Technical Report

2021-IETR021ESAIP013-V1.7

ESA 4000122242/17/NL/LF

Deliverable: CCN2/D1+D2
Version: V1.7
Status: draft
Date: 2021/05/03

Benchmark Specifications and Report

Martin Daněk
martin@daiteq.com

Contents

1 Preface

The NOEL-V processor is an implementation of the RISC-V processor instruction set architecture (ISA) designed and provided by Cobham Gaisler. At present the GRLIB distribution and documentation defines six configuration of the NOEL-V pipeline (see [GRI], [XCK] and the VHDL sources). The RISC-V ISA, together with the currently flourishing software development ecosystem, is seen as a strong advantage in the assessment of available processor architectures to be used in future ESA missions.

The RISC-V ISA definition is defined as a collection of basic (i.e. compulsory) and optional sets of instructions that are supported by individual RISC-V implementations provided by different vendors. At present the NOEL-V GRLIB distribution supports the I, M, A, F and D instruction sets, with additional ones to be implemented in future releases (namely the C and H set). This approach is compatible with application-specific specialization of the processor instruction set that has been developed in the preceeding LEON2FT activity (the configurable daiteq floating-point operations in daiFPU and integer operations in the SWAR unit).

This document defines the benchmark set that has been used to measure the NOEL-V performance, together with the NOEL-V configurations of potential interest that should be benchmarked. A starting point defined in previous discussions with the ESA defined two areas for benchmarking - floating-point performance, and performance scaling in multi-core configurations.

Subsequently, this document provides performance data measured using common benchmarks, namely

- *Paranoia*, *Whetstone*, *Linpack* and *Stanford*,
- *CoreMark*,
- *CoreMark-Pro*[1] .

The *CoreMark* benchmark ([RD4], [Cora]) used in this work has been derived from the version that is distributed as part of the *rtems-noel-1.0.4* toolchain. The benchmark has been extended to support multi-context execution, and computation both in the integer and floating-point domains.

The *CoreMark-Pro* benchmark suite ([RD5], [Corb]) has been extended to support execution in RTEMS platforms.

The NOEL-V performance is evaluated in the context of existing LEON-based systems.

1.1 How to read the report

This report is organized as follows.

- Sections 2 to 5 provide an overview of the benchmarks and tools used together with a definition and description of the evaluated NOEL-V and LEON configurations.
- Section 6 presents performance results for 4-core NOEL-V systems for each of the defined configurations.
- Section 7 presents performance results for LEON-based systems.
- Section 8 compares the performance and scaling of the individual benchmarks and workloads in NOEL-V and LEON-based systems.

[1] *FPMark* will be added in the next release of this report.

- Section 9 provides final conclusions of the performed experiments.

Each of Sections 6, 7 and 8 are organized in the same way. The discussion always starts with the results of the PWLS benchmarks, followed with the floating-point and integer versions of the *CoreMark* benchmark, and finally with the results of the *CoreMark-Pro* benchmark suite.

Sections 6 and 7 present benchmark performance normalized for 100MHz execution (i.e. the original results measured for GR740 were scaled down by a factor of 2.5), while Section 8 in addition presents relative benchmark performance related to the best achieved performance for each benchmark so as to highlight scalability properties of each system. Tables with the measured performance data are shown only in Sections 6 and 7.

Applicable documents

AD1 D03 - FPU Survey Report, V1.4

AD2 IEEE Std 754-2019 - IEEE Standard for Binary Floating-Point Arithmetic. June 13, 2019

AD3 The RISC-V Instruction Set Manual, Volume I: User-Level ISA. Document Version 2.2

AD4 The RISC-V Instruction Set Manual, Volume II: Privileged Architecture. Document Version 1.10

AD5 AT697 Evaluation Board User Manual

AD6 GR-CPCI-GR740 User's Manual

AD7 UG885 - VC707 Evaluation Board for the Virtex-7 FPGA User Guide (v1.8)

AD8 UG917 - KCU105 Board User Guide (v1.10)

AD9 RTEMS C User's Guide, Version 4.11.3

Reference documents

RD1 GRLIB IP Core User's Manual, Dec 2020, Version 2020.4

RD2 NOEL-XCKU User's Manual, Dec 2020, Version 3.0

RD3 GRLIB IP Library

RD4 CoreMark - An EEMBC Benchmark (www.eembc.org/coremark)

RD5 CoreMark-Pro - An EEMBC Benchmark (www.eembc.org/coremark-pro)

RD6 FPMark - An EEMBC Benchmark (www.eembc.org/fpmark)

RD7 Embench: A Modern Embedded Benchmark Suite (www.embench.org)

Abbreviations

```
daiFPU daiteq FPU
FPU    floating-point unit
GRFPU  Gaisler Research FPU
Meiko  Meiko FPU
PWLS   Paranoia, Whetstone, Linpack, Stanford
RISC   reduced instruction set computer
RTOS   real-time operating system
SPARC  scalable processor architecture
SWAR   SIMD-within-a-register
VHDL   VHSIC hardware description language
VHSIC  very high speed integrated circuits
```

2 Benchmark Selection

As a starting point for the definition of the benchmarking set, the following existing benchmarks have been discussed. The PWLS set has already been used in the LEON2FT activity that preceded the NOEL-V activity.

The *Paranoia* benchmark is usually used to assess the quality of the results provided by available floating-point implementations, both software and hardware,

The *Whetstone*, *Linpack* and *Stanford* benchmarks are the classical trio that has been used in the past to assess single-core floating-point performance [AD1].

The *CoreMark* benchmark, provided for free by the *EEMBC* consortium on *github* [CMR], has become a standard benchmark for measuring multi-core performance, not focussing on floating-point performance. The benchmark is also included in the *rtems-noel-1.0.4* toolchain package distributed by Cobham Gaisler.

The *CoreMark-Pro* benchmark has also been used in the benchmarking activity, due to its long-term existence, availability of reference data for other embedded and desktop systems, and acceptance in the industry. The source code of the benchmark suite has been made available by the *EEMBC* consortium on *github* [CMR], but its use must be in accordance with the *EEMBC* licensing conditions. Namely the results cannot be published without a license agreement signed mutually with the *EEMBC* consortium.

The *FPMark* benchmark will be used in the next release of this report to further extend the evaluation of floating-point performance.

The advantage of the *CoreMark-Pro* and *FPMark* benchmarks is that their implementation decouples the actual computing workloads from the task allocation and synchronization framework, denoted as *MITH*. The framework supports execution in POSIX threads. An identical *MITH* framework is used in both benchmarks, thus easing the porting task.

In addition, the use of the benchmarks defined by the new *EmBench* initiative [EmB] has been discussed. It was decided that the *EmBench* benchmarks will be used for additional performance measurements if there is time, i.e. after completing the *FPMark* measurements, to provide performance figures for the *EmBench* suite for the NOEL-V and LEON-based systems.

The RTEMS Real-Time Operating System (RTOS) [AD9] has been selected as the execution platform for the benchmarking activity due to its flight heritage and ability to execute and automatically schedule multi-context workloads on multiple processing cores in SMP systems [RTE], also supporting POSIX threads[1] .

[1] See also comments in Section 4.1.

3 Evaluated platforms and configurations

The hardware platforms used for the performance evaluation of NOEL-V and LEON-based systems are summarized in Table 3.1. The individual processor configurations used are listed in Table 3.2 for the NOEL-V systems and Table 3.3 for the LEON-based systems.

Table 3.1: Platforms used for performance evaluation.

Platform ID	Board	Processor	Frequency [MHz]	FPU	Max. cores
P1	AT697	AT697F	100	Meiko	1
P2	VC707	LEON2	100	daiFPU	1
P3	VC707	LEON3	100	GRFPU	4
P4	GR-CPCI-GR740	LEON4	250	GRFPU	4
P5	KCU105	NOEL-V	100	nanoFPU	4
P6	KCU105	LEON5	100	GRFPU5	4

Table 3.2: NOEL-V setups used for benchmarking.

ID	Altern.	Platform	BSP	CFG	#CORES
.	ID	ID	RTEMS	ID	TOTAL
NV01	CFG0-int	P5	noel64im	CFG0	1
NV02	CFG0-fp	P5	noel64imafd	CFG0	1
NV03	.	P5	noel64ima_smp	CFG0	4
NV04	CFG0	P5	noel64imafd_smp	CFG0	4
NV11	CFG1-int	P5	noel64im	CFG1	1
NV12	CFG1-fp	P5	noel64imafd	CFG1	1
NV13	.	P5	noel64ima_smp	CFG1	4
NV14	CFG1	P5	noel64imafd_smp	CFG1	4
NV21	CFG2-int	P5	noel64im	CFG2	1
NV22	CFG2-fp	P5	noel64imafd	CFG2	1
NV23	.	P5	noel64ima_smp	CFG2	4
NV24	CFG2	P5	noel64imafd_smp	CFG2	4
NV31	CFG3-int	P5	noel64im	CFG3	1
NV32	CFG3-fp	P5	noel64imafd	CFG3	1
NV33	.	P5	noel64ima_smp	CFG3	4
NV34	CFG3	P5	noel64imafd_smp	CFG3	4
NV41	CFG4-int	P5	noel64im	CFG4	1
NV43	CFG4	P5	noel64ima_smp	CFG4	4
NV51	CFG5-int	P5	noel64im	CFG5	1
NV61	.	P5	noel64im	CFG6	1
NV63	CFG6	P5	noel64ima_smp	CFG6	4

Table 3.3: LEON setups used for benchmarking[2] .

ID	Altern.	Platform	BSP	L2 Cache	#CORES	
.	ID	ID	RTEMS	.	USED	TOTAL
LE1	AT697F	P1	at697f	N	1	1
LE2	LEON2	P2	leon2	N	1	1
LE3	LEON3	P3	leon3	N	1	1
LE4	LEON3SMP	P3	leon3_smp	N	1	4
LE5	LEON3	P3	leon3_smp	N	4	4
LE6	GR740	P4	gr740	Y	1	4
LE7	GR740	P4	gr740_smp	Y	4	4
LE8	LEON5	P6	leon3	N	1	4
LE9	LEON5SMP	P6	leon3_smp	Y	1	4
LE10	LEON5L2	P6	leon3_smp	Y	4	4
LE11	LEON5	P6	leon3_smp	N	4	4
LE12	LEON52CL2	P6	leon3_smp	Y	2	2

[2] The identical *Alternative IDs* for LEON setups, used for non-SMP and SMP configurations, can be distinguished by the context of the benchmark used: for the PWLS benchmarks *LEON3*, *GR740* and *LEON5* refer to *LE3*, *LE6* and *LE8* respectively, while for the other benchmarks that can execute in multiple contexts they refer to *LE5*, *LE7* and *LE11*. The idea was to not decrease single-core performance by using the SMP BSPs for single-context benchmarks since the SMP BSP would also have to manage the unused processor cores and thus achieve a slightly lower performance.

4 Tool versions

The following tools and packages were used to conduct the evaluation:

Processor sources:

- grlib-com-nv-basic-2020.4-b4258 (equivalent to grlib-gpl-2020.4-b4261)
- leon2ft_2020.1_daifpu_swar

Development kits:

- See Table 3.1.

FPGA design tools:

- Xilinx ISE 14.7
- Xilinx Vivado 2020.1 (KCU105)
- Xilinx Vivado 2016.3 (VC707)

RTEMS compilers:

- rtems-noel-1.0.4-2020-12-02.tar.bz2
- sparc-rtems-5-gcc-10.2.0-1.3.0-linux.txz

Benchmarks:

- *Paranoia*, *Whetstone*, *Linpack*, *Stanford* - versions customized for LEON/RTEMS and NOEL-V/RTEMS based on C sources available on the web
- *CoreMark* (version derived from the one distributed with rtems-noel-1.0.4)
- *CoreMark-Pro* 1.1.2743

GRLIB/LEON configuration:

- For RTEMS execution the *LEON* processors have to suppport a 32-bit hardware time counter. If the DSU timer is present and mapped to *ASR22/23*, the parameter `tbits` has to be set to a value *32* when instantiating the *DSU* .

4.1 Notes

Several experiments have been conducted to establish a working NOEL-V software toolchain. Both the GRLIB versions 2020.2 and 2020.4 were tested, and the following observations made:

- GRLIB version 2020.2 is compatible only with rtems-noel-1.0.3
- GRLIB version 2020.4 is compatible only with rtems-noel-1.0.4 (changed memory layout)
- The *nanoFPU* (e.g. the *noel64imafd* BSP) is supported only in rtems-noel-1.0.4
- SMP (e.g. the *noel64imafd_smp* BSP) is supported only in rtems-noel-1.0.4

- The sample linux image built by Cobham Gaisler[3] web works fine with GRLIB version 2020.4
- The NOEL-V distribution *noel-buildroot-2020.08-1.0*[3] cannot be used to build a working linux binary image for NOEL-V. To be precise, the generated image fails during the boot sequence when it tries to mount the *rootfs* filesystem.

[3] available for download from the Cobham Gaisler web

5 Analysis of NOEL-V configurations

The NOEL-V RISC-V processor is distributed as part of the GRLIB IP core library. The GRLIB library specifies six pre-defined configurations for NOEL-V that should cover the computing range from high-performance systems down to tiny microcontrollers. The configurations are listed in Table 5.2. An additional seventh configuration was added to the pre-defined configurations that supports multi-core systems built using the *TINY* core.

The configurations differ mostly in the size of the branch prediction tables, caches, the number of issues, presence of an FPU, and support for SMP.

5.1 Dual-pipeline execution

The dual-pipeline configurations can execute up to two instructions in parallel. However, there are certain limitations to instruction execution as certain classes of instructions can execute only in one specific lane:

LANE #1

- `jal`, `jalr`, `branch`
- `csrxx` if `csr_lane` is configured to 1[4]
- FPU instructions (except load and store) if `fpu_lane` is configured to 1[5]

LANE #0

- *load* and *store* instructions (for both integer and floating-point registers)
- *atomic* instructions
- `fence`, `sfence.vma`
- `csrxx` if `csr_lane` is configured to 0[4]
- FPU instructions (except load and store) if `fpu_lane` is configured to 0[5]

5.2 Fixed configuration parameters

Certain NOEL-V pipeline parameters are defined in the VHDL code and cannot be changed through the `CFG_CFG` parameter. Table 5.1 provides an overview of the fixed parameters together with their values that were used to generate the NOEL-V configurations evaluated in this report.

Table 5.1: NOEL-V configuration parameters with fixed assignments in the VHDL files. List order as in *cpucorenv.vhd*.

Generic	Description	Range	Configuration
dmen	use RISC-V debug module	0,1	1

Continued on next page

[4] In the current NOEL-V `csr_lane` is set to 0 in all configurations.
[5] In the current NOEL-V `fpu_lane` is set to 0 in all configurations.

Table 5.1 – continued from previous page

Generic	Description	Range	Configuration
pbaddr	program buffer execute address(upper 20b)	.	16#90000#
cached	cacheability address mask (if not zero)	0,1	0
wbmask	writeback address mask	.	16#50FF#
busw	AHB bus width	32,64	32
cmemconf	configuration of cache tag memories	0-2	0
rfconf	regfile configuration:0-blkram,1-distram	0,1	0
clk2x	not used	.	.
ahbpipe	not used	.	.
irepl	not used	.	.
drepl	not used	.	.
dsnoop	not used	.	.
ilram	not used	.	.
ilramsize	not used	.	.
ilramstart	not used	.	.
dlram	not used	.	.
dlramsize	not used	.	.
dlramstart	not used	.	.
mmupgsz	not used	.	.
tlb_type	not used	.	.
tlb_rep	not used	.	.
tlbforepl	number of TLB entries	.	.
riscv_mmu	MMU ID (0=sparc, 1-3=riscv)	0-3	2
physaddr	physical addressing	32-56	32
rstaddr	reset address (upper 20 bits)	.	16#C0000#
disas	debug instruction execution	0-3	0
illegalTval0	zero tval on illegal instruction	0,1	0
no_muladd	multiply-add not supported in the FPU	0,1	0
mularch	multiplier architecture	0-3	0
hw_fpu	use HW FPU (if fpulen not zero)	0,1	1
rfreadhold	use read hold register for regfile	0,1	0
ft	not used	.	.
scantest	scantest enable	0,1	0
endian	little endian	1	1
nodbus	no debug bus	0,1	1

5.3 Configurations

An overview of user-selectable NOEL-V configurations is presented in Table 5.2. The first six configurations, *CFG0* to *CFG5* are defined in the file *noelvcpu.vhd*. The seventh configuration *CFG6* was created from *CFG5* by setting the parameter `ext_a` to *1* so as to enable evaluation of NOEL-V systems with multiple *TINY* cores.

The resources required to implement each configuration are shown in Table 5.3.

Table 5.2: NOEL-V configurations as defined in *GRLIB 2020.4*. *CFG_CFG* is defined in *config.vhd* inside the design directory. Values that change are shown in **bold**.

Name	HPP	GPP		MIN		TINY	
.	.	2ISSUES	1ISSUE	FPU	no FPU	no SMP	SMP
CFG_CFG	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Dual issue pipeline							
single_issue	0	**0**	**1**	1	1	1	1
ISA extensions							
ext_m	1	1	1	1	1	1	1
ext_a	1	1	1	1	**1**	**0**	**1**
ext_c	0	0	0	0	0	0	0
ext_h	0	0	0	0	0	0	0
Execution mode support							
mode_s	1	1	**1**	**0**	0	0	0
mode_u	1	1	1	1	**1**	**0**	0
FPU enable/data width							
fpulen[6]	64	64	64	**64**	**0**	0	0
Physical memory protection							
pmp_no_tor	0	0	0	0	0	0	0
pmp_entries	8	8	8	8	**8**	**0**	0
pmp_g	10	10	10	10	10	10	10
Performance counters							
perf_cnts	16	16	16	16	**16**	**0**	0
perf_evts	16	16	16	16	**16**	**0**	0
Trace buffer							
tbuf	4	4	4	4	4	1	1
trigger	16*1 + 2	**16*1 + 2**	**16*1 + 2**	**16*0 + 2**	16*0 + 2	16*0 + 0	16*0 + 0
Caches							
icen	1	1	1	1	**1**	**0**	0
iways	4	4	**4**	**2**	2	(1)	(1)
iwaysize	4	4	4	4	4	(1)	(1)
ilinesize	8	8	8	8	8	(8)	(8)
dcen	1	1	1	1	1	(0)	(0)
dways	4	4	**4**	**2**	2	(1)	(1)
dwaysize	4	4	4	4	4	(1)	(1)
dlinesize	8	8	8	8	8	(8)	(8)
MMU							
mmuen	1	1	**1**	**0**	0	0	0
itlbnum	8	8	8	(2)	(2)	(2)	(2)
dtlbnum	8	8	8	(2)	(2)	(2)	(2)
Pipeline config							
div_hiperf	1	1	**1**	**0**	0	0	0
div_small	0	0	0	0	**0**	**1**	1
late_branch	1	1	1	1	**1**	**0**	0
late_alu	1	1	1	1	**1**	**0**	0
Branch history							
bhtentries	128	128	**128**	**64**	**64**	**32**	32
bhtlength	5	5	5	5	**5**	**2**	2

Continued on next page

Table 5.2 – continued from previous page

Name	HPP	GPP		MIN		TINY	
.	.	2ISSUES	1ISSUE	FPU	no FPU	no SMP	SMP
CFG_CFG	**0**	**1**	**2**	**3**	**4**	**5**	**6**
predictor	2	2	2	2	2	2	2
Branch target prediction							
btbentries	**32**	**16**	16	16	**16**	**8**	8
btbsets	2	2	2	2	2	2	2

The effect of selected configuration constants in Table 5.2 is as follows:

- ext_a - enable support for atomic instruction in *iunv* and *cctrlnv*
- div_hiperf - shift two bits at once during division
- div_small - do not precompute the amount of bits to shift

5.4 Resources

Table 5.3 shows resources needed to implement each of the NOEL-V configurations shown in Table 5.2, and in addition two equivalent LEON2 configurations with 32KB instruction and 16KB data cache. The target frequency was set to 100MHz and achieved for both NOEL-V and LEON2 targets.

The table shows resources required to implement one processor core as well as resources needed to implement a complete processor system with 4 cores (1 core for *CFG5* and the LEON2 configurations).

The module *noelvsys* instantiates

- ahbctrl
- apbctrl (or apbctrldp when the debug bus is enabled by setting *nodbus* to *0*)
- **noelvcpu** - a number of processor cores as defined by `CFG_NCPU`
- rvdm - RISV-C debug module
- ahbtrace_mmb - AHB trace unit
- apbuart
- gptimer
- clint_ahb - RISC-V core local interrupt controller
- grplic_ahb - RISC-V platform interrupt controller

The module *noelvcpu* is a wrapper that defines the NOEL-V configurations listed in Table 5.2 and instantiates **cpucorenv**.

The module *cpucorenv* instantiates

- **iunv** - the actual RISC-V processor pipeline
- mul64 and div64 - integer multiplier and divider when `ext_m` is set to *1*
- cctrlnv - cache controller
- bhtnv - branch history table

[6] The valid `fpulen` configurations are *32* and *64*, other values result in the FPU buses not connected to the *nanoFPU*. In both configurations the same *nanoFPU* is instantiated, but the configuration *32* connects the higher 32 bits in the FPU buses to zero.

- btbnv - branch target buffer
- rasnv - return address stack
- regfile64sramnv (regfile64dffnv when `rfconf` is set to *1*) - the integer register file
- regfile64sramnv (regfile64dffnv when `rfconf` is set to *1*) - the floating-point register file when *fpulen* is greater than *0*
- cachememnv - cache memories
- tbufmemnv - instruction buffer when `tbuf` is not *0*
- nanofpunv - the *nanoFPU* floating-point unit when `fpulen` is greater than *0* and `hwfpu` is set to *1*[6]

In comparison, the module *mcore* in LEON2 instantiates

- ahbarb - AHB arbitter
- apbmst - AHB master
- **proc** - one processor core
- dsu - debug support unit
- dsu_mem - DSU trace memory
- mctrl - memory and peripheral controller
- ahbram - on-chip RAM
- ahbstat - AHB status register
- lconf, lconf_fpu - configuration registers
- 2x uart
- timers
- irqctrl - interrupt controller
- ioport - parallel I/O port

The module *proc* in LEON2 instantiates

- **iu** - the actual SPARC V8 pipeline
- regfile_iu - integer register file, also including the floating-point registers when FPU is enabled
- cache (or mmu_cache)
- cachemem
- fpucore - floating-point unit when FPU is enabled

Table 5.3: NOEL-V / LEON2 - resources used. The table lists implementation resources per each defined configuration, and it also lists resources for two common LEON2 configurations. The first part of the table gives resources for a complete processor subsystem that consists of 4 NOEL-V cores, with the exception of configuration 5 that consists of a single NOEL-V core. The second part of the table specifies resources for one processor core.

Name	HPP	GPP		MIN (1 ISSUE)		TINY (1 ISS,no FPU,no $)		LEON2 (1 ISSUE)	
.	.	2 ISSUES	1 ISSUE	FPU	no FPU	no SMP	SMP	FPU	no FPU
CFG_CFG	**0**	**1**	**2**	**3**	**4**	**5**	**6**	.	.
noelvsys								**mcore**	
processor cores	4	4	4	4	4	1	4	1	1
LUTs	165743	164309	126446	119575	94714	16279	60194	18251	9594
FFs	79401	74870	66958	55952	52231	9777	35074	8045	4872
RAMB36	4	4	4	4	4	0	0	0	0
RAMB18	175	175	175	119	119	11	23	44	44
DSP blocks	72	72	72	72	64	16	64	15	0
noelvcore								**proc**	
LUTs	39554	39198	30248	28046	21534	12544	13356	15507	6762
FFs	18675	17535	15630	12473	11616	7611	7763	6573	3359
RAMB36	1	1	1	1	1	0	0	0	0
RAMB18	42	42	42	28	28	4	4	20	20
DSP blocks	18	18	18	18	16	16	16	15	0

5.5 Maximum number of NOEL-V cores

A synthesis experiment was conducted that explored the maximal number of cores that can fit in the *XCKU040-2* devices when the target frequency is set to 100MHz[7] . The results are summed in Table 5.4.

Table 5.4: NOEL-V - maximum number of processor cores that fit in XCKU040.

Configuration	CFG0	CFG1	CFG2	CFG3	CFG4	CFG6
Max. NOEL-V cores	5	5	6	7	8	12

5.6 Performance limits

Performance of current processor-based systems is influenced by two factors:

- computing performance, i.e. the number of instructions that can be executed per second, and
- memory bandwidth, i.e. the number of bytes transfered per second.

Added together, these constituents form the commonly accepted roofline model. For the LEON and NOEL-V based systems evaluated in this report the upper bounds of the roofline model would be determined by the theoretical performance and bandwidth as follows.

The peak integer performance for a NOEL-V system running at 100MHz would be 2x100MOPS for dual-issue configurations, and 100MOPS for single-issue configurations. For LEON systems running at 100MHz this would be 100MOPS.

