Best Practice Guide - ARM64

Xu Guo, EPCC, UK

Cristian Morales, BSC, Spain

Ole Widar Saastad, University of Oslo, Norway

Anastasia Shamakina, HLRS, Germany

Wim Rijks (Editor), SURFsara, Netherlands

Volker Weinberg (Editor), LRZ, Germany

Version 2.0 by 08-02-2019

Table of Contents

1. Introduction

The ARM processor is heavily used in mobile phones and has the reputation of being very energy efficient. This has peaked interest to use this processor as building block in HPC systems, because energy efficiency has always been a major concern in developing Peta-scale and Exa-scale computers.

This best practice guide provides information about the ARM64 architecture and the programming models that programmers can use in order to achieve good performance with their applications on this architecture.

The guide gives a description of the hardware of the ARM64 processor. It provides information on the programming models and development environment as well as information about porting programs. Furthermore it provides information about tools and strategies on how to analyze and improve the performance of applications.

Finally, there is a description of test and production systems, that already exist in Europe or are planned in the near future.

2. System Architecture / Configuration

2.1. Processor Architecture / MCM Architecture

2.1.1. Huawei / High Silicon

The ARM version used for testing is produced by Huawei and High Silicon (a company owned 100% by Huawei). The system used for evaluation is a Huawei Taishan 2280 dual socket system with the following characteristics:

Huawei TaiShan 2280 characteristics

- Dual socket system
- Processors: Two ARM Hi1616 64 bit, 32 core, 2.4 GHz processors, Cortex-A72 cores
- Memory: 256 GiB RAM, DDR4, 8x32 GiB, Micron, 2400 MHz
- Memory is distriobuted in Four NUMA banks, 64 GiB each
- Network: 2x GbE + 2x 10GbE

HiSilicon 1616 processor cores

- Full ARMv8-A support
- Superscalar structure and out-of-order pipeline execution
- Dynamic branch prediction, including branch target buffer (BTB), global history buffer (GHB), return stack, and indirect prediction
- Prefetch is supported. The regular sequence for reading addresses will lead to the pre-placement of non-accessed cachelines for later addresses.
- Cacheline size 128 bytes (1024 bits)
- Level 1 cache 1 MiB, 2-way Set-associative
- Level 2 cache 8 MiB, 16-way Set-associative
- Level 3 cache 32MiB, 16-way Set-associative
- SIMD instruction set (32× 128-bit registers)

The exact internal specifications of the Hi1616 processor is available in the processor manual from HiSilicon (available upon request). Hi1616 does accept the compiler flags : -march=armv8-a[1] -mcpu=cortex-a72. E.g. a Cortex-72 version of the ARM architecture.

2.1.2. Cavium / ThunderX

Cavium is dedicated as a customer reference platform of the ThunderX® systems family that hosts ThunderX 2K – one of two platforms of the ThunderX family [3].

Cavium serves foremost the testing purposes for the state-of-the-art hardware and software solutions of ARM in order to evaluate ThunderX performance and features based on customers’ specific applications.

ThunderX 2K reference platform is a 1/2 SSI form factor sled inside of a 2U chassis. The 2U chassis can accommodate 4 of such sleds and each of the sleds being a dual socket design.

[1] It should also be noted that the vector instructions have been renamed from version 7 to version 8. They no longer start with the prefix 'v', example: from 'vmul d0, d0, d1' in version 7 to 'mul v0.u8, v0.u8, v1.u8' in version 8.

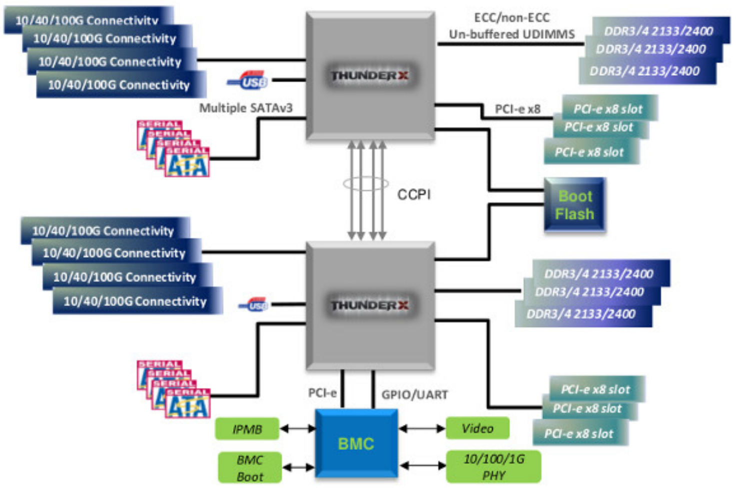
The ThunderX 2K reference platform has several configurations including CN8800-2S1N-128G-2.0-CDK. The considered configuration has the CN8890-Series processor as a bootable system-on-chip (SoC). ThunderX implements a fully-compliant ARMv8 64-bit instruction set architecture (ISA). The processor has the dual-socket configuration with 48 cores per socket.

Cavium ThunderX CN8890-Series processor characteristics

- Dual Socket Configuration with 48 cores per socket
- 64-bit ARMv8 server SoC
- Running up to 2.5GHz each, up to 160 GOPS
- 78 KiB-L cache and 32 KiB -D cache per core, 16 MiB shared L2
- 128 virtual NICs Package: 52.5x52.5 mm 2601 BGA

The ThunderX family supports dual socket solutions (Figure 1) using Cavium Coherent Processor Interconnect (CCPI^{TM}). This allows creation of nodes with up to 96 cores. The dual socket CCPI systems allow creation of many core nodes. Developers can expand the number of nodes using Cavium's scalable fabric available on some chips in the family. The fabric provides high speed connectivity in an XY mesh with connections to the four nearest neighbors in a 2D fabric. The architecture also supports 3D fabric connections. Nodes connected via the fabric do not share I/O or memory but they can deliver messages to applications anywhere in the fabric. This provides a way to build systems with thousands of nodes. Fabric monitoring and SLA features allow management and tracking of very large clusters [4].

Figure 1. Cavium ThunderX design for the dual-socket system



2.1.3. Cavium / ThunderX2

Figure 2. Cavium ThunderX2 ARM Processor

The second generation ThunderX2® product family introduced by Cavium has been released for general availability in early 2018. ThunderX2 is a family of 64-bit ARMv8 processors rebranded by Cavium based on the original design of Broadcom’s Vulcan. ThunderX2 is fully compliant with ARMv8 architecture and ARM's Server Base System Architecture (SBSA) standard.[5][6]

In the ThunderX2 CN99XX series, the ARM based System-on-Chip (SoC) integrates high performance custom fully out-of-order (OOO) cores, supporting up to 32 ARMv8.1 cores in a single socket configuration and 64 cores in a dual socket configuration, with the frequency up to 2.5GHz in nominal mode and 3GHz in Turbo mode. Simultaneous Multithreading (SMT) is supported which allows up to 4 threads per physical core. [7]

ThunderX2 ARM based SoC is fully cache coherent across dual sockets using the 2nd generation of Cavium Coherent Processor Interconnect (CCPI) with a speed of 600Gbps. Each core has 32 KB L1 instruction and data cache, as well as 256 KB L2 cache. The 32 MB L3 cache is distributed among the cores. [5][7]

Each ThunderX2 ARM processor provides multiple, up to 8, DDR4 memory controllers per chip with the capability of up to 4TB in a dual socket configuration. Up to 56 PCIe Gen3 lanes (supported widths including x1, x2, x4, x8, and x16) are supported with integrated IPO and SATAv3 ports. [5][7]

2.2. Building Block Architecture

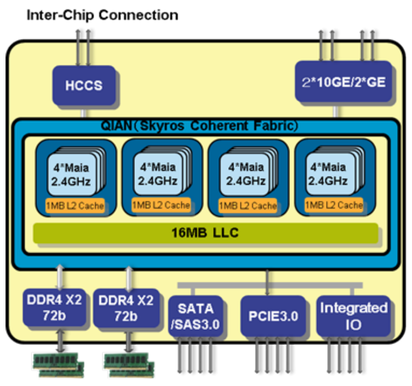
2.2.1. Huawei / High Silicon

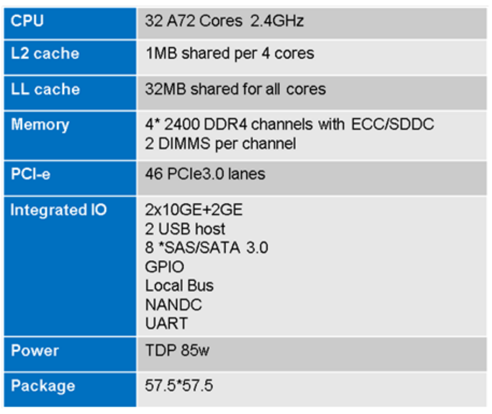
An overview of the units within the HiSilicon Hi1616 processor is shown in Figure 3.

Dual socket systems are using the HCCS cache coherence interconnect for inter chip connection. It's assumed that the cache-coherence system in the QIAN (Skyros Coherent Fabric) extend through the HCCS to make a dual socket cache coherent system.

Two memory controllers per processor yield 4 NUMA banks for a dual socket system. See (Section 2.3.1) for more on NUMA banks.

Figure 3. HiSilicon 1616 block diagram





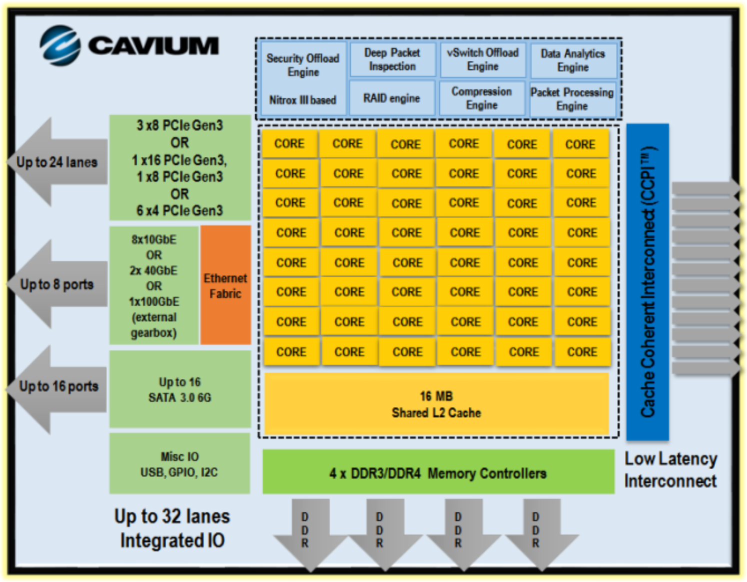
CPU	32 A72 Cores 2.4GHz
L2 cache	1MB shared per 4 cores
LL cache	32MB shared for all cores
Memory	4* 2400 DDR4 channels with ECC/SDDC 2 DIMMS per channel
PCI-e	46 PCIe3.0 lanes
Integrated IO	2x10GE+2GE 2 USB host 8 *SAS/SATA 3.0 GPIO Local Bus NANDC UART
Power	TDP 85w
Package	57.5*57.5

2.2.2. Cavium / ThunderX

An overview of the building blocks that Cavium has used to build the ThunderX is shown in Figure 4. Depending on the target market, some of these building blocks are removed to reduce power consumption or to increase the clock speed [8].

The dual socket CCPI systems allow creation of many core nodes. Developers can expand the number of nodes using Cavium's scalable fabric available on some chips in the family. The fabric provides high speed connectivity in an XY mesh with connections to the four nearest neighbors in a 2D fabric. The architecture also supports 3D fabric connections. Nodes connected via the fabric do not share I/O or memory but they can deliver messages to applications anywhere in the fabric. This provides a way to build systems with thousands of nodes. Fabric monitoring and SLA features allow management and tracking of very large clusters.

