイルした結果、ベクトル化の対象となったループ数を表 3 に示す。ベクトル化の有無の判断は、コンパイル時のメッセージやコンパイルリストから目視で確認を行った。ベクトル化可能だが性能向上に効果が無いと判断されたケースや、部分的なベクトル化についてもベクトル化可能として数えている。

Cray コンパイラが最も多い 107 種であり、Intel コンパイラは 106 種、GNU コンパイラは 70 種であった。Cray コンパイラのみがベクトル化できたループ数は 3 種、Intel コンパイラのみがベクトル化したループは 7 種、GNU コンパイラのみがベクトル化したループは 1 種であった。

2014 年に当時のシステム及び導入コンパイラで同様な評価を行った際の結果[2]を表 4 に示す。2014 年の結果と比較し、Cray コンパイラは 18 ループ、Intel コンパイラは 20 ループがベクトル化の対象と扱えるよう機能強化されていることが分かる。プロセッサのアーキテクチャの進化とコンパイラの自動ベクトル化機能の判定機能が強化された結果と考えられる。GNU コンパイラは当時の評価対象ではなかったため比較データは無い。

表 3　自動ベクトル化されたループ数

	Intel	Cray	GNU
ベクトル化ループ数	106	107	70
対象外のループ数	29	28	65
当該コンパイラのみがベクトル化したループ数	7	3	1

表 4　自動ベクトル化されたループ数（2014 年調査時）

	Intel	Cray
ベクトル化ループ数	86	89
対象外のループ数	49	46
コンパイラのバージョン	14.0.3	8.3.0

次に、ベンチマークのループ長 10,000 における全ループの実行時間の合計と、ループごとに各コンパイラがベストタイム、ワーストタイムとなったループ数を集計した結果を表 5 に示す。ベンチマークの実行時間およびベストタイムのループ数の双方で Intel コンパイラが優位の結果となった。Cray コンパイラは、個々のループの性能で見ると Intel コンパイラよりも優れているループもあるが、Intel コンパイラが群を抜いて高速なループが 2 つあり、その差が全体の実行時間の差に表れた結果となった。GNU コンパイラについては、HPC 向けにチューニングされているコンパイラと比較すると大きく差がついた結果となった。

表 5　実行時間

	Intel	Cray	GNU
実行時間合計[秒]	111.4	161.6	250.0
ベストタイムのループ数	77	51	6
ワーストタイムのループ数	23	18	94

4.2 Intel コンパイラが優位なケース

Intel コンパイラの性能が特に優位なループを一部紹介する。次に示すループ s233 は、ループの分割と入れ替えによってベクトル化が可能となるコードであり、Intel コンパイラのみがベクトル化の対象としている。実行性能については、ベクトル化した Intel コンパイラが Cray や GNU コンパイラと比較し 34 倍以上の高速な結果となっている。

```
!! s233   loop interchange
!! interchanging with one of
!! two inner loops
do 10 i = 2,n
  do 20 j = 2,n
    aa(i,j) = aa(i,j-1) + cc(i,j)
  20 END DO
  do 30 j = 2,n
    bb(i,j) = bb(i-1,j) + cc(i,j)
  30 END DO
10 END DO
!cray:  22.046795 sec
!intel:  0.583825 sec
!gnu:   20.150582 sec
```

次の s275 のループについても、Intel コンパイラはループの入れ替えによりベクトルを行っており、他のコンパイラと比べて 68 倍以上の性能となっている。

```
!! s275 control flow
!! if around inner loop,
!! interchanging needed
do 10 i = 2,n
  if(aa(i,1) > 0.d0)then
    do 20 j = 2,n
      aa(i,j) = aa(i,j-1) + bb(i,j) &
                * cc(i,j)
    20 END DO
  endif
10 END DO
!cray: 16.828276 sec
!intel: 0.244440 sec
!gnu:  19.269898 sec
```

ソースコードの紹介は省略するが、ループ S234、S235 についてもループの入れ替えが必要な構造となっており、Intel コンパイラのみがベクトル化の対象としており、Intel コンパイラはループの変換を伴う最適化に強みがあると言える結果となった。