The peak floating-point performance for a NOEL-V system with *nanoFPU* running at 100MHz would be about 5-10MFLOPS. For LEON systems with a blocking FPU (*Meiko*, *daiFPU*) running at 100MHz this would be 10MFLOPS. For LEON systems with a non-blocking FPU (*GRFPU*, *GRFPU5*) running at 100MHz this would be up to 100MFLOPS.

The peak memory bandwidth for a 32-bit AHB bus running at 100MHz would be somewhere between 100MB/s and 400MB/s. For a 64-bit AHB bus the peak would be between 200MB/s and 800MB/s.

Table 5.5: NOEL-V and LEON - single-core performance limits at 100MHz.

System	Issues	Bus width	BW bound	Performance bound	
.	.	[bits]	[MB/s]	[MOPS]	[MFLOPS]
NOEL-V	2	64	200-800	200	5-10
NOEL-V	1	64	200-800	100	5-10
LEON/daiFPU	1	32	100-400	100	10
LEON/GRFPU	1	32	100-400	100	100

The upper-bound values are presented in Table 5.5. These can be projected to the naive roofline plot such that the upper bound π is determined by the *Performance bound*, and the slope β of the left line is

[7] The next release of this report will include information on maximum frequency of 4-core NOEL-V systems in *XCKU040-2*.

determined by the *BW bound*. Our initial assumption is that NOEL-V should outperform any LEON-based system for workloads that have low floating-point arithmetic intensity and higher demands on the memory bandwidth.

6 NOEL-V performance

6.1 PWLS benchmarks

Initial benchmarking experiments were carried out using the *Paranoia, Whetstone, Linpack* and *Stanford* benchmarks in order to

1. check the compliance of the *nanoFPU* to the IEEE Std. 754,
2. validate the ability of the toolchain to target BSPs with and without floating-point instructions through observing a lower performance for floating-point computation in software (SoftFloat) when compared to execution in hardware (FPU),
3. check consistency of the initial RTEMS implementation of the benchmarks when comparing performance to LEON-based systems running bare-metal and RTEMS versions of the benchmarks.

6.1.1 About the benchmarks

The *Whetstone* benchmark evaluates both floating-point and integer operations. The benchmark uses a high number of floating-point operations to mimic a behaviour of a typical numerical programs. Other notable features are a high usage of global variables, which decreases the importance of optimizations of register allocations by the compiler, and calls to mathematical library routines. The benchmark has high code locality.

The *Linpack* benchmark shows performance measured for basic linear algebra kernels. It exercises floating point additions and multiplications and no divisions, since the benchmark focuses mostly on the *saxpy* (*daxpy*) computation. For a given matrix dimension performance measurements are reported four times - the first three measurements are reported for a single computation of the kernels, and the last measurement is reported for the kernels computed 1000 times in a row.

The *Stanford* benchmark represents a collection of small programs that were used to compare performance of RISC and CISC processors during the development of the RISC architecture at the Stanford university.

Explanation of Stanford non-floating point and floating-point composite values [Muc89]: *The Stanford, or Hennessy, benchmark suite consists of several routines that compute permutations, solve some puzzles (Towers of Hanoi, Eight Queens, and Forest Baskett's blocks puzzle), multiply matrices (integer and floating point), compute a floating-point fast Fourier transform, and sort in three different ways (quick, bubble, and tree). Only matrix multiplication and fast Fourier transform involve floating point.*

6.1.2 Taxonomy

Explanation of benchmark acronyms:

- whets-DP - Whetstone that uses double-precision arithmetic.
- whets-SP - Whetstone that uses single-precision arithmetic.
- lpack-DP-ROL - Rolled Linpack that uses double-precision arithmetic.
- lpack-SP-ROL - Rolled Linpack that uses single-precision arithmetic.

- lpack-DP-UNR - Unrolled Linpack that uses double-precision arithmetic.
- lpack-SP-UNR - Unrolled Linpack that uses single-precision arithmetic.
- sford-DP-NFP - Stanford that uses double-precision arithmetic. Non-floating point composite result (permutations, towers of Hanoi, Eight Queens, Forest Baskett's block puzzle, matrix multiplication, quick-sort, bubble-sort, tree-sort).
- sford-DP-FP - Stanford that uses double-precision arithmetic. Floating point composite result (matrix multiplication, FFT).
- sford-SP-NFP - Stanford that uses single-precision arithmetic. Non-floating point composite result (permutations, towers of Hanoi, Eight Queens, Forest Baskett's block puzzle, matrix multiplication, quick-sort, bubble-sort, tree-sort).
- sford-SP-FP - Stanford that uses single-precision arithmetic. Floating point composite result (matrix multiplication, FFT).

6.1.3 Compiler options

The following options were used for compiling the PWLS benchmarks for NOEL-V:

```
--pipe
-march=rv64imafd
-mabi=lp64d
-O2
-g
-ffunction-sections
-fdata-sections
-Wall
-Wmissing-prototypes
-Wimplicit-function-declaration
-Wstrict-prototypes
-Wnested-externs
-B/opt/rtems-noel-1.0.4/kernel/riscv-rtems5/noel64imafd/lib
-specs bsp_specs
-qrtems
```

6.1.4 Measurements

The results shown in Table 6.1 were measured with PWLS benchmarks compiled for the *noel64imafd* BSP (hardware FPU). The results shown in Table 6.2 were measured with PWLS benchmarks compiled for the *noel64im* BSP (SoftFloat).

Table 6.1: NOEL-V single-core performance summary - PWLS, native floating-point. Performance at 100MHz.

Benchmark	Units	NV02	NV12	NV22	NV32
whets-DP	MWIPS	15.801	15.776	15.629	15.614
whets-SP	MWIPS	19.419	19.378	19.148	19.139
lpack-DP-ROL	Kflops	2565	2564	2529	2513
lpack-SP-ROL	Kflops	2726	2726	2688	2661
lpack-DP-UNR	Kflops	2616	2615	2601	2583
lpack-SP-UNR	Kflops	2780	2780	2764	2735
sford-DP-NFP	ms	18	18	21	23
sford-DP-FP	ms	54	55	59	63
sford-SP-NFP	ms	18	18	22	23
sford-SP-FP	ms	54	54	58	60

Table 6.2: NOEL-V single-core performance summary - PWLS, SoftFloat. Performance at 100MHz.

Benchmark	Units	NV01	NV11	NV21	NV31	NV41	NV51
whets-DP	MWIPS	6.651	6.561	4.736	4.178	4.177	1.187
whets-SP	MWIPS	7.558	7.406	5.286	4.960	4.958	1.217
lpack-DP-ROL	Kflops	1224	1103	823	815	815	217
lpack-SP-ROL	Kflops	1280	1268	912	907	907	239
lpack-DP-UNR	Kflops	1133	1115	836	827	827	237
lpack-SP-UNR	Kflops	1302	1274	923	914	914	228
sford-DP-NFP	ms	18	18	22	23	23	34
sford-DP-FP	ms	103	105	141	147	147	544
sford-SP-NFP	ms	18	18	22	23	23	42
sford-SP-FP	ms	99	101	139	142	142	572

The values in the table are also shown in Figures 6.1 to 6.3. The figures show the corresponding benchmark performance at 100MHz. For Figures 6.1 and 6.2 higher values denote better processor performance, while for Fig. 6.3 lower values indicate better performance.

6.1.5 Analysis

The PWLS benchmarks have probed the basic performance characteristics of the single-core NOEL-V configurations.

The *Paranoia* benchmark has detected a flaw in the *nanoFPU* - lack(s) of guard digits or failure(s) to correctly round or chop - for both single- and double-precision operations. This flaw is probably due to the use of the fused multiply-add instruction; the use of this instruction can be disabled by the compiler option `-ffp-contract-off`.

The *Whetstone*, *Linpack* and *Stanford* benchmarks have shown that adding the *nanoFPU* to a NOEL-V core increases its floating-point performance by at least 2x.

There is almost no difference in the single-core performance between the NOEL-V configurations that use *nanoFPU* - *CFG0* to *CFG3*.

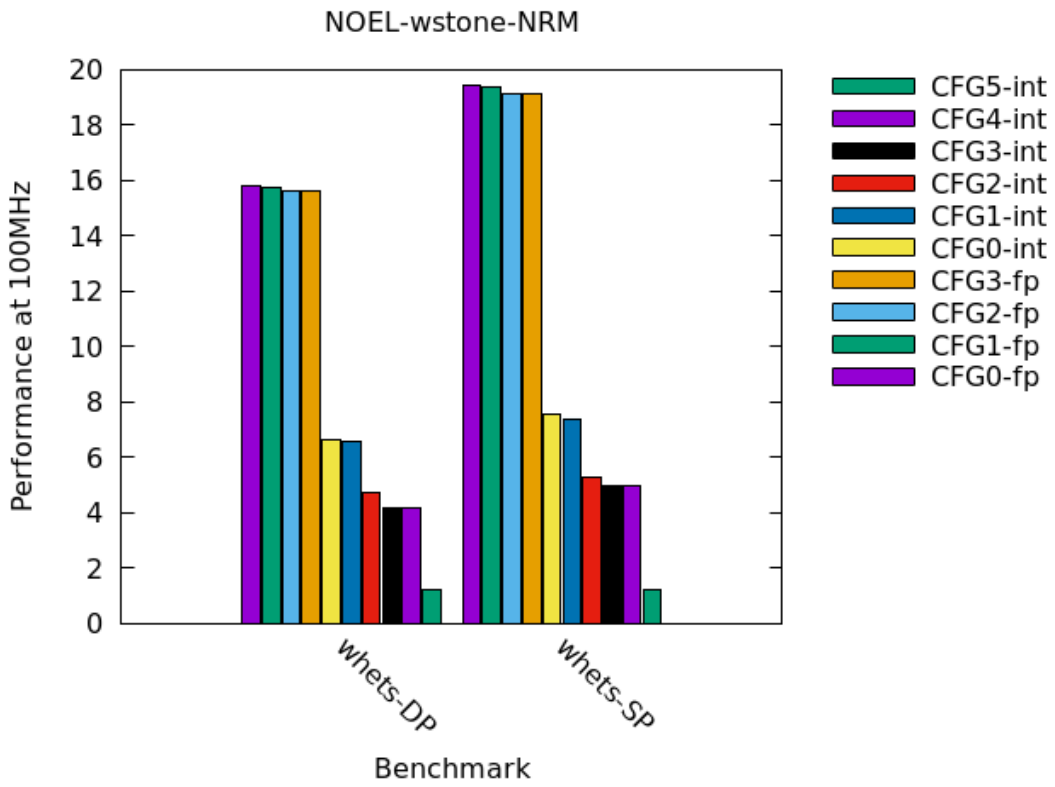

Figure 6.1: NOEL - Whetstone performance for NOEL targets, MWIPS at 100MHz. Higher values denote better performance. Configurations marked with `-fp` denote execution using the *nanoFPU*, while `-int` denotes execution using the *SoftFloat* library.

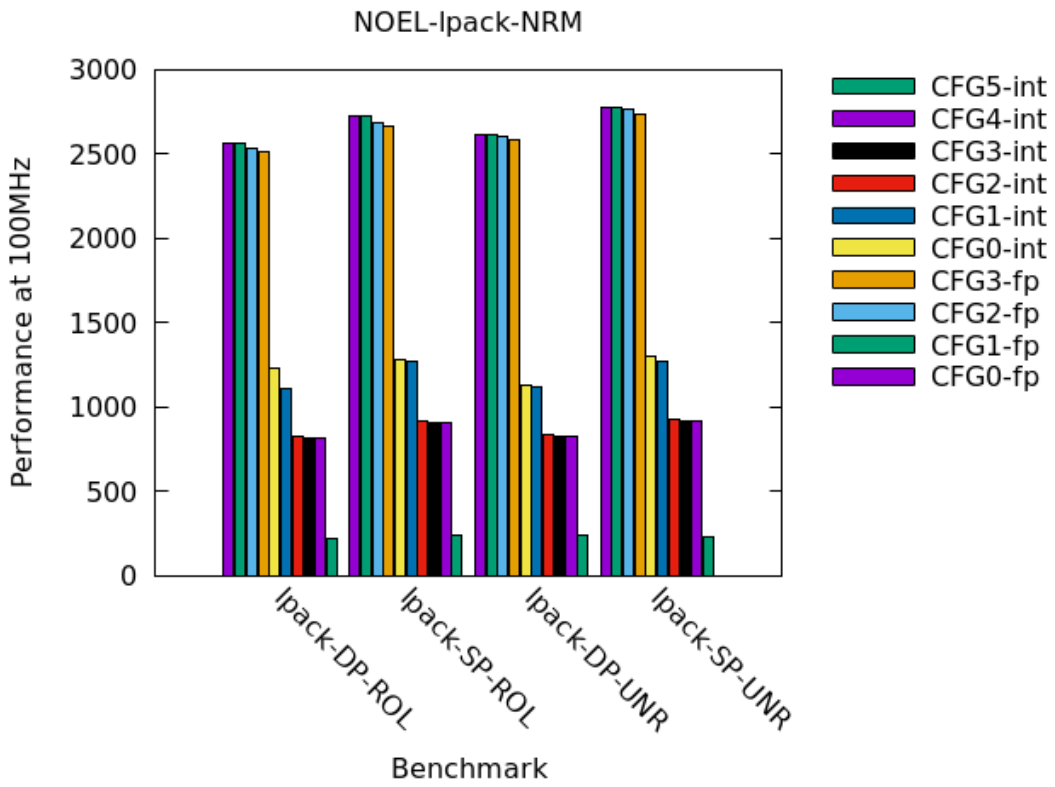

Figure 6.2: NOEL - Linpack performance for NOEL targets, Kflops at 100MHz. Higher values denote better performance. Configurations marked with `-fp` denote execution using the *nanoFPU*, while `-int` denotes execution using the *SoftFloat* library.

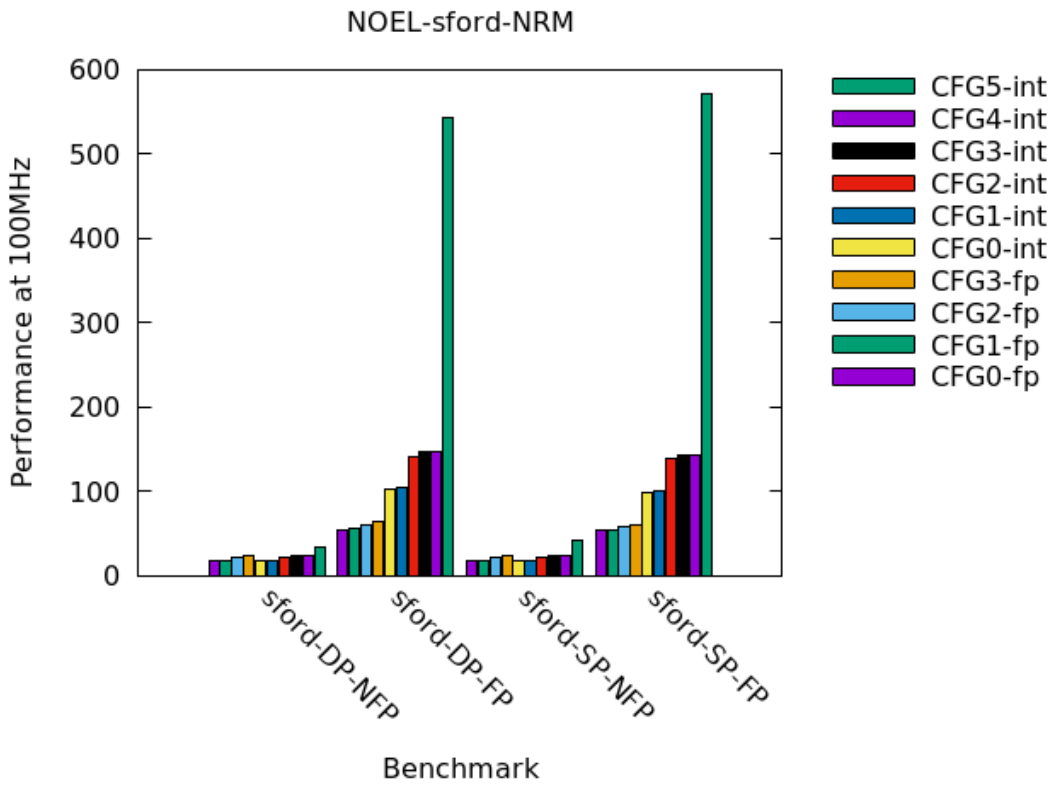

Figure 6.3: NOEL - Stanford performance for NOEL targets, milliseconds at 100MHz. Lower values denote better performance. Configurations marked with `-fp` denote execution using the *nanoFPU*, while `-int` denotes execution using the *SoftFloat* library.

The lack of caches in *CFG4* has a very significant negative impact on the performance of namely integer workloads that emulate floating-point operations using *SoftFloat*.

The positive effect of a larger *branch history table* and *branch target buffer* is visible in *SoftFloat* versions of the benchmarks that have higher number of subroutine calls and possibly deeper nesting of subroutine calls.

When using *SoftFloat* to emulate floating-point operations, the NOEL-V configurations can be divided into the following performance groups:

- high performance - *CFG0*, *CFG1* with almost the same performance,
- medium performance - *CFG2*, *CFG3*, *CFG4*, with *CFG2* having a slightly higher performance than the other two in the group,
- very low performance - *CFG5*, with about a 5x lower performance than *CFG0/1* and 3x lower performance than *CFG2/3/4*.

6.2 CoreMark

The *CoreMark* benchmark was used as a simple means to evaluate processor scaling on simple parallel workloads. The *CoreMark* benchmark is delivered as part of the *rtems-noel-1.0.4* distribution[8] , but to fit the needs of the benchmarking activity it has been extended to support parallel, multi-context execution in RTEMS using POSIX threads. The partial support for floating-point execution, implemented in the original *CoreMark* sources [CMR], was completed so that the benchmark can compute the `core_matrix()` function in the floating-point domain and thus enable evaluation of the floating-point performance.

6.2.1 Compiler options

The following options were used for compiling the *CoreMark* benchmark for NOEL-V:

```
-g
-march=rv64imafd
-mabi=lp64d
-O2
-ffunction-sections
-fdata-sections
-Wall
-Wmissing-prototypes
-Wimplicit-function-declaration
-Wstrict-prototypes
-Wnested-externs
-B/opt/rtems-noel-1.0.4/kernel/riscv-rtems5/noel64imafd_smp/lib
-specs bsp_specs
-qrtems
-funroll-all-loops
-funswitch-loops
-fgcse-after-reload
-fpredictive-commoning
-mtune=sifive-7-series
-finline-functions
-fipa-cp-clone
-falign-functions=8
-falign-loops=8
-falign-jumps=8
--param max-inline-insns-auto=20
```

6.2.2 Measurements

The goal was to validate the multi-context *CoreMark* implementation and execution in RTEMS against execution of identical sources compiled for LEON-based systems, and to provide initial evaluation of NOEL-V performance scaling. The results are shown in Table 6.3 for floating-point execution with binaries compiled for the *noel64imafd_smp* BSP, and Table 6.4 for integer execution with binaries compiled for the *noel64ima_smp*. Note that only the four configurations are listed in Table 6.3 that use the *nanoFPU*.

[8] To improve the quality of the generated code and the benchmark performance, the version of *CoreMark* distributed in *rtems-noel-1.0.4* defines ee_u32 as int32_t instead of uint32_t in core_portme.h so as to reduce the use of expensive unsigned 32 bit numbers where they are not needed. This modification has been kept both for the NOEL-V and LEON versions of the *CoreMark* benchmark.

Table 6.3: NOEL-V scalability - CoreMark, floating-point version, iterations/second at 100MHz for noel64imafd_smp.

THREADS	NV04	NV14	NV24	NV34
1	108.46	108.40	99.96	99.29
2	215.81	215.38	199.59	197.97
3	319.23	318.89	298.45	293.89
4	423.61	422.77	396.82	390.90
5	268.49	267.53	249.26	248.05
6	322.12	321.56	299.66	297.40
7	372.49	370.72	346.95	344.40
8	420.63	421.72	393.16	391.00
9	318.63	319.62	297.92	295.19
10	355.85	355.52	332.44	328.29
11	389.62	389.43	364.03	361.55
12	421.47	418.79	396.27	391.76

Table 6.4: NOEL-V scalability - CoreMark, integer version, iterations/second at 100MHz for noel64ima_smp.

THREADS	NV03	NV13	NV23	NV33	NV43	NV63
1	442.72	439.09	306.04	298.86	298.89	166.50
2	864.16	857.59	607.91	592.09	592.10	267.66
3	1216.80	1225.33	900.03	868.33	868.27	290.72
4	1467.80	1473.09	1157.22	1123.99	1121.23	302.86
5	1002.72	1007.03	745.65	732.17	732.74	265.61
6	1204.48	1190.23	891.42	872.54	872.93	286.70
7	1355.72	1367.37	1031.24	999.81	1001.50	315.15
8	1473.92	1497.38	1160.97	1125.08	1122.59	301.00
9	1181.15	1179.68	885.57	869.59	867.96	289.83
10	1310.71	1291.05	982.39	961.73	962.28	304.77
11	1406.78	1404.54	1074.15	1054.80	1053.38	304.97
12	1486.11	1488.03	1153.41	1131.54	1132.50	311.19

The values in the tables are also shown in Figures 6.4 and 6.5. The figures show the *CoreMark* iterations per second at 100MHz. Higher values denote better processor performance.

6.2.3 Analysis

First, all the tested NOEL-V configurations demonstrated performance scaling for varying number of *CoreMark* contexts. The zig-zag nature of the curves shown in Figures 6.4 and 6.5 indicates that the execution of the benchmark is compute-bound, and is caused by the low number of processor cores and the properties of the RTEMS scheduler[9] . Thread migration across processor cores, if implemented by the scheduler, probably results in eviction of cache lines by the migrating threads, and leads to the observable decrease

[9] In all the experiments described in this document the RTEMS scheduler was configured for pre-emptive scheduling using time slicing.

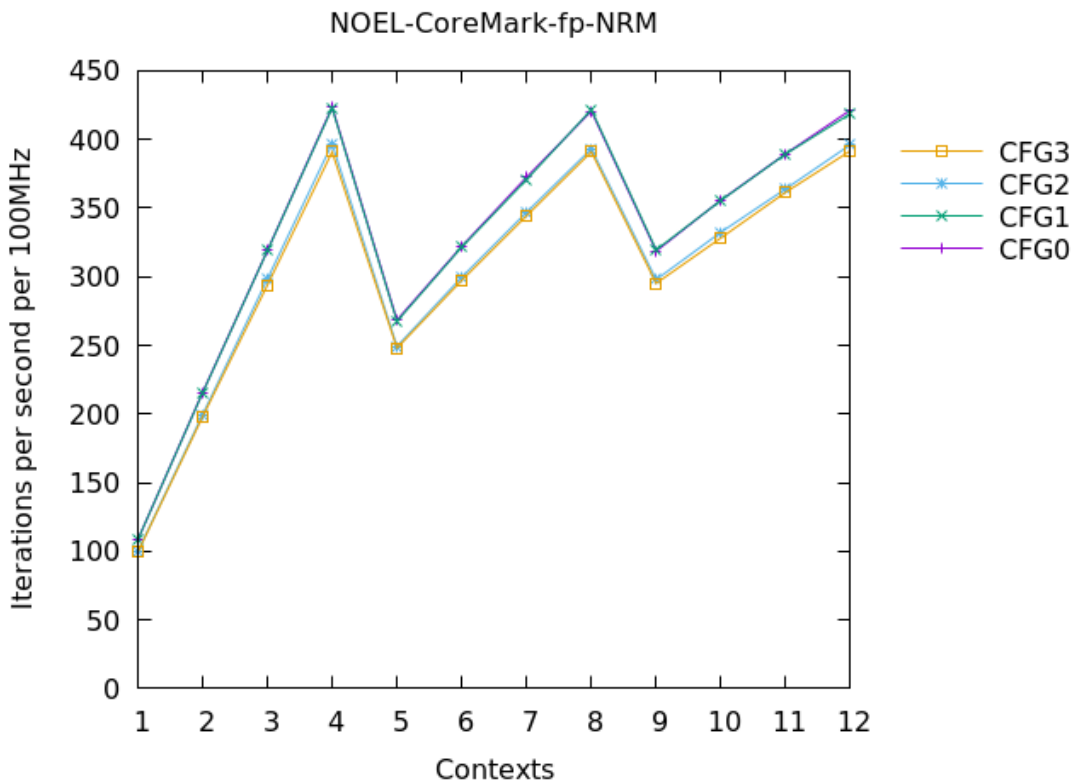

Figure 6.4: NOEL - CoreMark - floating-point performance for NOEL targets, CoreMark iterations at 100MHz. Higher values denote better performance.