Figure 4. Cavium ThunderX building blocks

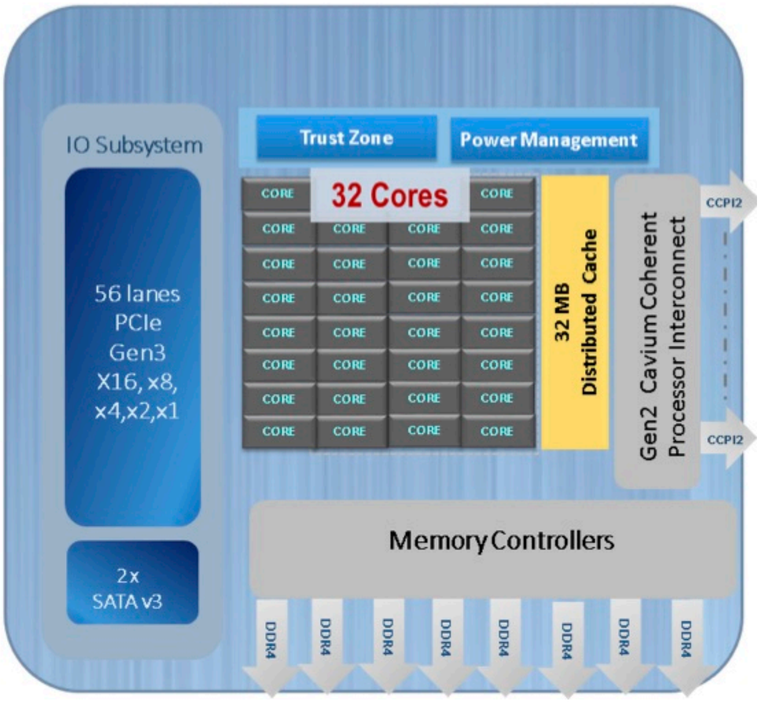


2.2.3. Cavium / ThunderX2

Figure 5 shows the design for the Cavium Thunder X2 CN99XX product with the single socket configuration. Dual-socket configuration is also supported - the interconnection between the two sockets/NUMA nodes has been upgraded to the Cavium Coherent Processor Interconnect 2 (CCPI2) from the original CCPI that used for the first generation ThunderX processors. The speed of CCPI2 is 600Gbps. [5] [7]

The nodes are connected with different interconnects according to the vendors' choices. Please refer to the Chapter about "European ARM-Based systems" for the specific ARM-based systems' interconnect info.

Figure 5. Cavium ThunderX2 CN99XX product design (single socket) [7]



2.3. Memory Architecture

2.3.1. Huawei / High Silicon

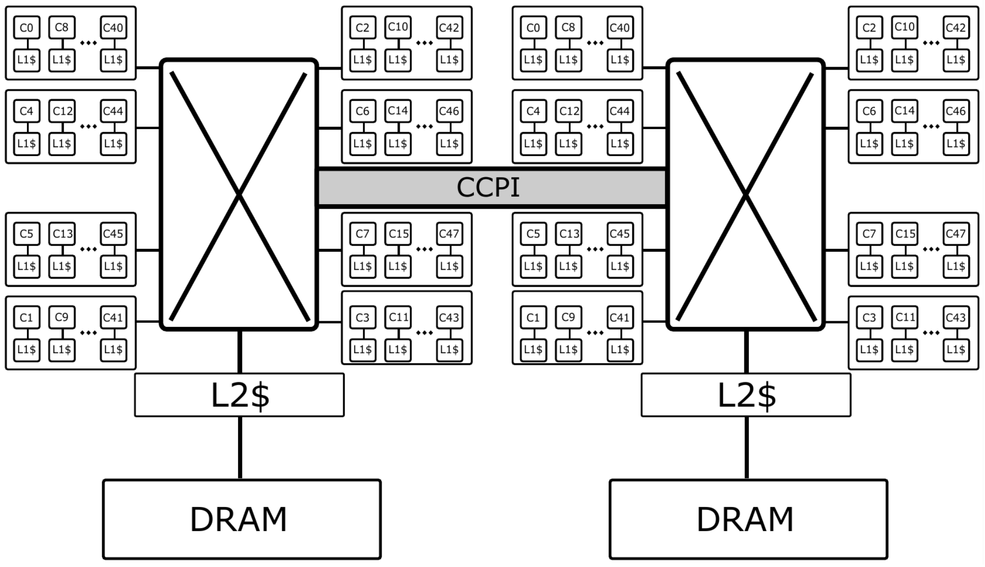
From a programmers point of view the command *numactl -H* provides an overview of the NUMA architecture. Below is shown the output obtained from the Huawei system:

```
numactl -H
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
node 0 size: 64353 MB
node 0 free: 59467 MB
node 1 cpus: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
node 1 size: 64446 MB
node 1 free: 62318 MB
node 2 cpus: 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
node 2 size: 64446 MB
node 2 free: 57445 MB
node 3 cpus: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
node 3 size: 64443 MB
node 3 free: 91 MB
node distances:
node   0   1   2   3
  0:  10  15  20  20
  1:  15  10  20  20
  2:  20  20  10  15
  3:  20  20  15  10
```

2.3.2. Cavium / ThunderX

A single ThunderX SoC can include up to 48 cores. Full cache coherence is maintained across all 96 cores on a node using CCPI as shown in Figure 6 [37]. Each core has its own 78 KiB L1 instruction cache and a 32 KiB L1 data cache. The cores on the same socket share a 16 MiB L2 cache. Cavium ThunderX has no L3 cache. All cores in the system are cache coherent in respect to L1, L2 and DMA accesses. The node has 128 GiB DDR4 DRAM: 288-pin UDIMM with ECC per socket up to 2133MHz.

Figure 6. Two Cavium ThunderX SoC connected via CCPI



The main properties of the L1 and L2 caches are summarized in the following table (from [9]).

L1 data cache	**Policy**	Write-through
	Type	Private
	Size	32Kb
	Associativity	32-way
	Block	128-bytes
L1 instruction cache	**Size**	78Kb
	Associativity	39-way
	Block size	128-bytes
L2 cache	**Policy**	Write-back
	Type	Shared
	Size	16MB
	Associativity	16-way
	Block	128-bytes

2.3.3. Cavium / ThunderX2

Full cache coherency is supported across the dual sockets. Each core has 32 KB L1 instruction and data cache. L2 cache per core is 256 KB. 32 MB L3 cache is distributed among the cores. [7]

Multiple DDR4 72-bit memory controllers (up to 8) are supported on each chip, with the speed of Up to 2666MHz in 1DPC and up to 2400MHz in 2DPC. The dual socket configuration has the capability of 1+TB and up to 4TB of memory.[5][7]

Please refer to the Chapter about "European ARM-Based systems" for the specific ARM-based systems’ memory architecture info, e.g. the NUMA architecture overview, etc.

3. Programming Environment / Basic Porting

3.1. Available Compilers

There are several compilers available for the ARM architecture. The GNU set of compilers comes with the Linux distribution, while a set of open-source compilers (FLANG, CLANG, CLANG C++) for integration with LLVM comes from the public GitHub repository [https://github.com/flang-compiler/]. This project is being spearheaded by the Lawrence Livermore, Sandia and Los Alamos national laboratories. The ARM HPC compiler is a set of compilers made by the ARM consortium, ARM HPC development tools [https://developer.arm.com/products/software-development-tools/hpc]. Cray's CCE compilers, included in the Cray Programming Environment, are also supported for the ThunderX2 ARM processor. [10]

Compilers installed on Huawei High Silicon:

- GNU compiler suite *gcc, gfortran, g++* (version 7.2.0)
- ARM compilers for HPC *armclang, armclang++, armflang* (version 18.0)

Currently the GNU and ARM compilers have been tested and evaluated on Ubuntu 16.04.

Compilers installed on Cavium ThunderX:

- GNU compiler suite *gcc, gfortran, g++* (version 7.2.0)
- LLVM compilers *clang, clang++, flang* (version 6.0.1)

Currently the GNU and LLVM compilers have been tested and evaluated on Ubuntu 16.04.

Compilers installed on Cavium ThunderX2 (GW4 Isambard's Cray XC50/ARM-based early access system)

- Cray compilers (included in PrgEnv-cray/6.0.5 with cce version 8.7.0.5323)
- GNU compiler suite *gcc, gfortran, g++* (version 6.1.0 , 7.1.0, 7.2.0, 8.1.0)
- ARM compilers for HPC *armclang, armclang++, armflang* (version 18.1, 18.2, 18.3, 18.4, 18.4.2)

Currently the Cray, GNU, and ARM compilers have been tested and evaluated on the GW4 Isambard's Cray XC50/ARM-based early access system with SUSE Linux 12.3

3.1.1. Compiler Flags

3.1.1.1. Huawei / High Silicon

Table 1. Suggested compiler flags for ARM HPC compilers

Compiler	Suggested flags
armclang	-Ofast -march=armv8-a -mcpu=cortex-a72 -fomit-frame-pointer
armclang++	-Ofast -march=armv8-a -mcpu=cortex-a72 -fomit-frame-pointer
armflang	-Ofast -march=armv8-a -mcpu=cortex-a72 -fomit-frame-pointer

The ARM compiler flags do not differ very much from the GNU compiler flags. Loop unrolling etc are similar, but prefetch will work differently as there are no software prefetch instructions.

Table 2. Suggested compiler flags for GNU compilers

Compiler	Suggested flags
gcc	-Ofast -march=armv8-a -mcpu=cortex-a72 -floop-optimize -falign-loops -falign-labels -falign-functions -falign-jumps -fomit-frame-pointer
g++	-Ofast -march=armv8-a -mcpu=cortex-a72 -floop-optimize -falign-loops -falign-labels -falign-functions -falign-jumps -fomit-frame-pointer
gfortran	-Ofast -march=armv8-a -mcpu=cortex-a72 -floop-optimize -falign-loops -falign-labels -falign-functions -falign-jumps -fomit-frame-pointer

3.1.1.2. Cavium / ThunderX

Table 3. Suggested compiler flags for GNU compilers

Compiler	Suggested flags
gcc	-Ofast -march=armv8-a -mcpu=thunderxt88 -fopenmp -floop-optimize -falign-loops -falign-labels -falign-functions -falign-jumps -fomit-frame-pointer
g++	-Ofast -march=armv8-a -mcpu=thunderxt88 -fopenmp -floop-optimize -falign-loops -falign-labels -falign-functions -falign-jumps -fomit-frame-pointer
gfortran	-Ofast -march=armv8-a -mcpu=thunderxt88 -fopenmp -floop-optimize -falign-loops -falign-labels -falign-functions -falign-jumps -fomit-frame-pointer

Table 4. Suggested compiler flags for LLVM compilers

Compiler	Suggested flags
clang	-Ofast -march=armv8-a -mcpu=thunderxt88 -fopenmp -fomit-frame-pointer
clang++	-Ofast -march=armv8-a -mcpu=thunderxt88 -fopenmp -fomit-frame-pointer
flang	-Ofast -march=armv8-a -mcpu=thunderxt88 -fopenmp -fomit-frame-pointer

The *-mcmodel=large* option was used by all compilers to compile the FT application. This application uses more than 2GB of memory per thread/process and requires the large memory model.

