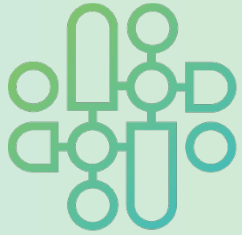


SC23
Denver, CO | i am hpc.

From Zero to Hero:
Conquering the Arm Neoverse

Brendan Bouffler (AWS)
Csaba Csoma (AWS)
John Linford (NVIDIA)
Matt Vaughn (AWS)
Conrad Hillairet (Arm Ltd)
Filippo Spiga (NVIDIA)

Compile and Execute

PART 1

I have no experience with Arm

Should I be worried ?

Overthinking:
The art of creating problems that weren't even there.

Most applications compile on Arm architecture with little to no modification

- All major Linux distributions support Arm, with extensive library of common packages (AArch64).
- Upgrading to newer library / software versions might be needed.
- GNU Compiler Collection (GCC) and LLVM are fully supported.
- Commercial compilers for Arm are available.

GNU compilers are a solid option on Arm

- GNU compilers are first class Arm compilers
  - Arm is one of the largest contributors to GCC
  - Focus on enablement and performance
  - Key for Arm to succeed in Cloud/Data center segment
- GNU toolchain ships with ACFL
  - Good Fortran support
  - Bug fixes and performance improvements in upcoming GNU releases
- Arm significant contributor to GNU

NVIDIA HPC Compiler Suite supporting Neoverse

- Former PGI Compilers & Tools evolved into the NVIDIA HPC SDK™
- Released multiple times year
  - Bundled with CPU and GPU libraries, Communication Library and Tools.
  - Free of charge and commercially supported.
- Support all three major CPU families used in HPC systems
  - X86_64
  - aarch64 (sbsa)
  - ppc64le
- Delivers state-of-the-art SIMD vectorization for both V1 and V2

arm COMPILER

Arm provided C/C++/Fortran compiler with best-in-class performance

Compilers tuned for Scientific Computing and HPC

Latest features and performance optimizations

Freely available

Tuned for Scientific Computing and HPC workloads

- Processor-specific optimizations for Neoverse-based platforms
- Optimal shared-memory parallelism using latest Arm-optimized OpenMP runtime

Linux user-space compiler with latest features

- C++ 17 and Fortran 2003 language support with OpenMP 4.5
- Support for Armv8-A and SVE architecture extension
- Based on LLVM and Flang, a leading open-source compiler project

Freely available on most Linux distributions

Documentation

- Fortran : https://developer.arm.com/documentation/101380/2210
- C/C++ : https://developer.arm.com/documentation/101458/2210

Compilers

How do I call them ?

arm
COMPILER
FOR LINUX

NVIDIA

`gcc`	`armclang`	`nvc`
`g++`	`armclang++`	`nvc++`
`gfortran`	`armflang`	`nvfortran`

Edit your Makefile, CFLAGS and FCFLAGS

Compiler flags might be different between ifort, gfortran, armflang, nvfortran and frt

- -qopenmp
- -mp
- -Kopenmp
- -fopenmp

Ensure that you are using the correct compiler options.
No SMT. Relaxed memory model.

Compiler flags

arm COMPILER FOR LINUX

FLAG		DESCRIPTION
`-mcpu=native or neoverse-v1, neoverse-v2, neoverse-512tvb`		Target the detected or specific CPU (the default mcpu is more generic/slower)
`-O1, -O2, O3, -Ofast`		Optimization level
`-fopenmp`		Turns on OpenMP
`-f(no-)vectorize` *acfl*	`-f(no)tree-vectorize` *gcc*	Enables vectorization
`-f(no-)stack-arrays`		Place all automatic arrays on stack memory (enabled by default with -Ofast).
`-f(no-)inline-functions`		Enables function inlining.
`-f(no-)associative-math`		Allows the re-association of FP operands
`-f(no-)lto`		Enables Link Time Optimizations (LTO)
`-f(no-)unroll-loops`		Enables loop unrolling
`-ffast-math`		Enables aggressive, lossy FP optimizations.
`-ffp-contract=fast -ffp-contract=off`		Enables FP expression contraction(i.e. FMAs)