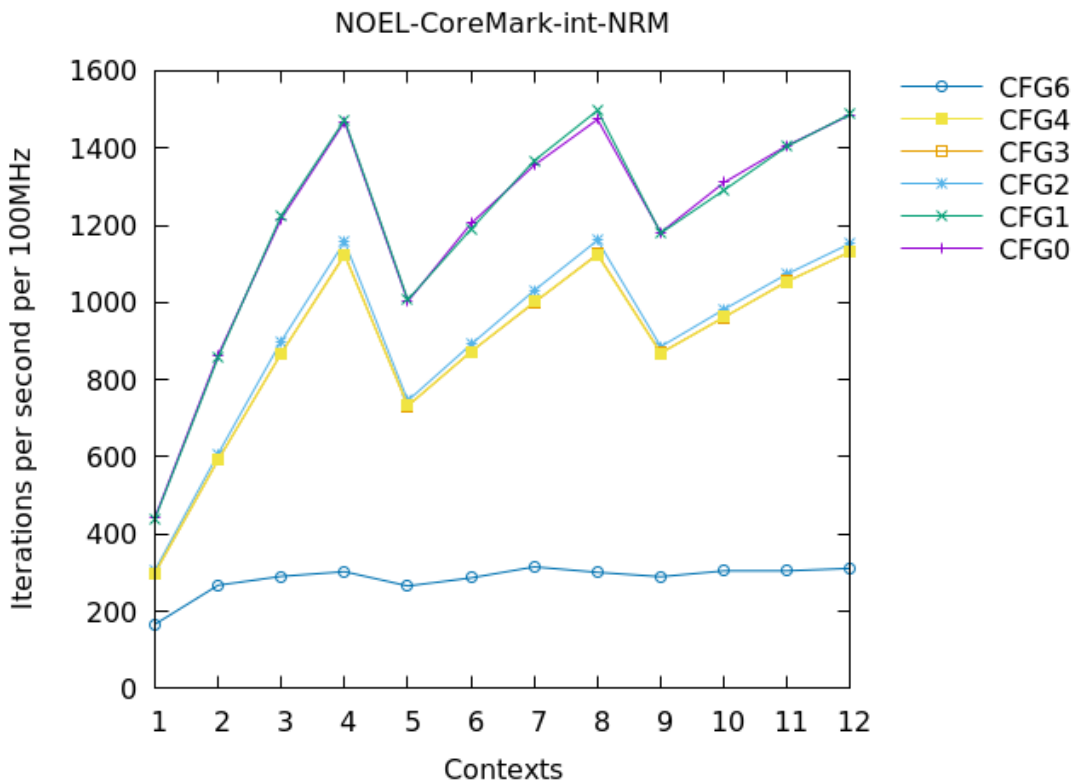

Figure 6.5: NOEL - CoreMark - integer performance for NOEL targets, CoreMark iterations at 100MHz. Higher values denote better performance.

in performance for five or more contexts that are not divisible by the number of available processor cores. Alternatively, the decreased performance may be caused by not fully utilizing all processing slots in configurations where the number of contexts is not divisible by the number of processor cores.

Fig. 6.4 indicates that the floating-point performance of dual-issue NOEL-V configurations *CFG0* and *CFG1* is nearly the same irrespective of the smaller *branch target buffer* used in *CFG1*. The floating-point performance of single-issue configurations is nearly the same irrespective of the smaller cache size, missing MMU, low-performance integer divider and smaller *branch history table* in *CFG3*. The actual performance difference between *CFG0*, *CFG1*, *CFG2* and *CFG3* is very small.

The results of the *CoreMark* integer execution shown in Fig. 6.5 confirm the previous observations from the *CoreMark* floating-point execution, but the performance difference between *CFG0/CFG1* and *CFG2/CFG3/CFG4* is bigger. The performance of the *TINY* NOEL-V configuration *CFG6* is very low, and its performance beyond 2 contexts is almost constant.

6.3 CoreMark-Pro

The *CoreMark-Pro* benchmark suite consists of a mix of integer and floating-point workloads that should provide better insight into the performance and scaling properties of NOEL-V[10] .

6.3.1 Specification of the workloads

CoreMark-Pro consists of nine workloads that sample different characteristics of parallel execution in multi-core systems. An overview of the workloads is shown in Table 6.5 together with abbreviated names that are used in the tables with the measured data[11] . Table 6.6 specifies for each workload the number of so-called iterations[12] that have to be completed to get a reliable *CoreMark-Pro* performance estimate.

Note that the *CoreMark-Pro* scores published in this report are not the certified scores as their measurement would require several order of magnitude longer execution time (days instead of hours)[13] .

Table 6.5: CoreMark-Pro - workload names and characteristics.

Full workload name	Abbr.	Type	Description
cjpeg-rose7-preset	cjpeg	Int	JPEG: colour space conversion and compression
core	core	Int	Improved CoreMark: list, matrix and FSM processing
linear_alg-mid-100x100-sp	linear	FP	Linpack: GEM
loops-all-mid-10k-sp	loops	FP	Livermore loops (selected): 24 kernels
nnet_test	nnet	FP	Neural Network for pattern evaluation
parser-125k	parser	Int	Simple XML parser: builds ezxml structure from XML
radix2-big-64k	radix2	FP	Radix-2 FFT: generic FFT decimation in time in radix 2
sha-test	sha	Int	SHA256 hashing: long code sequence with few branches
zip-test	zip	Int	zlib deflation: pattern matching, Huffman encoding

[10] *FPMark* results will be published in the next release.

[11] For a detailed explanation of the *CoreMark-Pro* workloads see the *datasheet.txt* files in the *benchmarks* directory in the *CoreMark-Pro* distribution package (e.g. [CMP]).

[12] *CoreMark-Pro* and also *FPMark* iterations refer to the number of runs of complete workloads. *Iterations* do not refer to actual iterations of individual workloads, but to complete executions of independent replicas of the workloads. The performance scalability is measured using the common coarse-grain process-based approach rather than more fine-grain multi-threaded execution of each workload.

[13] Certified scores are computed in two phases; each phase consists of one verification run, computing just one iteration of each workload, and subsequent three performance runs that compute the number of iterations as shown in Table 6.6. The result is determined as the median of the three performance runs. The first phase determines the single-core performance baseline, the second phase determines the multi-core performance for a user-selected number of contexts together with the scaling factor.

Table 6.6: CoreMark-Pro workloads, iterations computed per measurement. *-cx* denotes the number of parallel execution contexts.

.												Iterations
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	10	10	10	10	10	10	10	10	10	10	10	10
core	1	2	3	4	5	6	7	8	9	10	11	12
linear	50	50	50	50	50	50	50	50	50	50	50	50
loops	50	50	50	50	50	50	50	50	50	50	50	50
nnet	10	10	10	10	10	10	10	10	10	10	11	12
parser	1	2	3	4	5	6	7	8	9	10	11	12
radix2	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
sha	10	10	10	10	10	10	10	10	10	10	11	12
zip	1	2	3	4	5	6	7	8	9	10	11	12

6.3.2 Worst-case execution time

An approximate worst-case execution time for the *CoreMark-Pro* suite executed in NOEL-V is shown in Table 6.7.

Table 6.7: NOEL-V - worst-case execution times.

Configuration	CFG0	CFG1	CFG2	CFG3	CFG4	CFG6
Frequency [MHz]	100	100	100	100	100	100
Execution time [hrs]	2	2	2	2	6	6

6.3.3 Compiler options

The following options were used for compiling the *CoreMark-Pro* benchmark suite for NOEL-V:

```
-g
-march=rv64imafd
-mabi=lp64d
-O2
-ffunction-sections
-fdata-sections
-Wall
-Wmissing-prototypes
-Wimplicit-function-declaration
-Wstrict-prototypes
-Wnested-externs
-B/opt/rtems-noel-1.0.4/kernel/riscv-rtems5/noel64imafd_smp/lib
-specs bsp_specs
-qrtems
-funroll-all-loops
```

(continues on next page)

(continued from previous page)

```
-funswitch-loops
-fgcse-after-reload
-fpredictive-commoning
-mtune=sifive-7-series
-finline-functions
-fipa-cp-clone
-falign-functions=8
-falign-loops=8
-falign-jumps=8
--param max-inline-insns-auto=20
```

6.3.4 Measurements

The measurements are presented in one set of tables that show workload iterations computed per second.

Table 6.8: NOEL-V - CoreMark-Pro results for configuration NV04 - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	2.5967	4.5249	5.0429	5.1020	4.6318	4.6838	4.7551	4.5746	4.5475	4.5935	4.7015	4.5434
core	0.0263	0.0520	0.0735	0.0865	0.0598	0.0712	0.0804	0.0862	0.0694	0.0767	0.0835	0.0883
linear	0.1826	0.3638	0.5329	0.6940	0.6960	0.6921	0.7010	0.6934	0.7113	0.6909	0.7166	0.7034
loops	0.0101	0.0188	0.0249	0.0290	0.0291	0.0291	0.0291	0.0290	0.0292	0.0292	0.0292	0.0290
nnet	0.0106	0.0211	0.0264	0.0351	0.0351	0.0351	0.0351	0.0351	0.0351	0.0351	0.0385	0.0420
parser	0.2853	0.2420	0.2045	0.1865	0.1565	0.1574	0.1555	0.1548	0.1534	0.1542	0.1522	0.1520
radix	1.0961	2.1270	3.0566	3.8406	3.8363	3.8360	3.8354	3.8353	3.8355	3.8351	3.8350	3.8355
sha	0.9193	1.1127	1.0743	1.0739	1.0828	1.0628	1.0769	1.0838	1.0711	1.0771	1.0988	1.0912
zip	1.2195	1.5736	1.5617	1.5456	1.4723	1.5516	1.5531	1.5465	1.5060	1.5454	1.5486	1.5480

Table 6.9: NOEL-V - CoreMark-Pro results for configuration NV14 - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	2.5893	4.5005	5.0176	5.2029	4.6275	4.6948	4.7371	4.5662	4.5310	4.6253	4.7081	4.5683
core	0.0263	0.0519	0.0733	0.0874	0.0597	0.0712	0.0803	0.0871	0.0694	0.0767	0.0831	0.0883
linear	0.1826	0.3638	0.5329	0.6940	0.6946	0.7140	0.6959	0.6920	0.7113	0.6910	0.7166	0.6995
loops	0.0101	0.0188	0.0249	0.0290	0.0293	0.0291	0.0292	0.0291	0.0292	0.0292	0.0292	0.0290
nnet	0.0106	0.0211	0.0263	0.0351	0.0351	0.0351	0.0351	0.0351	0.0351	0.0351	0.0385	0.0420
parser	0.2851	0.2484	0.2064	0.1854	0.1564	0.1582	0.1552	0.1549	0.1528	0.1536	0.1523	0.1522
radix	1.0949	2.1252	3.0543	3.8377	3.8333	3.8328	3.8341	3.8319	3.8328	3.8319	3.8356	3.8358
sha	0.9193	1.1094	1.0745	1.0792	1.0595	1.0700	1.0732	1.0655	1.0718	1.0792	1.0928	1.0906
zip	1.2195	1.5711	1.5666	1.5468	1.4697	1.5512	1.5538	1.5483	1.5058	1.5487	1.5495	1.5500

Table 6.10: NOEL-V - CoreMark-Pro results for configuration NV24 - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	2.0446	3.7679	4.8123	5.0839	4.5683	4.6620	4.7619	4.5413	4.5126	4.5683	4.6816	4.4984
core	0.0176	0.0351	0.0524	0.0687	0.0435	0.0521	0.0607	0.0687	0.0515	0.0571	0.0628	0.0681
linear	0.1798	0.3583	0.5249	0.6834	0.6806	0.6889	0.6878	0.6815	0.7005	0.6828	0.6954	0.6888
loops	0.0101	0.0187	0.0248	0.0289	0.0290	0.0290	0.0291	0.0291	0.0291	0.0291	0.0291	0.0290
nnet	0.0105	0.0209	0.0261	0.0348	0.0348	0.0348	0.0348	0.0348	0.0348	0.0348	0.0382	0.0416
parser	0.2814	0.2499	0.2016	0.1851	0.1558	0.1581	0.1545	0.1550	0.1515	0.1524	0.1511	0.1510
radix	1.0913	2.1234	3.0465	3.8328	3.8280	3.8304	3.8269	3.8298	3.8307	3.8281	3.8287	3.8293
sha	0.7188	1.0685	1.0503	1.0844	1.0873	1.0839	1.0799	1.0851	1.1179	1.1161	1.1341	1.1242
zip	1.0204	1.5256	1.5666	1.5498	1.4069	1.5424	1.5566	1.5507	1.4679	1.5427	1.5510	1.5508

Table 6.11: NOEL-V - CoreMark-Pro results for configuration NV34 - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	1.7522	3.0451	3.4758	3.5537	3.1596	3.2342	3.2415	3.1496	3.1377	3.1726	3.2206	3.1358
core	0.0174	0.0347	0.0517	0.0672	0.0428	0.0514	0.0597	0.0674	0.0507	0.0562	0.0616	0.0667
linear	0.1783	0.3549	0.5194	0.6756	0.6790	0.6725	0.6845	0.6724	0.6913	0.6725	0.6949	0.6795
loops	0.0099	0.0182	0.0241	0.0283	0.0284	0.0282	0.0282	0.0281	0.0282	0.0281	0.0282	0.0280
nnet	0.0099	0.0197	0.0246	0.0325	0.0325	0.0325	0.0325	0.0325	0.0325	0.0325	0.0356	0.0384
parser	0.2667	0.2173	0.1670	0.1382	0.1186	0.1186	0.1180	0.1177	0.1164	0.1162	0.1163	0.1159
radix	1.0863	2.1121	3.0305	3.8036	3.7994	3.7974	3.7979	3.7970	3.7987	3.7979	3.7987	3.7976
sha	0.2981	0.3504	0.3456	0.3486	0.3484	0.3485	0.3481	0.3479	0.3482	0.3481	0.3479	0.3474
zip	0.9099	1.2895	1.2898	1.2788	1.1812	1.2731	1.2804	1.2727	1.2245	1.2763	1.2768	1.2723

Table 6.12: NOEL-V - CoreMark-Pro results for configuration NV43 - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	1.7298	3.0184	3.5311	3.6114	3.2605	3.3124	3.3212	3.2394	3.2383	3.2669	3.3047	3.2383
core	0.0173	0.0344	0.0517	0.0672	0.0429	0.0514	0.0597	0.0674	0.0511	0.0567	0.0623	0.0674
linear	0.0623	0.1223	0.1725	0.2056	0.2142	0.2127	0.2111	0.2158	0.2147	0.2114	0.2122	0.2145
loops	0.0036	0.0067	0.0091	0.0104	0.0105	0.0105	0.0105	0.0105	0.0106	0.0106	0.0106	0.0105
nnet	0.0030	0.0058	0.0069	0.0084	0.0084	0.0084	0.0084	0.0084	0.0084	0.0084	0.0090	0.0094
parser	0.2657	0.2057	0.1705	0.1422	0.1219	0.1218	0.1215	0.1205	0.1203	0.1205	0.1199	0.1197
radix	0.3071	0.5614	0.7801	0.8503	0.8446	0.8510	0.8564	0.8592	0.8549	0.8581	0.8589	0.8597
sha	0.3335	0.3938	0.3862	0.3918	0.3923	0.3911	0.3913	0.3914	0.3913	0.3919	0.3923	0.3910
zip	0.9066	1.2812	1.2815	1.2706	1.1723	1.2653	1.2748	1.2654	1.2147	1.2694	1.2695	1.2641

The values shown in Tables 6.8 to 6.12 are plotted in Figures 6.6 to 6.14.

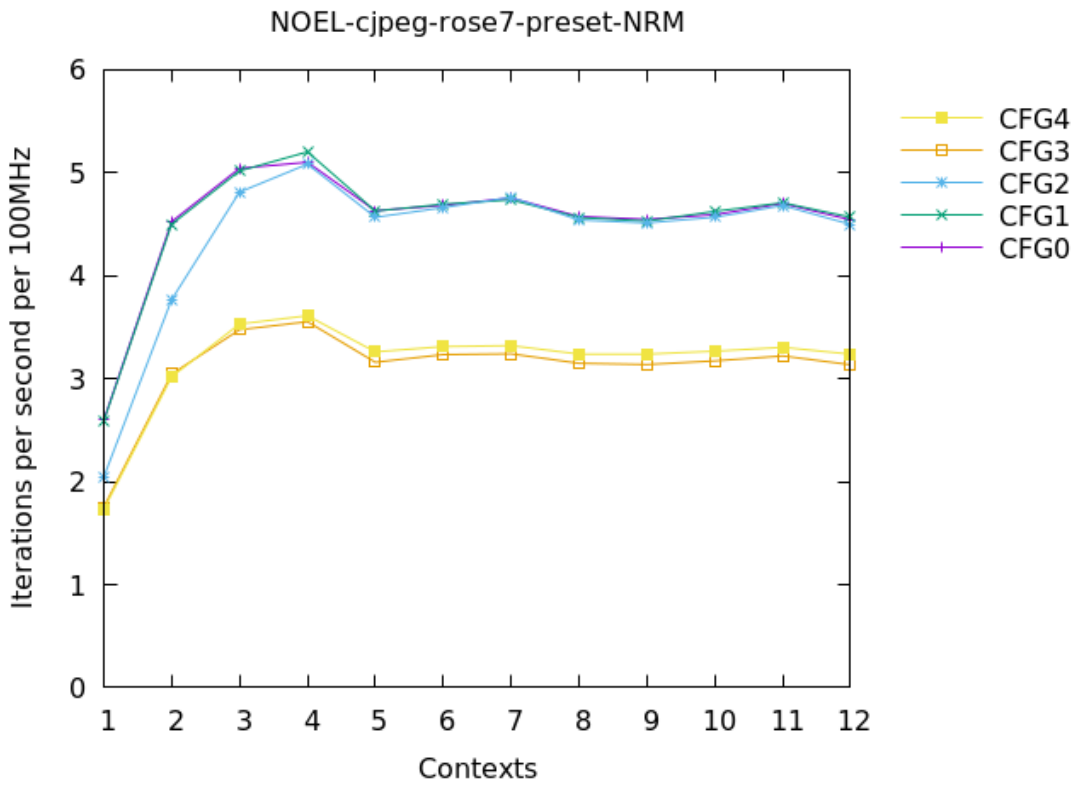

Figure 6.6: NOEL - CoreMark-Pro - cjpeg, iterations per second at 100MHz. Higher values denote better performance.

6.3.5 Analysis

Observations based on Figures 6.6 to 6.14 are as follows:

- Branch target prediction: There is no observable performance difference between *CFG0* and *CFG1*.
- Dual vs. single issue: The performance of *CFG2* is almost identical to *CFG0* and *CFG1* with the exception of *core*.
- Cache size: *CFG3* achieves a significantly lower performance in *cjpeg*, *parser*, *sha*, *zip*.
- FPU performance: There is almost no performance difference between *CFG0*, *CFG1*, *CFG2* and *CFG3* in *linear*, *loops*, *nnet*, *radix2*.
- *nanoFPU* vs *SoftFloat*: Tables 6.11 and 6.12 present the *CoreMark-Pro* performance for execution of floating-point operations in *nanoFPU* and in *SoftFloat* respectively. It can be seen that for the workloads without floating-point operations the performance is nearly identical. For the four floating-point workloads (*linear*, *loops*, *nnet*, *radix2*) the *SoftFloat* performance in *CFG4* is about 4x lower than the *nanoFPU* performance in *CFG3*.

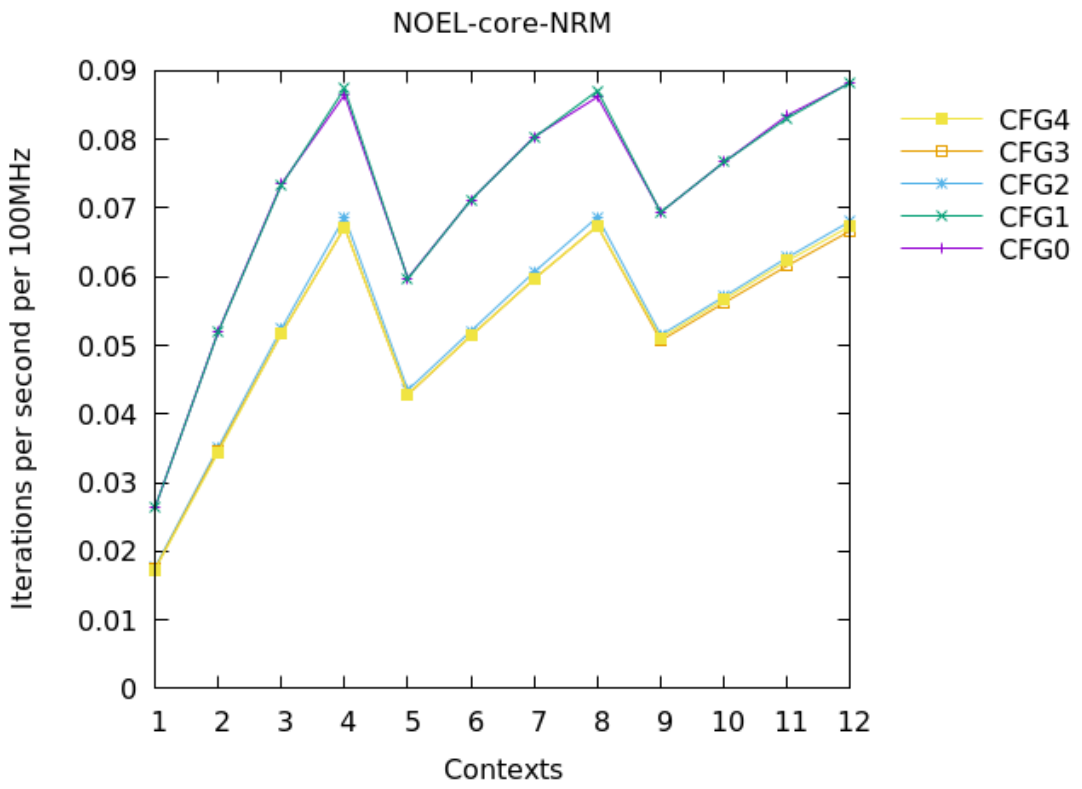

Figure 6.7: NOEL - CoreMark-Pro - core, iterations per second at 100MHz. Higher values denote better performance.

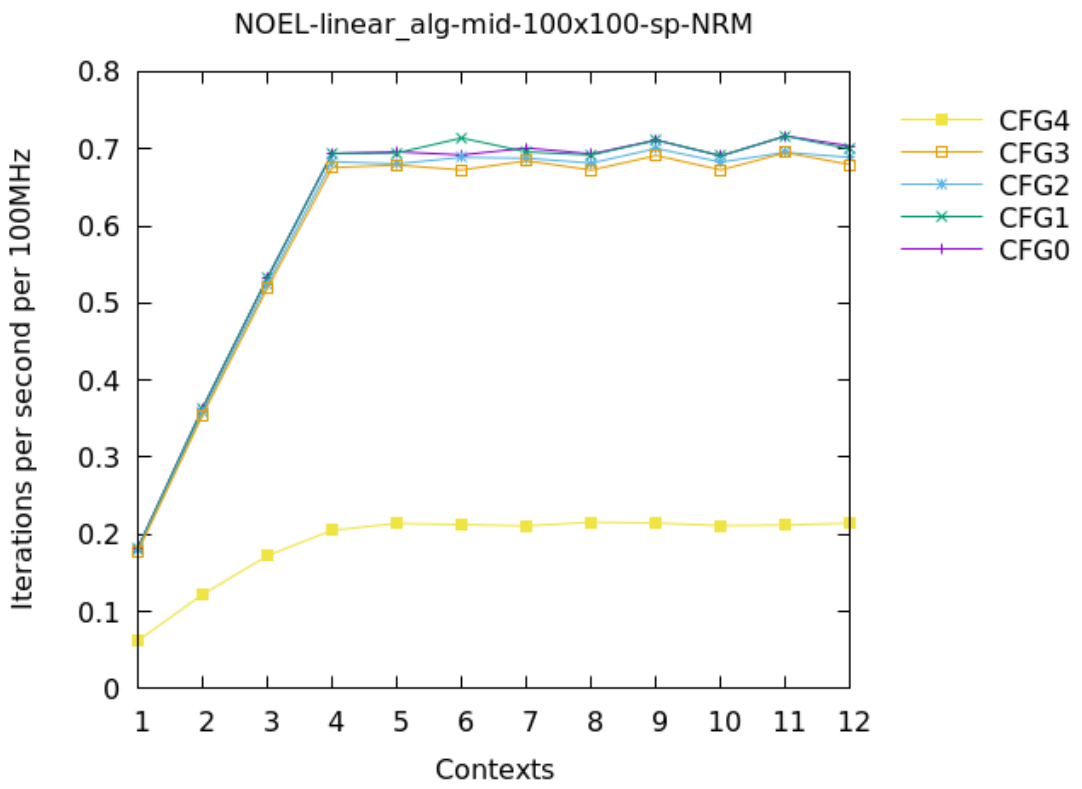

Figure 6.8: NOEL - CoreMark-Pro - linear, iterations per second at 100MHz. Higher values denote better performance.