3.1.1.3. Cavium / ThunderX2

Table 5. Suggested compiler flags for Cray compilers

Compiler	Suggested flags
cc (C compiler wrapper)	default options
CC (C++ compiler wrapper)	default options
ftn (Fortran compiler wrapper)	default options

Table 6. Suggested compiler flags for GNU compilers

Compiler	Suggested flags
gcc	-Ofast -march=armv8.1-a -mcpu=thunderx2t99 -mtune=thunderx2t99 -fopenmp -funroll-loops
g++	-Ofast -march=armv8.1-a -mcpu=thunderx2t99 -mtune=thunderx2t99 -fopenmp -funroll-loops

Compiler	Suggested flags
gfortran	-Ofast -march=armv8.1-a -mcpu=thunderx2t99 -mtune=thunderx2t99 -fopenmp -funroll-loops

Table 7. Suggested compiler flags for ARM HPC compilers

Compiler	Suggested flags
armclang	-Ofast -march=armv8.1-a -mcpu=native -ffast-math -fopenmp
armclang++	-Ofast -march=armv8.1-a -mcpu=native -ffast-math -fopenmp
armflang	-Ofast -march=armv8.1-a -mcpu=native -ffast-math -fopenmp

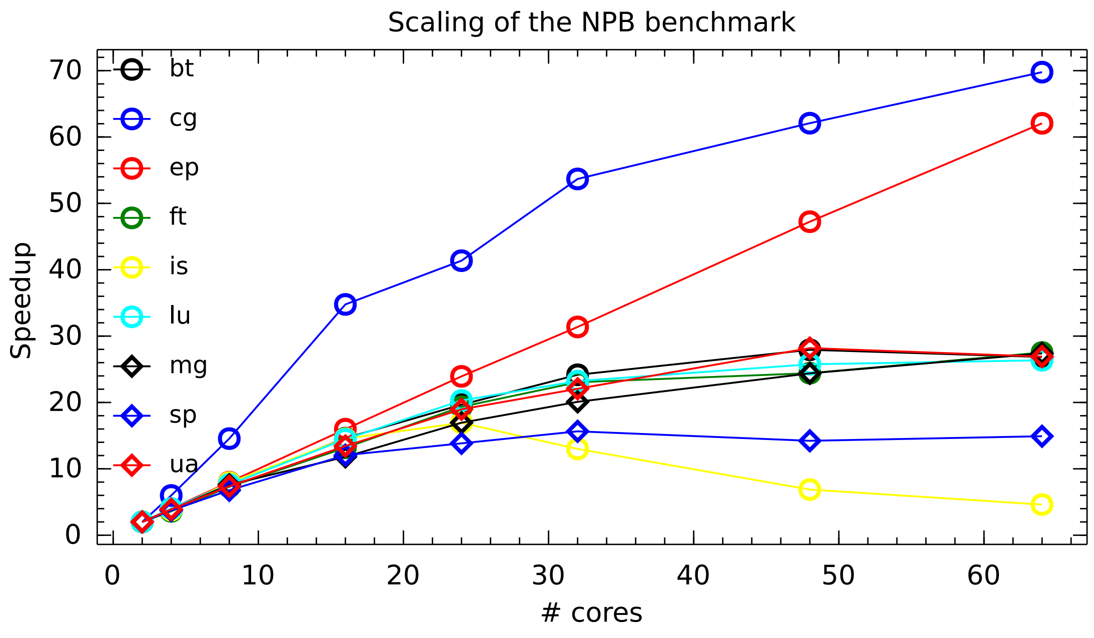
3.1.2. Compiler Performance

The well known set of benchmarks found in the NPB [38] suite is used for several examples in this guide. The different compilers show varying performance with the different NPB benchmarks. The figure below show the performance recorded using the OpenMP version of the NPB benchmarks. OpenMP version is choosen over MPI as the OpenMP thread library is an integral part of the compiler and should be evaluated together with the code generation. From the figure it's evident that they all do a reasonably good job.

The High Performance Conjugated Gradients (HPCG) benchmark [39] is gaining more and more interest as the Linpack benchmark used to assess the 500 fastest systems in the world has some shortcomings [40]. HPCG generally yields a very low processor efficiency due to the fact that this benchmark is highly memory bound.

3.1.2.1. Huawei / High Silicon

Figure 7. Compiler scaling performance using the ARM compiler



The figure above shows that not all of the different benchmarks in the NPB suite scale equally well. Some expose limited suitability for running parallel using a thread model like OpenMP. It's also a performance measure of the OpenMP library used. The GNU and ARM compiler use different OpenMP libraries. GNU uses the default supplied with the Linux distribution while ARM compiler supplies it's own library. The figure below illustrates the difference in scaling when using a different threading model with different OpenMP libraries.

Figure 8. Compiler scaling performance using the GNU compilers

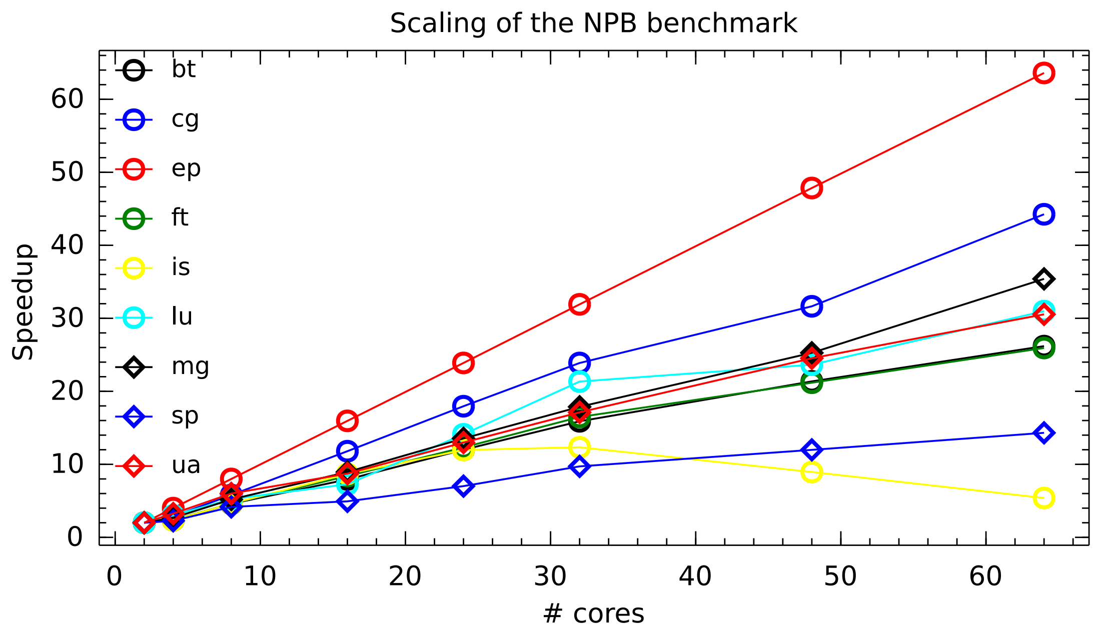
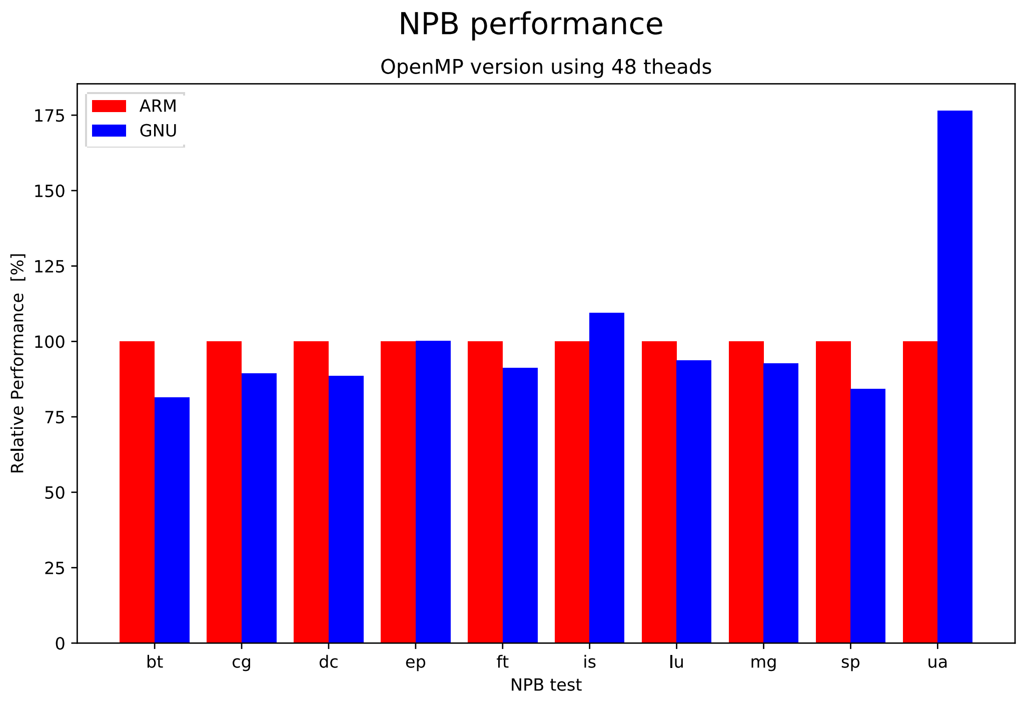
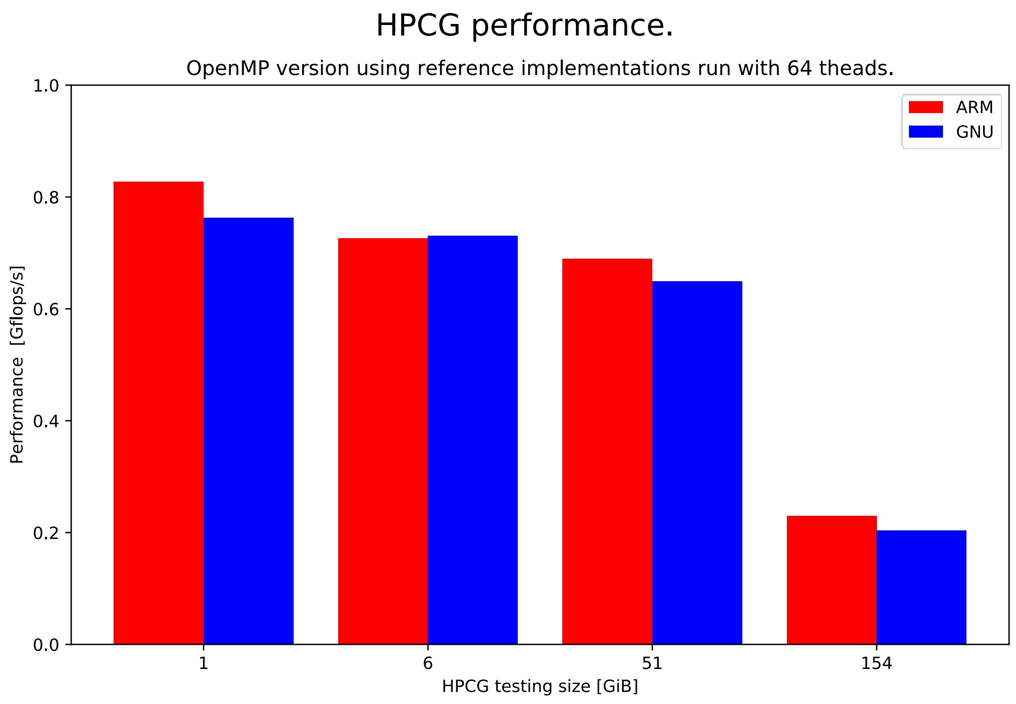


Figure 9. Compiler performance comparison



There are some differences in performance between the ARM compiler and the GNU compilers. The GNU compilers seem to have an edge with integer sort (IS, a bucket sort, in C) and unstructured grids (unstructured memory access, in Fortran) like the UA benchmark.

Figure 10. Compiler performance comparison



The HPCG benchmark is compiled using the reference version of the linear algebra and other functions that normally are called from an optimized library. This is by choice as the test should illustrate the different compiler's ability to generate efficient code. It's clear that all the C++ compilers tested generate code that performs this task reasonably well.

It's interesting to note that performance drops as soon as the benchmark footprint spans more than one NUMA memory node. The total memory of 256 GiB is made up of four NUMA nodes of 64 GiB each. This performance drop is a consequence of the difference of memory accesses in a None Uniform Memory Access (NUMA) system. It might well be that this problem should have been attacked using a hybrid model with one MPI rank per NUMA node and 16 threads per MPI rank keeping the fine granular memory accesses within the local NUMA node. It's beyond the scope of this guide to go into details of hybrid programming.