NVIDIA

FLAG	DESCRIPTION
`-mcpu=[...] or –tp host,native,neoverse-v1,neoverse-v2`	Target the given or detected CPU Enabled by default
`-O1, -O2, -O3,-O4 -fast (-O2 –Munroll=c:1 –Mlre –Mautoinline –Mvect=simd) -Ofast (-O3 -Mfprelaxed -Mstack_arrays)`	Optimization level
`-mp`	Turns on OpenMP (‘*multicore*’ is the default)
`-Munroll`	Enables loop unrolling
`-Mvect[=…] / -Mnovect simd/nosimd,assoc/noassoc,fuse,gather`	Enables loop vectorization Enables/Disables SIMD instructions code generation
`-Mipa=fast,inline`	Option invokes interprocedural analysis (IPA) and enables automatic procedure inlining.
`-Minline`	Enables function inlining.

arm PERFORMANCE LIBRARIES

Optimized BLAS, LAPACK and FFT for HPC applications

Freely available

Best-in-class performance

Validated with
NAG test suite

Arm provided 64-bit Armv8-A math libraries

- Optimized BLAS, LAPACK, FFT and math.h routines
- FFTW compatible interface for FFT routines
- Sparse linear algebra and batched BLAS support
- Minimum Linux kernel version 5.10 is needed for SVE

Best-in-class serial and parallel performance

- Generic Armv8-A optimizations by Arm
- Tuned for Arm Neoverse family of processors

Validated by Arm Engineers

- Validated with NAG's test suite, a de-facto standard
- Community supported

Documentation

- https://developer.arm.com/documentation/102620/0100

Arm Performance Libraries

Optimized BLAS, LAPACK and FFT

FLAG	DESCRIPTION
`–I$ARMPL_DIR/include`	Include directory
`-L$ARMPL_DIR/lib -larmpl_mp`	Link against ArmPL Here OpenMP version (_mp)
`-L$ARMPL_DIR/lib -lamath`	Link against libamath
`(-lgfortran) -lm`	Always finish by linking with -lm

arm COMPILER FOR LINUX

FLAG	DESCRIPTION
`-armpl`	Use with –mcpu(=native)
`-armpl=serial`	Single threaded version
`-armpl=parallel`	Default with -fopenmp

NVIDIA

FLAG	DESCRIPTION
`–I$ARMPL_DIR/include`	Include directory
`-L$ARMPL_DIR/lib -larmpl_mp`	Link against ArmPL Here OpenMP version (_mp)
`-lm -lnvf -lnvc -lrt`	Always finish by linking with these libraries

MPI & OpenMP binding and placement

On which cores am I running?

Some flags, commands and environment variables

- `--report-bindings`
- `htop`
- `OMP_DISPLAY_ENV`
- `OMP_DISPLAY_AFFINITY`

- `--map-by ppr:32:node:PE=2`
- `OMP_NUM_THREADS=2`
- `OMP_PROC_BIND=close`
- `OMP_PLACES=cores`

Useful OpenMP Runtime Environment Variables

OMP_NUM_THREADS

- Normally defaults to the number of cores on the system, i.e. 64 on HPC7g/C7g.16xlarge

OMP_DISPLAY_ENV

- Set to “true” or “verbose” to see values of run-time variables

OMP_PROC_BIND

- Strongly recommended to set OMP_PROC_BIND to “true”, “close” or “spread” depending on needs
- If unset, the threads are not pinned to cores and may migrate around the system

OMP_PLACES

- Description of unit of pinning ‘threads’, ‘cores’ or ‘sockets’

OMP_DYNAMIC=true and OMP_NESTED=true

- Necessary for “nested parallelism” (multi-threaded functions from within an already parallel environment)

Guided Hands-Ons || Bring Your Own Code

50 MINUTES

Part 1 – Practical hands-on

Goal: compile the code(s), check correctness, experiment with different compilers and libraries *(which combination is faster?)*, single/multi-instances

Options:

