D3.2 IFS - ECMWF

Manual and Automated Vectorisation for the Integrated Forecasting System

Andrew Beggs, Olivier Marsden, Ioan Hadade

Andrew.Beggs@ecmwf.int, Olivier.Marsden@ecmwf.int, Ioan.Hadade@ecmwf.int

EUPEX

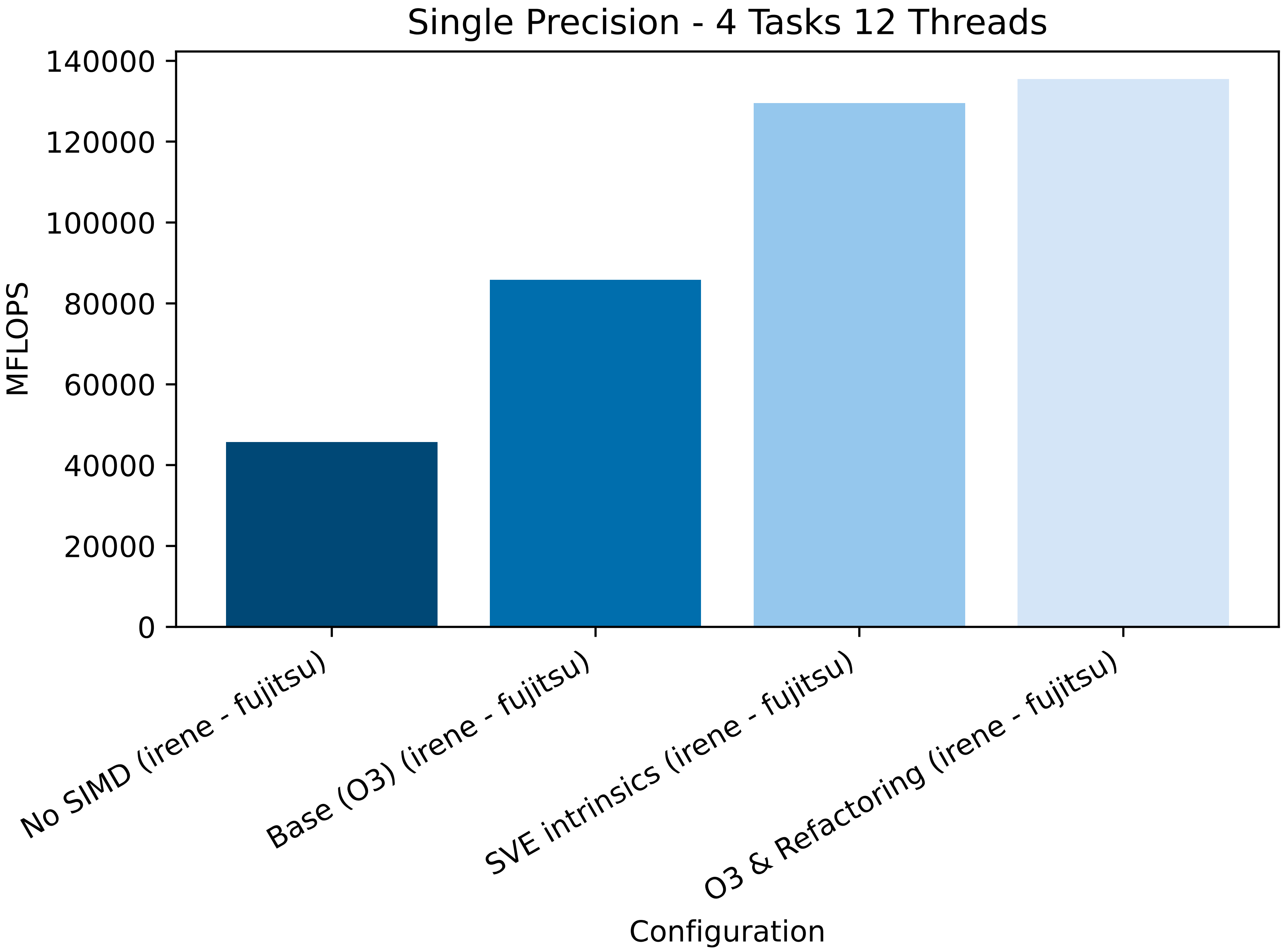
EuroHPC Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101033975. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Germany, Italy, Greece, United Kingdom, Czech Republic, Croatia.

Introduction

- Summary of effects and strategies to utilise Arm ISAs
  - Scalable Vector Extension (SVE)
  - HBM (available on Fujitsu’s A64FX processor)
- CloudSC
  - Physics component of the Integrated Forecasting System (IFS)
    - A cloud microphysics parameterisation
  - Known for being computationally demanding
- Brief overview of co-design with ECMWF, Eviden, EUPEX, ESiWACE, and the EPI