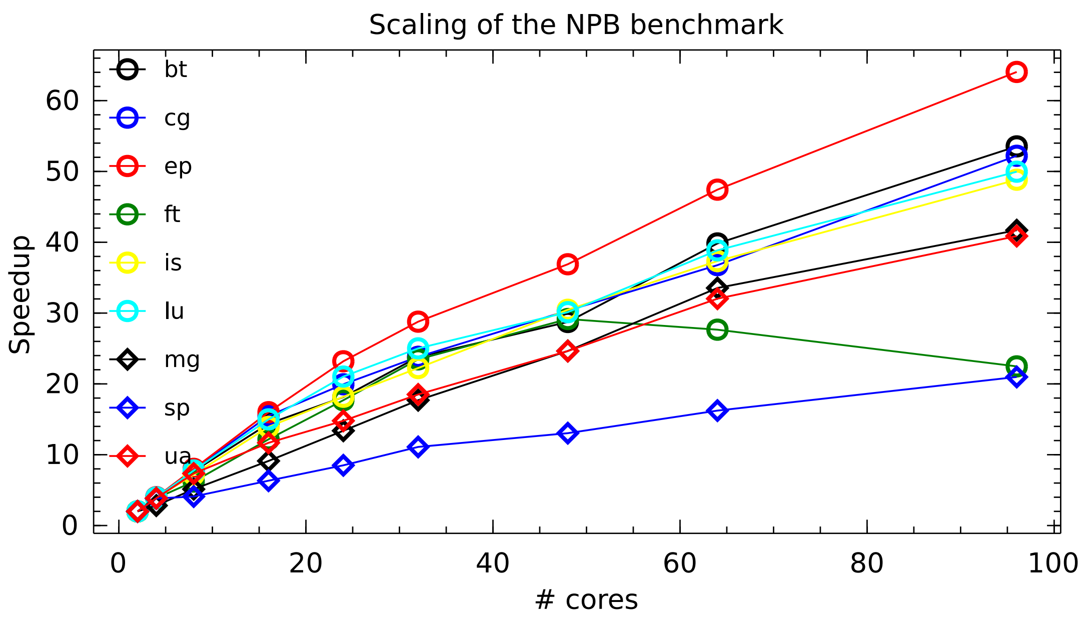
As always with HPCG the absolute performance compared to the theoretical peak performance is very low. This is part of the reason that this benchmark now shows growing interest and is used as an alternative benchmark as top500 HPCG along with the top500 HPL benchmark.

3.1.2.2. Cavium / ThunderX

The compiler scaling performance of Cavium ThunderX using the GNU compiler is shown in Figure 11 [2].

[2] For each compiler, a significant dissipation in the runtime of the benchmarks was observed during these experiments. Values with the greatest deviation from the mean were removed, the plots were built without taking them into account. In addition, the processor has thermal problems: overheating during computation on all cores.

Figure 11. Compiler scaling performance of Cavium ThunderX using the GNU compiler



The compiler scaling performance of Cavium ThunderX for the LLVM compiler is shown in Figure 12.

Figure 12. Compiler scaling performance of Cavium ThunderX using the LLVM compiler

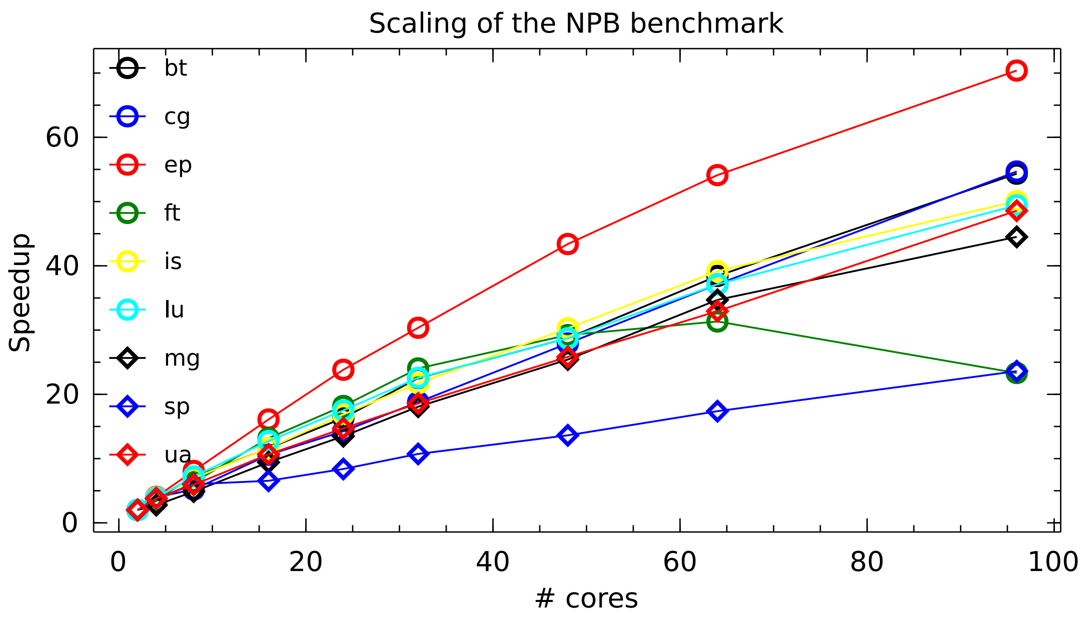
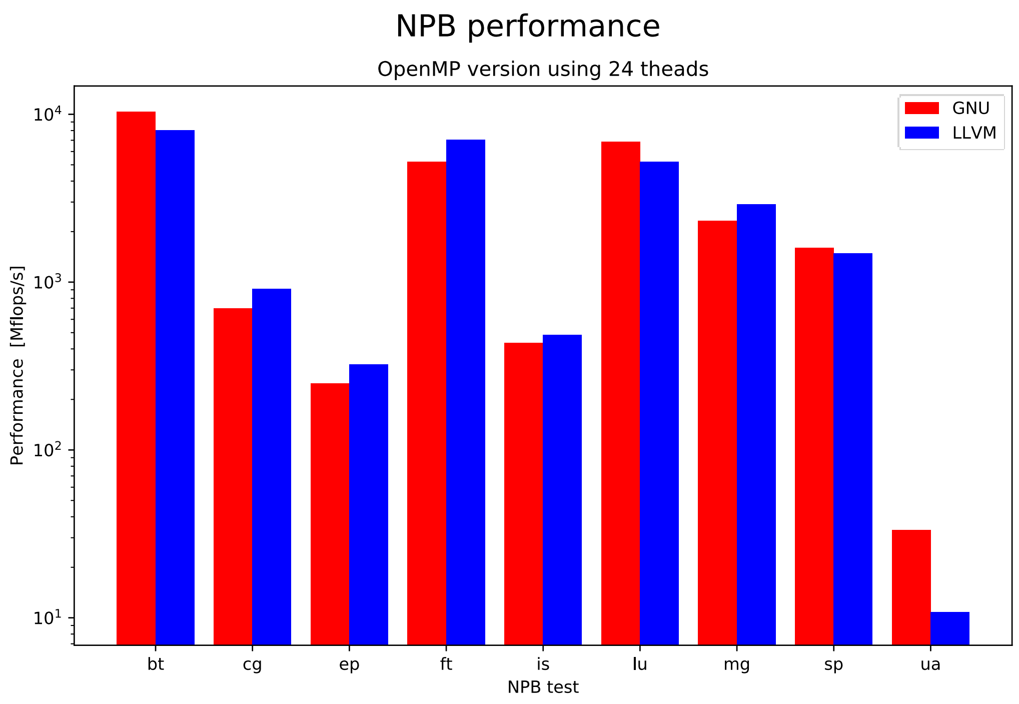


Figure 13. Compiler performance comparison



There are some differences in performance between the GNU and LLVM compilers.

3.2. Available (Vendor Optimised) Numerical Libraries

The ARM performance libraries [https://developer.arm.com/products/software-development-tools/hpc/arm-performance-libraries] are a set of libraries containing BLAS, LAPACK and FFT (icluding fftw wrappers)in both serial and multithreaded (mp) versions for both 32 and 64 bits (int64 / ilp64) integers.

Name of ARM performance libraries

- libarmpl.so
- libarmpl_mp.so
- libarmpl_int64.so
- libarmpl_int64_mp.so

Only dynamically linked versions or ARM Performance libraries are available.

Documentation is available in different formats at the ARM.com web site. [42] The BLAS and LAPACK syntax should be compatible with the usual Netlib syntax. There is also an interface for FFTW so applications written to use FFTW can be compiled without changes.

Another performance library often used is the OpenBLAS library.[41] While not vendor optimised it is often used in HPC. It builds on the framework of the well known Goto and Goto2 libraries (see the OpenBLAS wiki page). OpenBLAS is ported to the ARM architecture and works well with the top 500 HPL benchmark.

3.2.1. Performance evalution

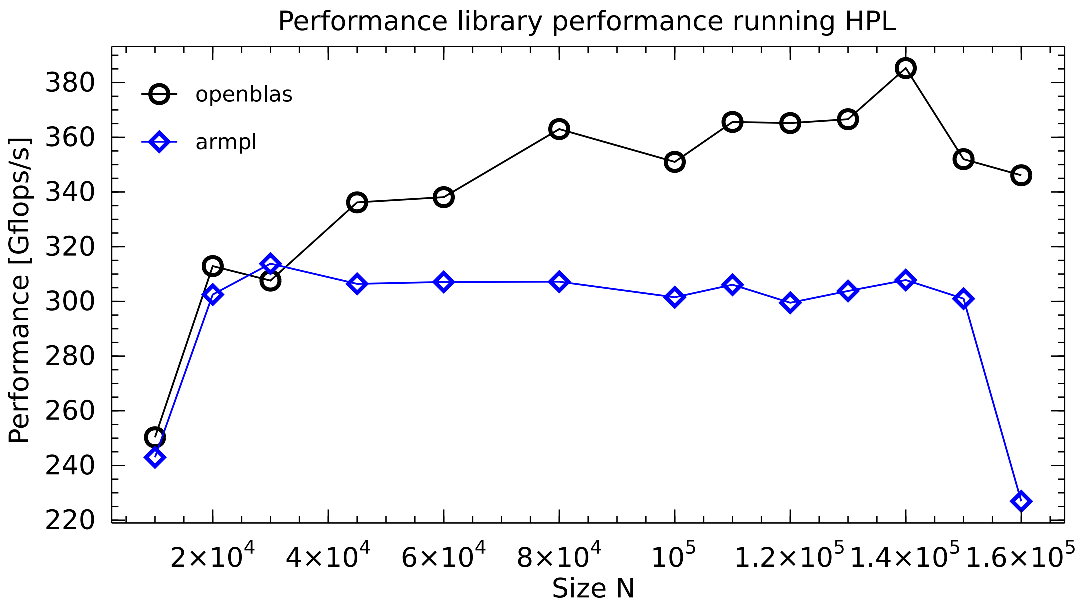
For this guide only a superficial study of the performance of the libraries is performed. A quick test of the most common functions both single threaded and multi threaded. The goal is to show that libraries exist for the most common functions needed for scientific programs.

3.2.1.1. HPL (top500) test

The well known top 500 test HPL is a good candidate to evaluate the two performance libraries head to head. While only using a few routines from the library it will expose differences. The HPL test is an MPI program and hence only single threaded versions of the BLAS library are tested. Hybrid models could also be tested, but they are beyond the scope of this guide. In the HPL test the OpenBLAS seems to perform quite well.

As with the compilers the threading model can vary from library to library, this is addressed in the next test below.

