Programming RISC-V accelerators via Fortran

Nick Brown (n.brown@epcc.ed.ac.uk) , Jake Davies

Why Fortran?

Fortran is the lingua franca of scientific computing. A mix of legacy and new HPC applications are written in Fortran.

To give you an idea, on ARCHER2 the UK national supercomputer:

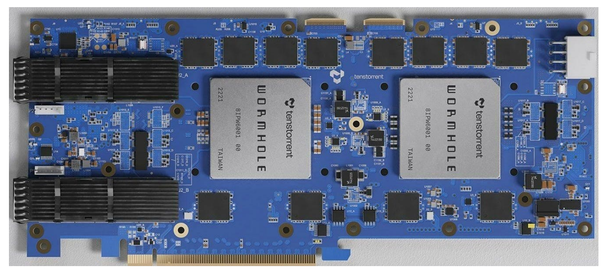
- Around 60% of code running on the machine are written in Fortran
- Which account for approximately 62% of the machine's overall runtime
- These use around 70% of the total energy consumed by the supercomputer

We therefore care, not only about making Fortran codes run as fast as possible but also in an energy efficient manner

A wealth of RISC-V accelerator cards

Starting to see a range of accelerators built upon RISC-V. Primarily designed for machine learning workloads, compute capabilities can be used for scientific computing.

Arguably, accelerator cards built around RISC-V are where we are going to first start seeing RISC-V be adopted by HPC due to the lower barrier to entry compared to CPU-based RISC-V machines.



Programming these is a major challenge as they all leverage bespoke APIs and SDKs, HPC programmers are not going to port their codes!

Building upon MLIR and xDSL

MLIR is a composable compiler ecosystem built upon Intermediate Representation (IR) dialects and transformations between them. Promotes reuse of compiler infrastructure.

MLIR

xDSL

https://xdsl.dev

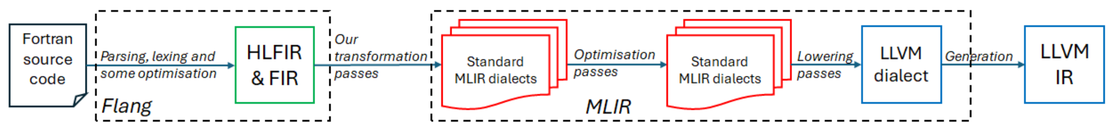
MLIR is nice, but a challenge is the steep learning curve. xDSL is a fast prototyping environment enabling rapid exploration of compiler techniques and approaches.