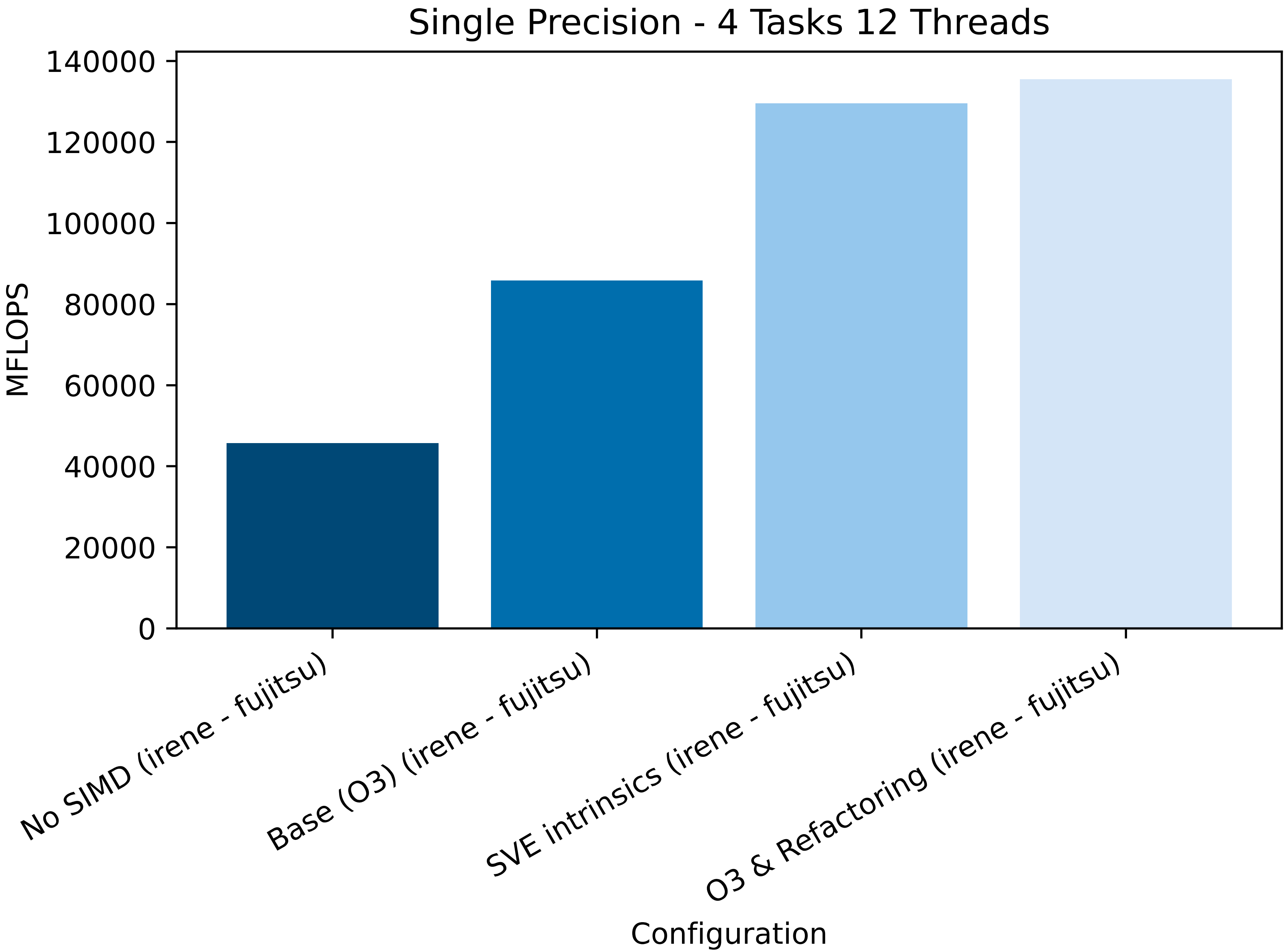
Effects of vectorisation on performance



Performance of CloudSC without vectorisation, with auto-vectorisation, a handwritten SVE kernel, & auto-vectorised refactored kernel