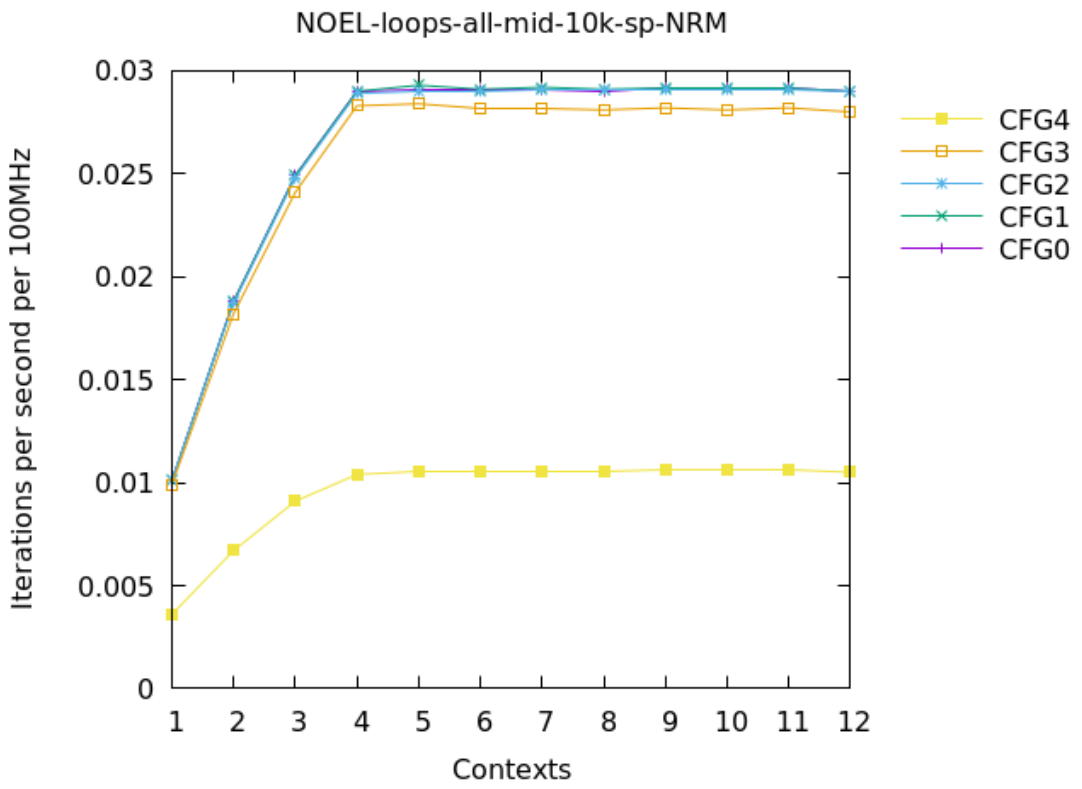

Figure 6.9: NOEL - CoreMark-Pro - loops, iterations per second at 100MHz. Higher values denote better performance.

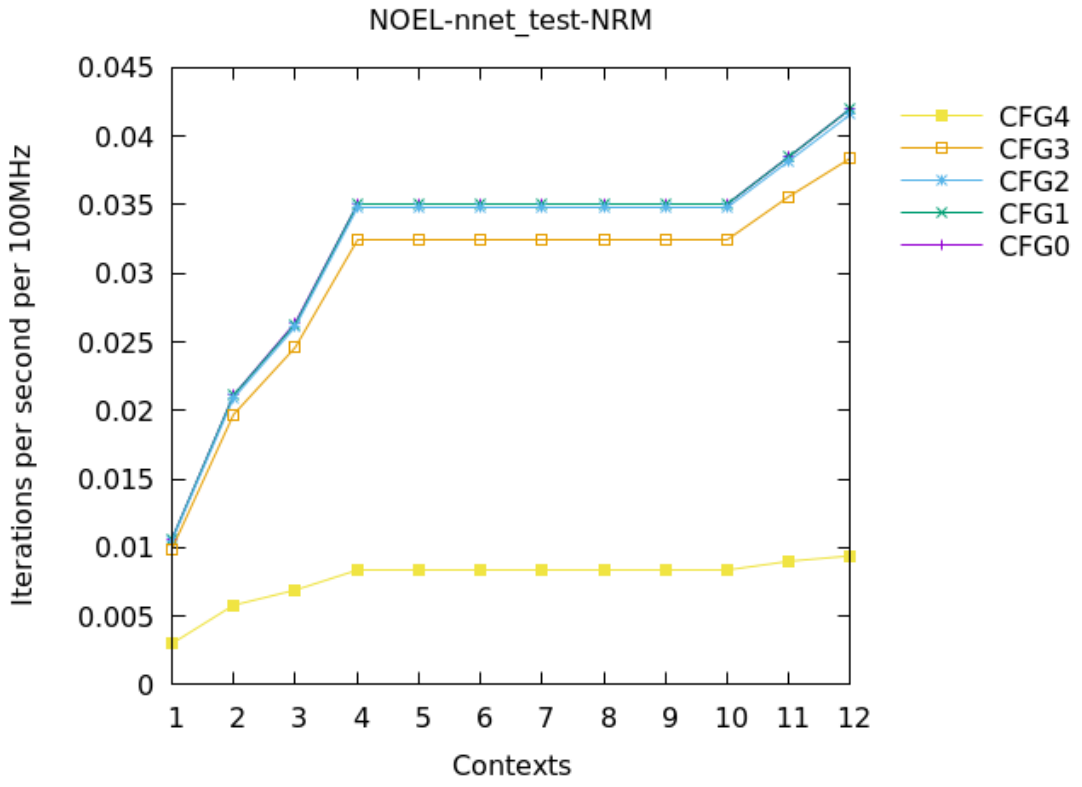

Figure 6.10: NOEL - CoreMark-Pro - nnet, iterations per second at 100MHz. Higher values denote better performance.

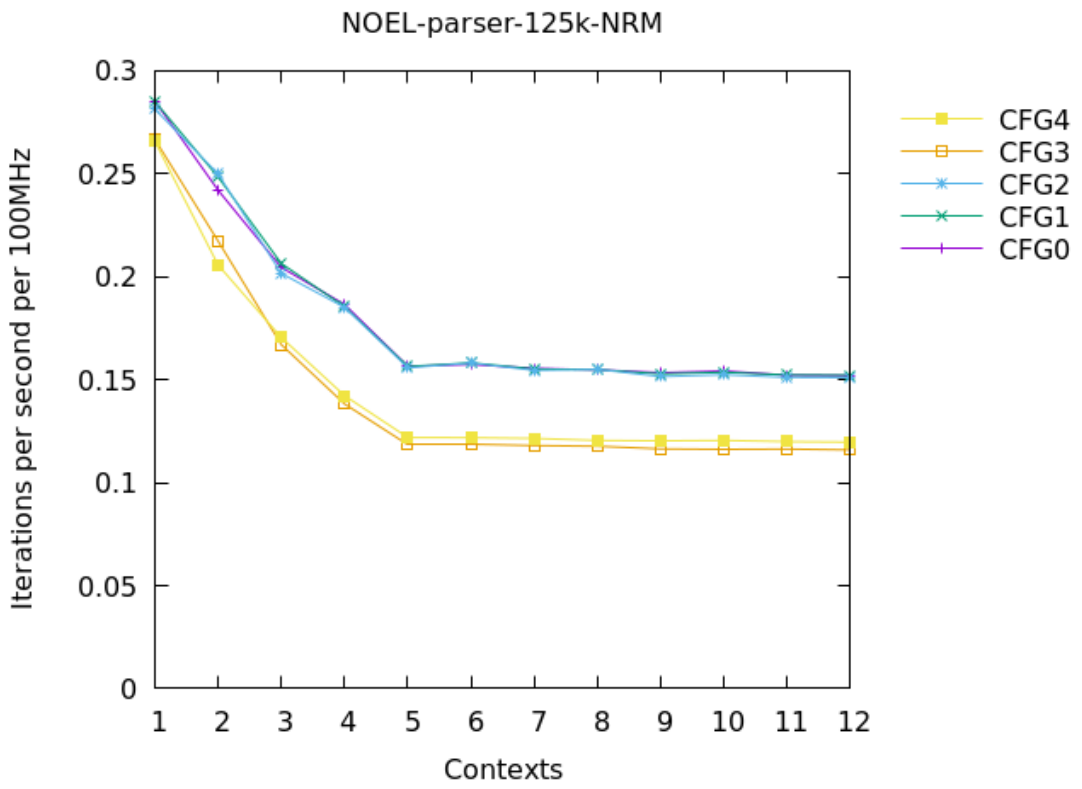

Figure 6.11: NOEL - CoreMark-Pro - parser, iterations per second at 100MHz. Higher values denote better performance.

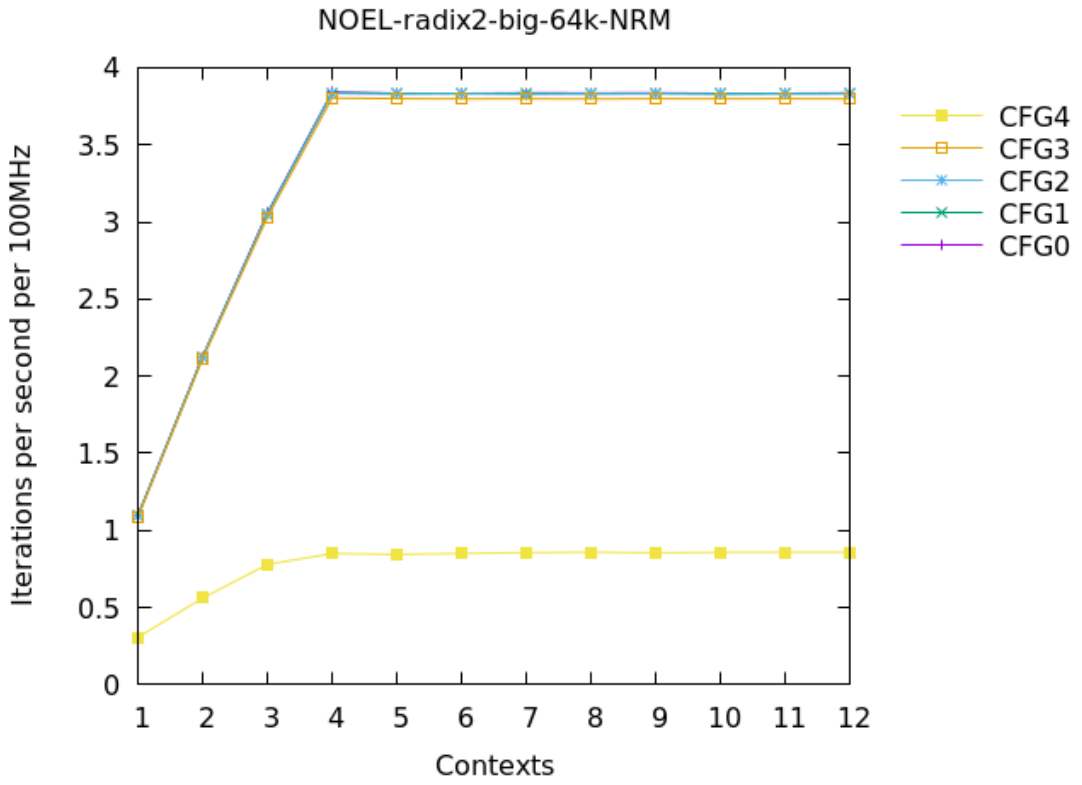

Figure 6.12: NOEL - CoreMark-Pro - radix2, iterations per second at 100MHz. Higher values denote better performance.

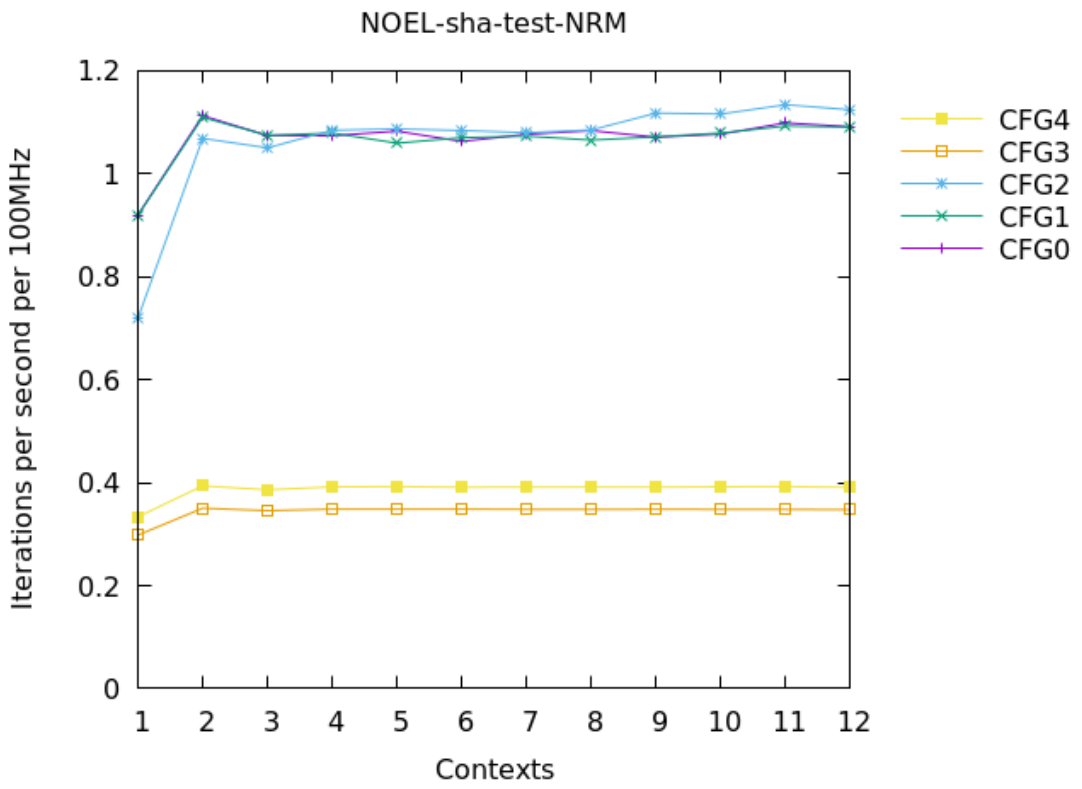

Figure 6.13: NOEL - CoreMark-Pro - sha, iterations per second at 100MHz. Higher values denote better performance.

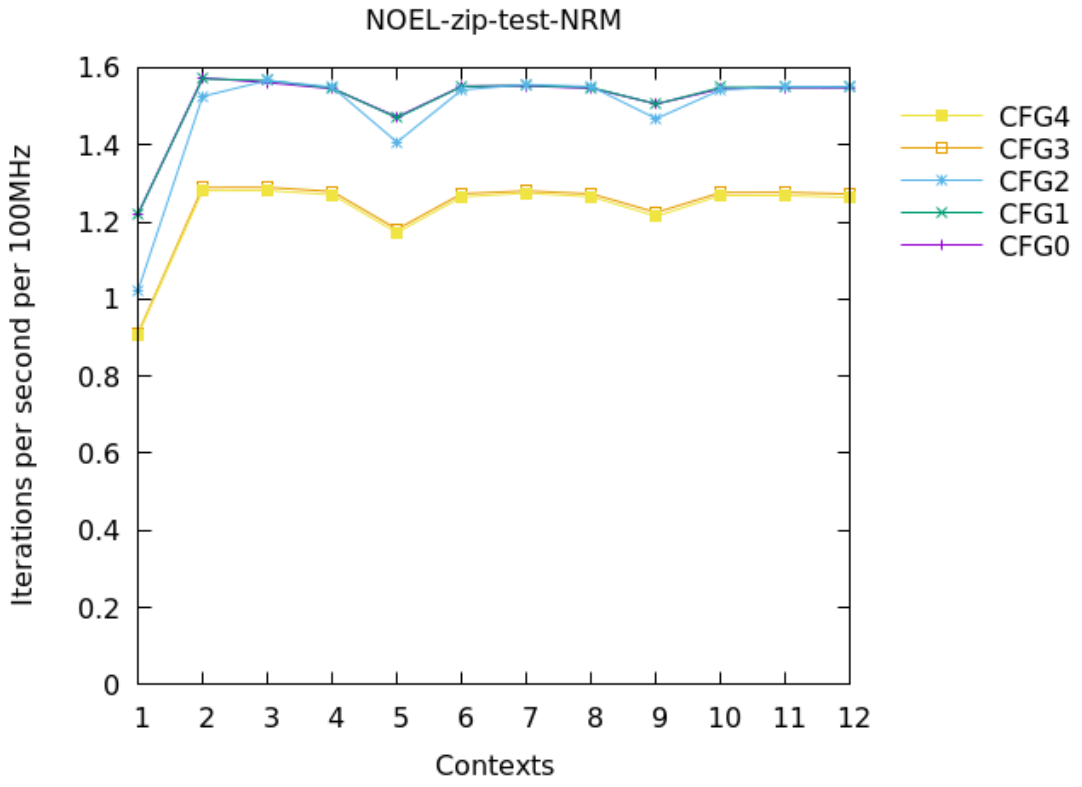

Figure 6.14: NOEL - CoreMark-Pro - zip, iterations per second at 100MHz. Higher values denote better performance.

7 LEON performance

7.1 PWLS benchmarks

7.1.1 Compiler options

The following options were used for compiling the PWLS benchmarks for LEON:

```
--pipe
-O2
-g
-ffunction-sections
-fdata-sections
-Wall
-Wmissing-prototypes
-Wimplicit-function-declaration
-Wstrict-prototypes
-Wnested-externs
-B, -mcpu, -qbsp as per the selected system
```

7.1.2 Measurements

For an explanation of the benchmark acronyms see Section 6.1.2.

Table 7.1 provides performance results for the *Paranoia*, *Whetstone*, *Linpack* and *Stanford* benchmarks. For LEON3 and LEON5 results are shown both for the *leon3* and *leon3_smp* BSP to highlight possible performance inconsistencies in the compiled binaries.

Table 7.1: LEON single-core performance summary - PWLS, native floating-point. Performance at 100MHz.

Benchmark	Units	AT697	LEON2	LEON3	LEON3SMP	GR740	LEON5	LEON5SMP
.	.	LE1	LE2	LE3	LE4	LE6	LE9	LE10
whets-DP	MWIPS	37.042	39.894	76.740	76.713	83.263	85.118	85.162
whets-SP	MWIPS	51.530	48.102	83.028	83.007	84.540	85.751	85.748
lpack-DP-ROL	Kflops	5184	5534	5166	5162	9968	7584	9839
lpack-SP-ROL	Kflops	7597	6842	6690	6681	8718	7090	7749
lpack-DP-UNR	Kflops	5477	5776	5262	5256	9880	7389	9294
lpack-SP-UNR	Kflops	8368	7443	7813	7798	10176	8416	9356
sford-DP-NFP	ms	24	26	21	23	20	14	13
sford-DP-FP	ms	53	50	37	39	33	28	27
sford-SP-NFP	ms	25	26	21	23	20	12	13
sford-SP-FP	ms	47	46	34	36	33	26	26

The values in the table are also shown in Figures 7.1 to 7.3. The figures show the corresponding benchmark

performance at 100MHz. For Figures 7.1 and 7.2 higher values denote better processor performance, while for Fig. 7.3 lower values indicate better performance.

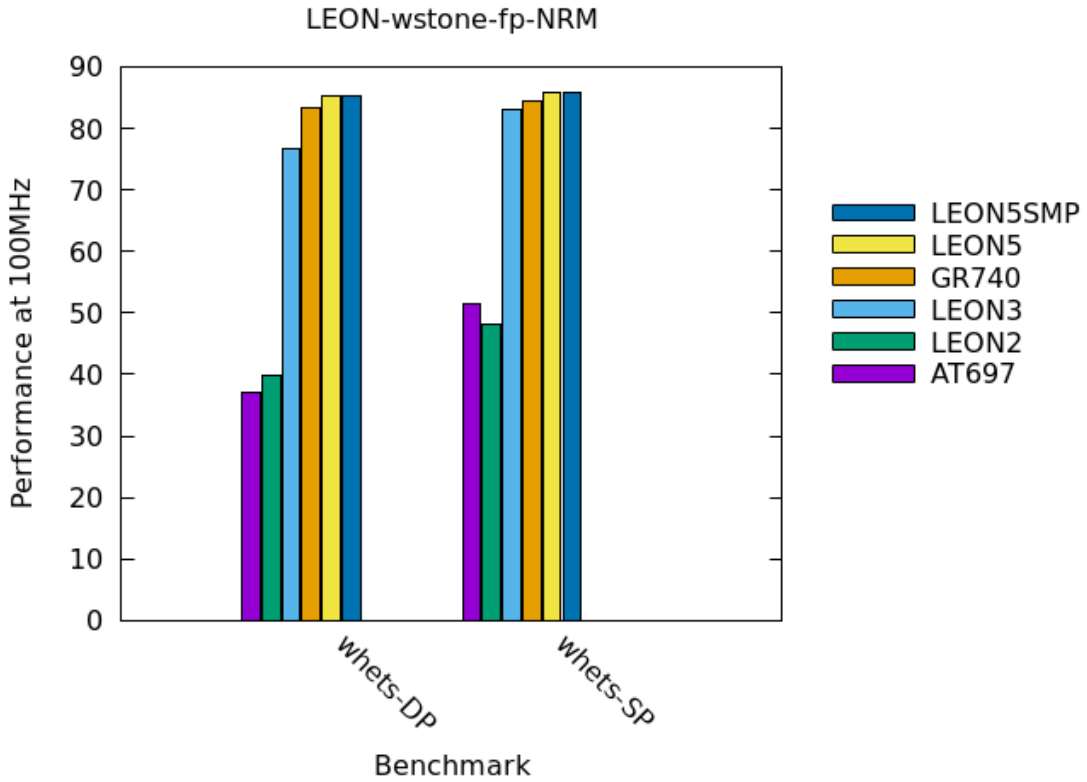

Figure 7.1: LEON - Whetstone performance for LEON targets, MWIPS at 100MHz. Higher values denote better performance.

7.1.3 Analysis

The PWLS benchmarks have shown the basic performance characteristics of single-core LEON-based systems.

The *Paranoia* benchmark has detected flaws in the *GRFPU* calculations that are related to the missing support for subnormal representations. *Meiko*, *daiFPU* and *GRFU5* passed without any flaws, defects or errors.

The *Whetstone* benchmark has shown that the blocking FPUs used in AT697F (*Meiko*) and LEON2FT (*daiFPU*) achieve about half the floating-point performance of the non-blocking FPUs used in LEON3 (*GRFPU*), GR740 (*GRFPU*) and LEON5 (*GRFPU5*). Overall, *GRFPU5* has a slightly higher performance than *GRFPU*.

The *Linpack* benchmark divides the LEON-based systems in two groups: AT697F, LEON2FT and LEON3 achieve about 0.5x-0.8x the performance of GR740 and LEON5. The outstanding performance of GR740 and LEON5 is caused by the presence of the level-2 cache that is missing in the rest of the systems.

The *Stanford* benchmark shows that in general each generation of the LEON processor achieves a better performance[14] than the previous generation.

[14] both integer and floating-point performance

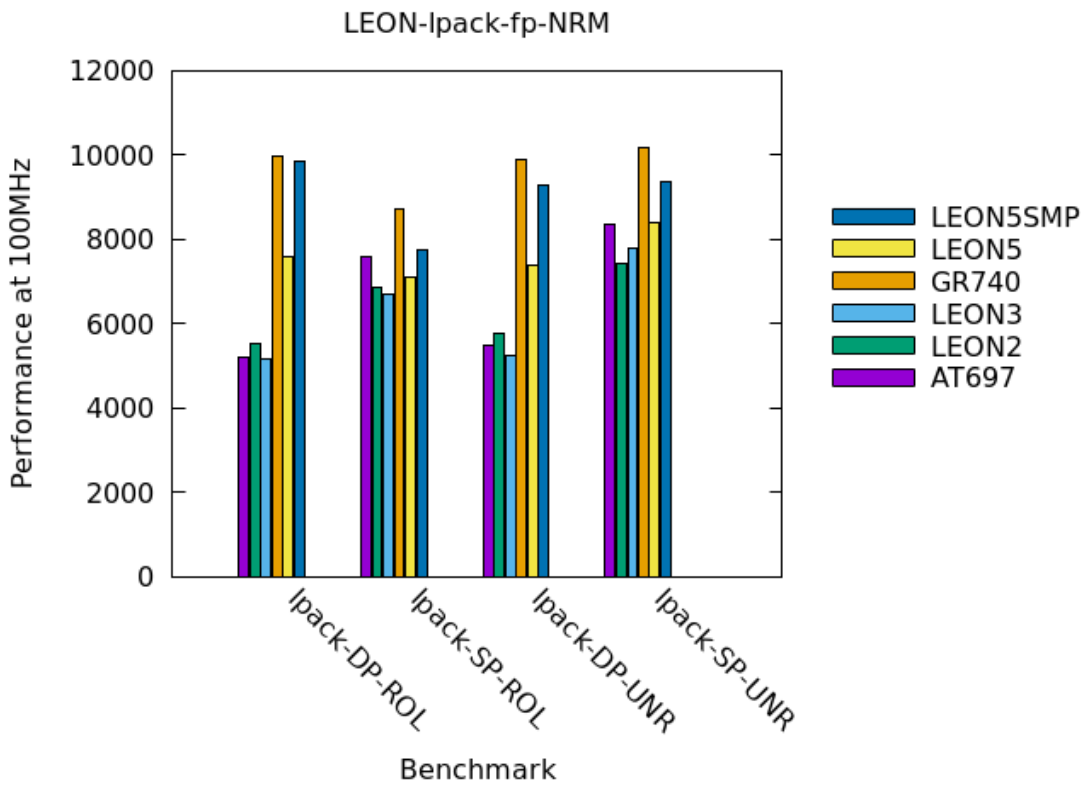

Figure 7.2: LEON - Linpack performance for LEON targets, Kflops at 100MHz. Higher values denote better performance.