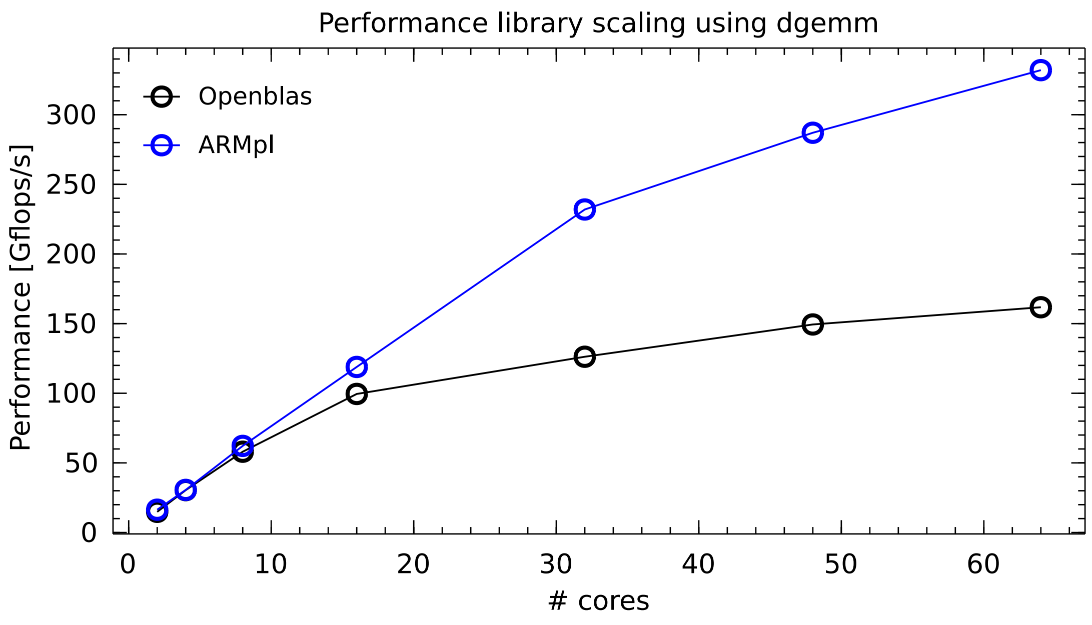
Figure 14. Performance Libraries running HPL using 64 cores



3.2.1.2. DGEMM performance

Multithreaded libraries normally have their own threading model, they normally have slightly different algorithms for the different routines. The thread performance of the libraries can be tested using the commonly used matrix matrix multiplication (dgemm) test. While this is only a single routine it is nevertheless one of the best known and is commonly selected for performance evaluation.

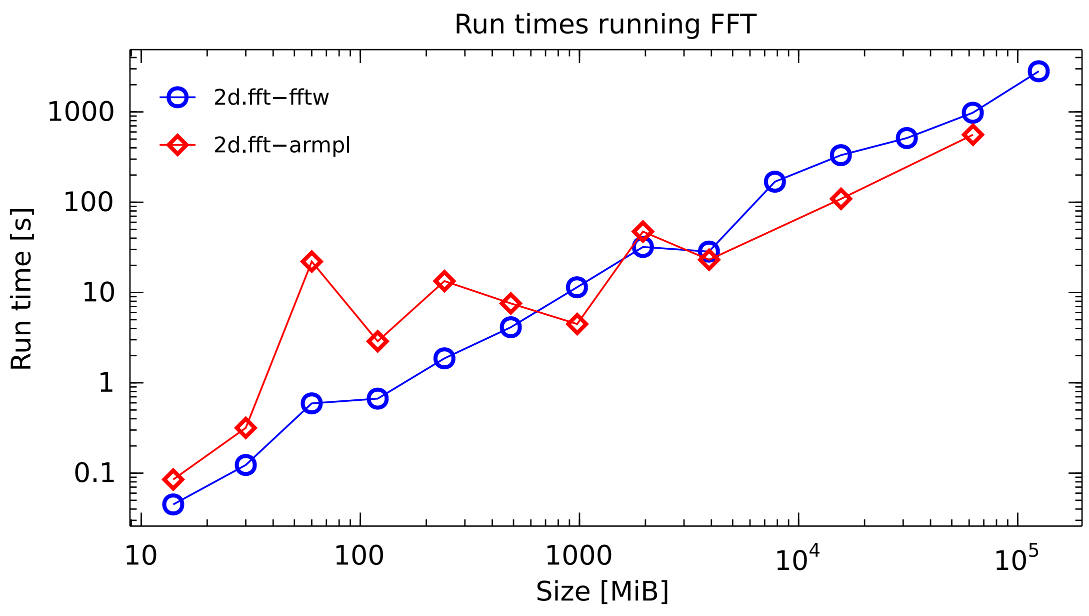
Figure 15. Performance Libraries testing dgemm



3.2.1.3. FFT performance

Fast Fourier Transforms are widely used and also implemented in the ARM performance library. The scaling in the FFTs are often limited and with the current two dimensional test the scaling was found to be limited with both FFTW and ARMpl. Hence the performance evaluation below is run using a single thread on only one core. The figure below shows run times (where lower is better) for the commonly used FFTW library and the ARM performance library.

Figure 16. Performance Libraries testing fft (single threaded, run times means that lower is better)



The performance picture is somewhat mixed, it's not easy to make good FFT implementations. The numbers are not stable using the ARM performance library (reason for this is unknown). The FFTW library on the other hand has been around for a long time and is more mature. The ARM performance library seems to be a bit less stable. Clearly more focus should have been put on the FFT implementations. It must be noted that only two dimensional FFT functions have been tested.

3.3. Available MPI Implementations

3.3.1. OpenMPI

The open source OpenMPI works nicely with the ARM architecture. Both the configure and the make process run smoothly and the resulting binaries and libraries work well.

3.3.2. Cray MPI

Cray MPI is provided and supported on the ARM-based Cray XC50.

3.4. OpenMP

OpenMP is implemented in a range of compilers, it's normally a part of the compiler. Some compilers come with an embedded OpenMP library while some rely on the GNU gomp library. Most of the OpenMP libraries use the POSIX thread library that comes with the Linux installation, libpthread. Therefore it is expected that performance differences show up when evaluating multithreaded applications with different compilers.

3.4.1. Compiler Flags

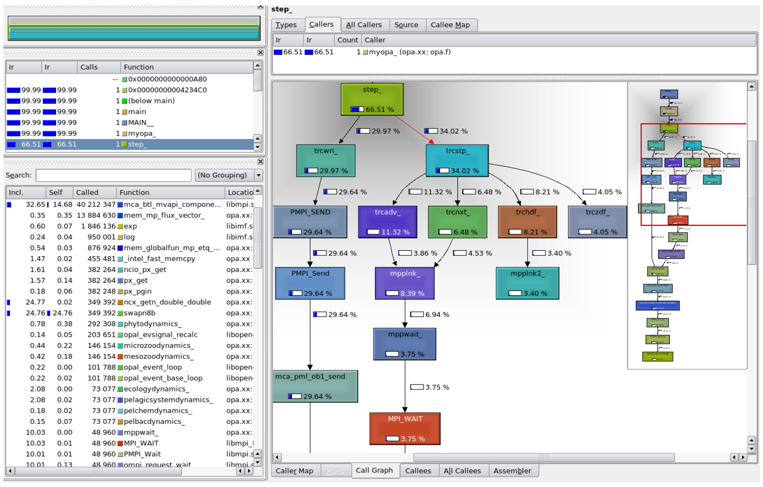
The most common flag to invoke parsing of the OpenMP pragmas in the source code is *-fopenmp*. GNU, clang/flang and the ARM compiler all honor this flag.

4. Performance Analysis

4.1. Available Performance Analysis Tools

Performance is a crucial characteristic of most applications, regardless whether they are serial or parallel. Performance analysis is usually a challenging process, as it requires a lot of efforts and skills from the analyst. Fortunately, there are a number of tools and frameworks that facilitate the application code analysis and optimization. There are two major techniques that are applied by those tools: *profiling* and *tracing*. Profiling is a general term for an inventory of performance of application events (which might be methods, functions, classes, etc.) and timings for their execution, based on the hardware counters information obtained as a result of sampling (program interruption at every given time interval). A profile typically contains aggregated (minimum, maximum, average, maximal deviation etc.) statistics for all application events. The profiling information is pretty much static – it does not contain any details about the timing of the events included into the profile, but only their consolidated values. One of the typical results of profiling is a call-graph (Figure 17), which shows the inclusive costs (absolute time or a percentage of the total execution) for the most time-consuming events of the application, but not gives details on when (at which time) these parts were actually running. However, exactly this latter information is strictly required in order to trace back the in-depth statistics of every particular call to the event during the application execution, for example when analyzing the bottlenecks of an MPI communication. This task can be accomplished by tracing – a technique that allows logging the timing of all application events (i.e. tracking the start and end timestamps). A trace (Figure 18) allows to retrieve the exact duration of each event and align it with the other events, thus enabling the identification of sources of performance degradation.

Figure 17. Example of a profile

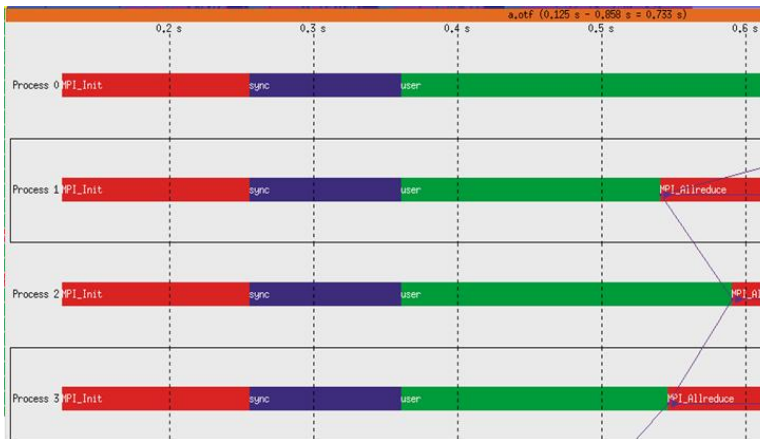


Ir	Ir	Count	Caller
66.51	66.51	1	myopa_ (opa.xx: opa.f)

Ir	Ir	Calls	Function
		--	0x0000000000000A80
99.99	99.99	1	0x00000000004234C0
99.99	99.99	1	(below main)
99.99	99.99	1	main
99.99	99.99	1	MAIN__
99.99	99.99	1	myopa_
66.51	66.51	1	step_

Incl.	Self	Called	Function	Locatio
32.65	14.68	40 212 347	mca_btl_mvapi_compone...	libmpi.s
0.35	0.35	13 884 630	mem_mp_flux_vector_	opa.xx:
0.60	0.07	1 846 136	exp	libimf.s
0.24	0.04	950 001	log	libimf.s
0.54	0.03	876 924	mem_globalfun_mp_etq_...	opa.xx:
1.47	0.02	455 481	_intel_fast_memcpy	opa.xx
1.61	0.04	382 264	ncio_px_get	opa.xx:
1.57	0.14	382 264	px_get	opa.xx:
0.18	0.06	382 248	px_pgin	opa.xx:
24.77	0.02	349 392	ncx_getn_double_double	opa.xx:
24.76	24.76	349 392	swapn8b	opa.xx:
0.78	0.38	292 308	phytodynamics_	opa.xx:
0.14	0.05	203 651	opal_evsignal_recalc	libopen-
0.44	0.22	146 154	microzoodynamics_	opa.xx:
0.42	0.18	146 154	mesozoodynamics_	opa.xx:
0.22	0.00	101 788	opal_event_loop	libopen-
0.22	0.02	101 788	opal_event_base_loop	libopen-
2.08	0.00	73 077	ecologydynamics_	opa.xx:
2.08	0.02	73 077	pelagicsystemdynamics_	opa.xx:
0.18	0.02	73 077	pelchemdynamics_	opa.xx:
0.15	0.07	73 077	pelbacdynamics_	opa.xx:
10.03	0.00	48 960	mppwait_	opa.xx:
10.03	0.01	48 960	MPI_WAIT	libmpi_
10.01	0.01	48 960	PMPI_Wait	libmpi.s
10.01	0.13	48 960	ompi_request_wait	libmpi.s

Figure 18. Example of a profile



The eco-system of performance analysis tools spans over a wide range of frameworks for profiling, tracing and (most often) combination of both techniques. Below an outlook of the most wide-spread tools for the standard CPUs as well as tools for ARM SoCs is given, which can be used on the high-performance ARM systems.

perf

Perf is a profiler tool for Linux 2.6+ based systems, which also allow to create a trace [27].

ftrace

Ftrace is the open-source tracing framework for the Linux kernel [18].