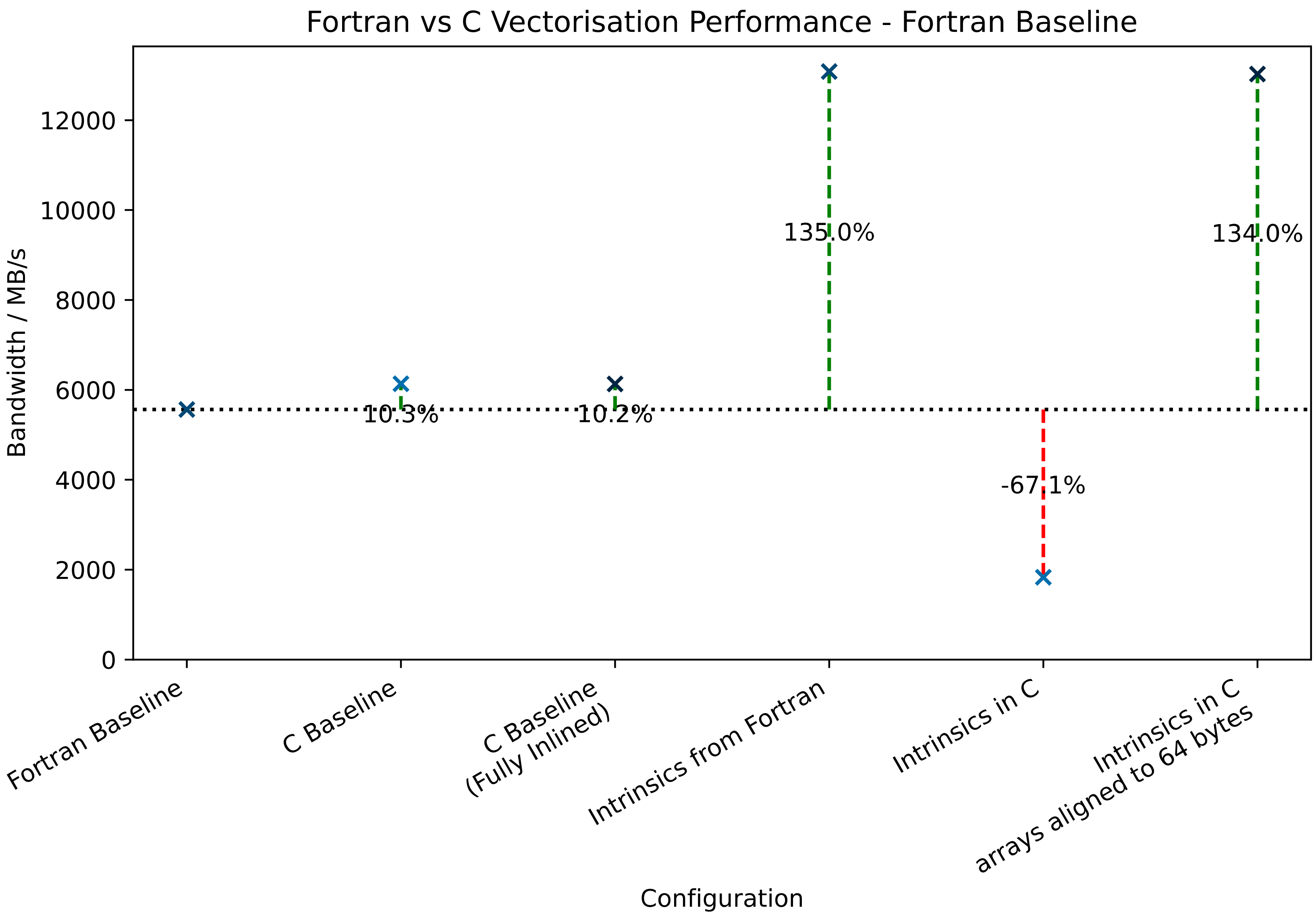
EUPEX ECMWF

- **Auto-vectorisation boosts performance** by 87.6%
- Refactoring the hottest loop yields an increase of 57.9% over base performance
- **Handwritten SVE** intrinsics kernel written in C and called from Fortran **achieves 95.6%** of performance **of the auto-vectorised** refactored loop
- Only **5.87% of maximum** theoretical **FLOPS per node**



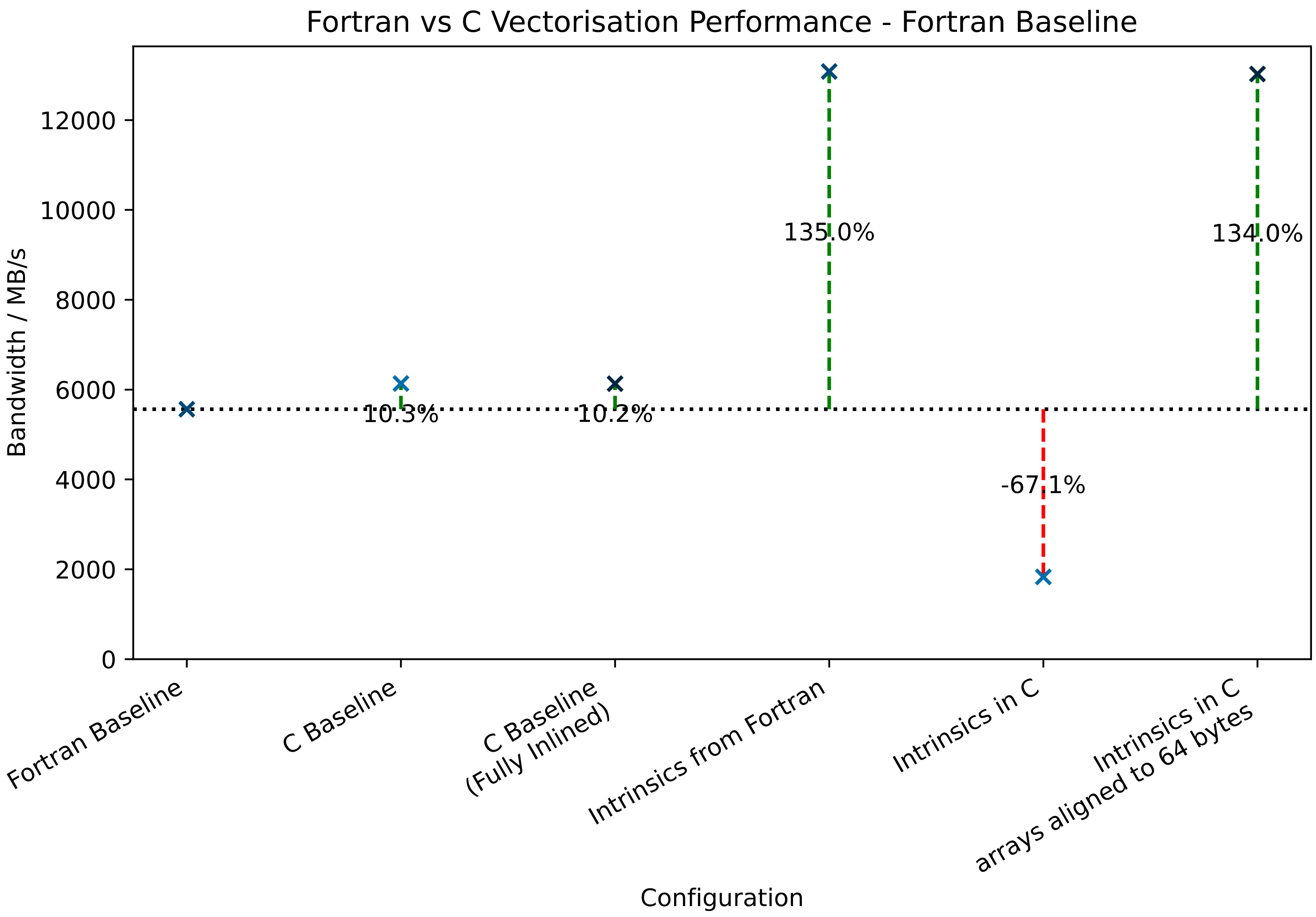
Performance of CloudSC without vectorisation, with auto-vectorisation, a handwritten SVE kernel, & auto-vectorised refactored kernel