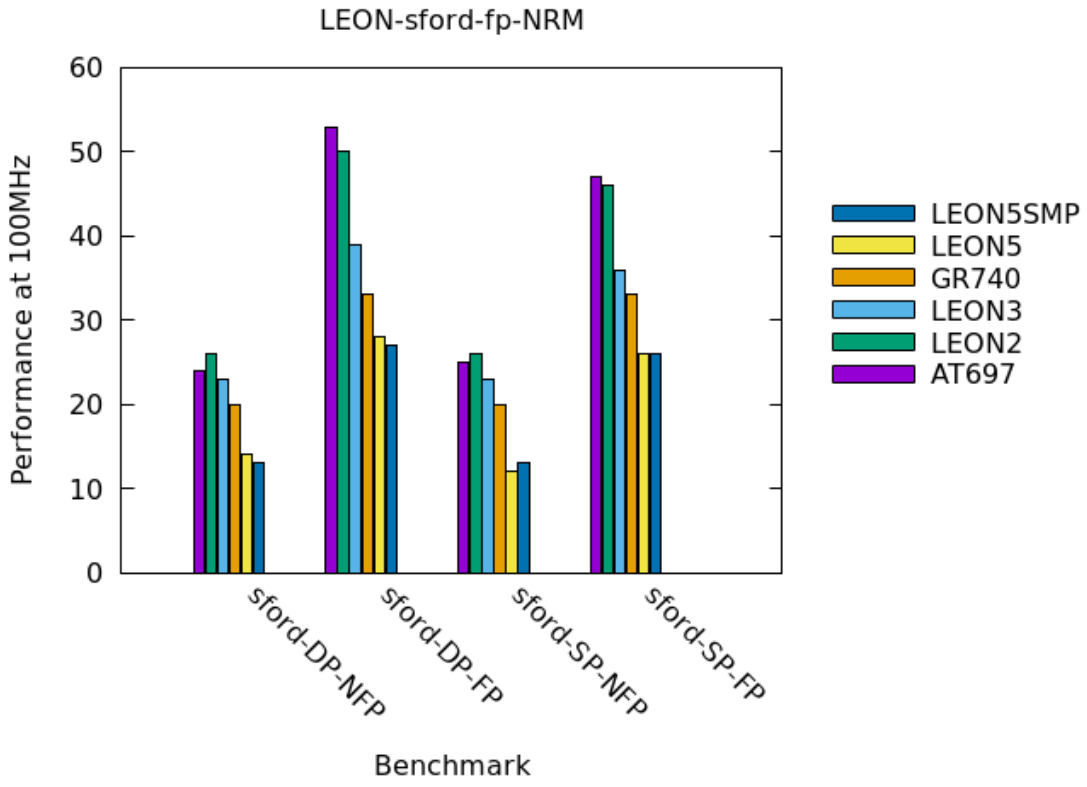

Figure 7.3: LEON - Stanford performance for LEON targets, milliseconds at 100MHz. Lower values denote better performance.

7.2 CoreMark

The *CoreMark* data are presented only for AT697F, LEON2FT, LEON3, GR740 and LEON5 in two configurations - with and without a level-2 cache.

7.2.1 Compiler options

The following options were used for compiling the *CoreMark* benchmark for LEON:

```
-g
-O2
-qrtems
-funroll-all-loops
-funswitch-loops
-fgcse-after-reload
-fpredictive-commoning
-finline-functions
-fipa-cp-clone
-falign-functions=8
-falign-loops=8
-falign-jumps=8
--param max-inline-insns-auto=20
-B, -mcpu, -qbsp as per the selected system
```

7.2.2 Measurements

Table 7.2 shows results for the *CoreMark* benchmark when `the core_matrix()` function is computed in the floating-point domain. Table 7.3 shows results for the *CoreMark* benchmark when `the core_matrix()` function is computed in the integer domain.

Table 7.2: LEON scalability - CoreMark, floating-point version, iterations per second at 100MHz.

THREADS	AT697F	LEON2	LEON3	GR740	LEON5L2	LEON5
.	LE1	LE2	LE5	LE7	LE10	LE11
1	137.65	133.18	191.18	198.17	299.26	295.35
2	136.86	132.28	374.16	392.79	596.26	578.14
3	136.98	132.34	488.87	584.28	890.00	821.63
4	137.03	132.10	523.03	778.65	1183.10	998.22
5	137.08	132.17	389.20	495.32	746.25	687.23
6	137.09	132.29	466.13	593.66	895.62	815.24
7	137.12	132.23	507.00	691.81	1039.67	920.81
8	137.14	132.08	529.18	783.59	1190.48	1031.64
9	137.16	132.12	442.82	595.98	896.47	806.81
10	137.48	132.06	487.55	659.70	993.63	881.88
11	137.48	132.11	518.53	723.71	1087.85	957.56
12	137.48	132.07	531.00	781.79	1182.95	1010.96

Table 7.3: LEON scalability - CoreMark, integer version, iterations per second at 100MHz.

THREADS	AT697F	LEON2	LEON3	GR740	LEON5L2	LEON5
.	LE1	LE2	LE5	LE7	LE10	LE11
1	202.64	204.48	221.83	227.07	411.41	407.41
2	201.36	201.09	442.33	452.16	821.04	797.04
3	201.52	201.33	650.37	677.04	1229.08	1157.95
4	201.30	201.59	821.67	899.35	1635.31	1464.46
5	201.36	201.69	525.83	567.50	1026.52	956.13
6	201.40	201.99	622.68	679.04	1233.32	1141.51
7	201.41	201.88	723.73	791.14	1436.78	1303.84
8	201.25	203.85	818.94	900.82	1639.80	1429.38
9	201.30	204.00	620.02	681.16	1233.58	1116.31
10	201.32	203.98	678.56	755.85	1368.65	1244.74
11	201.36	203.97	740.90	830.81	1506.03	1361.30
12	201.36	203.94	805.42	902.16	1640.58	1444.95

The values in the tables are also shown in Figures 7.4 and 7.5. The figures show the *CoreMark* iterations per second at 100MHz. Higher values denote better processor performance.

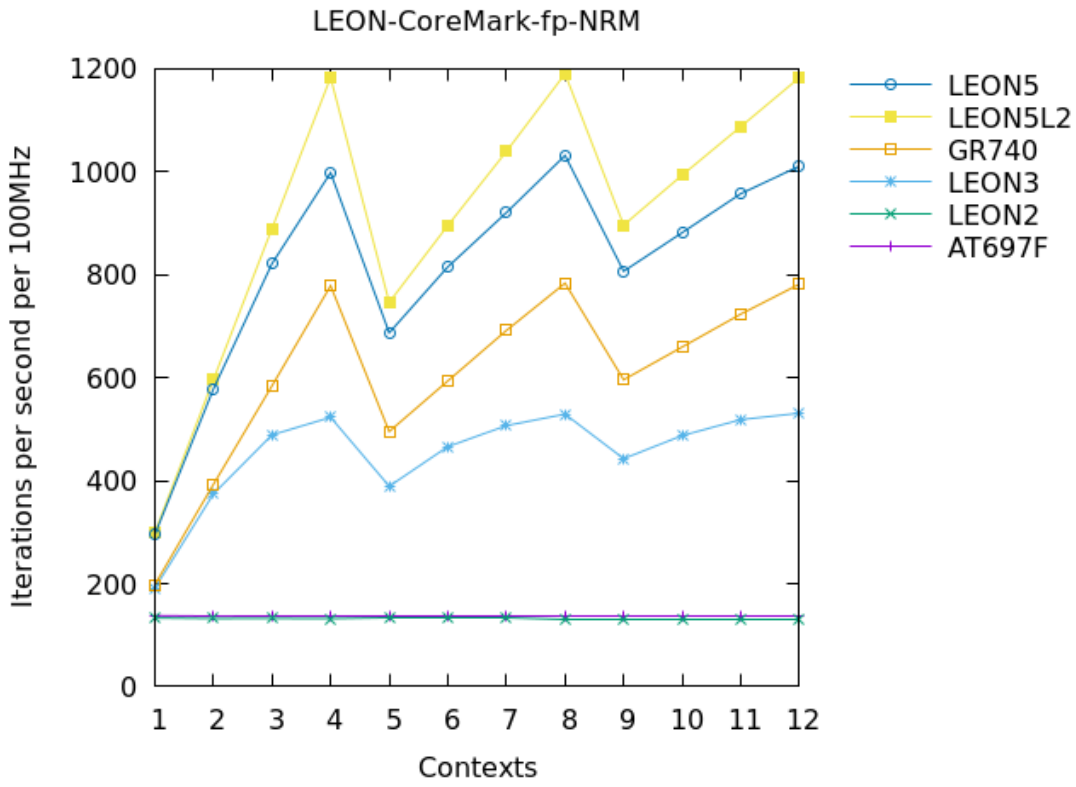

Figure 7.4: LEON - CoreMark - floating-point performance for LEON targets, CoreMark iterations at 100MHz. Higher values denote better performance.

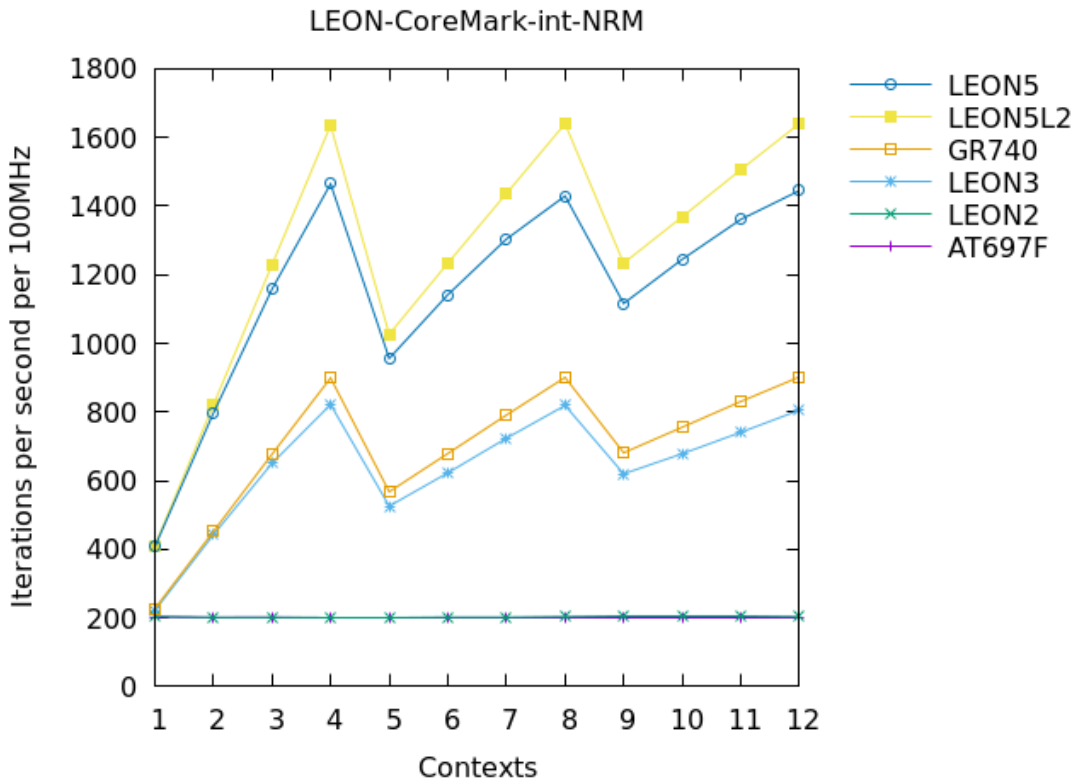

Figure 7.5: LEON - CoreMark - integer performance for LEON targets, CoreMark iterations at 100MHz. Higher values denote better performance.

7.2.3 Analysis

The integer version of the *CoreMark* benchmark shows an almost identical performance for the single-context version, and perfect scaling for LEON3, GR740 and the two LEON5 configurations. The zig-zag nature of the curve indicates that the benchmark is compute-bound (see also Section 6.2.3).

The floating-point version of the benchmark shows a superior performance of the systems with non-blocking FPUs, and nearly perfect scaling for GR740 and the two LEON5 configurations. The scaling for LEON3 shows a suboptimal performance for 3 and more contexts that may be caused by a saturation of the memory busses due to the missing level-2 cache.

The zero-scaling performance of AT697F and LEON2FT is caused by the fact that these are single-core systems.

The two LEON5 configurations achieved the best performance for both the floating-point and integer version of the CoreMark benchmark, with LEON5L2 achieving a significantly higher performance than GR740 (1.5x and 1.8x for the floating-point and integer version respectively).

7.3 CoreMark-Pro

For a definition of the workloads and specification of the measurements see Section 6.3.1.

7.3.1 Worst-case execution time

An approximate worst-case execution time for the *CoreMark-Pro* suite executed in LEON-based systems is shown in Table 7.4.

Table 7.4: LEON - worst-case execution times.

Configuration	LEON2	LEON3	GR740	LEON5
Frequency [MHz]	100	100	250	100
Execution time [min]	90	50	35	50

7.3.2 Compiler options

The following options were used for compiling the *CoreMark-Pro* benchmark suite for LEON:

```
-g
-O2
-qrtems
-funroll-all-loops
-funswitch-loops
-fgcse-after-reload
-fpredictive-commoning
-finline-functions
-fipa-cp-clone
-falign-functions=8
-falign-loops=8
-falign-jumps=8
--param max-inline-insns-auto=20
-B, -mcpu, -qbsp as per the selected system
```

7.3.3 Measurements

The measurements are presented in one set of tables that show workload iterations computed per second.

The *CoreMark-Pro* data are presented for AT697F, LEON2FT, LEON3, GR740, and three LEON5 configurations - 2-core with a level-2 cache, 4-core without a level-2 cache and 4-core with a level-2 cache.

Table 7.5 lists performance results when *CoreMark-Pro* was computed in the AT697 evaluation board equipped with AT697F mounted on an extension board. Sample read and write memory accesses in GRMON identified that the on-board memory beyond 28MB is not reliable; this is probably the reason why the workloads *loops* and *zip* failed to complete due to memory allocation errors even when the board was configured to use only the 64MB SDRAM memory mapped from address 0x40000000.

Table 7.5: LEON - CoreMark-Pro results for configuration LE1 (AT697F/Meiko) - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations/sec.	
Workload	-c1	-c2
cjpeg	1.61	1.61
sha	1.57	1.57
core	0.01	0.01
nnet	0.02	0.02
linear	0.48	0.48
parser	0.53	0.53
loops	FAILED	FAILED
radix2	2.15	2.15
zip	FAILED	FAILED

Table 7.6: LEON - CoreMark-Pro results for configuration LE2 (LEON2/daiFPU) - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations/sec.	
Workload	-c1	-c2
cjpeg	1.6407	1.6402
core	0.0135	0.0136
linear	0.4642	0.4650
loops	0.0151	0.0151
nnet	0.0232	0.0232
parser	0.2837	0.2838
radix2	1.9172	1.9173
sha	1.5423	1.5423
zip	0.7047	0.7052

Table 7.7: LEON - CoreMark-Pro results for configuration LE5 (GRLIB/LEON3) - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	1.9157	2.7503	2.6781	2.5786	2.4752	2.4564	2.4931	2.4728	2.4366	2.4528	2.4649	2.4378
core	0.0140	0.0280	0.0409	0.0441	0.0309	0.0369	0.0426	0.0431	0.0354	0.0392	0.0426	0.0429
linear	0.5936	0.8226	0.7458	0.7376	0.7318	0.7312	0.7357	0.7314	0.7314	0.7318	0.7319	0.7318
loops	0.0237	0.0322	0.0326	0.0327	0.0327	0.0327	0.0326	0.0327	0.0327	0.0327	0.0327	0.0327
nnet	0.0377	0.0458	0.0469	0.0488	0.0488	0.0489	0.0489	0.0488	0.0489	0.0488	0.0493	0.0498
parser	0.2194	0.2236	0.1805	0.1687	0.1383	0.1406	0.1374	0.1395	0.1363	0.1381	0.1352	0.1357
radix2	2.2789	2.7670	2.7750	2.7253	2.7219	2.7212	2.7222	2.7232	2.7215	2.7226	2.7222	2.7219
sha	1.5555	3.0628	3.7922	4.9727	4.9727	4.9702	4.9702	4.9702	4.9677	4.9677	5.4348	5.8423
zip	0.7955	0.8925	0.8378	0.8288	0.8217	0.8447	0.8433	0.8288	0.8239	0.8409	0.8374	0.8332

Table 7.8: LEON - CoreMark-Pro results for configuration LE7 (GR740) - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	2.1448	4.2061	6.1069	8.0972	7.6775	8.0972	8.0808	7.6336	7.6190	7.9840	7.9365	7.5472
core	0.0144	0.0287	0.0430	0.0573	0.0358	0.0430	0.0501	0.0575	0.0430	0.0477	0.0525	0.0573
linear	0.8437	1.6694	2.4411	3.1706	3.1676	3.1802	3.1995	3.1995	3.2446	3.2092	3.2452	3.2103
loops	0.0374	0.0685	0.0815	0.0812	0.0816	0.0817	0.0821	0.0816	0.0817	0.0825	0.0821	0.0816
nnet	0.0479	0.0935	0.1180	0.1544	0.1551	0.1550	0.1544	0.1547	0.1545	0.1546	0.1710	0.1847
parser	0.4796	0.7797	0.6993	0.6788	0.5739	0.5690	0.5585	0.5521	0.5325	0.5130	0.4966	0.4761
radix2	4.5834	8.6736	10.2559	7.5793	7.5654	7.5562	7.5100	7.5146	7.5379	7.5143	7.4706	7.5399
sha	1.6502	3.3113	4.1195	5.4945	5.4795	5.4870	5.4870	5.5021	5.4870	5.4795	6.0357	6.6025
zip	1.1396	2.0305	2.7714	3.3755	2.4038	2.7907	3.1042	3.3827	2.7692	3.0120	3.2070	3.3708

Table 7.9: LEON - CoreMark-Pro results for configuration LE10 (GRLIB/LEON5 w/ L2Cache) - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	2.8662	5.6402	8.1766	10.7181	10.1833	10.5263	10.6045	9.8619	9.9701	10.2987	10.4058	9.8232
core	0.0258	0.0514	0.0773	0.1030	0.0645	0.0773	0.0901	0.1030	0.0773	0.0858	0.0944	0.1027
linear	0.8052	1.5966	2.3475	3.0401	3.0652	3.1293	3.0639	3.0505	3.0473	3.0503	3.1376	3.0519
loops	0.0360	0.0638	0.0812	0.0898	0.0897	0.0898	0.0902	0.0897	0.0897	0.0896	0.0901	0.0895
nnet	0.0492	0.0980	0.1224	0.1626	0.1627	0.1628	0.1626	0.1627	0.1627	0.1627	0.1790	0.1950
parser	0.5417	0.8107	0.7413	0.6282	0.5052	0.4662	0.4400	0.4254	0.4084	0.4002	0.3919	0.3821
radix2	4.0103	5.1139	5.7918	5.9454	5.9649	5.9561	5.9209	5.9604	5.9133	5.9360	5.9095	5.9319
sha	2.8612	5.7078	7.1124	9.4967	9.4967	9.4877	9.4877	9.4967	9.4877	9.4877	10.4167	11.3529
zip	1.6051	2.9283	3.5377	3.9024	3.0960	3.4803	3.7493	3.8760	3.3040	3.6724	3.7288	3.8511

Table 7.10: LEON - CoreMark-Pro results for configuration LE11 (GRLIB/LEON5) - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	2.7255	4.8356	5.7013	5.9988	5.4083	5.5371	5.6433	5.4318	5.3735	5.4437	5.5835	5.3937
core	0.0256	0.0504	0.0732	0.0839	0.0585	0.0697	0.0801	0.0863	0.0677	0.0748	0.0816	0.0876
linear	0.6971	1.3287	1.6340	1.6557	1.6446	1.6593	1.6561	1.6414	1.6652	1.6606	1.6548	1.6538
loops	0.0253	0.0363	0.0406	0.0418	0.0418	0.0418	0.0419	0.0418	0.0418	0.0418	0.0418	0.0418
nnet	0.0449	0.0759	0.0918	0.1053	0.1053	0.1053	0.1054	0.1054	0.1054	0.1053	0.1115	0.1145
parser	0.3368	0.3617	0.2875	0.2585	0.2144	0.2166	0.2119	0.2138	0.2105	0.2123	0.2098	0.2097
radix2	3.0524	4.4679	4.5051	4.5097	4.4796	4.5047	4.4994	4.5064	4.5037	4.4966	4.4985	4.4829
sha	2.8257	5.6148	6.9735	9.2421	9.2336	9.1241	9.2166	9.2336	9.2336	9.2166	10.1010	10.9290
zip	1.2626	1.5420	1.5385	1.5296	1.4754	1.5436	1.5405	1.5340	1.4985	1.5387	1.5376	1.5355

Table 7.11: LEON - CoreMark-Pro results for configuration LE12 (GRLIB/LEON5 - 2 cores w/ L2Cache) - workload iterations per second at 100MHz. *-cx* denotes the number of parallel execution contexts.

.	Iterations per second											
Workload	-c1	-c2	-c3	-c4	-c5	-c6	-c7	-c8	-c9	-c10	-c11	-c12
cjpeg	2.8645	5.6243	5.5556	5.4585	5.5006	5.4437	5.4705	5.4171	5.4555	5.3821	5.4230	5.3879
core	0.0257	0.0514	0.0387	0.0515	0.0429	0.0515	0.0450	0.0515	0.0464	0.0515	0.0472	0.0515
linear	0.8046	1.5954	1.5739	1.5820	1.5930	1.5862	1.5994	1.5859	1.5791	1.5934	1.5901	1.5902
loops	0.0360	0.0640	0.0643	0.0636	0.0640	0.0636	0.0638	0.0637	0.0637	0.0636	0.0637	0.0637
nnet	0.0491	0.0979	0.0979	0.0979	0.0979	0.0979	0.0979	0.0979	0.0979	0.0979	0.0899	0.0979
parser	0.5423	0.7965	0.5466	0.4920	0.4717	0.4444	0.4201	0.3928	0.3933	0.3766	0.3754	0.3742
radix2	4.0175	5.0762	5.0661	5.0725	5.0527	5.0680	5.0719	5.0829	5.0602	5.0672	5.0644	5.0916
sha	2.8563	5.7143	5.7143	5.7110	5.7110	5.7110	5.7110	5.7110	5.7110	5.7078	5.2356	5.7088
zip	1.6077	2.9283	2.3548	2.9283	2.5733	2.9513	2.6545	2.9412	2.6874	2.9129	2.7528	2.9119

The values shown in Tables 7.6 to 7.11 are plotted in Figures 7.6 to 7.14.

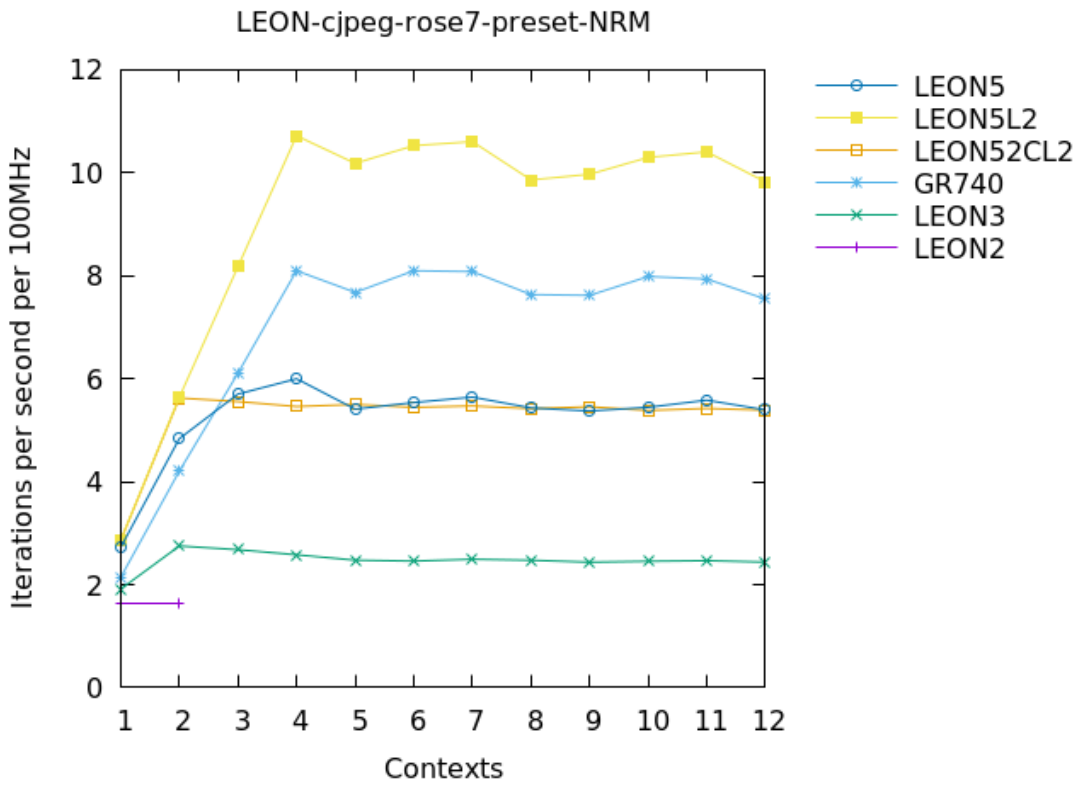

Figure 7.6: LEON - CoreMark-Pro - cjpeg, iterations per second at 100MHz. Higher values denote better performance.

