Reducing FPGA Memory Footprint of Stencil Codes

Robert Szafarczyk, Syed Waqar Nabi, Wim Vanderbauwhede

School of Computing Science, University of Glasgow

Problem

The stencil computation pattern is common in many scientific codes, especially in the climate domain. State of the art stencil implementations on FPGAs employ on-chip buffering of the stencil windows for data reuse [1]. This buffering technique severely decreases the supported resolution size of stencil codes in domains higher than 2-D because of limited on-chip memory FPGA resources.

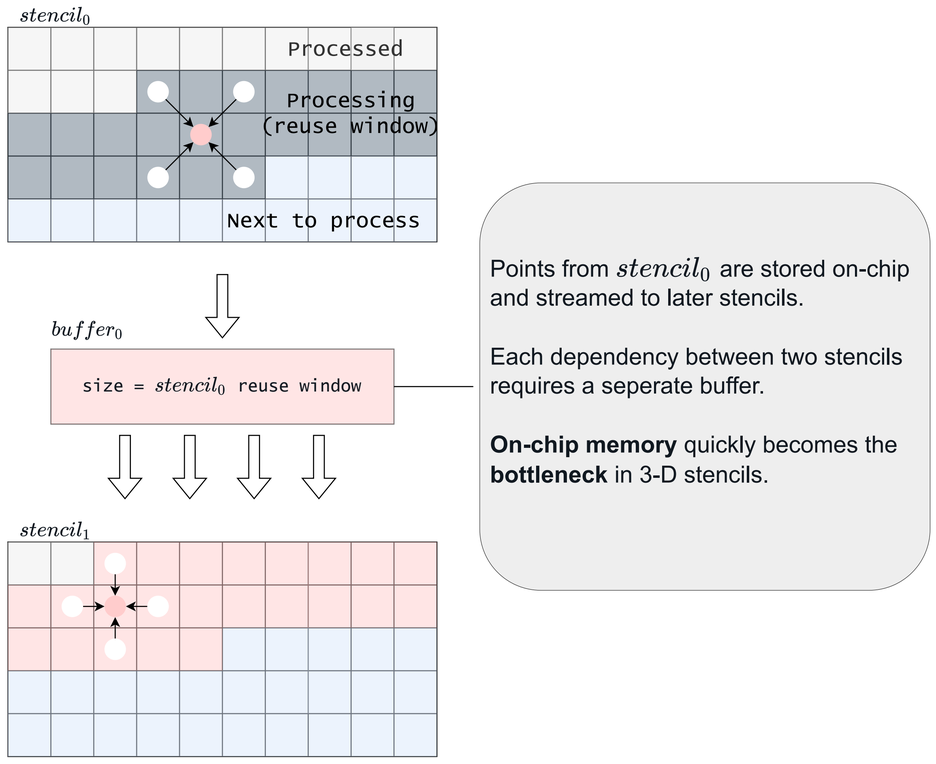
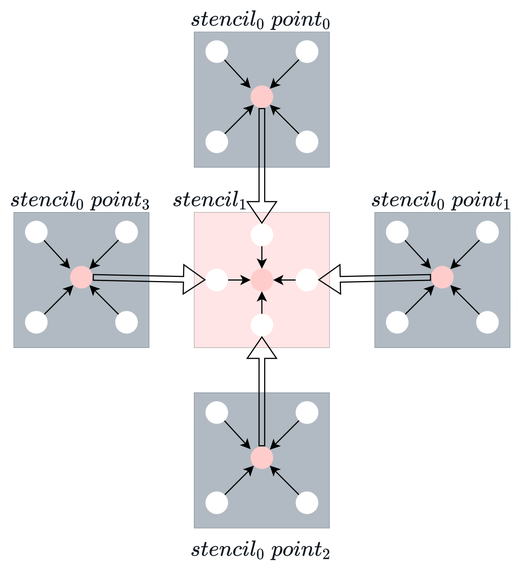


Figure 1: Visual representation of data reuse in a streaming stencil accelerator.

Method

We rewrite the TyTraCL program dataflow such that, instead of storing a value calculated in a stencil for later re-use, we re-calculate the value every time it is needed. Additional rewrite rules perform other well-known stencil optimisations, like stencil fusion. The data reuse buffer in Figure 1 is transformed into additional computations (one per point): The previous code with two stencils would become:



(a) Visual representation

```
do idx=1,500*500
  ! Initial stencil point collection for first stencil.
  integer, dimension(1:4) :: s2=(/ 1, 502, 504, 1005 /)
  integer, dimension(1:3) :: s1=(/ 0, 2, 1004 /)
  do s_idx_1 = 1,4
    do s_idx_2 = 1,3
      stencil_idx = idx + s2(s_idx_1) + s1(s_idx_2)
      in_pts(s_idx_1, s_idx_2) = in(stencil_idx)
    end do
  end do

  ! Values are recomputed (unrolled loop on FPGAs)
  do i_fmaps=1,4
    v1_pts(i_fmaps) = (in_pts(i_fmaps, 1) +
                       in_pts(i_fmaps,2) +
                       in_pts(i_fmaps, 3) +
                       in_pts(i_fmaps,3)) * a
  end do

  out(idx) = (v1_pts(2) + v1_pts(3) +
              v1_pts(1) + v1_pts(4)) / b
end do
```

(b) Transformed code

Figure 3: Stencil points are re-computed instead of re-used.

