SIMD 率の向上

問題は、ある二つの量の掛け算の和を求める Fortran90 並列計算の K コンピュータ上での Single-instruction-multi-data（SIMD）率を上げるということです。依頼者によって与えられたコードの SIMD 率は 0 でした。

このコードの SIMD 率向上のためのポイントはループ番号が隣接する二つの処理が SIMD 化されるようにコードを作るということです。これは SIMD 最適化のオプションをコンパイル時につけ（-Kfast で自動的につきます）、かつソースコードでそのループの独立性を明示して実行するとそのようなループが SIMD 化されるからです。二つの改良版を以下に載せます。以下、コンパイルオプションは -Kfast,openmp,ocl です。

```
program test

  implicit none
  include 'mpif.h'
  integer, parameter :: dp=kind(1.0d0), lvec=2, nthrd=8
  real(dp), dimension(8,2) :: array1
  integer :: omp_get_num_threads, omp_get_thread_num
  integer :: i,j,l,nprocs,ierr,myrank
  integer :: ithrd
  real(dp) :: s
  real(dp), dimension(8,lvec,nthrd) :: stmp

  call mpi_init(ierr)
  call mpi_comm_size(mpi_comm_world,nprocs,ierr)
  call mpi_comm_rank(mpi_comm_world,myrank,ierr)

  array1 = 1.11111111111111_dp
  stmp   = 0.0d0

!$omp parallel private(ithrd)
  if(nthrd /= omp_get_num_threads()) stop
  ithrd = omp_get_thread_num()
  ithrd = ithrd+1
!$omp do private(j,l) &
!$omp reduction(+:stmp)
  do i=1,10000
!ocl novrec
  do j=1,125000
   l = mod(j,lvec)+1
   stmp(1,l,ithrd) = stmp(1,l,ithrd) +array1(1,1)*array1(1,2)
   stmp(2,l,ithrd) = stmp(2,l,ithrd) +array1(2,1)*array1(2,2)
   stmp(3,l,ithrd) = stmp(3,l,ithrd) +array1(3,1)*array1(3,2)
   stmp(4,l,ithrd) = stmp(4,l,ithrd) +array1(4,1)*array1(4,2)
   stmp(5,l,ithrd) = stmp(5,l,ithrd) +array1(5,1)*array1(5,2)
   stmp(6,l,ithrd) = stmp(6,l,ithrd) +array1(6,1)*array1(6,2)
   stmp(7,l,ithrd) = stmp(7,l,ithrd) +array1(7,1)*array1(7,2)
   stmp(8,l,ithrd) = stmp(8,l,ithrd) +array1(8,1)*array1(8,2)
```

```
  enddo
  enddo
!$omp end do
!$omp end parallel

  s=0.0D0
  do ithrd=1,nthrd
  do l=1,lvec
   s =&
s+stmp(1,l,ithrd)+stmp(2,l,ithrd)+stmp(3,l,ithrd)+stmp(4,l,ithrd)&
+stmp(5,l,ithrd)+stmp(6,l,ithrd)+stmp(7,l,ithrd)+stmp(8,l,ithrd)
  enddo
  enddo

  call mpi_finalize(ierr)

end program test
```

```
program test

  implicit none
  include 'mpif.h'
  integer, parameter :: dp=kind(1.0d0), lvec=2
  real(dp), dimension(8,2) :: array1
  integer :: i,j,l,nprocs,ierr,myrank
  real(dp) :: s
  real(dp), dimension(8,lvec) :: stmp

  call mpi_init(ierr)
  call mpi_comm_size(mpi_comm_world,nprocs,ierr)
  call mpi_comm_rank(mpi_comm_world,myrank,ierr)

  array1 = 1.11111111111111_dp
  stmp   = 0.0d0

!$omp parallel do private(j,l) &
!$omp reduction(+:stmp)
  do i=1,10000
!ocl novrec
  do j=1,125000
   l = mod(j,lvec)+1
   stmp(1,l) = stmp(1,l) +array1(1,1)*array1(1,2)
   stmp(2,l) = stmp(2,l) +array1(2,1)*array1(2,2)
   stmp(3,l) = stmp(3,l) +array1(3,1)*array1(3,2)
   stmp(4,l) = stmp(4,l) +array1(4,1)*array1(4,2)
   stmp(5,l) = stmp(5,l) +array1(5,1)*array1(5,2)
   stmp(6,l) = stmp(6,l) +array1(6,1)*array1(6,2)
   stmp(7,l) = stmp(7,l) +array1(7,1)*array1(7,2)
   stmp(8,l) = stmp(8,l) +array1(8,1)*array1(8,2)
  enddo
  enddo
```

```
!$omp end parallel do

  s=0.0D0
  do l=1,lvec
   s =&
s+stmp(1,l)+stmp(2,l)+stmp(3,l)+stmp(4,l)+stmp(5,l)+stmp(6,l)+stmp(7,l)&
+stmp(8,l)
  enddo

  call mpi_finalize(ierr)

end program test
```

いずれのコードでも改良点は、和を格納する配列 stmp の配列要素をループ番号が偶数か奇数かに応じて別にしたことです。第一のコードでは、各スレッドが別個に計算を行うことが stmp の第三の引数によって明示されています。K コンピュータでの計算では、SIMD 率はどちらのコードでも同じで、約１１％でした。Flops-peak 比は約８％でした。

SIMD 率の向上を含むチューニングに関する指南が、K コンピュータですと
https://k.aics.riken.jp/cgi-bin/K.ja/index.cgi　の
Start → ドキュメント → マニュアル → Fortran 手引書
の第９章　最適化機能、および
Start → ドキュメント → チュートリアル → 2.1 CPU チューニングチュートリアル
にあります。