7.3.4 Analysis

Observations based on Figures 7.6 to 7.14 are as follows:

GR740 and LEON5

- show the best performance and scalability across all the workloads.
- achieve near-perfect scaling in *core* and *zip* (the compute-bound zig-zag curve) in the configurations with a level-2 cache.
- indicate a potential for further performance scaling in *nnet* and *sha* (the curve rises, stagnates and then rises again).
- the performance reaches its ceiling in *cjpeg*, *linear*, *loops* (the memory-bound curve rises and gets flat).
- the performance indicates negative performance scaling in *parser* and *radix2* (the curve rises and then decreases, probably due to small cache size).

LEON5 w/ level-2 cache

- using a level-2 cache improves the LEON5 performance by up to 2x in the memory-bound workloads - *cjpeg*, *linear*, *loops*, *nnet*, *parser*, *radix2*, *zip*. For the *zip* workload the level-2 cache changes the workload characteristics from memory-bound to compute-bound.

LEON3

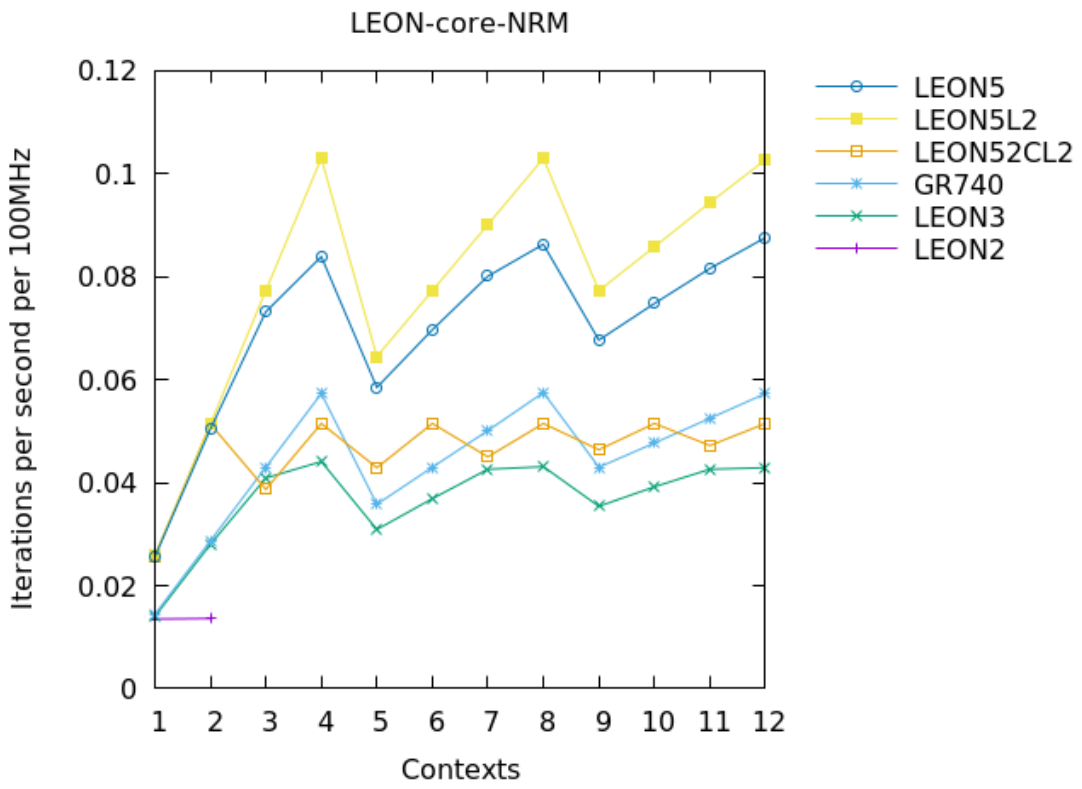

Figure 7.7: LEON - CoreMark-Pro - core, iterations per second at 100MHz. Higher values denote better performance.

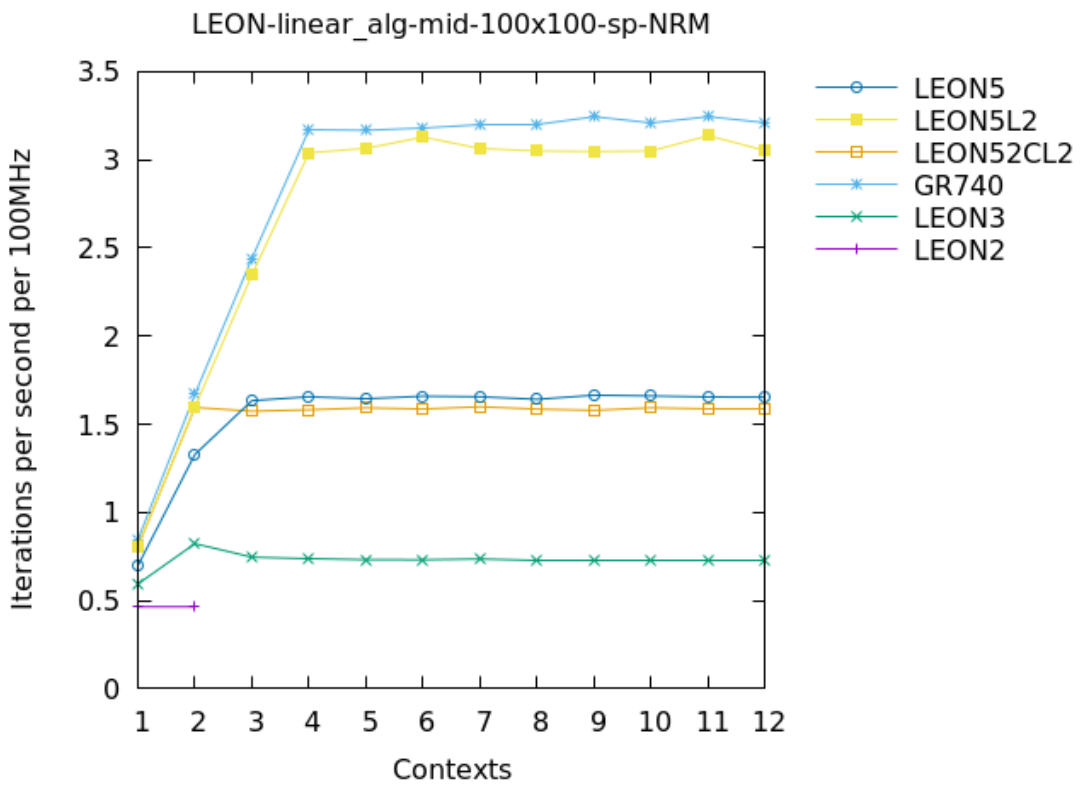

Figure 7.8: LEON - CoreMark-Pro - linear, iterations per second at 100MHz. Higher values denote better performance.

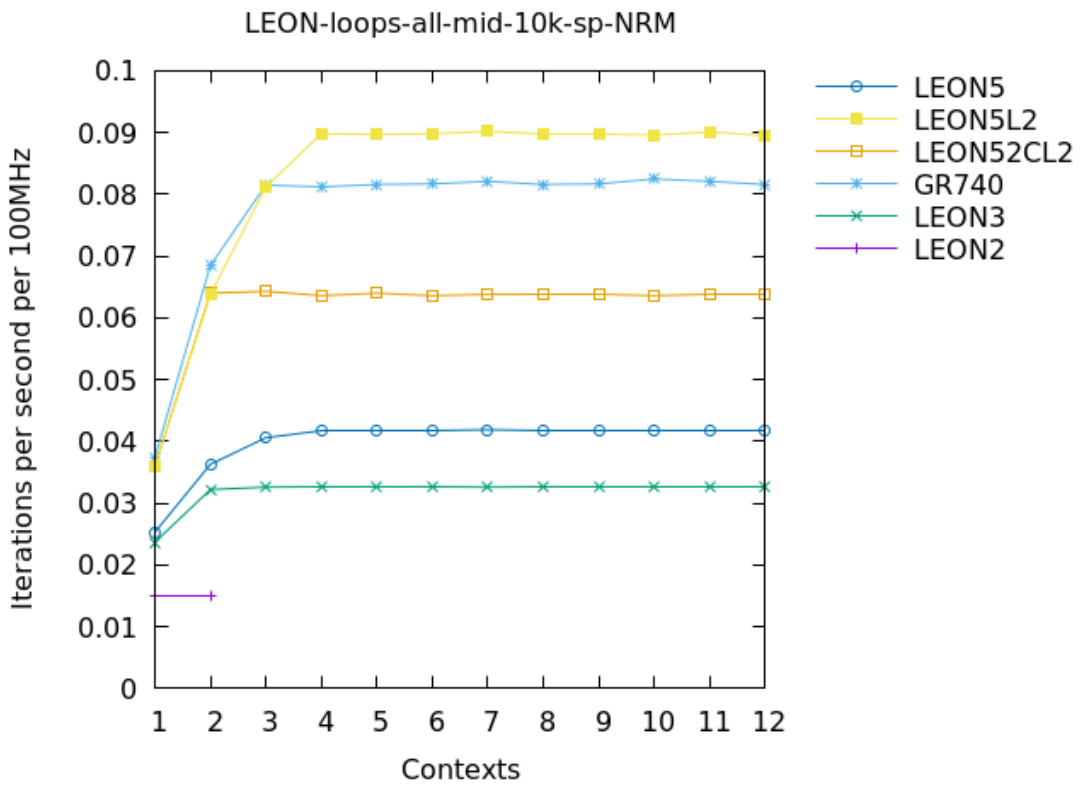

Figure 7.9: LEON - CoreMark-Pro - loops, iterations per second at 100MHz. Higher values denote better performance.

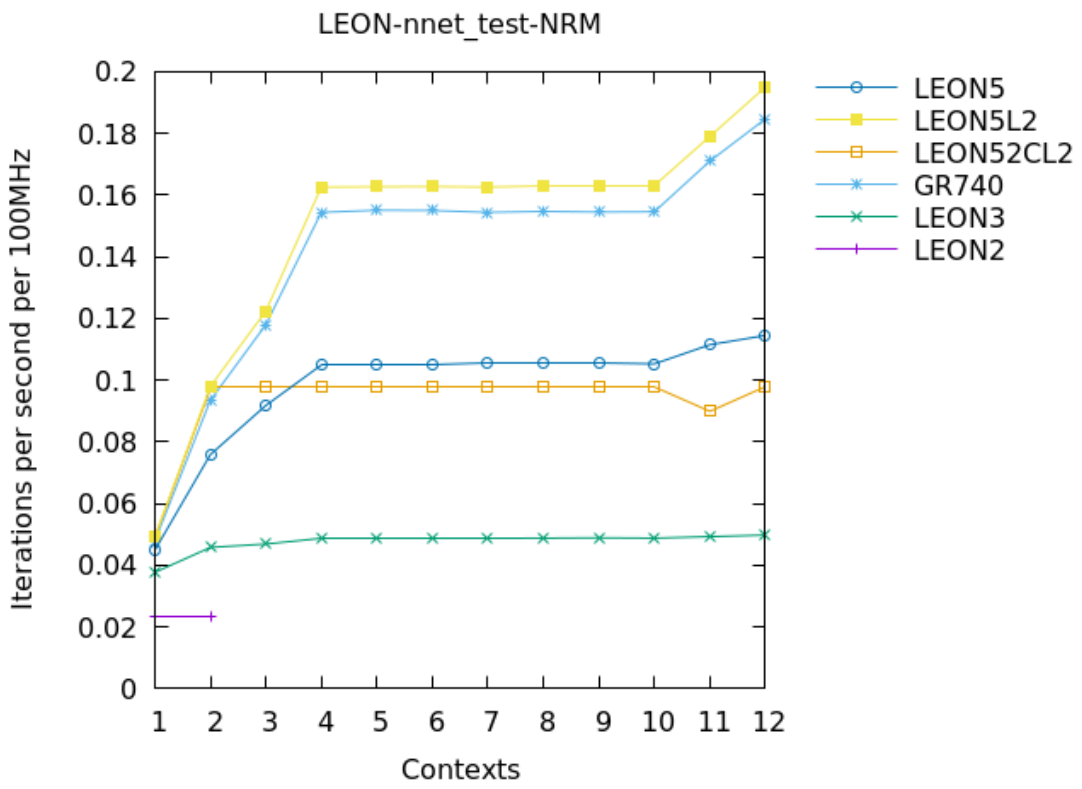

Figure 7.10: LEON - CoreMark-Pro - nnet, iterations per second at 100MHz. Higher values denote better performance.

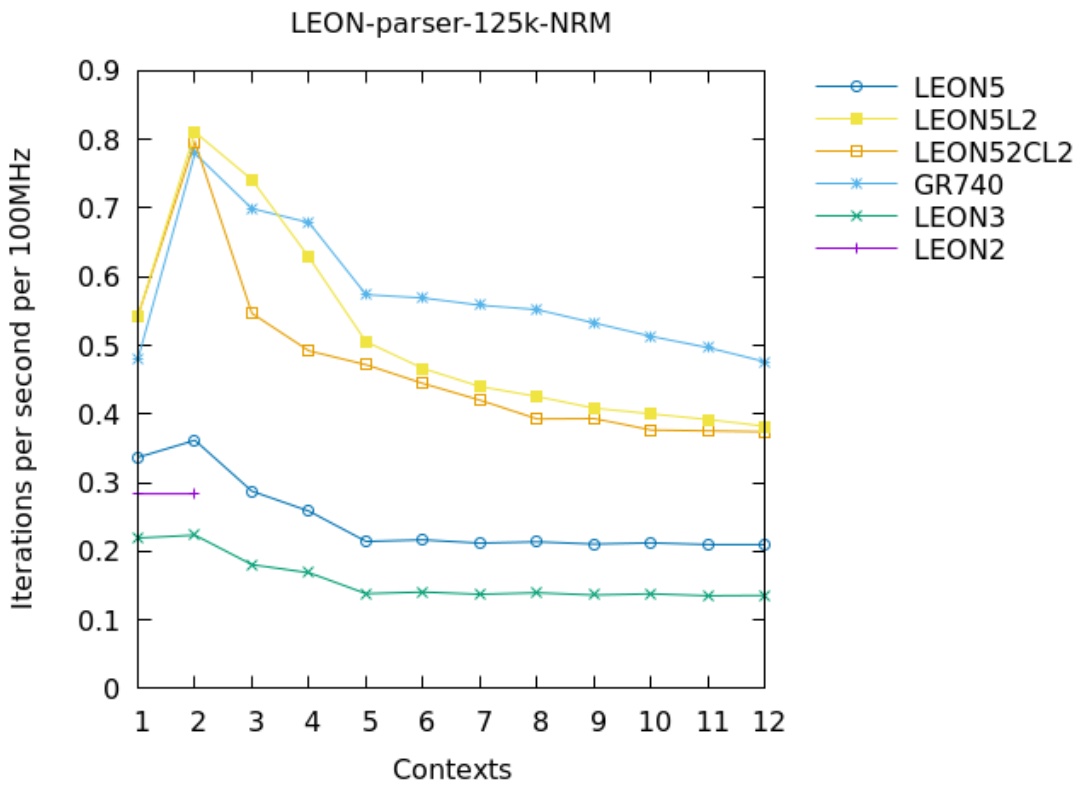

Figure 7.11: LEON - CoreMark-Pro - parser, iterations per second at 100MHz. Higher values denote better performance.

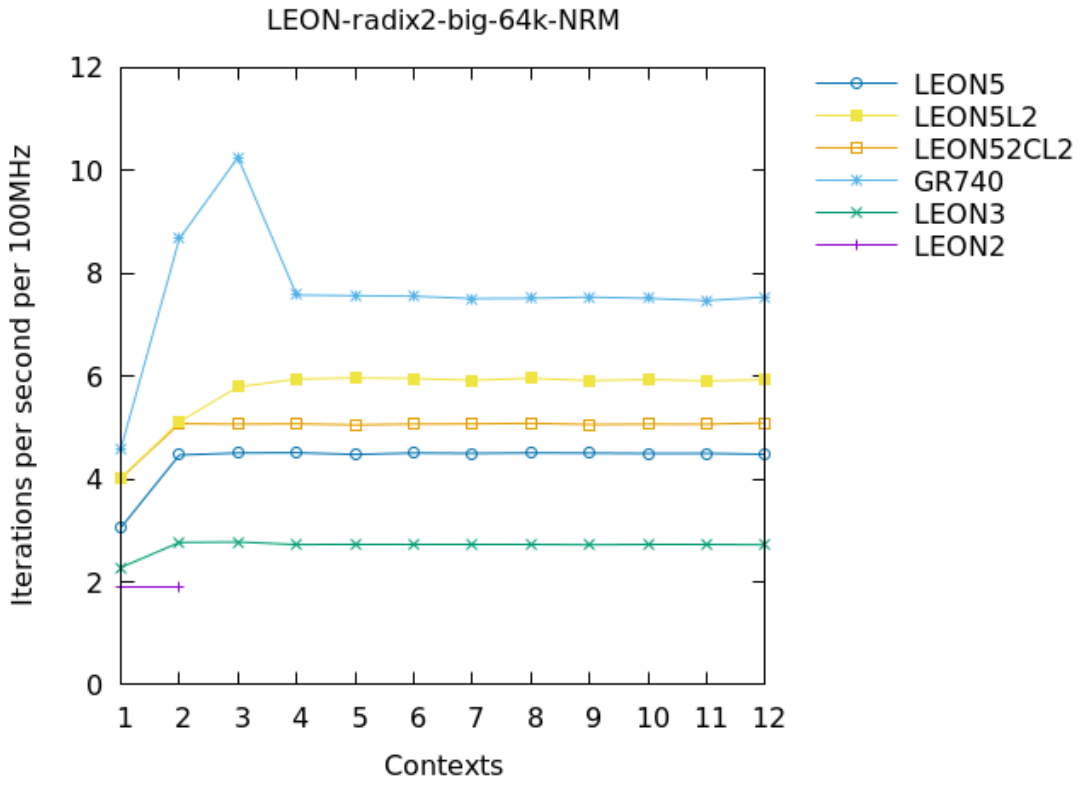

Figure 7.12: LEON - CoreMark-Pro - radix2, iterations per second at 100MHz. Higher values denote better performance.

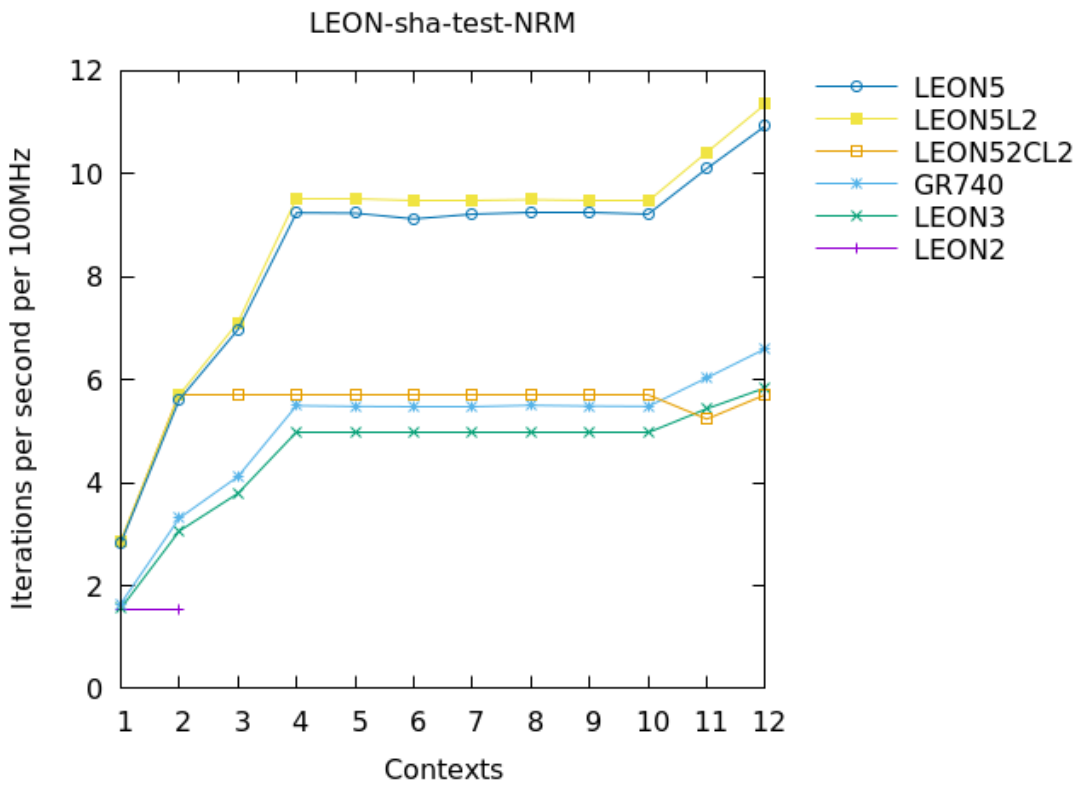

Figure 7.13: LEON - CoreMark-Pro - sha, iterations per second at 100MHz. Higher values denote better performance.

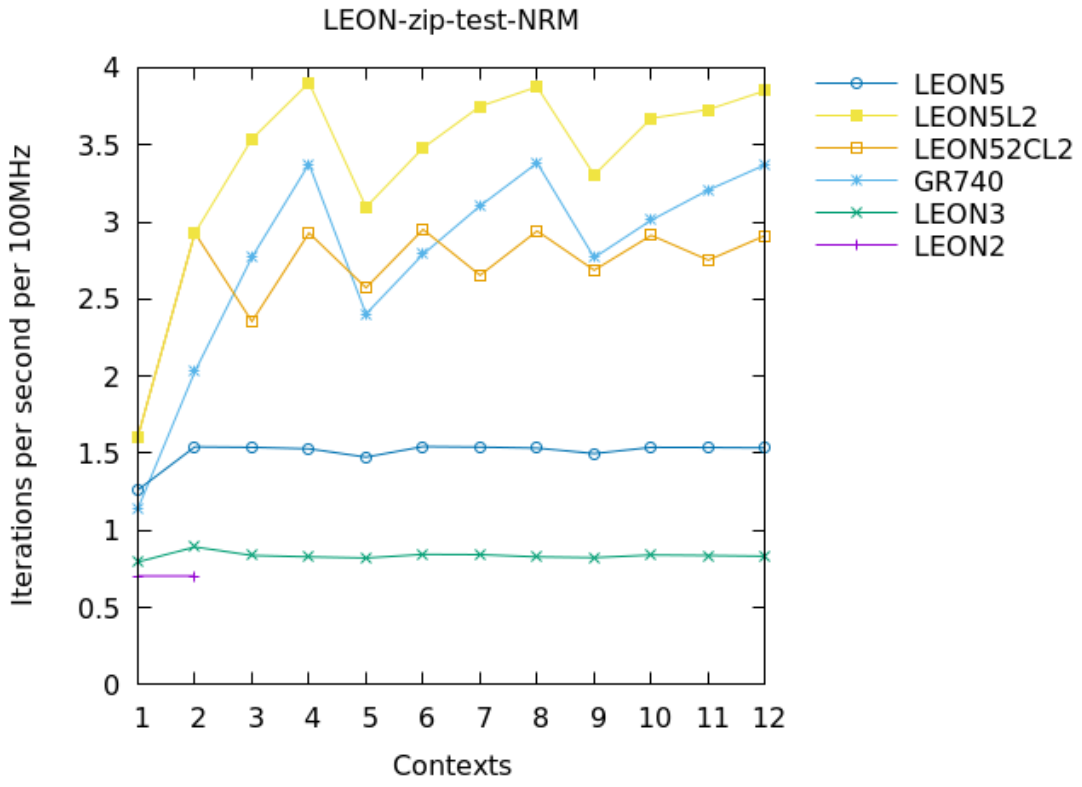

Figure 7.14: LEON - CoreMark-Pro - zip, iterations per second at 100MHz. Higher values denote better performance.