次のループ s292 については、配列の添え字に対するループ中の代入を含むコードとなっており、Intel コンパイラのみがベクトル化の対象としている。性能については、Intel コンパイラが他と比べて 5.0 倍以上の性能となっている。

```
!! s292 loop peeling
!! wrap around variable, 2 levels
im1 = n
im2 = n-1
do 10 i = 1,n
  a(i) = (b(i) + b(im1) &
       + b(im2)) * .333d0
  im2 = im1
  im1 = i
10 END DO

!cray:  0.318264 sec
!intel: 0.062691 sec
!gnu:   0.440623 sec
```

4.3 Intel コンパイラのみ遅いケース

逆に Intel コンパイラのみ性能が遅いループを一部紹介する。次のループ s171 は、配列の添え字に演算式が含まれるループとなっている。ループ s292 とは異なり、Intel コンパイラのみがベクトル化不可と判断しており、他と比べて 8.3 倍以上遅い結果となっている。

```
!! s171 symbolics
!! symbolic dependence tests
do 10 i = 1,n
  a(i*inc) = a(i*inc) + b(i)
10 END DO
!cray:   0.036014 sec
!intel:  0.301778 sec
!gnu:    0.047688 sec
```

次のループ s172 については、ループの誘導変数の刻み値がサブルーチン外部で定義された変数 n3 により指定されているループである。こちらも Intel コンパイラのみがベクトル化不可と判断し、5.6 倍以上の性能差が生じている。

```
!! s172 symbolics
!! vectorizable if n3 .ne. 0
do 10 i = n1,n,n3
  a(i) = a(i) + b(i)
10 END DO
!cray:  0.036112 sec
!intel: 0.202817 sec
!gnu:   0.048145 sec
```

次のループ s122 は、s171 と s172 を融合したようなループであるが、こちらも Intel コンパイラはベクトル化ができていない。

```
!! s122 induction variable recognition
!! variable lower and upper bound,
!! and stride
j = 1
k = 0
do 10 i=n1,n,n3
    k = k + j
    a(i) = a(i) + b(n-k+1)
10 END DO
!cray:  0.043333 sec
!intel: 0.268437 sec
!gnu:   0.048527 sec
```

4.4 Cray コンパイラが優位なケース

次に Cray コンパイラが優位なケースを紹介する。次のループ s162 については、Cray コンパイラが他と比べて 4.3 倍以上高速なケースである。ベクトル化自体はいずれのコンパイラも対象としているが、Cray コンパイラはより優れたコードを生成していると考えられる。

```
!! s162 control flow
!! deriving assertions
if ( k > 0 ) then
  do 10 i = 1,n-1
    a(i) = a(i+k) + b(i) * c(i)
  10 END DO
endif
!cray:  0.055939 sec
!intel: 0.241279 sec
!gnu:   0.268564 sec
```

次のループ s312 は乗算を含むリダクション演算であり、Cray コンパイラが 3.5 倍以上優れた性能となっている。

```
!! s312 reductions
!! product reduction
prod = 1.d0
do 10 i = 1,n
  prod = prod * a(i)
10 END DO
!cray:  0.014094 sec
!intel: 0.050663 sec
!gnu:   0.050653 sec
```

4.5 Cray コンパイラのみ遅いケース

次のループ s124 については、Cray コンパイラのみベクトル化不可と判断し、Intel、GNU コンパイラはベクトル化可能と判断している。その結果、Cray コンパイラの性能は、他と 4.9 倍以上の差が生じている。

```
!! s124
!! induction variable recognition
!! induction variable under both
!! sides of if (same value)
do 10 i = 1,n/2
    if(b(i) > 0.d0) then
        j = j + 1
        a(j) = b(i) + d(i) * e(i)
    else
        j = j + 1
        a(j) = c(i) + d(i) * e(i)
    endif
10 END DO
!cray:  0.435522 sec
!intel: 0.088209 sec
!gnu:   0.087927 sec
```

次のループ s4117 についても、いずれのコンパイラもベクトル化しているが、Cray コンパイラのみ生成したコードの最適化が不十分であり、2.8 倍以上の性能差が生じている。

```
!! s4117 indirect addressing
!! seq function
do 10 i = 2,n
  a(i) = b(i) + c(i/2) * d(i)
10 END DO
!cray:  0.236764 sec
!intel: 0.083908 sec
!gnu:   0.088450 sec
```