Using C SVE intrinsics in Fortran



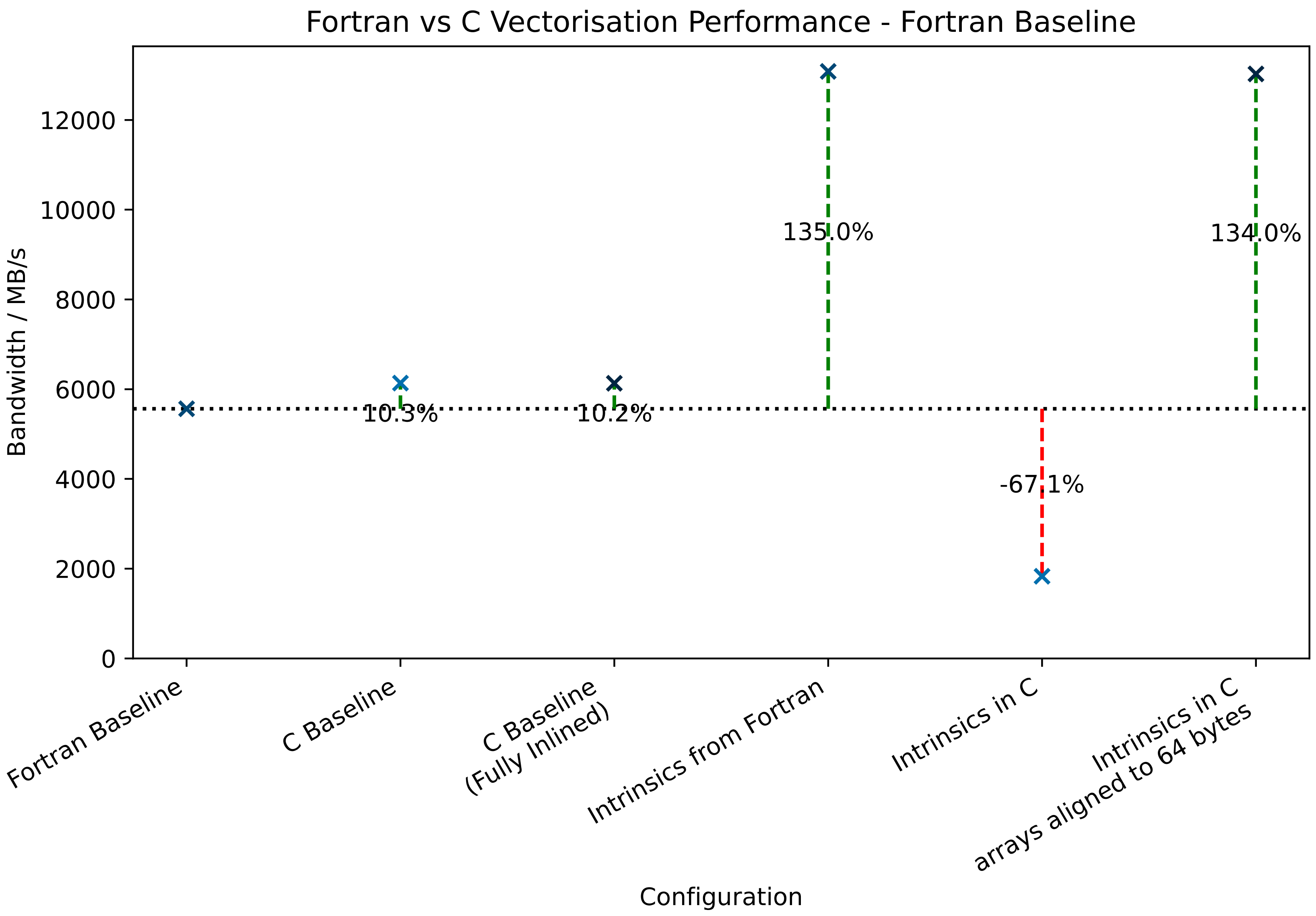
Performance of various kernels against a Fortran auto-vectorisation baseline

- The **IFS**, and by extension CloudSC, is **written** almost entirely **in Fortran**
  - Not feasible to translate the whole codebase
  - **SVE intrinsics** are only available **in C**



Performance of various kernels against a Fortran auto-vectorisation baseline

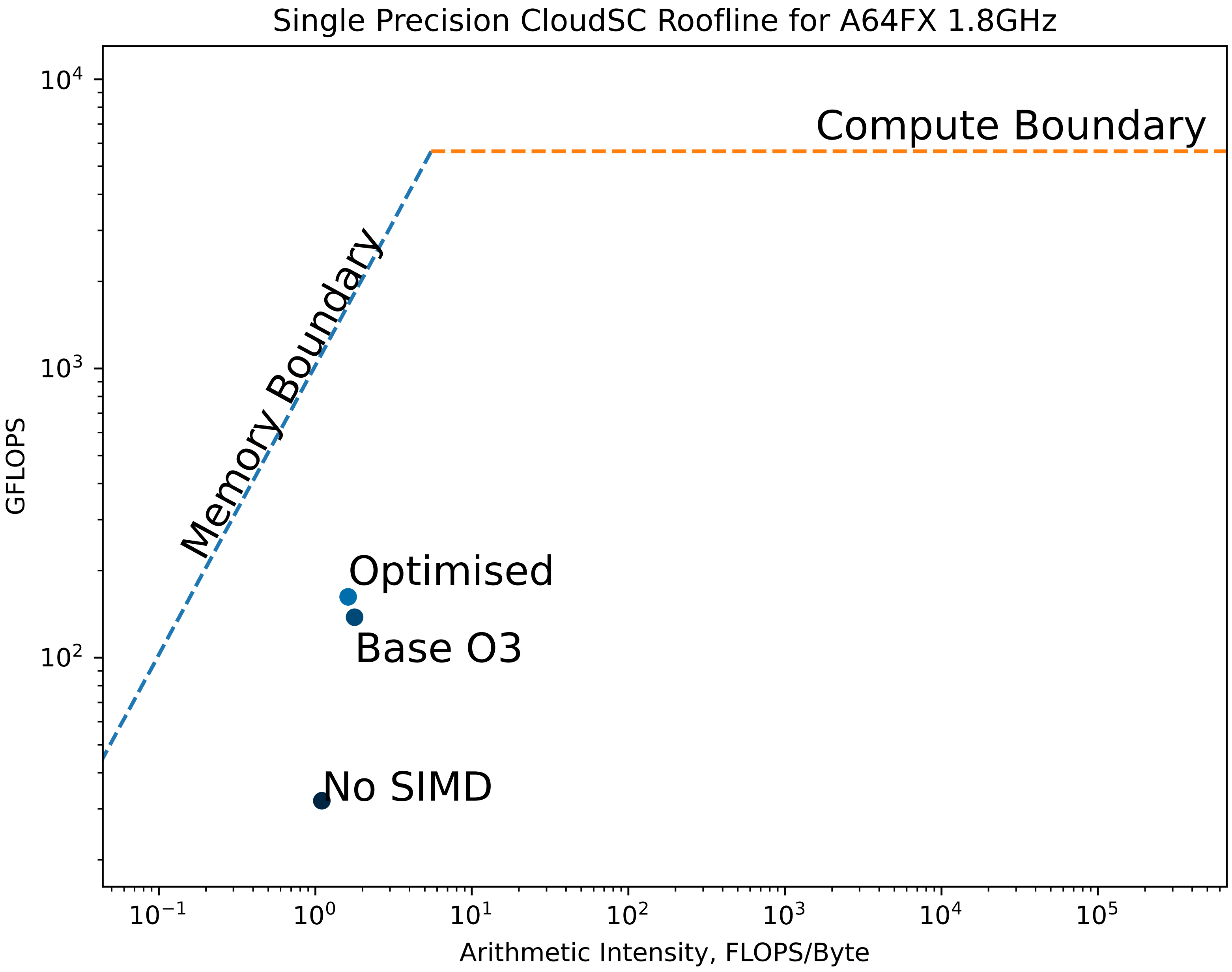
- The solution is to identify hot sections of the codebase and hand write SVE kernels for these in C, which can be called from Fortran
- Fortran is aligned to 64 bytes, while C is aligned to 16 by default
  - Only aligning to 64 bytes gave a performance increase over baseline
  - Aligning to 16 bytes gave a performance regression



