

Programming RISC-V accelerators via Fortran

Nick Brown^{1*}, Jake Davies¹, Felix LeClair²

¹EPCC, The University of Edinburgh, 47 Potterrow, Edinburgh, UK

²Tenstorrent, 2600 Great America Way, Santa Clara, CA, USA

Abstract

A range of RISC-V based accelerators are available and coming to market, and there is strong potential for these to be used for High Performance Computing (HPC) workloads. However, such accelerators tend to provide bespoke programming models and APIs that require codes to be rewritten. In scientific computing, where many of the simulation code are highly complex, extensive, and written in Fortran, this is not realistic. In this extended abstract we present an approach that enables driving such architectures via Fortran, avoiding code redevelopment.

Introduction

Whilst RISC-V has grown rapidly in areas such as embedded computing, it is yet to gain significant traction in High Performance Computing (HPC). However, as we move further into the exascale era the HPC community will be faced by a range of new challenges, for instance the requirement to decarbonise their workloads, and there is the potential for RISC-V to play an important role.

Arguably, it is likely that we will first see adoption of RISC-V in HPC via PCIe accelerator cards. These can be easily fitted into existing systems and because it enables centres to *dip their toe* into the RISC-V ecosystem it limits their risk as other parts of the supercomputer remain the same. Indeed, there are a range of RISC-V PCIe accelerator cards that are shipping, such as Esperanto's ET-SoC and the Tensix family from Tenstorrent, with other products such as Inspire Semiconductor's Thunderbird having been announced. However, the major challenge associated with all of these is that to actually run codes on them then the developer must learn a new programming model, restructure their codes to map to the architecture, and leverage the vendor API.

Fortran is the lingua franca of scientific computing, indeed around 65% of codes running on ARCHER2, the UK national supercomputer, are written in Fortran and these account for around 70% of the machine's runtime. Ultimately, developers of these high performance codes want to run more complex problems at reduced time to solution, and the specialisation provided by RISC-V accelerators means that they can potentially provide this whilst also delivering energy benefits. However, a major challenge to adoption of such technologies is the requirement for scientific programmers to significantly restructure their codes, potentially also having to rewrite them in different programming languages.

MLIR

Since it was first merged into mainstream LLVM in 2019, MLIR has become a popular for developing compilers. Comprising Intermediate Representation (IR) dialects, and transformations which undertake optimisations and convert the IR between dialects, it is possible to mix dialects which are at different levels of abstraction and progressively lower between them. Ultimately, MLIR provides reuse of compiler infrastructure, and via the MLIR framework one can define their own IR dialects and transformations.

Flang

Flang is the LLVM community's Fortran compiler and leverages MLIR by providing its own Fortran IR (FIR) and High Level Fortran IR (HLFIR) dialects. However, only a subset of MLIR standard dialects are integrated with Flang, and Flang itself transforms straight from HLFIR+FIR into LLVM-IR without using any of the existing MLIR transformations or optimisations.

Conversely, the *mlir-opt* MLIR driver tool is unaware of the Flang dialects and it is not possible to drive the wide range of MLIR transformations and optimisations via Flang's IR. To this end in ?? we developed a transformation pass that lowers Flang's HLFIR and FIR dialects into standard MLIR dialects. The first benefit of this is that the user is then able to leverage the existing MLIR transformations which are developed and maintained by a large community, including many vendor, to generate LLVM-IR. The second benefit is that it provides a much wider range of potential target architectures including GPUs.

Flang for RISC-V accelerators

Figure 1 provides an overview of our approach, where the existing work of [1] lowers Fortran into the standard dialects and a transformation is provided which lowers into the specific host and device dialects for

*Corresponding author:
(n.brown@epcc.ed.ac.uk)