1. "BYOC" - **B**ring **Y**our **O**wn **C**ode
2. `code_saturne` play-through
3. Other popular HPC apps (follow the links)

https://github.com/aws/aws-graviton-getting-started/tree/main/HPC

https://github.com/arm-hpc-devkit/nvidia-arm-hpc-devkit-users-guide

Code_Saturne

Reference - Build instructions on x86 with Intel toolchain

Install

```
source ~/intel/oneapi/setvars.sh
CC="mpiicc"
CXX="mpiicpc"
FC="ifort"
DEST=$HOME/code_saturne_7.2.0
mkdir build
cd build
../configure CC="$CC" CXX="$CXX" FC="$FC" \
            --with-blas=$MKLROOT --prefix=$DEST \
            --disable-gui --without-med \
            --without-hdf5 --without-cgns \
            --without-metis --disable-salome \
            --without-salome --without-eos \
            --disable-static --enable-long-gnum
make
make install
```

Run

- Download test cases https://github.com/code-saturne/saturne-open-cases
- Two test cases:
  - C016 allows very fast computations (single core)
  - F128 more precise simulations (single node)
- **First step**: generate the meshes that will be used later
  - `cd $REPBASE/BENCH_C016_PREPROCESS/DATA`
  - `$DEST/bin/code_saturne run`
    - Outputted data are in $REPBASE/BENCH_C016_PREPROCESS/RESU/extrude_16
    - C016 : has 1 024 cells as an input and 17 408 as the output
  - `cd $REPBASE/BENCH_F128_PREPROCESS/DATA`
  - `$DEST/bin/code_saturne run`
    - Outputted data are in $REPBASE/BENCH_F128_PREPROCESS/RESU/extrude_128
    - F128 : 100 040 cells in input, 12 905 160 cells in output
- **Second step**: run C016 on one node with 48 cores and 1 OMP thread:
  - `cd $REPBASE/BENCH_C016_01/DATA`
  - `$DEST/bin/code_saturne submit --wckey dtsi:undefined --nodes=1 -n 48 --exclusive --ntasks-per-node=48 --cpus-per-task=1`

Code: https://github.com/code-saturne/code_saturne/archive/refs/tags/v7.2.0.tar.gz

Adapt configure command to your environment

x86 to aarch64

```
CC="mpiicc"
CXX="mpiicpc"
FC="ifort"
DEST=$HOME/code_saturne_7.2.0
mkdir build
cd build
../configure CC="$CC" CXX="$CXX" FC="$FC" \
             --with-blas=$MKLROOT --prefix=$DEST \
             --disable-gui --without-med \
             --without-hdf5 --without-cgns \
             --without-metis --disable-salome \
             --without-salome --without-eos \
             --disable-static --enable-long-gnum
make
make install
```

```
CC=mpicc CXX=mpicxx FC=armflang ./configure \
 --with-blas=$ARMPL_DIR --prefix=$PWD/build \
 --disable-gui --without-med                \
 --without-hdf5 --without-cgns              \
 --without-metis --disable-Salome           \
 --without-salome --without-eos             \
--disable-static --enable-long-gnum
```

Adapt configure command to your environment

x86 to aarch64

```
[...]
checking for python3 extension module
directory...
${exec_prefix}/lib64/python3.7/site-packages
checking for dlopen... dlopen
checking for MKL libraries... no
checking for threaded ATLAS BLAS... no
checking for ATLAS BLAS... no
checking for legacy C BLAS... no
configure: error: in `/home/ec2-
user/code_saturne-7.2.0':
configure: error: BLAS support is requested,
but test for BLAS failed!
See `config.log' for more details
```

- **Need to introduce support for ArmPL**
- **Modify one file**
- **Adapt ATLAS and MKL detection mechanisms**

Adapt configure command to your environment

x86 to aarch64

```
[...]
checking for python3 extension module directory...
${exec_prefix}/lib64/python3.7/site-packages
checking for dlopen... dlopen
checking for MKL libraries... no
checking for threaded ATLAS BLAS... no
checking for ATLAS BLAS... no
checking for legacy C BLAS... no
configure: error: in `/home/ec2-user/code_saturne-
7.2.0':
configure: error: BLAS support is requested, but test
for BLAS failed!
See `config.log' for more details
```

- **Need to introduce support for ArmPL**
- **Modify one file**
- **Adapt ATLAS and MKL detection mechanisms**