Performance of various kernels against a Fortran auto-vectorisation baseline

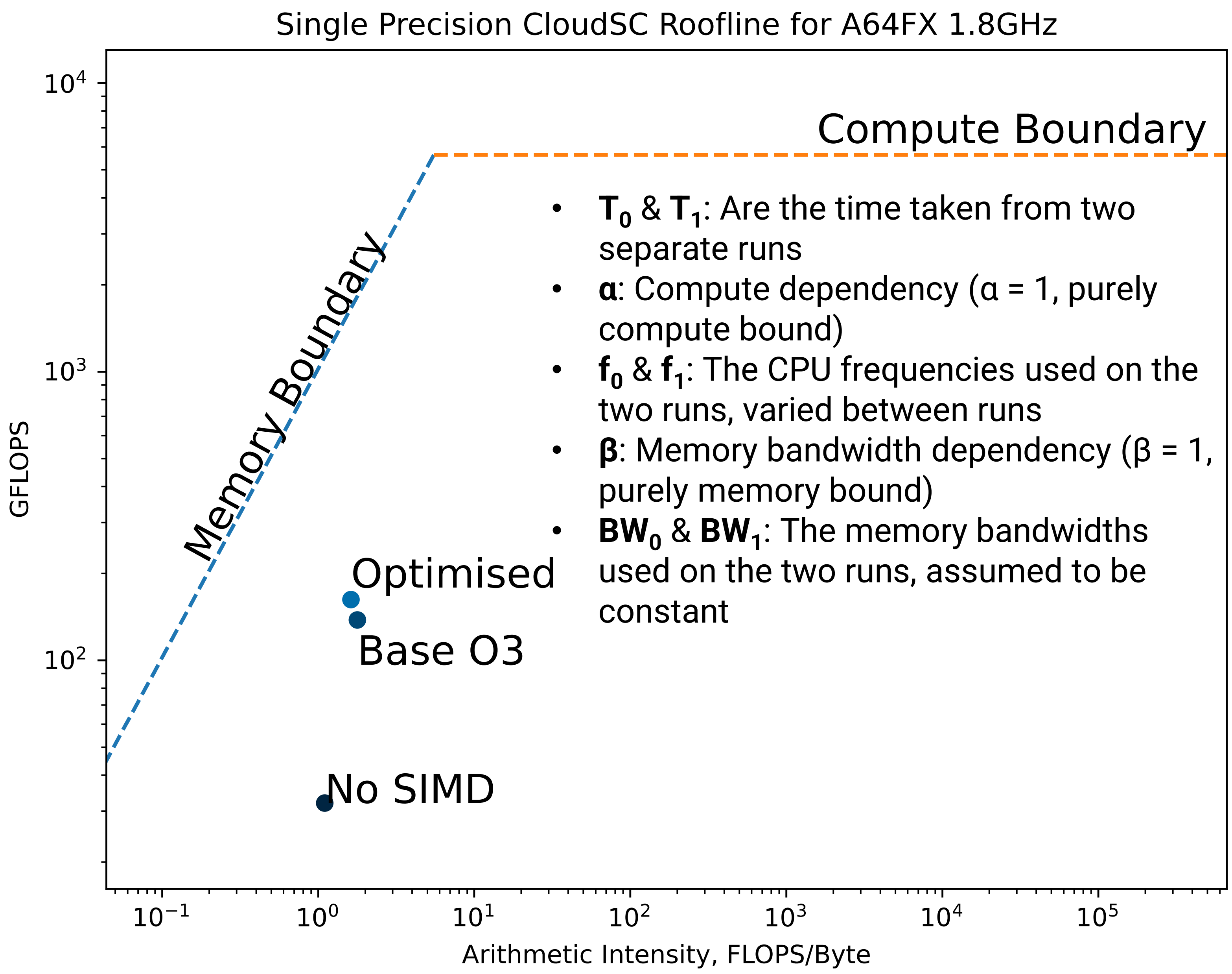
EUPEX ECMWF

Impact of memory bandwidth on CloudSC



Roofline graph for the A64FX processor of CloudSC with various optimisations applied. Theoretical bounds used