- achieves the fifth[15] best performance for all workloads but for *parser-125k*.
- shows a compute-bound performance nearly as optimal as GR740 and LEON5 in *core* and *sha*.
- in general achieves a significantly worse performance than a 2-core LEON5 system with a level-2 cache.
- shows a memory-bound performance with moderate results in *cjpeg*, *linear*, *loops* and *nnet*.
- shows a suboptimal multi-core performance in *parser*, *radix2* and *zip*, probably due to small cache size.

Single-core LEON2

- outperforms multi-core LEON3 in *parser*.

[15] i.e. after LEON5L2, LEON52CL2, LEON5 and GR740

8 NOEL vs. LEON

This section compares the performance of the NOEL-V configurations and the five LEON-based systems. The performance is analysed using two types of plots:

1. plots with absolute performance normalized for 100MHz execution to assess the absolute computing performance, and
2. plots with relative performance to assess performance scaling among the different configurations.

8.1 PWLS benchmarks

Figures 8.1 to 8.3 demonstrate the floating-point performance in the NOEL-V and LEON-based systems.

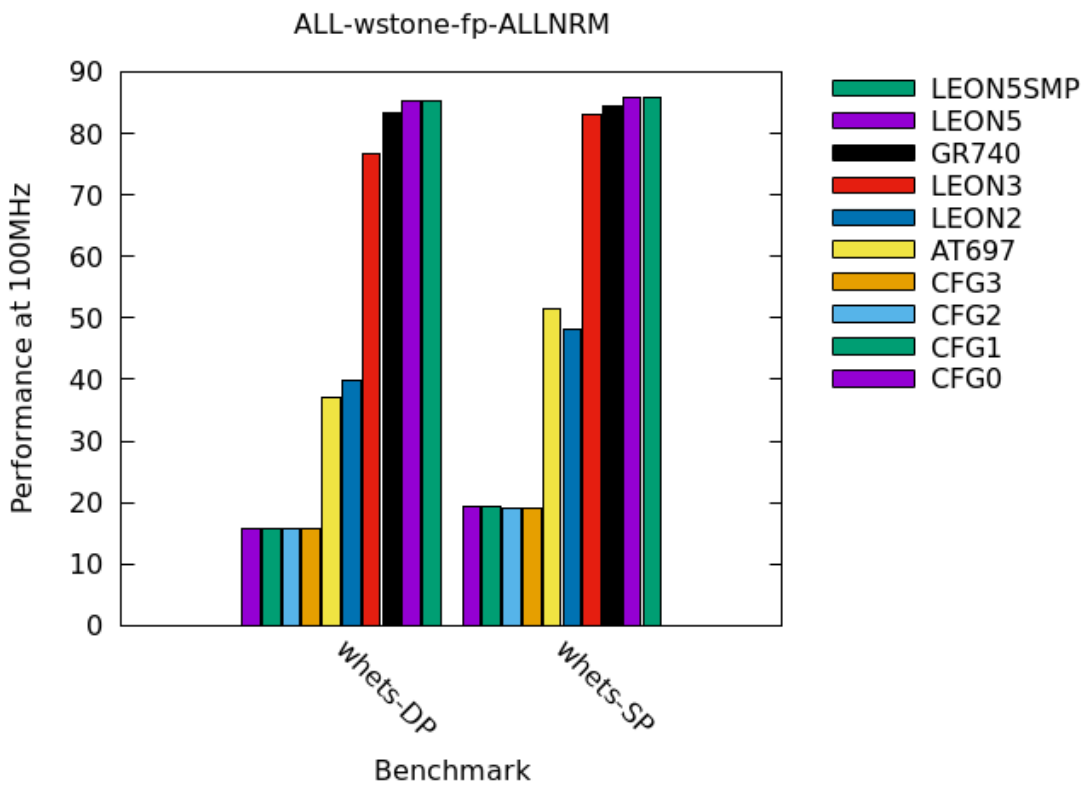

Figure 8.1: ALL - Whetstone performance for ALL targets, MWIPS at 100MHz. Higher values denote better performance.

Figures 8.1 and 8.2 clearly show an inferior floating-point performance of all NOEL-V configurations compared to any of the LEON-based systems.

Fig. 8.3 shows a superior performance of GR740 and LEON5 with *GRFPU5* compared to any of the NOEL-V configurations. In the integer part of the *Stanford* benchmark NOEL-V configurations *CFG0* and *CFG1* achieve a better performance than *LEON2*, *LEON3* and *GR740*, while *LEON5* scores better than *CFG0* and *CFG1*. This is attributed to wider instruction and data buses in NOEL-V (64bits), up to 4x the memory

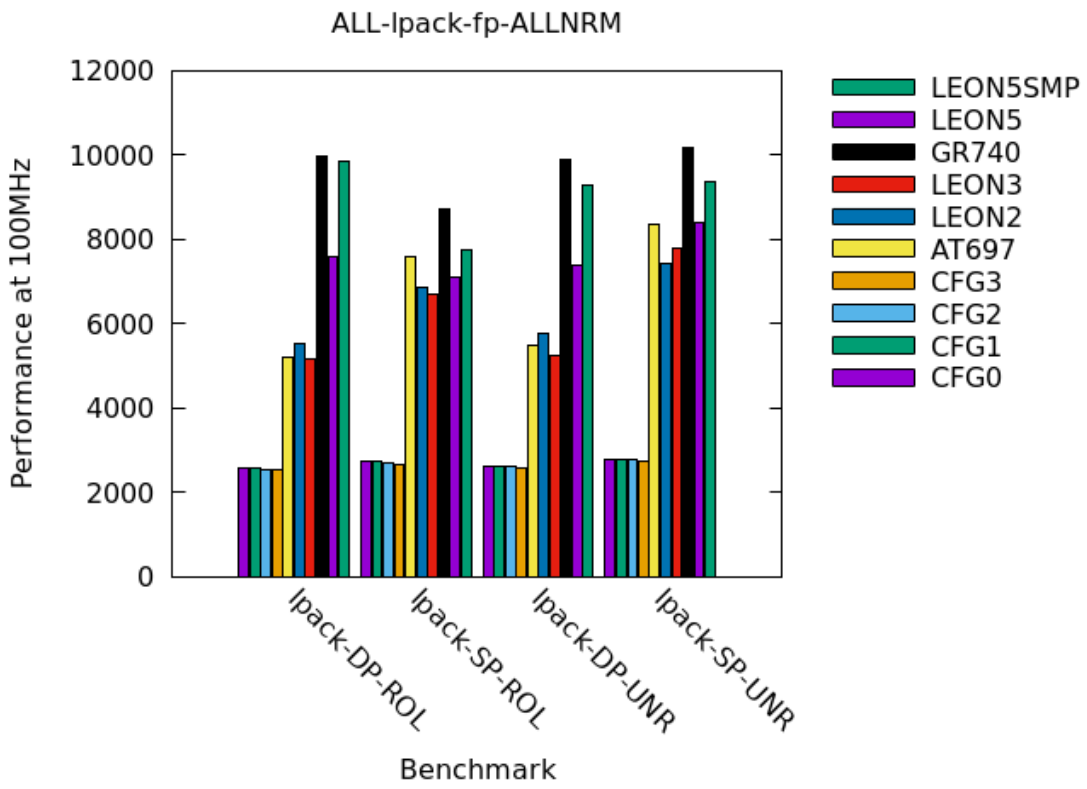

Figure 8.2: ALL - Linpack performance for ALL targets, Kflops at 100MHz. Higher values denote better performance.

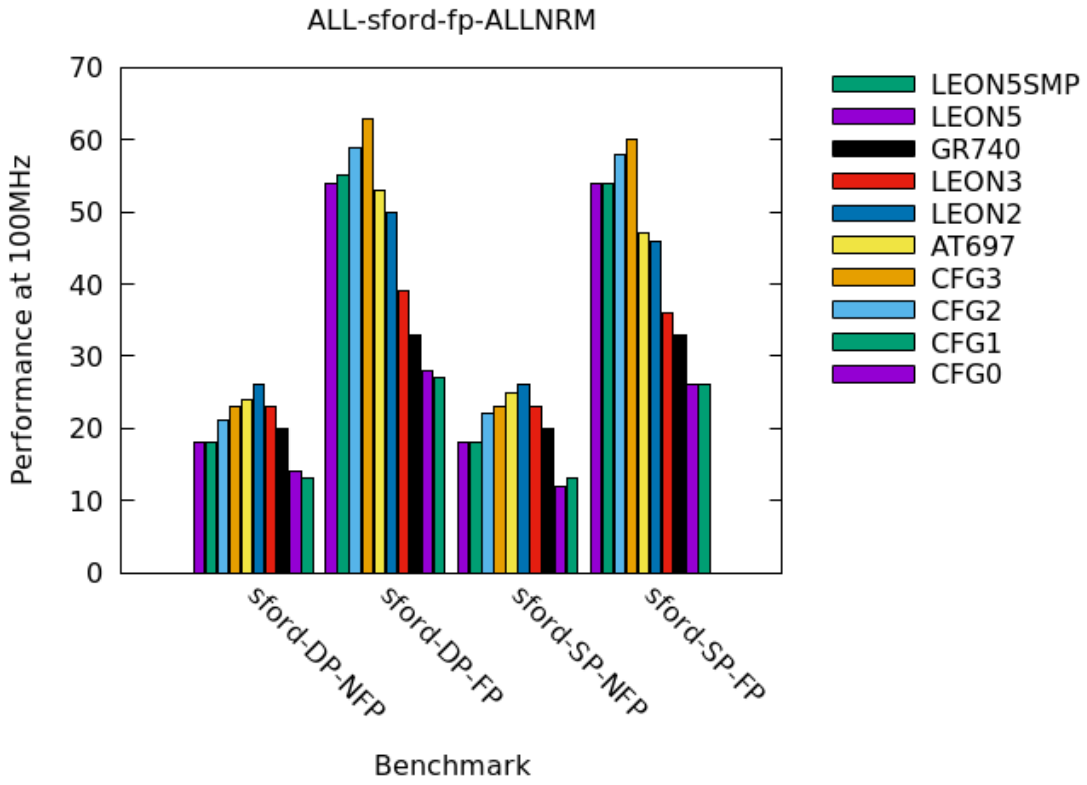

Figure 8.3: ALL - Stanford performance for ALL targets, milliseconds at 100MHz. Lower values denote better performance.

bandwidth in NOEL-V compared to the LEON-based systems (NOEL-V:128 vs LEON5:64 vs LEON3:32 data bits in on-chip AMBA buses), and to branch target prediction logic implemented both in NOEL-V and in LEON5, and a level-2 cache in *LEON5SMP* that is missing in NOEL-V.

8.2 CoreMark

Figures 8.4 and 8.5 show absolute performance scaling in the NOEL-V and LEON-based systems for both the floating-point and integer version of the *CoreMark* benchmark.

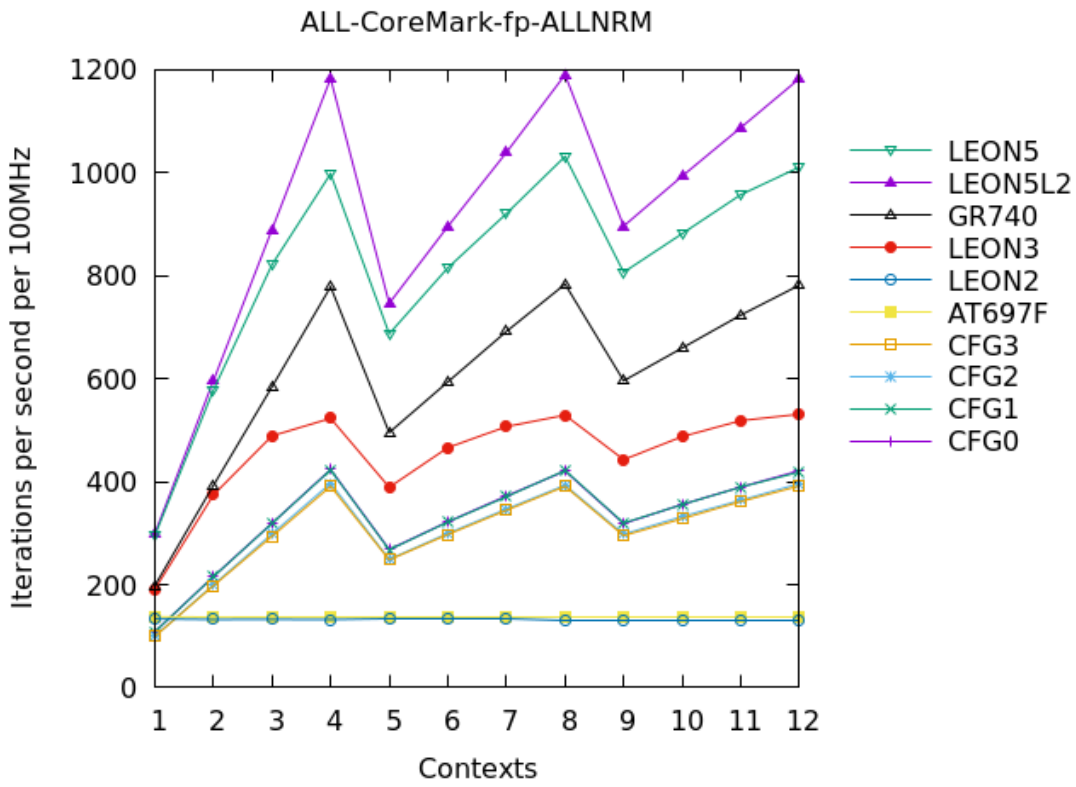

Figure 8.4: ALL - CoreMark - floating-point performance for all targets, CoreMark iterations at 100MHz. Higher values denote better performance.

Fig. 8.4 shows that

- LEON5 achieves 2.8x the floating-point performance of the NOEL-V *CFG0*,
- GR740 achieves 2x the floating-point performance of the NOEL-V *CFG0*, and
- LEON3 with *GRFPU* about 1.25x the performance of NOEL-V *CFG0*.

Fig. 8.5 shows that NOEL-V achieves a higher integer performance than any of the LEON-based system except *LEON5L2* which is about 1.11x better. Overall, the integer performance of NOEL-V configurations *CFG0* and *CFG1* are slightly worse than *LEON5L2* and slightly better than *LEON5*. The performance of NOEL-V *CFG0* is about 1.8x the performance of GR740. The performance of LEON3 is slightly lower than GR740. An interesting observation is that the performance of a 4-core NOEL-V system in *CFG6* (no caches) is just slightly higher (about 1.2x) than the performance of a single-core LEON2/AT697F.

For a more detailed analysis of the *CoreMark* performance for NOEL-V and LEON see Sections 6.2 and 7.2 respectively.

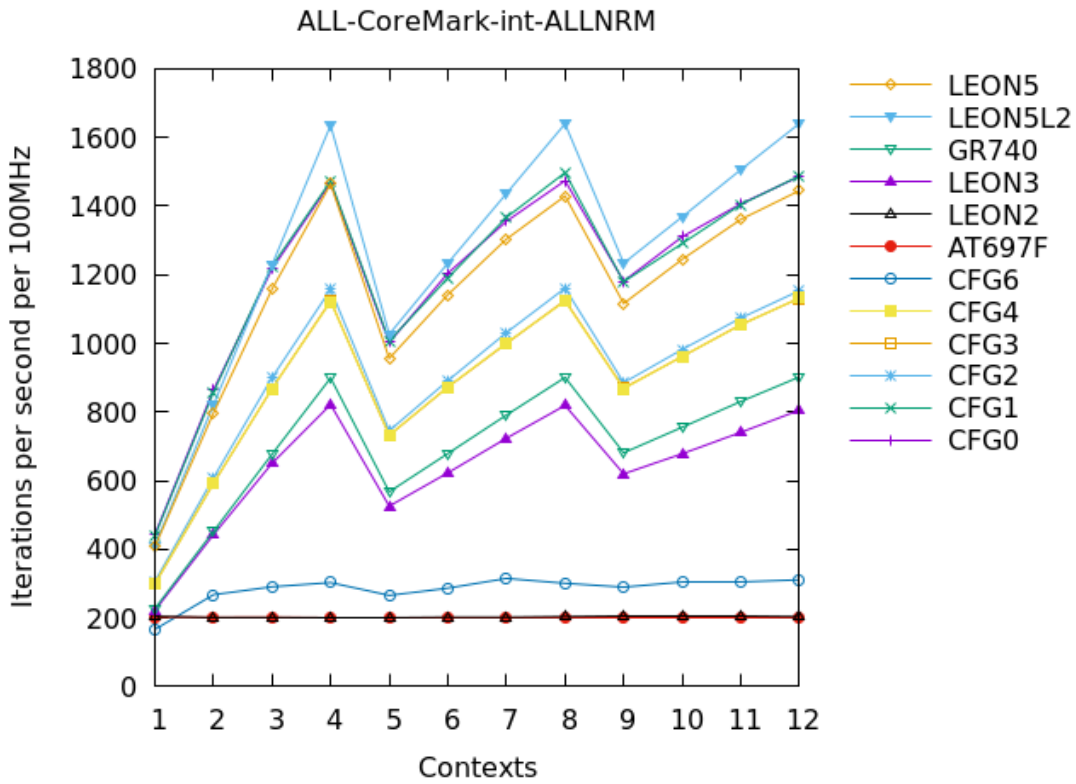

Figure 8.5: ALL - CoreMark - integer performance for all targets, CoreMark iterations at 100MHz. Higher values denote better performance.

8.3 CoreMark-Pro

8.3.1 Performance plots

Figures 8.6 to 8.14 show absolute performance scaling in the NOEL-V and LEON-based systems.

LEON5L2 achieves a performance that is superior to any of the NOEL-V configurations. Overall, it achieves the best performance in all workloads except *linear*, *parser* and *radix2* where *GR740* is the best.

GR740 achieves a performance superior to any of the NOEL-V configurations in all workloads but *core*.

Surprisingly *LEON3* with *GRFPU* achieves a better performance than any of the NOEL-V configurations in a number of workloads - *linear*, *loops*, *nnet*, and up to two contexts in *radix2* and *sha*.

8.3.2 Scaling plots

Figures 8.15 to 8.23 highlight performance scaling in the individual NOEL-V and LEON-based systems. For each benchmark the performance reference (*1.0* on the y-axis) was determined as the maximum performance achieved by the configuration across all evaluated contexts.

LEON3 is memory-bound in *cjpeg*, *core*, *linear*, *loops*. *radix2* and *zip*.

LEON52CL2 exhibits an interesting drop in performance for 11 contexts in *nnet* and *sha*.

The NOEL-V configurations with *nanoFPU* exhibit a similar performance pattern in *linear* and *nnet* like

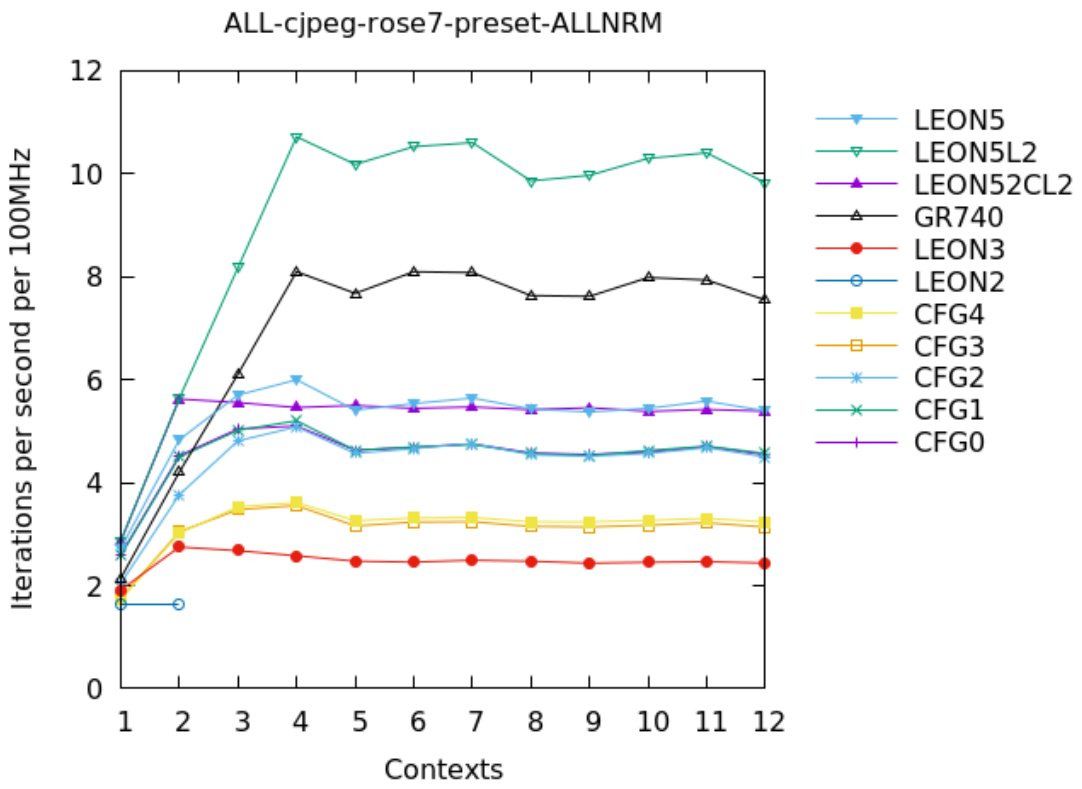

Figure 8.6: ALL - CoreMark-Pro - cjpeg, iterations per second at 100MHz. Higher values denote better performance.

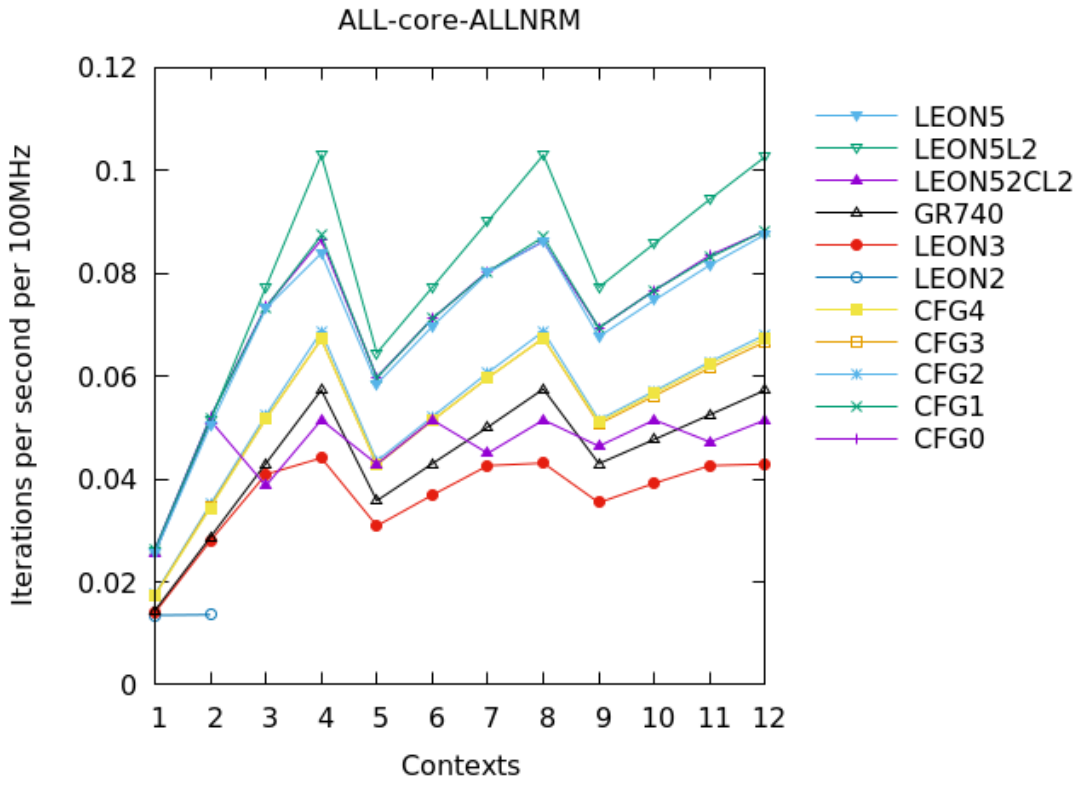

Figure 8.7: ALL - CoreMark-Pro - core, iterations per second at 100MHz. Higher values denote better performance.

