Progress report on OpenMP offloading in ORB5

Ville-Markus Yli-Suutala

Starting point

- Working OpenMP offloading implementation
- Performance not ideal
- Compiler: flang

Test case

- TAE_pb.in with 10x

FortranAAssign

- Function in the flang runtime library
- Used for assigning to arrays and derived types
- Call might be optimized away
- Very costly to call on GPUs

FortranAAssign

- Not obvious to the programmer where the calls actually happen
- Linker error shows at the function level
- For large functions still hard to tell where the calls happen

printf debugging the compiler

- MLIR pass to visit all function call nodes and print those that are calls to the Fortran runtime library
- MLIR operations carry source line information
- Run once and throw away

Workaround

- FortranAAssign found
  - `arr1 = arr2`
  - `arr1(1) = arr2(1)`
  - `arr1(2) = arr2(2)`
  - `arr1(3) = arr2(3)`
-

Atomics

- flang atomics were slow compared to clang
- Workaround: Call C function from Fortran
- Calling the C function has a cost
- The cost has increased for push in ORB5

Timings on AMD MI 250x on LUMI

- push (clang atomics) 18.7s / 440s
- deposition_by_part (flang atomics) 10.51s
- get_field_by_part 1.84s

Atomic add example in Fortran

```
subroutine atomic_add(a, b)
real(kind=8) :: a, b
!$omp declare target

!$omp atomic update
a = a + b
end subroutine
```

Slow atomic add

```
  %2 = load double, ptr %1, align 8, !tbaa !8
  %.atomic.load = load atomic i64, ptr %0 monotonic, align 8
  br label %.atomic.cont

.atomic.cont:                                     ; preds = %.atomic.cont, %entry
  %3 = phi i64 [ %.atomic.load, %entry ], [ %7, %.atomic.cont ]
  %.atomic.fltCast = bitcast i64 %3 to double
  %4 = fadd contract double %2, %.atomic.fltCast
  %5 = bitcast double %4 to i64
  %6 = cmpxchg ptr %0, i64 %3, i64 %5 monotonic monotonic
  %7 = extractvalue { i64, i1 } %6, 0
  %8 = extractvalue { i64, i1 } %6, 1
  br i1 %8, label %.atomic.exit, label %.atomic.cont

.atomic.exit:                                     ; preds = %.atomic.cont
  ret void
```

Fast atomic add

```
%2 = load double, ptr %1
%3 = atomicrmw fadd ptr %0, double %2 monotonic
ret void
```

Better atomics in flang

- Patch merged last week
- Improves atomic add/sub for 32 and 64 bit floats
- Also improved min/max
- min/max require -ffp-maxmin-behavior to use in flang

Timings on AMD MI 250x on LUMI

- push (clang atomics) 18.7s / 440s → 10.8s
- deposition_by_part (flang atomics) 10.51s → 3.36s
- get_field_by_part 1.84s → 0.79s

Nvidia GPUs

- Big difference between having code in the loop vs in a function that is called in the loop

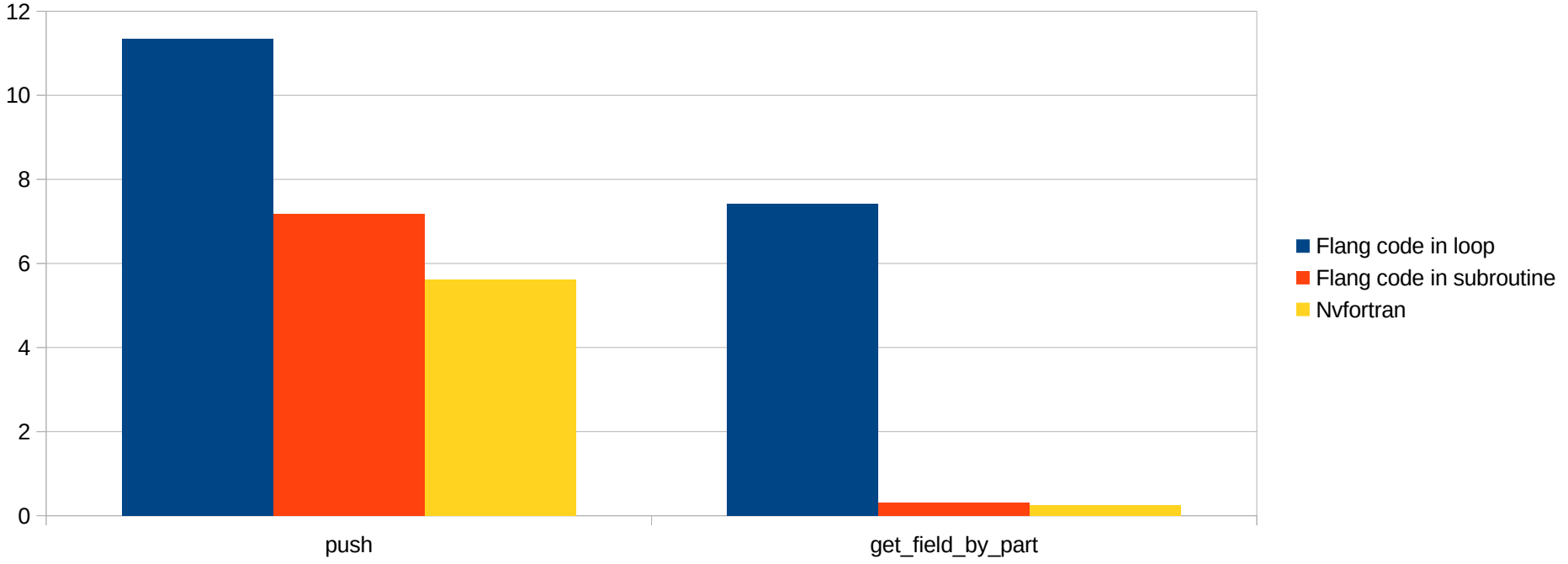
```
!$omp target teams distribute parallel do &
!$omp private(xs, ys...)
do i,n
  ! Kernel code
end do
```

```
!$omp target teams distribute parallel
do i,n
  call kernel()
end do
```

GH200 on Roihu

- Push 11.346s
- get_field_by_part 7.409s (nvfortran 0.242s)
- 30x faster get_field with nvfortran

Nvidia GH200 on Roihu



Bonus challenges

- Global declare target variable linking
- Push kernel body executing too many times with OpenMP optimizations enabled
- -gline-directives-only