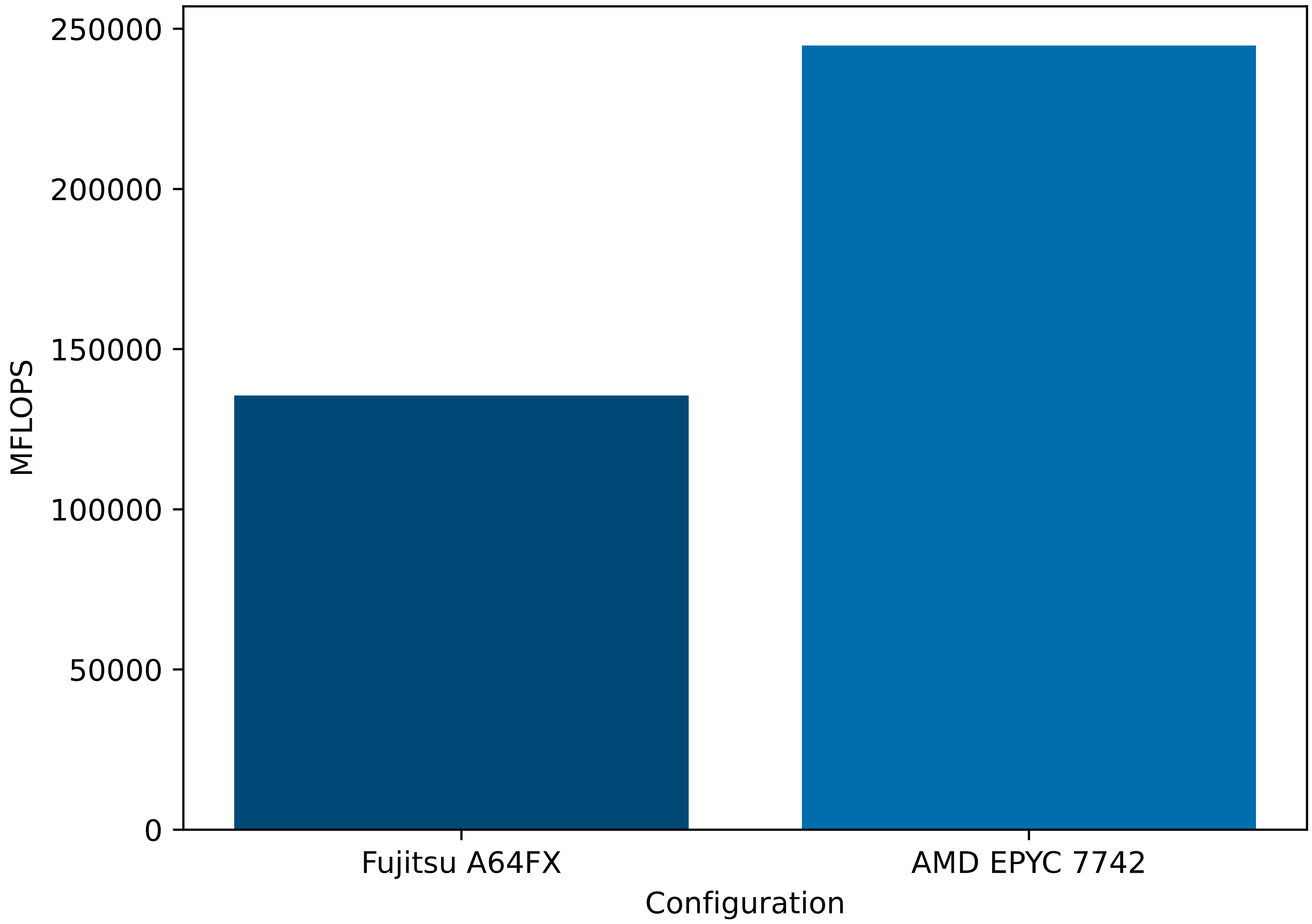
- CloudSC is known to be compute intensive and often not bound by memory bandwidth.
- Performance is modelled with a naïve equation: $T_1 = T_0\alpha(f_0/f_1) + T_0\beta(BW_0/BW_1)$
- Runs were taken varying only the CPU frequency, as it is the easiest of the two variables
- **α = 0.651 & β = 0.343 for CloudSC**
- **Not memory bound** at practical problem sizes



Roofline graph for the A64FX processor of CloudSC with various optimisations applied. Theoretical bounds used. Single precision used.