Adapt configure command to your environment

x86 to aarch64

```
[...]
${exec_prefix}/lib64/python3.7/site-packages
checking for dlopen... dlopen
checking for MKL libraries... no
checking for threaded ATLAS BLAS... no
checking for ATLAS BLAS... no
checking for legacy C BLAS... no
checking for ArmPL libraries... yes
checking for MPI (MPI compiler wrapper test)... yes
checking for MPI I/O... yes
checking for MPI in place... yes
checking for MPI nonblocking barrier... yes
CCMIO headers not found
[...]
```

Need to introduce support for ArmPL

Modify one file

Adapt ATLAS and MKL detection mechanisms

Adapt configure command to your environment

x86 to aarch64

```
[...]
checking for a sed that does not truncate output...
/usr/bin/sed
checking for ar... ar
checking the archiver (ar) interface... ar
configure: sourcing config/cs_auto_flags.sh (test for
known compilers)
compiler 'mpicc' is NVIDIA compiler
compiler 'mpicxx' is NVIDIA compiler
compiler 'armflang' is NVIDIA compiler
checking how to print strings... printf
[...]
```

- **Bug in compiler detection mechanism**
- **Arm Compiler for Linux detection overwritten by NVIDIA detection**
- **Two files to modify (~10-20 lines)**

Adapt configure command to your environment

x86 to aarch64

```
[...]
checking for ar... ar
checking the archiver (ar) interface... ar
configure: sourcing config/cs_auto_flags.sh
(test for known compilers)
compiler 'mpicc' is Arm C compiler
compiler 'mpicxx' is Arm C++
compiler 'armflang' is Arm Fortran compiler
checking how to print strings... printf
[...]
```

Bug in compiler detection mechanism

Arm Compiler for Linux detection overwritten by NVIDIA detection

Two files to modify (~10-20 lines)

Compile

x86 to aarch64

```
make -j 4 > resComp >&1
make install


grep –i err ./resComp | wc –l
0
```

No problem, compiles OoB

Flat MPI version

x86 to aarch64

```
CC=mpicc CXX=mpicxx FC=gfortran ./configure --
prefix=$PWD/build-noomp --disable-gui --without-hdf5 --
without-cgns --without-med --without-metis --disable-static  -
-disable-salome --without-salome --without-eos  --enable-long-
gnum --with-blas=$ARMPL_DIR --disable-openmp > resConf 2>&1

grep -i openmp ./resConf
[…]
AArch64_RHEL-7_aarch64-linux/bin/../lib/gcc/aarch64-linux-
gnu/11.2.0/../../.. -lgfortran -lm
checking for OpenMP (C)... yes
checking for OpenMP (Fortran)... yes
checking for Fortran libraries of gfortran... (cached) OpenMP
support: no
 OpenMP support: yes
 OpenMP accelerator support: no
 OpenMP Fortran support: yes
 Auto load environment modules:  binutils/11.2.0 gnu/11.2.0
armpl/22.1.0 openmpi/gcc/4.1.4
```

Bug in OpenMP configuration
not Arm specific
also happens on x86

One file to modify (~1 line)

Flat MPI version

x86 to aarch64

```
CC=mpicc CXX=mpicxx FC=gfortran ./configure --
prefix=$PWD/build-noomp --disable-gui --without-hdf5 --
without-cgns --without-med --without-metis --disable-static  -
-disable-salome --without-salome --without-eos  --enable-long-
gnum --with-blas=$ARMPL_DIR --disable-openmp > resConf 2>&1

grep -i openmp ./resConf
[…]
OpenMP support: no
OpenMP support: no
 Auto load environment modules:  binutils/11.2.0 gnu/11.2.0
armpl/22.1.0 openmpi/gcc/4.1.4


make > resComp 2>&1
grep fopenmp ./resComp | wc -l
0
```

Bug in OpenMP configuration
not Arm specific
also happens on x86

One file to modify (~1 line)