LTTng

LTTng is an open source tracing framework for Linux [19].

Extrae

Extrae [23] is the instrumentation package that captures information during the program execution and generates Paraver and Dimemas traces. The information collected by Extrae includes entry and exit to the programming model runtime, hardware counters, call stack reference, user functions, periodic samples and user events [22]. Extrae is distributed under the LGPL license.

Score-P

The Score-P (Scalable Performance Measurement Infrastructure for Parallel Codes) [26] measurement infrastructure is a highly scalable and easy-to-use tool suite for profiling, event tracing, and online analysis of HPC applications. Score-P offers the user a maximum of convenience by supporting a number of analysis tools. Currently, it works with Periscope, Scalasca, Vampir, and Tau and is open for other tools. Score-P comes together with the new Open Trace Format Version 2, the Cube4 profiling format and the Opari2 instrumenter. Score-P is available under the New BSD Open Source license [24].

ARM DS-5 Streamline performance analyzer

The commercial tools such as the ARM DS-5 Streamline performance analyzer [20] combine the functionality of a sample-based profiler with kernel trace data and processor performance counters, providing high-level visibility of how applications make use of the kernel and system-level resources [21].

Callgrind

Callgrind is an open-source profiler using execution-driven cache simulation via dynamic runtime instrumentation (provided by the open-source project Valgrind). This way, it can directly analyze cache behavior of program binaries for x86, x86-64, PPC32/64 and ARM. It comes with the visualization GUI KCachegrind, which provides various views of performance data such as annotated call graphs and tree maps for call nesting, as well as annotation of source and machine code [25].

Cray perftools

Cray Performance Analysis Tool (CrayPAT) is the full-featured performance analysis tool set provided on Cray systems, including ARM-based Cray XC50. CrayPAT-lite is a simplified, easy-to-use version of CrayPAT which provides basic performance analysis information automatically with a minimum of user interaction, i.e. less overhead. Using Cray-PAT-lite can be a good starting point for the users to explore the code behaviour and decide whether to use the full-featured CrayPAT for further performance profiling and analysis. [29] The latest version Cray Performance Measurement and Analysis Tools User Guide can be found here: [30].

Arm Forge

Arm Forge combines Arm DDT (the graphical debugger) and Arm MAP (the parallel profiler). Arm Forge is a cross platform tool and supports multiple parallel architectures and models, including MPI, UPC, CUDA and

OpenMP. The Arm Forge User Guide can be found here: [31]. Further info on Arm DDT and Arm MAP are introduced below.

Arm MAP

Arm MAP is a parallel profiler that supports MPI, OpenMP and single-threaded programs. Arm MAP supports both interactive and batch modes for gathering profile data. Further info on Arm MAP can be found here here: [32].. The Arm MAP User Guide is provided here: [33].

4.2. Access to Hardware Counters

Most of the performance tools (e.g. Extrae, Score-P) use performance counters in order to gather information regarding the application performance. The access to these counters is critical for performance analysis of the parallel applications. The hardware counters are available through standard libraries, e.g. Linux perf and PAPI (Performance Application Programming Interface). PAPI provides the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessors [17].

However, hardware counters are not always available on SoC by default. In the framework of the Mont-Blanc project, the PAPI support for advanced performance analysis on Cavium ThunderX CN8890 SoC was provided [47]. In order to gain access the following steps were performed:

- enabling the support for the ThunderX PMU in the Linux Kernel;
- extend PAPI event definition in order to support the ThunderX SoC and its hardware counters

The ARMv8 architecture foresees that performance counters can be accessed via a PMU (Processor Unit Component). The PMU component is handled as component 0 in PAPI - which is the default CPU component [46]. However, the PMU hardware module is optional on ARMv8 architecture definition. The availability of the PMU also depends on a kernel version of an operating system.

Moreover, extending PAPI to support Cavium ThunderX was provided by the authors [47]. This made it possible to install Extrae and Paraver and with their help the performance analysis for the LB (Lattice Boltzmann) application was performed.

4.3. Usage Examples of Performance Analysis Tools

perf

The perf package was installed on Huawei HiSilicon. Below are the commands for working with this tool.

```
# To install the perf
sudo yum install perf

# To show the list of all supported hardware and software events
perf list

# To collect basic statistics for the IS benchmark
perf stat -e task-clock,cycles,instructions,branch-misses ./bin/is.C.x
...
      Performance counter stats for './bin/is.C.x':

       44296,251400      task-clock:u (msec)    #    3,852 CPUs utilized
       105.470.817.037   cycles:u               #    2,381 GHz
       61.900.885.161    instructions:u         #    0,59  insn per cycle
       53.159            branch-misses:u
```

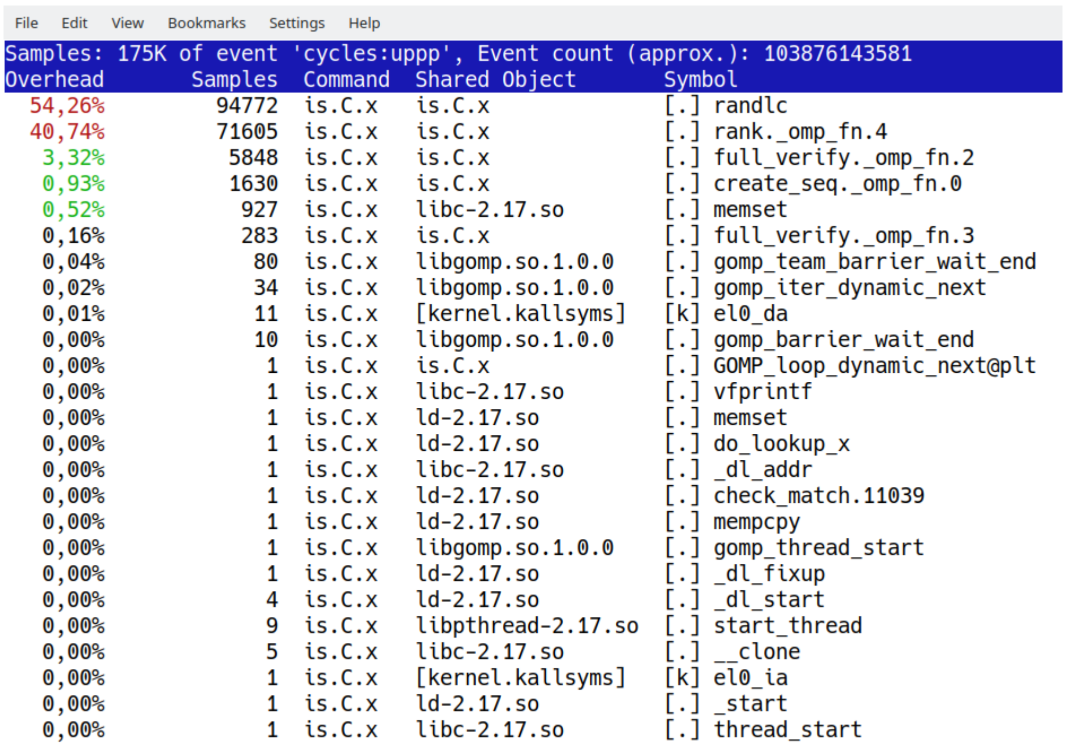
```
        11,498640056 seconds time elapse

 # To collect a profile data
 perf record ./bin/is.C.x

 # To display the perf.data from the current directory
 perf report -n
```

The perf profile for the IS benchmark is represented in Figure 19.

Figure 19. The perf profile for the IS benchmark

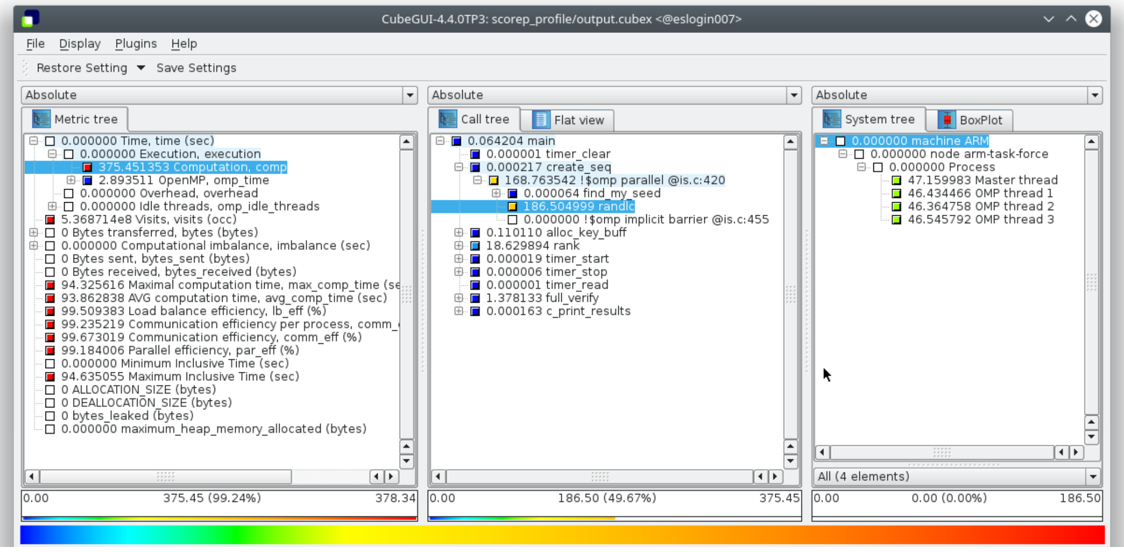


```
Samples: 175K of event 'cycles:uppp', Event count (approx.): 103876143581
Overhead         Samples  Command   Shared Object         Symbol
  54,26%           94772  is.C.x    is.C.x                [.] randlc
  40,74%           71605  is.C.x    is.C.x                [.] rank._omp_fn.4
   3,32%            5848  is.C.x    is.C.x                [.] full_verify._omp_fn.2
   0,93%            1630  is.C.x    is.C.x                [.] create_seq._omp_fn.0
   0,52%             927  is.C.x    libc-2.17.so          [.] memset
   0,16%             283  is.C.x    is.C.x                [.] full_verify._omp_fn.3
   0,04%              80  is.C.x    libgomp.so.1.0.0      [.] gomp_team_barrier_wait_end
   0,02%              34  is.C.x    libgomp.so.1.0.0      [.] gomp_iter_dynamic_next
   0,01%              11  is.C.x    [kernel.kallsyms]     [k] el0_da
   0,00%              10  is.C.x    libgomp.so.1.0.0      [.] gomp_barrier_wait_end
   0,00%               1  is.C.x    is.C.x                [.] GOMP_loop_dynamic_next@plt
   0,00%               1  is.C.x    libc-2.17.so          [.] vfprintf
   0,00%               1  is.C.x    ld-2.17.so            [.] memset
   0,00%               1  is.C.x    ld-2.17.so            [.] do_lookup_x
   0,00%               1  is.C.x    libc-2.17.so          [.] _dl_addr
   0,00%               1  is.C.x    ld-2.17.so            [.] check_match.11039
   0,00%               1  is.C.x    ld-2.17.so            [.] mempcpy
   0,00%               1  is.C.x    libgomp.so.1.0.0      [.] gomp_thread_start
   0,00%               1  is.C.x    ld-2.17.so            [.] _dl_fixup
   0,00%               4  is.C.x    ld-2.17.so            [.] _dl_start
   0,00%               9  is.C.x    libpthread-2.17.so    [.] start_thread
   0,00%               5  is.C.x    libc-2.17.so          [.] __clone
   0,00%               1  is.C.x    [kernel.kallsyms]     [k] el0_ia
   0,00%               1  is.C.x    ld-2.17.so            [.] _start
   0,00%               1  is.C.x    libc-2.17.so          [.] thread_start
```

Score-P

The Score-P v4.1 measurement infrastructure was installed on Huawei HiSilicon. The IS benchmark from the NPB3.3.1-OMP suite was selected as an example application for profiling. A profile created using Score-P is given in Figure 20. The Cube is used to visualize the information from the profile. The Cube [28] is a generic tool for displaying a multi-dimensional performance space consisting of the dimensions (i) performance metric, (ii) call path, and (iii) system resource. Each dimension can be represented as a tree, where non-leaf nodes of the tree can be collapsed or expanded to achieve the desired level of granularity. In addition, Cube can display multi-dimensional Cartesian process topologies.