Context

The TyTraCL framework [3] provides an automatic porting of Fortran code to a streaming architecture on FPGAs. It extracts the parallel patterns (e.g. maps, folds, stencils) from Fortran code to a functional coordination language (TyTraCL) which describes the dataflow between opaque functions from the original Fortran program. We can define provably correct rewrite rules of TyTraCL which allow for automatic Design Space Exploration (DSE).

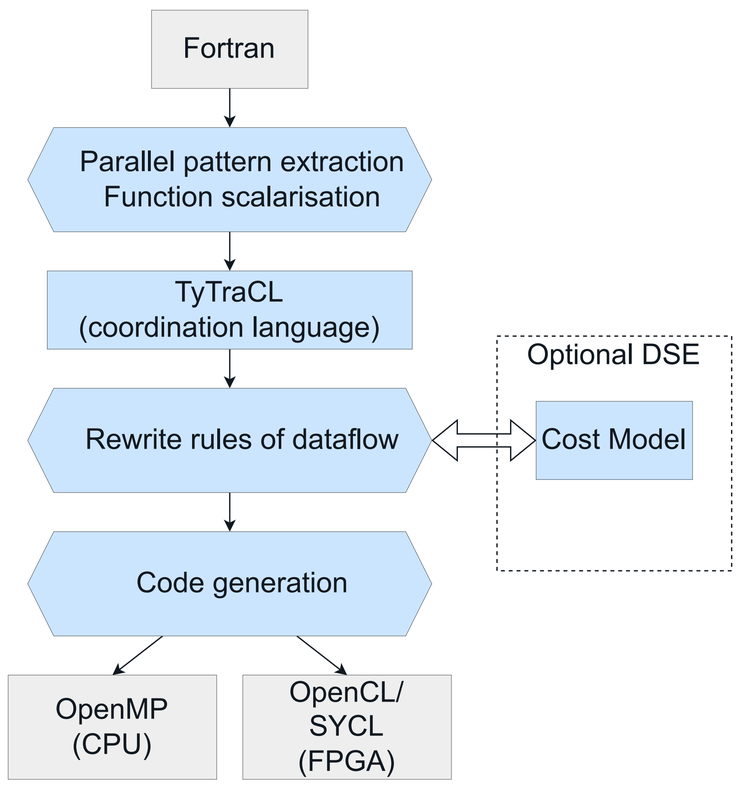


Figure 2: Automatic porting of Fortran to FPGAs through the TyTraCL framework.

Existing Fortran stencil codes make excessive use of redundant intermediate arrays storing values in-between two stencil applications for the whole domain - these patterns are easy to extract in TyTraCL. The rewrite rules proposed in our paper would transform the intermediate array `v1` from below code into additional computation.

```
do j = 1:500
  do k = 1:500    ! stencil_0
    v1(j, k) = (in(j-1, k-1) + in(j+1, k-1) + in(j-1, k+1) + in(j-1, k+1)) * a
  end do
end do

do j = 1:500
  do k = 1:500    ! stencil_1
    out(j, k) = (v1(j-1, k) + v1(j+1, k) + v1(j, k-1) + v1(j, k+1)) / b
  end do
end do
```

Evaluation

We evaluated our approach on a 2-D (3 intermediate buffers, 4 stencil applications [2]) and a 3-D (18 intermediate buffers, 3 stencil applications [4]) physics simulation code using an Intel Arria 10 FPGA. We compared our `reduced` approach against a `direct` translation of the Fortran code to FPGA, a `manual` FPGA implementation in SYCL using shift-registers for reuse buffers and common HLS optimisations [1].