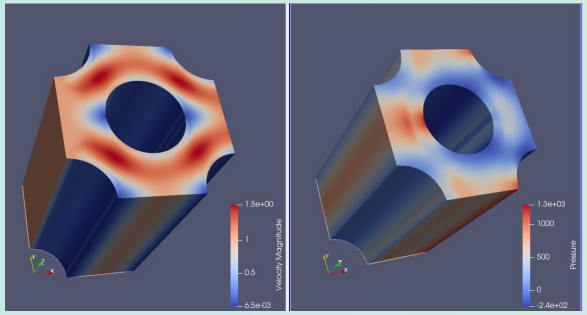
Run

x86 to aarch64

No problem at runtime

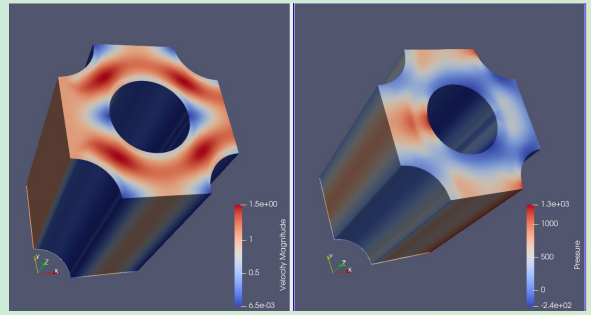
```
First step : generate the meshes that will be used later
cd $REPBASE/BENCH_F128_PREPROCESS/DATA
$DEST/bin/code_saturne run

Second step: run testcase C016 on one node with 48 cores
and 1 thread per MPI task :
cd $REPBASE/BENCH_C016_01/DATA
$DEST/bin/code_saturne submit --wckey dtsi:undefined --
nodes=1 -n 48 --exclusive --ntasks-per-node=48 --cpus-
per-task=1
```



```
First step : generate the meshes that will be used later
cd $REPBASE/BENCH_F128_PREPROCESS/DATA
$DEST/bin/code_saturne run

Second step: run testcase C016 on one node with 64 cores
and 1 thread per MPI task :
cd $REPBASE/BENCH_C016_01/DATA
$DEST/bin/code_saturne submit -n 64 --cpus-per-task=1
```

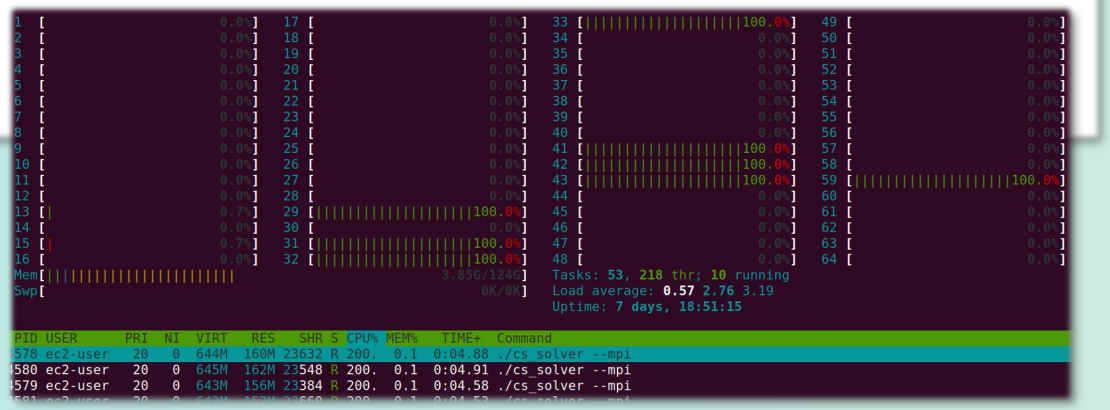


Another run with MPI & OpenMP

x86 to aarch64

Not Arm specific

```
code_saturne submit -n 4 --nt 2 --cpus-per-task=2
```




```
export CS_MPIEXEC_OPTIONS="--map-by
ppr:4:node:PE=2 --report-bindings"
code_saturne submit -n 4 --nt 2
[ip-172-31-77-39:04714] MCW rank 0 bound to
socket 0[core 0[hwt 0]], socket 0[core 1[hwt 0]]:
[B/B/./././/././././././././././././././././././././././././
././././././././././././././././././././././././././././././.
/././././././././././././././.]
[ip-172-31-77-39:04714] MCW rank 1 bound to
socket 0[core 2[hwt 0]], socket 0[core 3[hwt 0]]:
[././B/B/./././/./././././././././././././././././././././././
././././././././././././././././././././././././././././././.
/././././././././././././././.]
[…]
```