Figure 20. Visualization of the IS profile using the Cube



CubeGUI-4.4.0TP3: scorep_profile/output.cubex <@eslogin007>

```
0.000000 Time, time (sec)
  0.000000 Execution, execution
    375.451353 Computation, comp
    2.893511 OpenMP, omp_time
  0.000000 Overhead, overhead
  0.000000 Idle threads, omp_idle_threads
5.368714e8 Visits, visits (occ)
0 Bytes transferred, bytes (bytes)
0.000000 Computational imbalance, imbalance (sec)
0 Bytes sent, bytes_sent (bytes)
0 Bytes received, bytes_received (bytes)
94.325616 Maximal computation time, max_comp_time (se
93.862838 AVG computation time, avg_comp_time (sec)
99.509383 Load balance efficiency, lb_eff (%)
99.235219 Communication efficiency per process, comm_
99.673019 Communication efficiency, comm_eff (%)
99.184006 Parallel efficiency, par_eff (%)
0.000000 Minimum Inclusive Time (sec)
94.635055 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.000000 maximum_heap_memory_allocated (bytes)

0.00          375.45 (99.24%)          378.34
```

```
0.064204 main
  0.000001 timer_clear
  0.000217 create_seq
    168.763542 !$omp parallel @is.c:420
      0.000064 find_my_seed
      186.504999 randlc
      0.000000 !$omp implicit barrier @is.c:455
  0.110110 alloc_key_buff
  18.629894 rank
  0.000019 timer_start
  0.000006 timer_stop
  0.000001 timer_read
  1.378133 full_verify
  0.000163 c_print_results

0.00          186.50 (49.67%)          375.45
```

```
0.000000 machine ARM
  0.000000 node arm-task-force
    0.000000 Process
      47.159983 Master thread
      46.434466 OMP thread 1
      46.364758 OMP thread 2
      46.545792 OMP thread 3

0.00          0.00 (0.00%)          186.50
```

5. Debugging

5.1. Available Debuggers

5.1.1. Arm DDT

Arm DDT is a graphical debugger that can be used in many different development environments, such as MPI codes, OpenMP codes, GPU codes, and hybrid codes, etc. Further info on Arm DDT can be found here here: [34]. The Arm DDT User Guide is provided here: [35].

5.1.2. GDB4HPC

GDB4HPC, as a GDB-based parallel debugger included in Cray Programming Environment (Cray PE), can be used to debug Fortran, C/C++ applications. using MPI, SHMEM, PGAS and OpenMP. [29].

5.2. Compiler Flags

To compile the code to debug, the debug flag should be added to the compile command. For most compilers this flag is -g.

Please refer to the Arm DDT User Guide for further info on the compiler flags to use/avoid when debugging with Arm DDT: [35].

6. European ARM-based Systems

6.1. Isambard

GW4 Isambard, the first Arm-based Cray XC50 ’Scout’ system , was due to arrive in 2018 as part of the UK’s national Tier-2 HPC service. The system, funded by EPSRC, is delivered by the GW4 Alliance together with the UK’s Met Office, Cray, Arm and Cavium.[43]

6.1.1. System Architecture / configuration

The full Isambard system is a Cray XC50 system which contains more than 10,496 cores. Each of the 164 compute nodes has two 32-core Cavium ThunderX2 processors on it with the clock rate of 2.1GHZ and each processor has eight 2666 MHz DDR4 Channels. Isambard uses Cray Aries interconnect with a Dragonfly topology for the nodes connection. 480 terabytes of Lustre storage is to be provided by a Cray Sonexion 3000 storage cabinet. The Isambard User Documentation can be found here: [11]

6.1.2. System Access

Account requests for Isambard should be submitted via SAFE for EPSRC Tier2 system. Further information can be found here: [12].

Guidance on how to connect to Isambard can be found here:[13]

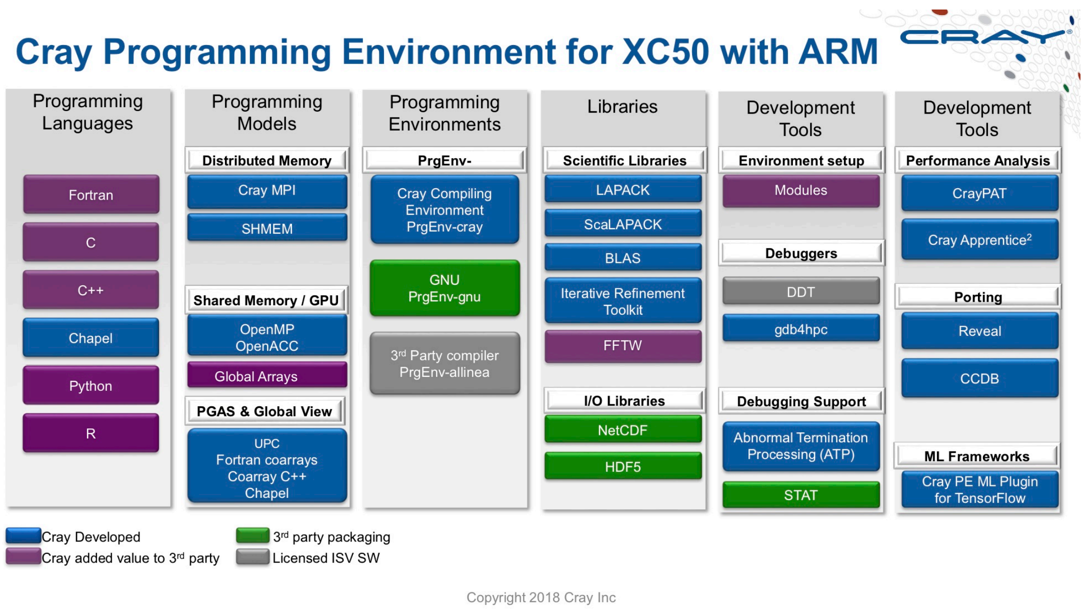
6.1.3. Production Environment

The production environment on the Isambard Phase 2 - XC50 ARM system, including the available modules, can be found here:[14] .

6.1.4. Programming Environment

A full Cray XC50 CPE (Cray Programming Environment) is to be provided on Isambard.

Figure 21. Cray Programing Environment for XC50 with ARM [45]



6.1.5. Further Useful Information

Further information about Isambard usage can be found in the Isambard User Documentation here: [11]

There are also a number of useful materials which are good source for better understanding / using Isambard:

- McIntosh-Smith, S., Price, J., Deakin, T., and Poenaru, A. Comparative Benchmarking of the First Generation of HPC-Optimised Arm Processors on Isambard. Cray User Group, CUG 2018, Stockholm, Sweden. [43]
- McIntosh-Smith, S., Price, J., Deakin, T., and Poenaru, A. A Performance Analysis of the First Generation of HPC-Optimised Arm Processors. In press, Concurrency and Computation: Practice and Experience, Feb 2019. [44]
- Isambard's bechmarks repository, which contains build instructions, run scripts and output files for all the benchmarks presented in the paper above:[15]
- The "isambard list" on the ARM HPC packages wiki:[16]

6.2. Mont-Blanc

Since October 2011, the aim of the European projects called Mont-Blanc has been to design a new type of computer architecture capable of setting future global HPC standards, built from energy efficient Arm solutions.

The first phase of the Mont-Blanc project established the following goals: to design and deploy a sufficiently large HPC prototype system based on the current mobile commodity technology; to port and optimize software stack and enable its use for HPC; to port and optimize a set of HPC applications to be run at this HPC system. The first phase of Mont-Blanc was coordinated by the Barcelona Supercomputing Center (BSC), and the first goal of the project was achieved with the deployment of the Mont-Blanc 1 Prototype on the BSC facilities. They investigated the usage of low-power Arm processors for HPC and gave rise to the world's first Arm-based HPC cluster, which helped demonstrate the viability of using Arm technology for HPC.

Phase 2 of the project was again coordinated by the BSC. Two years after the start of the initial project, the European Commission extended the Mont-Blanc project activities until September 2016. This three-year extension enabled further development of the OmpSs parallel programming model to automatically exploit multiple cluster nodes, transparent application check pointing for fault tolerance, support for ARMv8 64-bit processors, and the initial design of the Mont-Blanc Exascale architecture.

The third phase of the Mont-Blanc project started in October 2015: it is coordinated by Bull (Atos group), funded by the European Commission under the Horizon 2020 programme. The third phase adopts a co-design approach to ensure that hardware and system innovations are readily translated into benefits for HPC applications. It aims at designing a new high-end HPC platform that is able to deliver a new level of performance / energy ratio when executing real applications. As part of the Phase 3 of Mont-Blanc, Atos built a new prototype. It is named Dibona, after the Dibona peak in the French Alps, and it started operation in Fall 2017. It aims at designing a new high-end HPC platform that is able to deliver a new level of performance / energy ratio when executing real applications.

Following on from the three successive Mont-Blanc projects since 2011, the three core partners Arm, Barcelona Supercomputing Center and Bull (Atos Group) have united again to trigger the development of the next generation of industrial processor for Big Data and High Performance Computing. The Mont-Blanc 2020 consortium also includes CEA, Forschungszentrum Jülich, Kalray, and SemiDynamics. It intends to pave the way to the future low-power European processor for Exascale. To improve the economic sustainability of the processor generations that will result from the Mont-Blanc 2020 effort, the project includes the analysis of the requirements of other markets.

6.2.1. System Architecture / Configuration

The Mont-Blanc 1 Prototype has 63 blades and each of them contains 15 SDB (Samsung Daughter Board) nodes. The Mont-Blanc compute node is a Server-on-Module architecture. Each SDB is built around a Samsung Exynos 5250 mobile SoC integrating ARM Cortex-A15 CPUs at 1.7 GHz in a dual core configuration sharing 1 MB of on-die L2 cache, and a mobile ARM Mali-T604 GPU. The SoC connects to the on- board 4 GB of LPDDR3-1600 RAM through two memory channels shared among the CPUs and GPU, providing a peak memory bandwidth of 12.8 GB/s. The node's interconnect interface is provided through an on-die USB 3.0 interface utilizing a discrete on-board NIC (integrating USB 3.0 to 1 Gb Ethernet bridge), and the additional fabric providing Ethernet PHY. An external 16 GB µ SD card is connected to a µ SD memory slot and provides boot-loader, OS system image and local scratch storage. The blade hosts 15 Mont-Blanc nodes which are interconnected through an on-board 1 Gb Ethernet switch fabric. The switch provides two 10 GbE up-links. The operating system is Ubuntu 14.04, Kernel 3.11.0. Each node has 15GB at root partition, /home and /apps folders are mounted over the network (Lustre) from the file servers and are shared among all the nodes.

Figure 22. Rack Mont-Blanc 1 Prototype

During the Mont-Blanc 2 project, several ARM-based clusters have been deployed. We use the term mini-clusters for referring to the set of platforms deployed within the project, based on ARM SoCs from both the mobile and server markets. These platforms are usually deployed in the form of a small set of computational nodes made accessible as standard HPC clusters.