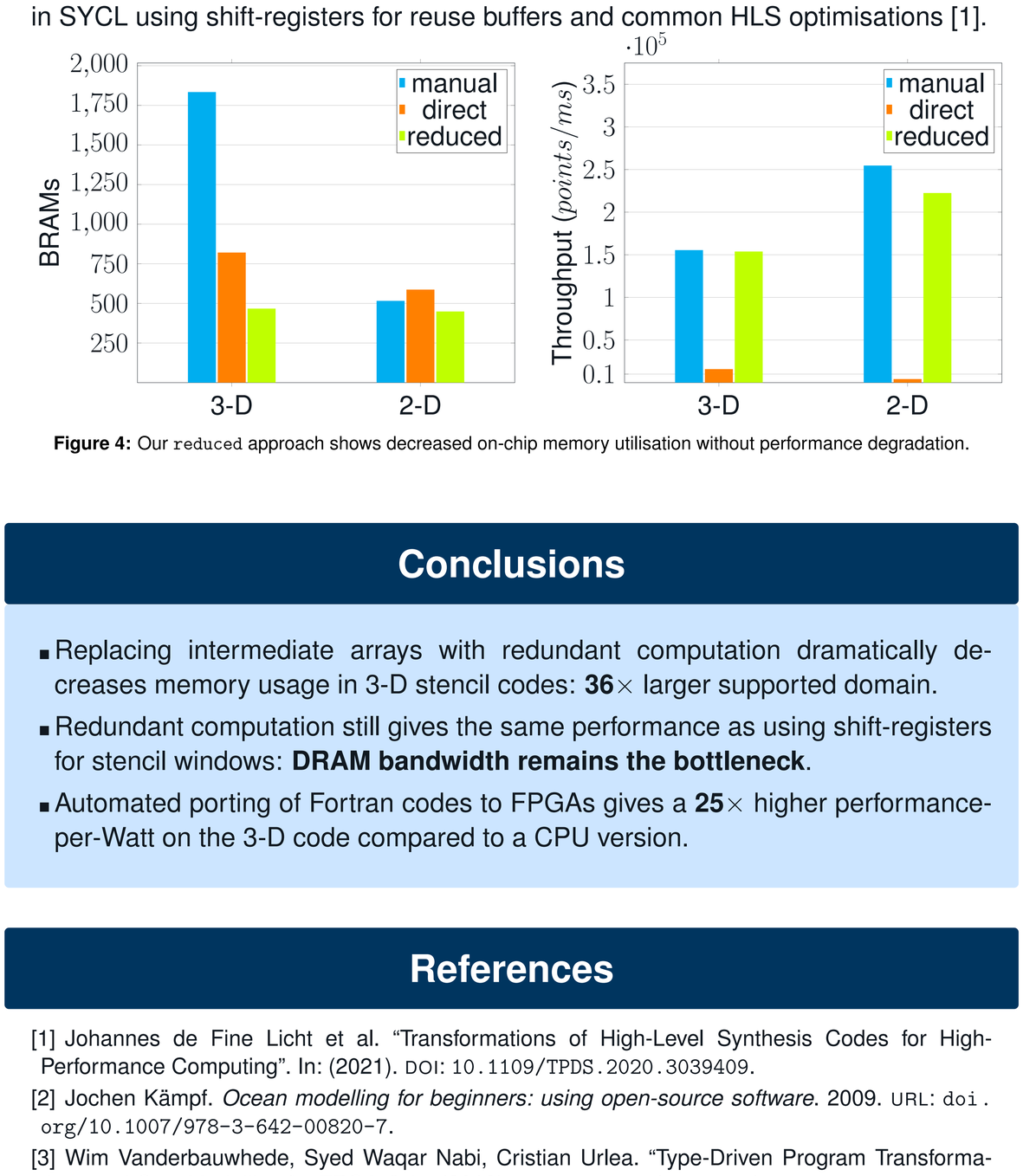


Figure 4: Our `reduced` approach shows decreased on-chip memory utilisation without performance degradation.

Conclusions

- Replacing intermediate arrays with redundant computation dramatically decreases memory usage in 3-D stencil codes: **36**× larger supported domain.
- Redundant computation still gives the same performance as using shift-registers for stencil windows: **DRAM bandwidth remains the bottleneck**.
- Automated porting of Fortran codes to FPGAs gives a **25**× higher performance-per-Watt on the 3-D code compared to a CPU version.

References

[1] Johannes de Fine Licht et al. “Transformations of High-Level Synthesis Codes for High-Performance Computing”. In: (2021). DOI: 10.1109/TPDS.2020.3039409.

[2] Jochen Kämpf. *Ocean modelling for beginners: using open-source software*. 2009. URL: doi.org/10.1007/978-3-642-00820-7.

[3] Wim Vanderbauwhede, Syed Waqar Nabi, Cristian Urlea. “Type-Driven Program Transformations and Cost Modelling for FPGAs”. In: (2019). URL: doi.org/10.1007/s10766-018-0572-z.

[4] Toshiya Yoshida et al. “Large-eddy-simulation study of the effects of building-height variability on turbulent flows”. In: (2018). URL: doi.org/10.1007/s10546-018-0344-8.