Work for the future

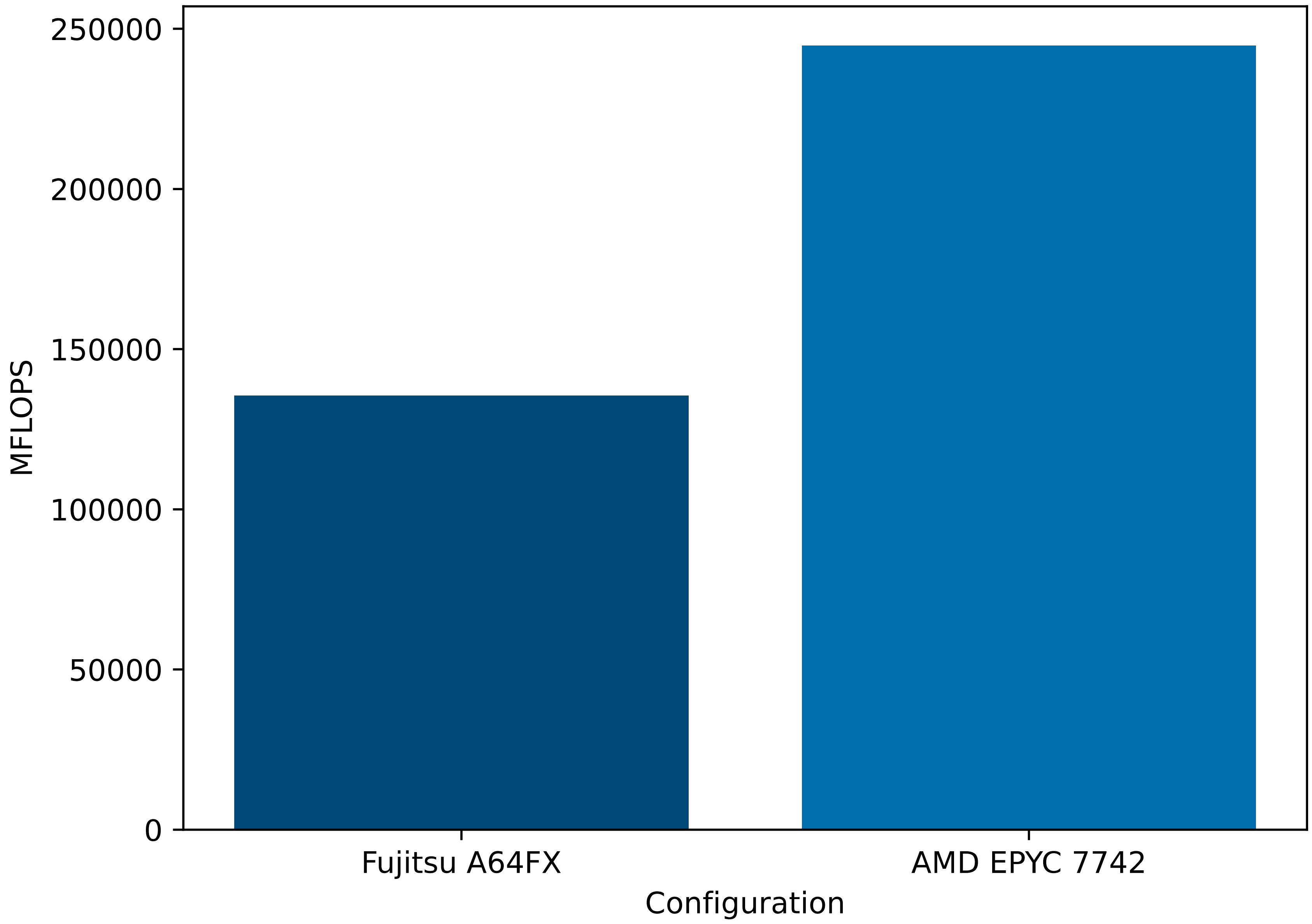
- Better overall performance
  - Currently CloudSC is **only achieving ~6%** of maximum theoretical FLOPS
  - How do we make it better?



CloudSC on a Fujitsu A64FX node vs ECMWF's current operational AMD EPYC 7742 node

EUPEX ECMWF

- Use **Loki to automate** the process of writing SVE kernels
  - A **source-to-source translation tool**
  - Recipes can be tailored to ECMWF code bases
  - We can make **more assumptions than the compiler can**
  - Directives can give more explicit hints to Loki
  - **Vectorise at the source code level**, so no reliance on new compilers



CloudSC on a Fujitsu A64FX node vs ECMWF’s current operational AMD EPYC 7742 node

EUPEX ECMWF

Loki in Action

```
s = 0
!Loop A
!$loki loop-unroll
do a=1, 3
    !Loop B
    do b=1, a
        s = s + a + b + 1
    end do
end do
```

Loop unroll →

```
s = 0
!Loop A
!$loki loop-unroll
!Loop B
s = s + 1 + 1 + 1
!Loop B
s = s + 2 + 1 + 1
s = s + 2 + 2 + 1
!Loop B
s = s + 3 + 1 + 1
s = s + 3 + 2 + 1
s = s + 3 + 3 + 1
```

Constant propagation →

```
s = 0
!Loop A
!$loki loop-unroll
!Loop B
s = 3
!Loop B
s = 7
s = 12
!Loop B
s = 17
s = 23
s = 30
```

Discoveries Since D3.2

- CloudSC wastes a lot of stack space on temporaries
  - This can pollute the cache and greatly decrease performance
- A lot of the branching can be eliminated with branchless programming techniques
  - CloudSC often branches within loops depending on the state of precipitation within clouds - rain, snow, hail, etc.
  - This enables more SIMD code generation

An Example of Branchless Programming

```fortran
DO JM=1,NCLV-1
  DO JK=1,KLEV
    DO JL=KIDIA,KFDIA
      IF (ZQX(JL,JK,JM)<RLMIN) THEN
        ZLNEG(JL,JK,JM) = ZLNEG(JL,JK,JM)+ZQX(JL,JK,JM)
        ZQADJ           = ZQX(JL,JK,JM)*ZQTMST
        tendency_loc%q(JL,JK) = tendency_loc%q(JL,JK)+ZQADJ
        IF (IPHASE(JM)==1) THEN
          tendency_loc%T(JL,JK) = tendency_loc%T(JL,JK)-RALVDCP*ZQADJ
        ENDIF
        IF (IPHASE(JM)==2) THEN
          tendency_loc%T(JL,JK) = tendency_loc%T(JL,JK)-RALSDCP*ZQADJ
        ENDIF
        ZQX(JL,JK,NCLDQV)   = ZQX(JL,JK,NCLDQV)+ZQX(JL,JK,JM)
        ZQX(JL,JK,JM)       = 0.0_JPRB
      ENDIF
    ENDDO
  ENDDO
ENDDO
```

```fortran
RALVVEC(1) = RALVDCP
RALVVEC(2) = RALSDCP

DO JM=1,NCLV-1
  DO JK=1,KLEV
    DO JL=KIDIA,KFDIA
      IF (ZQX(JL,JK,JM)<RLMIN) THEN
        ZLNEG(JL,JK,JM) = ZLNEG(JL,JK,JM)+ZQX(JL,JK,JM)
        ZQADJ           = ZQX(JL,JK,JM)*ZQTMST
        tendency_loc%q(JL,JK) = tendency_loc%q(JL,JK)+ZQADJ
        !IF (IPHASE(JM)==1) THEN
        !   tendency_loc%T(JL,JK) = tendency_loc%T(JL,JK)-RALVDCP*ZQADJ
        !ENDIF
        !IF (IPHASE(JM)==2) THEN
        !   tendency_loc%T(JL,JK) = tendency_loc%T(JL,JK)-RALSDCP*ZQADJ
        !ENDIF
        tendency_loc%T(JL,JK) = tendency_loc%T(JL,JK)-RALVVEC(JM)*ZQADJ
        ZQX(JL,JK,NCLDQV)   = ZQX(JL,JK,NCLDQV)+ZQX(JL,JK,JM)
        ZQX(JL,JK,JM)       = 0.0_JPRB
      ENDIF
    ENDDO
  ENDDO
ENDDO
```