Nick Brown

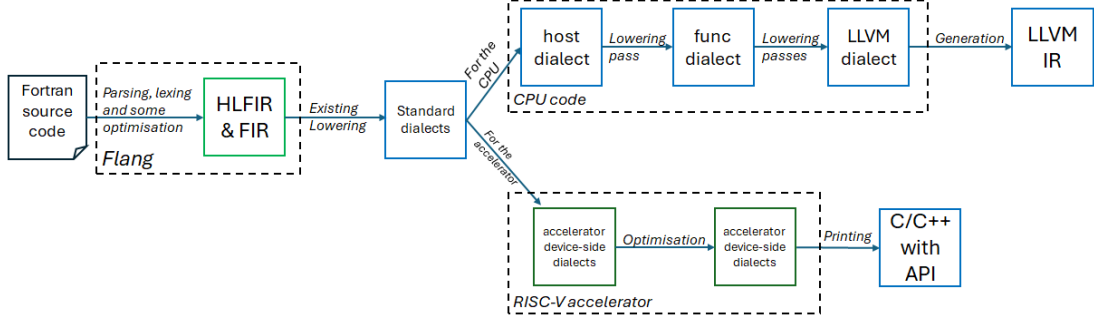


Figure 1: Illustration of our approach lowering Flang to target RISC-V accelerators

the RISC-V accelerator. Some PCIe based RISC-V accelerators already provide an MLIR-based compiler stack and-so we can then leverage their existing dialects in combination with their compilation pipeline to generate the resulting executables.

However, many of these accelerators either do not provide an MLIR stack or such a stack is immature. In such a case, as per Figure 1, we develop a backend for these accelerators which comprises host and device-side dialects that map one-to-one to the accelerator API. A lowering is then developed that converts the host-side dialect to the *func* dialect, calling runtime functions and eventually into LLVM-IR. On the device-side we develop a *printer* which accepts the device specific dialects and standard MLIR dialects, such as *memref* for memory management, and this prints out target code comprising a programming language, commonly C or C++, calling into the device’s API.

Tenstorrent example

Tenstorrent ship RISC-V PCIe accelerator cards that are built upon their Tensix technology. Each Tensix core comprises five RISC-V cores; one for data movement in, one for data movement out, and three drive a 16384 wide vector unit. The Wormhole n300, for example, contains 128 Tensix cores. The decoupling of data movement from compute makes this a very interesting potential architecture for HPC, and indeed early experiments porting a scientific computing workload to the Grayskull delivered similar performance to a 24-core Xeon Platinum CPU but at five times less energy usage [2]. However, to run codes on this architecture programmers must learn a new architecture and significantly recast their applications.

We developed a Tenstorrent specific backend which comprises a host dialect and three device-side dialects, one for data movement, one for circular buffers between RISC-V cores, and one for compute. We also developed a printer that, from the device-side dialects, generates C++ that calls into the Metalium API.

```

subroutine saxpy(a, x, y, n)

```

```

...

```

```

!$omp omp target parallel &

```

```

!$omp& do simd num_threads(20) simdlen(32) &

```

```

do i=1, n

```

```

    y(i) = a * x(i) + y(i)

```

```

end do

```

```

!$omp end target parallel do simd

```

```

end subroutine

```

Listing 1: Example Fortran code running Single-precision A times X Plus Y (SAXPY) on the Tenstorrent accelerator card (argument declarations omitted for brevity)

A question is how, in Figure 1, to lower from the standard dialects into the device-specific ones that map to the RISC-V accelerator. The programmer drives this via OpenMP target offload, and Listing 1 illustrates Single-precision A times X Plus Y (SAXPY) written in Fortran and offloaded to the Tenstorrent PCIe accelerator via OpenMP. HPC programmers are already familiar with OpenMP, both for threaded and GPU programming so it is a natural choice. The code example in Listing 1 will run the loop in parallel over two Tensix cores, due to the *num_teams(2)*, leveraging the SIMD capabilities of each Tensix core.

Conclusions

We have described offloading Fortran code to RISC-V based accelerators via Flang. OpenMP provides a clear abstraction which can be used to drive such an offloading, and MLIR is a powerful compiler technology for supporting these accelerators because it enables the sharing of compiler infrastructure between them.

References

- [1] Nick Brown. “Fully integrating the Flang Fortran compiler with standard MLIR”. In: *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2024.
- [2] Nick Brown. “Accelerating stencils on the Tenstorrent Grayskull RISC-V accelerator”. In: *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2024.