The LLVM Flang compiler

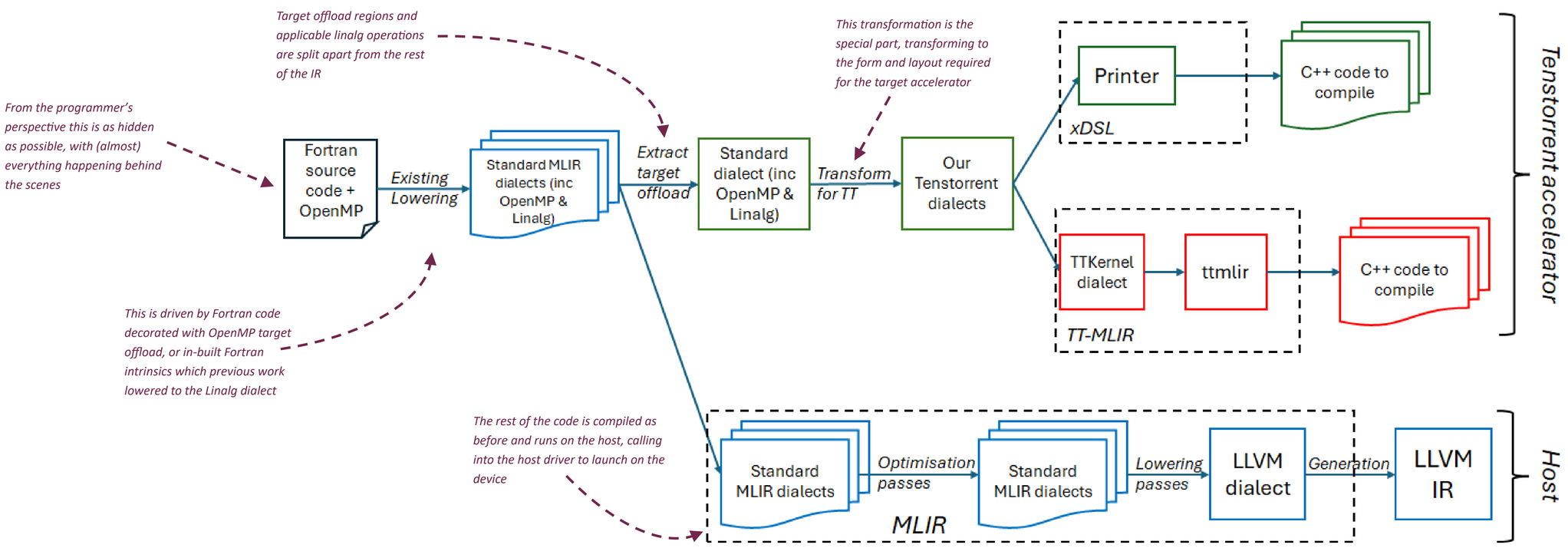
Flang is LLVM's Fortran compiler and this leverages MLIR by generating High Level Fortran Intermediate Representation (HLFIR) and FIR. However, this IR is then directly lowered to LLVM-IR rather than going via the rest of the MLIR ecosystem.

In previous work we lowered HLFIR and FIR to the *standard* MLIR dialects:

- Based on this, we can then lower the standard dialects to a range of RISC-V accelerators not just for Fortran but for a range of other frontend languages



Tenstorrent example: Compilation flow driven by Fortan and OpenMP



Ultimately, C++ with TT-Metal is generated. Either through our pipeline in xDSL (green) or lowering from our TT-Metal dialects into Tenstorrent's ttkernel and tt-mlir stack.

Our approach also generates the required host code to drive memory transfer and kernel launching on the device. This is integrated into the rest of the host-side Fortran code.

1) Driven by OpenMP target offload

Programmers can decorate loops via OpenMP's *target offload* directive to run this loop on the accelerator.

- The compiler determines data movement to and from the device.
- Generates necessary kernels for data in, data our and compute.

```
subroutine saxpy(a, x, y, n)
  ...
  !$omp omp target parallel do
  !$omp& num_threads(20) simd simdlen(32)
  do i=1, n
     y(i) = a * x(i) + y(i)
  end do
  !$omp end target parallel do simd
end subroutine
```

OpenMP directives and attributes are used to tune the mapping of iterations to device:

- The *parallel do* directive splits loop iterations across Tensix cores, with the number of cores to use provided by the *num_threads* attribute (or a default chosen).
- Within each Tensix core, the *simd* directive allocates loop iterations to SIMD lanes of the SFPU, with the *simdlen* attribute controlling the number of lanes (or a default chosen).

2) Driven by Fortran intrinsics

Intrinsics are built in functions provided by the compiler, including mathematical support such as matrix multiplication, dot product, matrix transposition, reductions. They are used extensively in HPC codes, and our previous work lowered them to the *linalg* dialect.

```
integer :: data(100000), result, i
do i=1, 100000
   data(i)=i
end do
result=sum(data)
```

Execute this sun intrinsic on the RISC-V accelerator

Our approach lowers applicable operations in the *linalg* dialect to the Tenstorrent dialects and RISC-V accelerator, seamlessly offloading and accelerating these language built-ins.

Conclusions

We can't expect HPC developers to come to us, we must go to them and supporting technologies that they are familiar with and that require minimal changes to their code is crucial!

Aim is to extend this to support a wide range of RISC-V accelerators