EUPEX

EUPEX and ESiWACE CoE

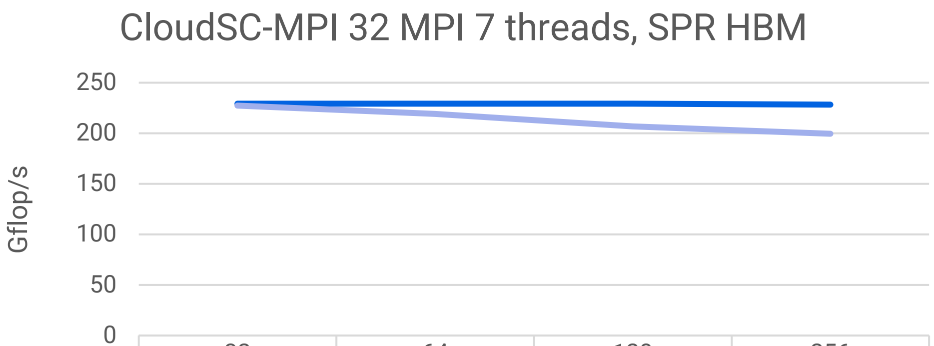
- ESiWACE: Excellence in Simulation of Weather and Climate in EU
  - **H**igh **P**erformance **C**limate and **W**eather benchmark suite
    - Compares performance & performance portability on different HPC systems (CPU, GPU, Memory hierarchy, etc.)
    - Deployed on various x86 and aarch64 CPUs, explore software stack capability and obtained performance metrics, identify code optimization opportunities and give feedback to developers and technology providers
    - Leads to exchange of ideas with developers, technology providers and HPC vendors (i.e. **co-design**)
    - Include IFS dwarfs: ecRad, ecTrans, Dwarf-P-CloudSC
      - The reference test cases are currently defined by ECMWF in ESiWACE: Configuration, reference input data, output data validation, timing extraction
    - Periodic meeting between Eviden and ECMWF to synchronize and exchange technical information

Co-funded by the European Union
This project has received funding from the EuroHPC JU under grant agreement No 101093054. Co-funded by the European Union.

EUPEX, ESiWACE CoE, and EPI

Co-Design in Action

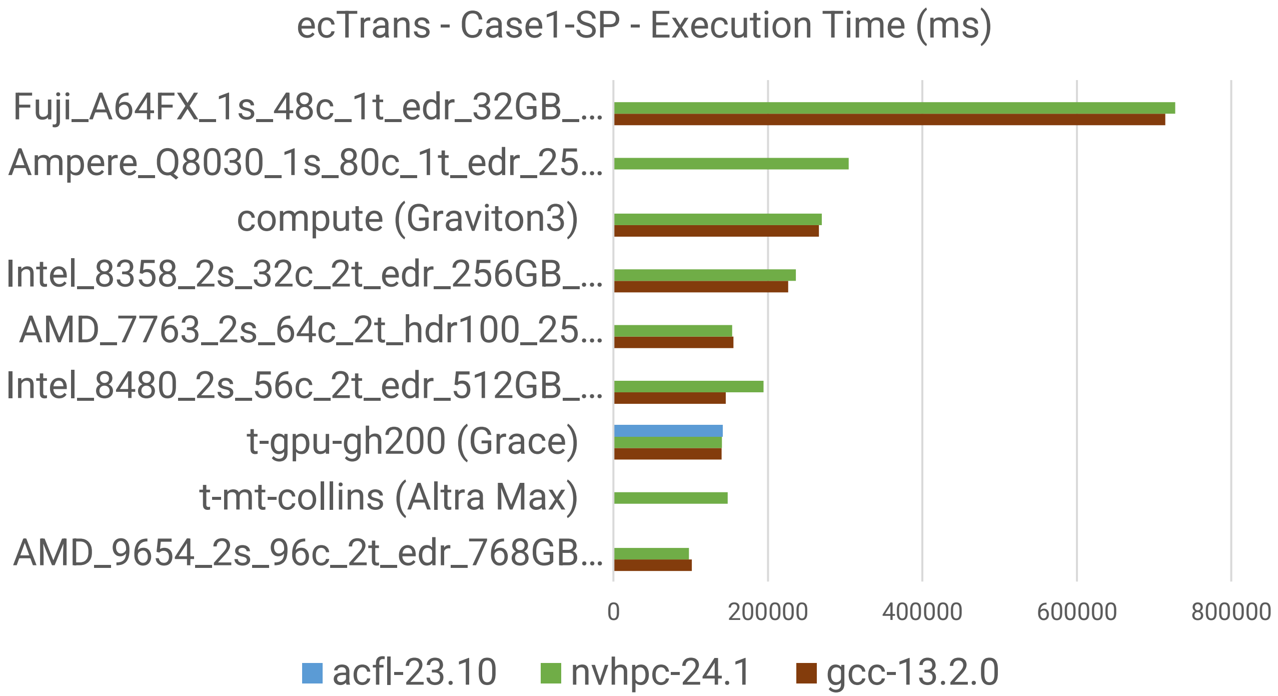
- ecTrans, early results
- CloudSC, HBM Assessment
  - *Intel Xeon Max (Sapphire Rapids w. HBM)*



	32	64	128	256
HBM	229,32	229,1	229,09	228,07
DDR	227,15	219,13	206,76	199,5

NPROMA

HBM DDR



+ many compiling / runtime issues recorded

Co-funded by the European Union

This project has received funding from the EuroHPC JU under grant agreement No 101033054. Co-funded by the European Union.