4.6 ループ種別ごとの結果

最後に、ループ種別ごとの傾向を把握するために、loopd.f90 ファイルのコメントに記載されているループの種別情報を元に、ベクトル化可能と判断されたループ数を分類して集計した結果を表 6 に示す。また、Intel コンパイラを 100%とした実行時間の比を表 7 に示す。Intel コンパイラは、loop interchange や loop peeling 等のループを変換する処理において群を抜いて良い結果を得ている。一方で indirect addressing や symbolics など配列の参照が間接的になる場合は苦手としていることが分かる。

表 6　ループ種別ごとのベクトル化状況

ループ種別	ループ数	Intel	Cray	GNU
linear dependence testing	8	7	8	6
call statements	1	0	0	0
control flow	14	12	12	7
control loops	13	12	13	12
crossing thresholds	1	0	0	1
diagonals	2	2	2	0
global data flow analysis	2	2	2	2
indirect addressing	6	6	6	6
induction variable recognition	4	2	2	3
induction variable recognition	3	2	2	2
induction variables	1	1	1	1
interprocedural data flow analysis	2	2	2	0
intrinsic functions	3	3	3	2
loop distribution	2	2	2	0
loop interchange	5	4	4	0
loop peeling	3	2	2	1
loop recognition	5	4	4	3
loop rerolling	3	3	3	1
node splitting	4	4	3	0
nonlinear dependence testing	1	0	0	0
non-local goto's	2	0	0	0
non-logical if's	3	3	2	1
packing	3	3	0	0
parameters	2	2	2	2
recurrences	3	0	0	0
reductions	13	11	10	8
scalar and array expansion	8	4	6	1
scalar renaming	1	1	1	0
search loops	2	1	1	0
statement functions	1	1	1	1
statement reordering	2	2	2	0
storage classes and equivalencing	4	4	4	3
symbolics	6	3	6	6
vector semantics	1	1	1	1
wavefronts	1	0	0	0
合計	135	106	107	70

表 7　ループ種別ごとの性能比

ループ種別	Intel	Cray	GNU
call statements	100%	111%	40%
control flow	100%	722%	1053%
control loops	100%	66%	83%
crossing thresholds	100%	104%	132%
diagonals	100%	121%	134%
global data flow analysis	100%	102%	117%
indirect addressing	100%	82%	79%
induction variable recognition	100%	101%	104%
induction variables	100%	126%	290%
interprocedural data flow analysis	100%	84%	1005%
intrinsic functions	100%	91%	1202%
linear dependence testing	100%	97%	104%
loop distribution	100%	111%	113%
loop interchange	100%	1683%	3797%
loop peeling	100%	238%	424%
loop recognition	100%	75%	1491%
loop rerolling	100%	67%	136%
node splitting	100%	107%	218%
nonlinear dependence testing	100%	101%	101%
non-local goto's	100%	119%	237%
non-logical if's	100%	76%	145%
packing	100%	68%	77%
parameters	100%	76%	99%
recurrences	100%	86%	95%
reductions	100%	120%	272%
scalar and array expansion	100%	159%	110%
scalar renaming	100%	232%	362%
search loops	100%	85%	449%
statement functions	100%	75%	97%
statement reordering	100%	145%	287%
storage classes and equivalencing	100%	94%	208%
symbolics	100%	31%	37%
vector semantics	100%	92%	95%
wavefronts	100%	100%	100%

5 まとめ

AVX512 に対応した Xeon Phi KNL を対象に、Netlib/Vectord ベンチマークを用いて 3 種のコンパイラにの自動ベクトル化機能と実行性能について評価を行った。コンパイラの性能の傾向を把握することで、スーパーコンピュータシステムを利用する上での役に立てば幸いである。

参考文献

[1] Netlib: Benchmark Programs and Reports. http://www.netlib.org/benchmark (参照: 2021-09-08)

[2] 山口倉平、池田健二、疋田淳一、Netlib/vector ベンチマークによる自動ベクトル化機能の検証、大学 ICT 推進協議会 年次大会論文集、2014