Table 8. Mini-clusters

Cluster	Number of CPUs in the cluster (CPUs/node)	Architecture	Feature of Interest
Jetson (Jetson TK1)	28 (4)	ARMv7-A	Ethernet via PCIe
Octodroid (ODROID-XU3)	112 (8)	ARMv7-A	big.LITTLE
Merlin (Applied Micro X-gene 2)	24 (8)	ARMv8	L3 cache
Thunder (ThunderX)	480 (96)	ARMv8	NUMA / CPU density
Jetson-TX (Nvidia Jetson TX1)	56 (4)	ARMv8	Cortex-A57 implementation

The Mont-Blanc Dibona prototype has 48 nodes, each node includes:

- 2 ThunderX2 CPU (32 cores per CPU, 64 per node, each core at 2GHz, 32MB L3 cache)
- 128 of main memory per node (16 DDR4 DIMM slots, 8 channels per CPU)
- 128 GB local storage e (+ 8TB nfs)

The Mont-Blanc Dibona prototype is equipped with a fat-tree with a pruning factor of ½ at L1 level interconnect topology with EDR 100Gb/s. The theoretical peak performance of the Dibona prototype is approximately 49 Tflops.

Figure 23. Rack Mont-Blanc 3 Prototype

6.2.2. System Access

The users must use Secure Shell (ssh) tools to login into the cluster or transfer files to it. On the Mont-Blanc 1 Prototype, the user login to one of the login nodes in a random way when connecting to the cluster (mb.bsc.es). On the Mont-Blanc Mini-Clusters, the user login to ssh.hca.bsc.es.

6.2.3. Production Environment

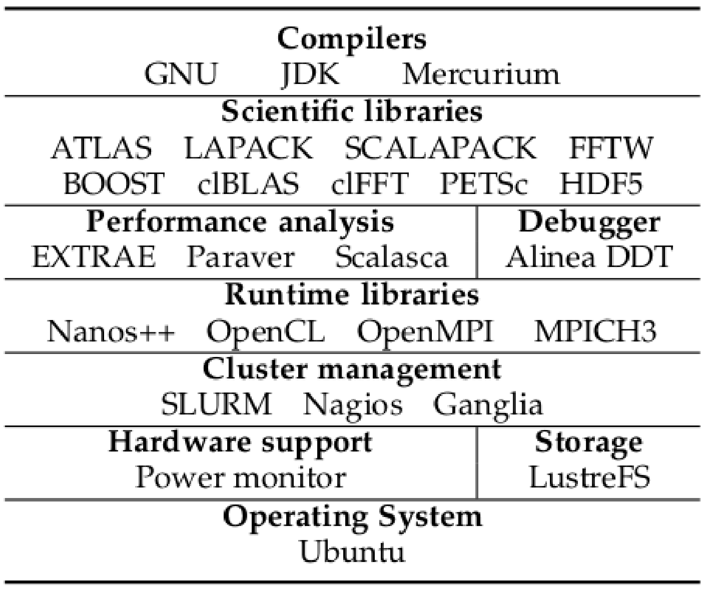
The Environment Modules package (http://modules.sourceforge.net/) provides a dynamic modification of a user's environment via modulefiles. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically, in a clean fashion.

The SLURM job scheduler is the tool selected by the Mont-Blanc consortium as the load balancing engine for the mini-clusters. SLURM is an open source and highly scalable job scheduling system for Linux clusters.

6.2.4. Programming Environment

On Mont-Blanc Prototypes and the mini-clusters there are different compilers and programming models available (GNU Compiler Suite, ARM Compiler, ARM HPC Compiler, ARM HPC Compiler, MPICH, OpenMPI, OmpSs), Scientific Libraries (ARM Performance Libraries, ATLAS, clBLAS, etc) and developer tools (Extrae, Paraver, AllineaDDT, PAPI).

Figure 24. Mont-Blanc Software Stack



Compilers	
GNU JDK Mercurium	
Scientific libraries	
ATLAS LAPACK SCALAPACK FFTW	
BOOST clBLAS clFFT PETSc HDF5	
Performance analysis	**Debugger**
EXTRAE Paraver Scalasca	Alinea DDT
Runtime libraries	
Nanos++ OpenCL OpenMPI MPICH3	
Cluster management	
SLURM Nagios Ganglia	
Hardware support	**Storage**
Power monitor	LustreFS
Operating System	
Ubuntu	

Further documentation

Books

[1] *Best Practice Guide - Intel Xeon Phi, January 2017, http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Intel-Xeon-Phi-1.pdf* .

Websites, forums, webinars

[2] *PRACE Webpage, http://www.prace-ri.eu/.*

[3] *Cavium ThunderX ARM processors, https://www.cavium.com/product-thunderx-arm-processors.html.*

[4] *Cavium ThunderX design for the dual-socket system, http://www.electronicdesign.com/microprocessors/64-bit-cortex-platform-take-x86-servers-cloud.*

[5] *ThunderX2® ARM Processors, https://www.cavium.com/product-thunderx2-arm-processors.html.*

[6] *ThunderX2 - Cavium, https://en.wikichip.org/wiki/cavium/thunderx2.*

[7] *ThunderX2® CN99XX Product Brief, https://www.cavium.com/pdfFiles/ThunderX2_PB_Rev2.pdf?x=2.*

[8] *Investigating Cavium's ThunderX: The First ARM Server SoC With Ambition, https://www.anandtech.com/show/10353/investigating-cavium-thunderx-48-arm-cores/2/.*

[9] *Puzovic, M., Quantitative and Comparative Analysis of Manycore HPC Nodes, http://www.oerc.ox.ac.uk/sites/default/files/uploads/ProjectFiles/CUDA/ Presentations/2017/2017-06-14-Milos-Puzovic.pdf.*

[10] *Cray XC50 ARM Product Brief, https://www.cray.com/sites/default/files/Cray-XC50-ARM-Product-Brief.pdf.*

[11] *Isambard User Documentation, https://gw4-isambard.github.io/docs/index.html.*

[12] *Isambard User Documentation - Request Account, https://gw4-isambard.github.io/docs/user-guide/requestaccount.html.*

[13] *Isambard User Documentation - Connecting to Isambard, https://gw4-isambard.github.io/docs/user-guide/connecting.html.*

[14] *Isambard User Documentation - Phase 2 XC50 ARM, https://gw4-isambard.github.io/docs/user-guide/phase2.html.*

[15] *Isambard Benchmarks Repository, https://github.com/UoB-HPC/benchmarks.*

[16] *Isambard-list on ARM HPC package wiki, https://gitlab.com/arm-hpc/packages/wikis/categories/isambard-list* .

[17] *The PAPI official site [https://icl.utk.edu/papi/index.html]* .

[18] *The ftrace [https://elinux.org/Ftrace]* .

[19] *The LTTng tracing framework [https://lttng.org]* .

[20] *The ARM DS-5 Streamline performance analyzer [http://ds.arm.com/ds-5/optimize/]* .

[21] *Orensanz, J. Performance Analysis on ARM Embedded Linux and Android Systems [https://community.arm.com/tools/b/blog/posts/ performance-analysis-on-arm-embedded-linux-and-android-systems---by-javier-orensanz]* .

[22] *The Extrae instrumentation package [https://www.vi-hps.org/Tools/Extrae.html]* .

[23] *The Extrae official site [https://tools.bsc.es/extrae] .*

[24] *The Score-P measurement infrastructure [https://www.vi-hps.org/projects/score-p/] .*

[25] *The Callgrind profiler [https://www.vi-hps.org/Tools/callgrind.html] .*

[26] *The Score-P official site [http://scorepci.pages.jsc.fz-juelich.de/scorep-pipelines/docs/scorep-4.1/html] .*

[27] *The perf tutorial [https://perf.wiki.kernel.org/index.php/Tutorial] .*

[28] *The Cube: performance report explorer [http://www.scalasca.org/software/cube-4.x/download.html] .*

[29] *The Cray Programming Environment https://www.cray.com/sites/default/files/SB-Cray-Programming-Environment.pdf [https://www.cray.com/sites/default/files/SB-Cray-Programming-Environment.pdf] .*

[30] *Cray Performance Measurement and Analysis Tools User Guide, https://pubs.cray.com/content/S-2376/7.0.0/cray-performance-measurement-and-analysis-tools-user-guide/ [https://pubs.cray.com/content/S-2376/7.0.0/cray-performance-measurement-and-analysis-tools-user-guide/] .*

[31] *Arm Forge User Guide, https://developer.arm.com/docs/101136/latest/arm-forge [https://developer.arm.com/docs/101136/latest/arm-forge] .*

[32] *Arm MAP webpage, https://developer.arm.com/products/software-development-tools/hpc/arm-forge/arm-map [https://developer.arm.com/products/software-development-tools/hpc/arm-forge/arm-map] .*

[33] *Arm MAP User Guide, https://developer.arm.com/docs/101136/latest/map [https://developer.arm.com/docs/101136/latest/map] .*

[34] *Arm DDT webpage, https://developer.arm.com/products/software-development-tools/hpc/arm-forge/arm-ddt [https://developer.arm.com/products/software-development-tools/hpc/arm-forge/arm-ddt] .*

[35] *Arm DDT User Guide, https://developer.arm.com/docs/101136/latest/ddt [https://developer.arm.com/docs/101136/latest/ddt] .*

Manuals, papers

[36] *PRACE Public Deliverable 7.6 Best Practice Guides for New and Emerging Architectures, http://www.prace-ri.eu/IMG/pdf/D7.6_4ip.pdf.*

[37] *Puzovic, M., Manne S., GalOn S., Ono M., Quantifying Energy Use in Dense Shared Memory HPC Node, 4th International Workshop on Energy Efficient Supercomputing, 2016, p. 16-23.*

[38] *NPB Benchmark. The High NPB benchmark [https://www.nas.nasa.gov/publications/npb.html].*

[39] *The High Performance Conjugate Gradients (HPCG) Benchmark project is an effort to create a new metric for ranking HPC systems. The High Performance Conjugate Gradients (HPCG) Benchmark [http://www.hpcg-benchmark.org].*

[40] *The High Performance Conjugate Gradients (HPCG) Benchmark top500. The High Performance Conjugate Gradients top500 list [https://www.top500.org/hpcg/].*

[41] *OpenBLAS library Wiki OpenBLAS library [https://github.com/xianyi/OpenBLAS/wiki].*

[42] *ARM performance libraries documentation [https://developer.arm.com/docs/101004/latest/1-introduction], in printable format for download [https://static.docs.arm.com/101004/1810/arm_performance_libraries_reference_manual_101004_1801_00_en.pdf] .*

[43] *McIntosh-Smith, S., Price, J., Deakin, T., and Poenaru, A. Comparative Benchmarking of the First Generation of HPC-Optimised Arm Processors on Isambard. Cray User Group, CUG 2018, Stockholm, Sweden. http://uob-hpc.github.io/assets/cug-2018.pdf.*

[44] *McIntosh-Smith, S., Price, J., Deakin, T., and Poenaru, A. A Performance Analysis of the First Generation of HPC-Optimised Arm Processors. In press, Concurrency and Computation: Practice and Experience, Feb 2019. DOI: 10.1002/cpe.5110.*

[45] *D Ernst, The Arm Technology Ecosystem: Current Products and Future Outlook, 2018. https://iadac.github.io/events/adac5/ernst.pdf.*

[46] *McCraw, H., Terpstra, D., Dongarra, J., Davis, K., Musselman, R. Beyond the CPU: Hardware Performance Counter Monitoring on Blue Gene/Q [http://www.netlib.org/utk/people/JackDongarra/PAPERS/hw-counter-bg.pdf] .*

[47] *Ruiz, D., Calore, E., Mantovani, F. Enabling PAPI support for advanced performance analysis on ThunderX SoC [http://montblanc-project.eu/wp-content/uploads/2017/12/2Enabling-PAPI-support-for-advanced.pdf] .*