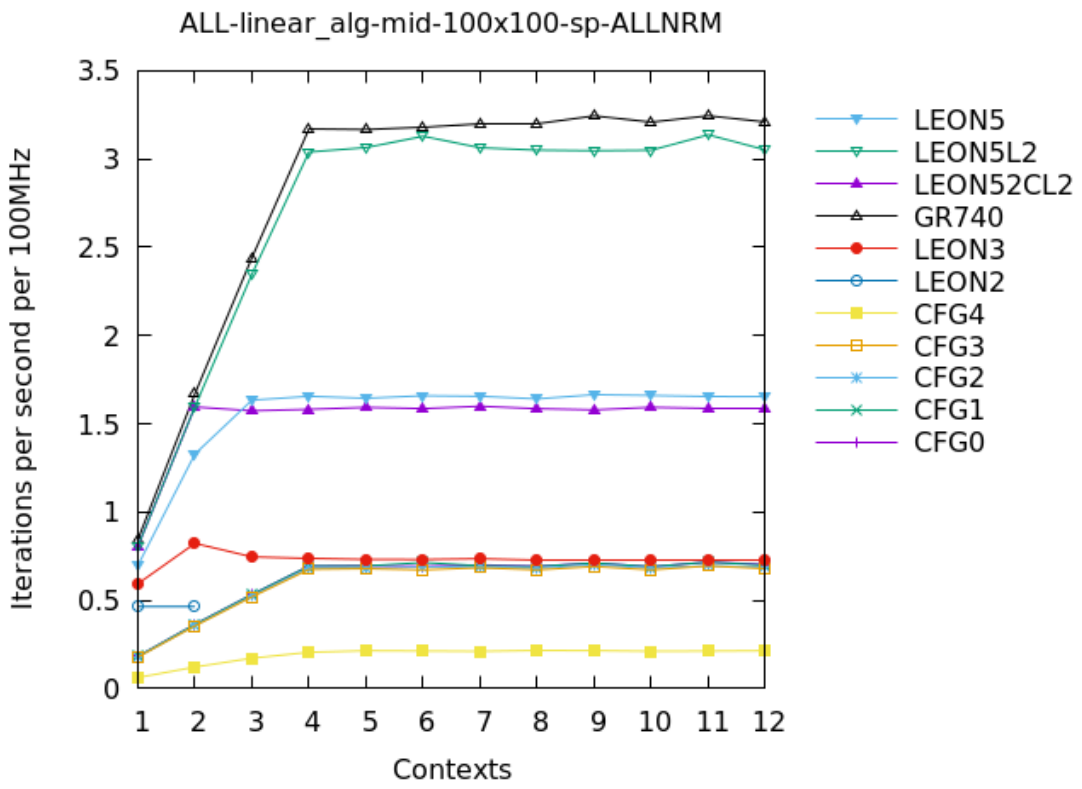

Figure 8.8: ALL - CoreMark-Pro - linear, iterations per second at 100MHz. Higher values denote better performance.

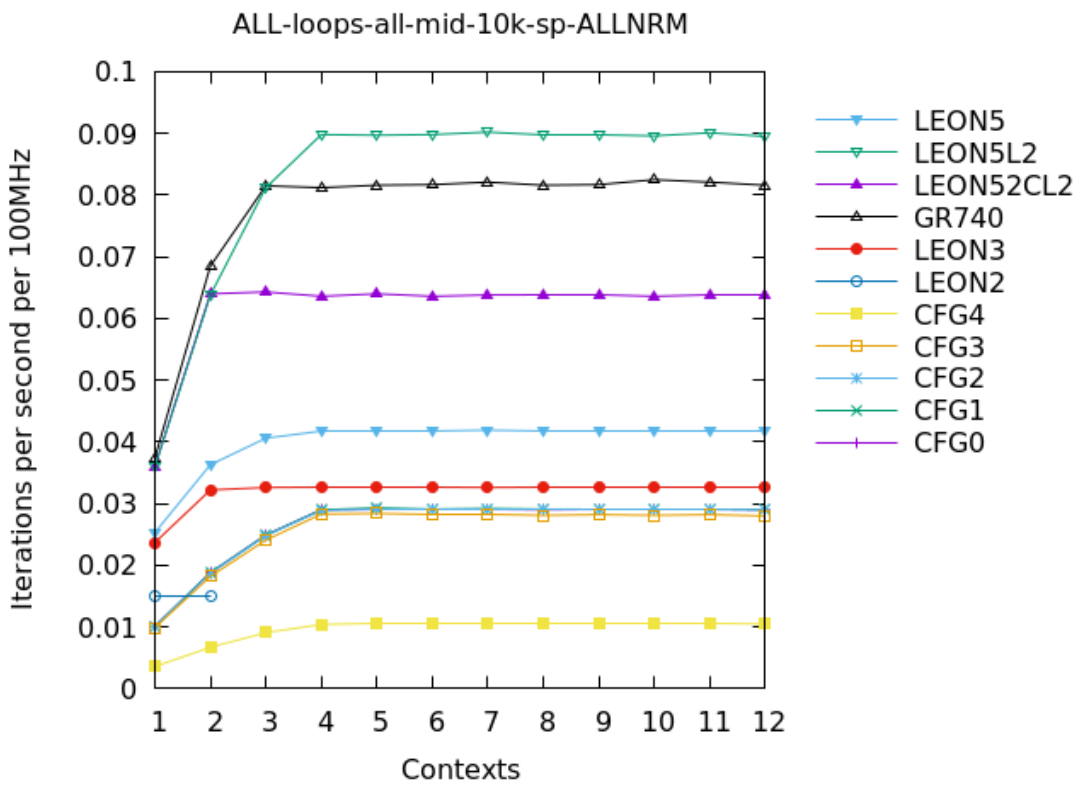

Figure 8.9: ALL - CoreMark-Pro - loops, iterations per second at 100MHz. Higher values denote better performance.

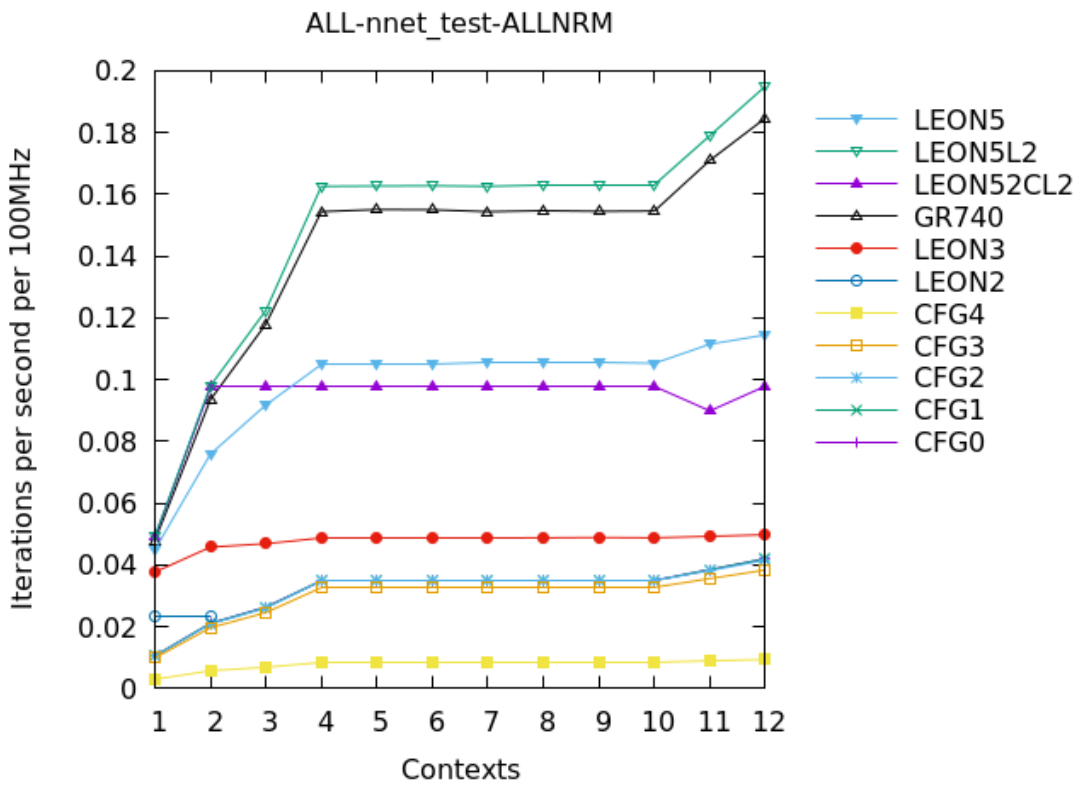

Figure 8.10: ALL - CoreMark-Pro - nnet, iterations per second at 100MHz. Higher values denote better performance.

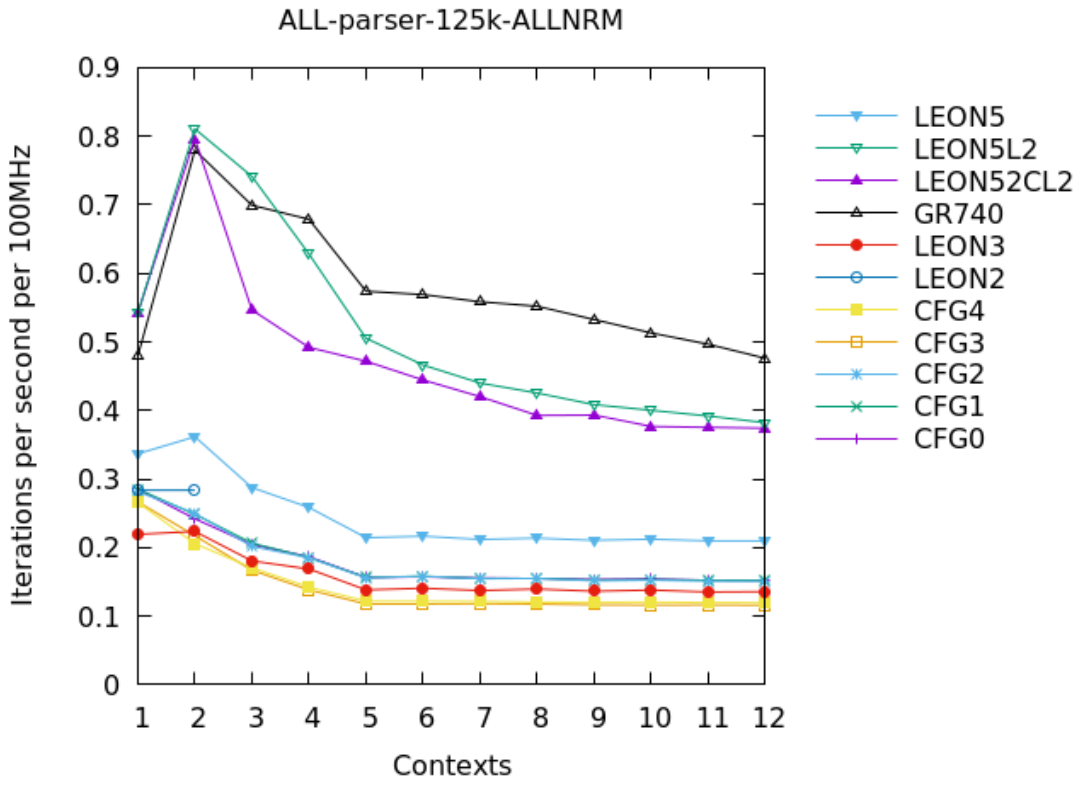

Figure 8.11: ALL - CoreMark-Pro - parser, iterations per second at 100MHz. Higher values denote better performance.

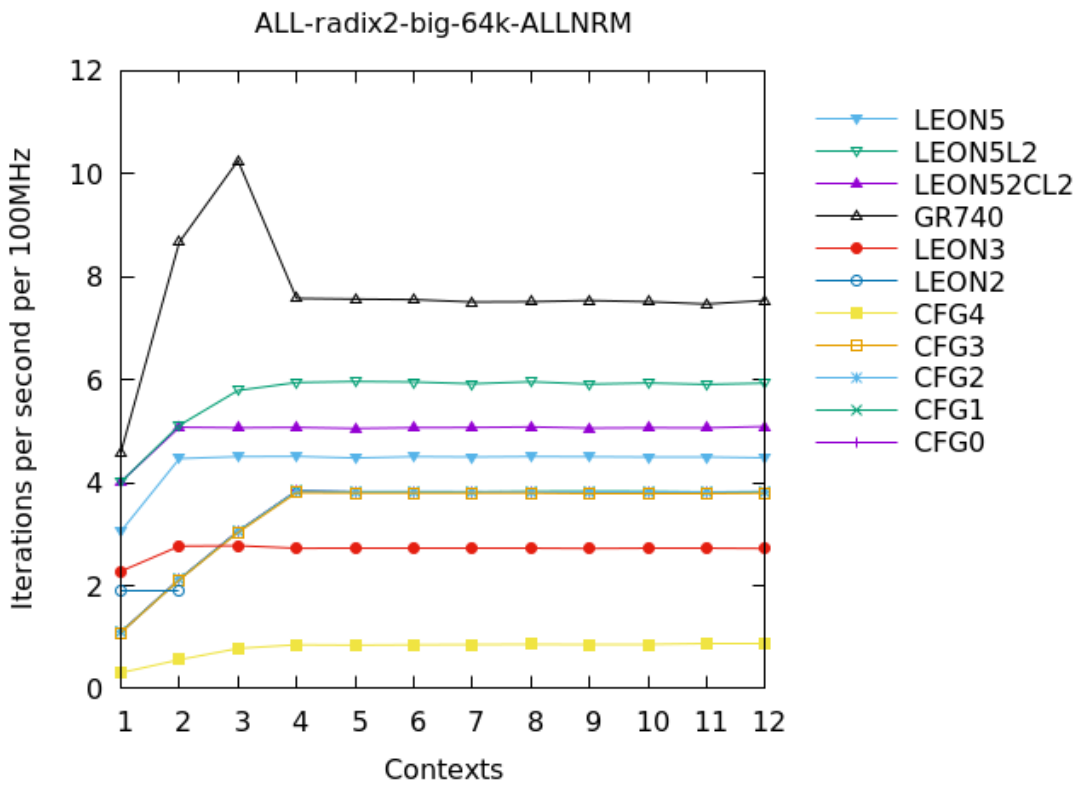

Figure 8.12: ALL - CoreMark-Pro - radix2, iterations per second at 100MHz. Higher values denote better performance.

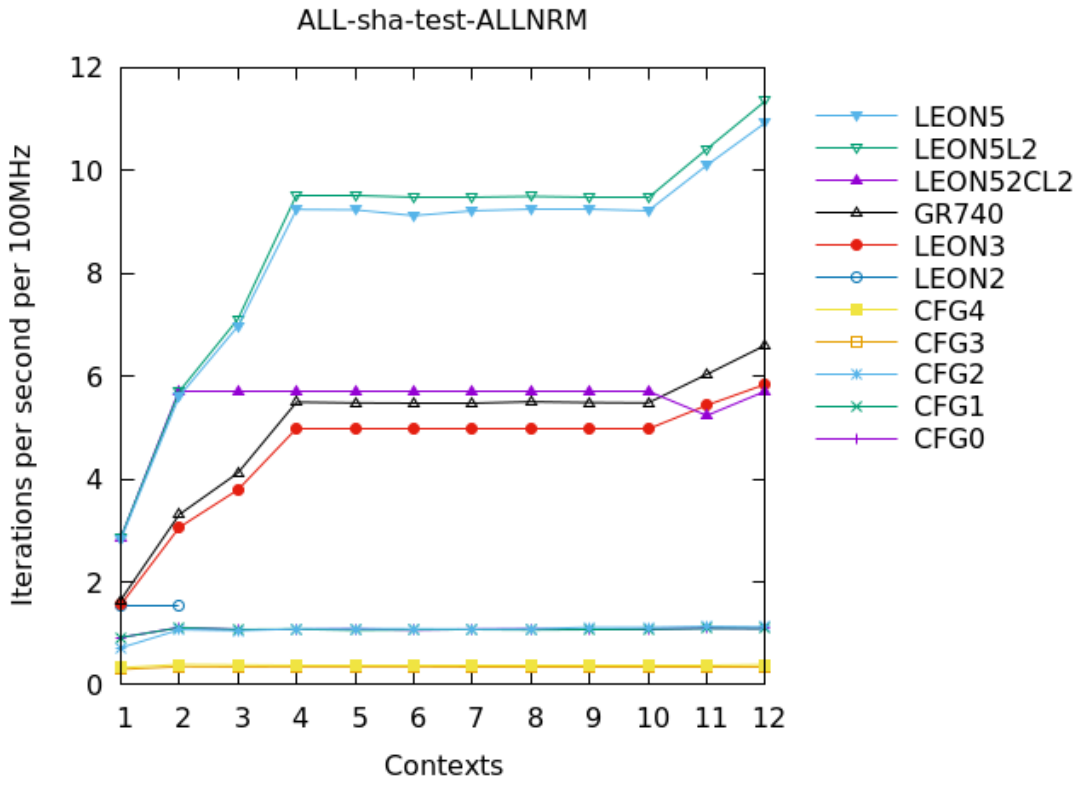

Figure 8.13: ALL - CoreMark-Pro - sha, iterations per second at 100MHz. Higher values denote better performance.

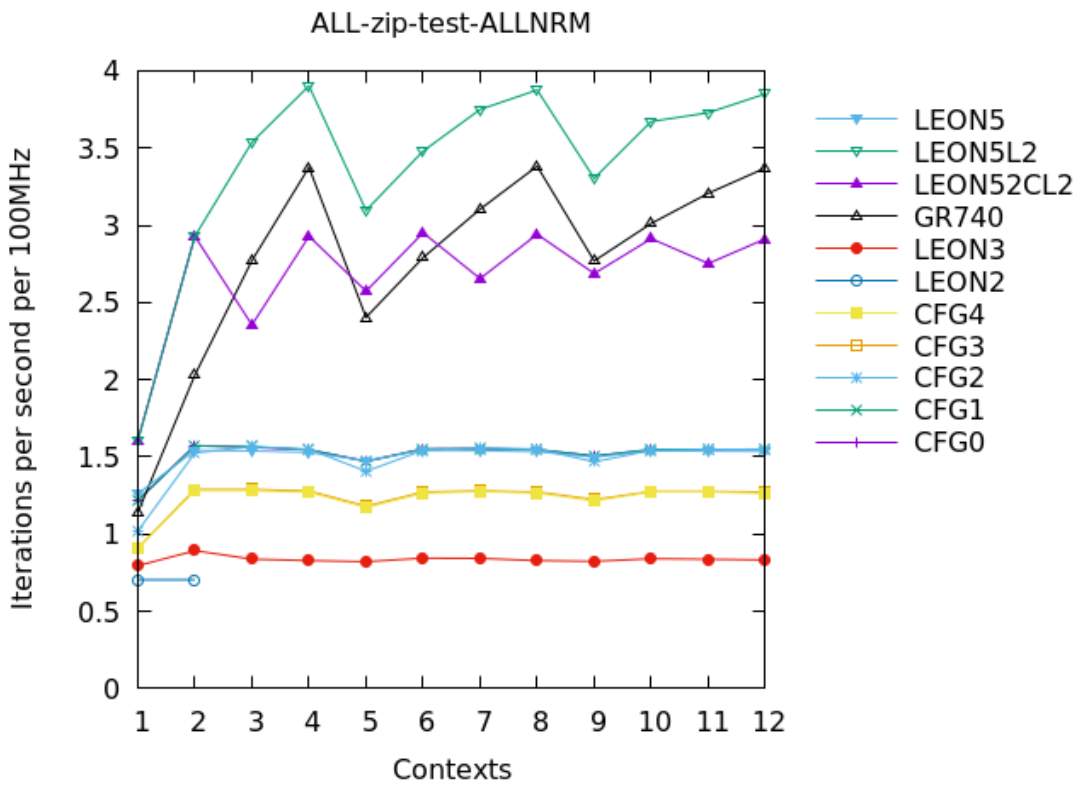

Figure 8.14: ALL - CoreMark-Pro - zip, iterations per second at 100MHz. Higher values denote better performance.

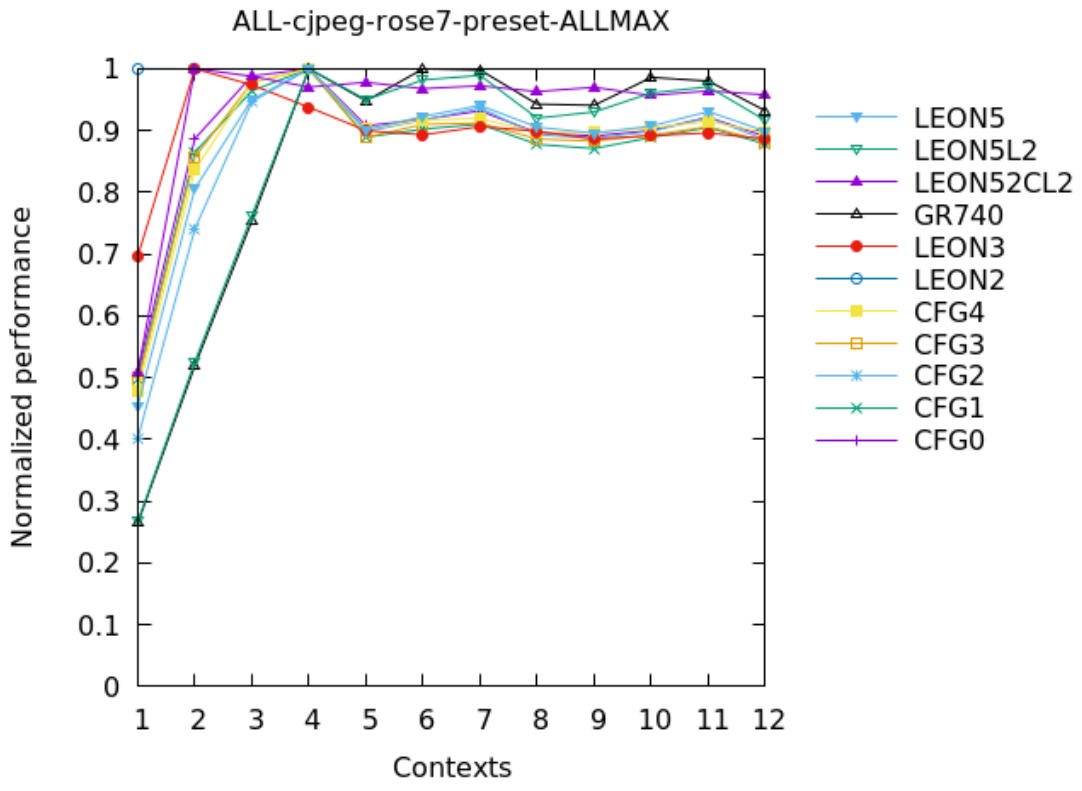

Figure 8.15: ALL - CoreMark-Pro - cjpeg, relative performance.

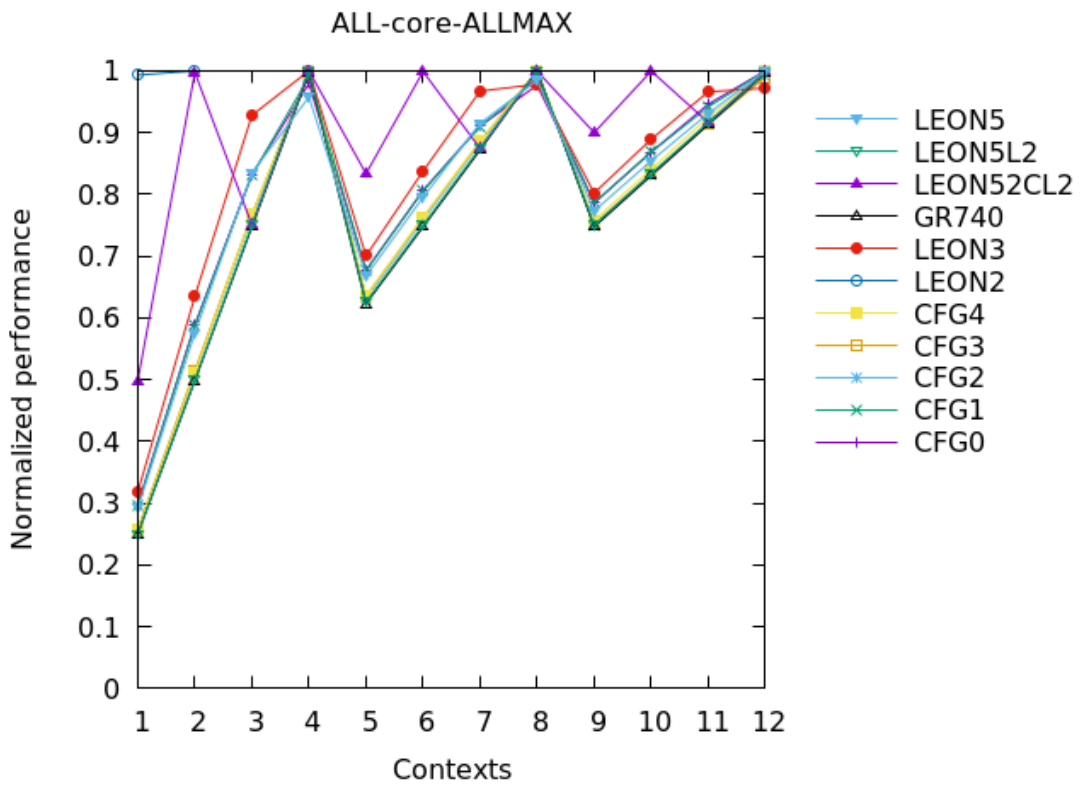

Figure 8.16: ALL - CoreMark-Pro - core, relative performance.

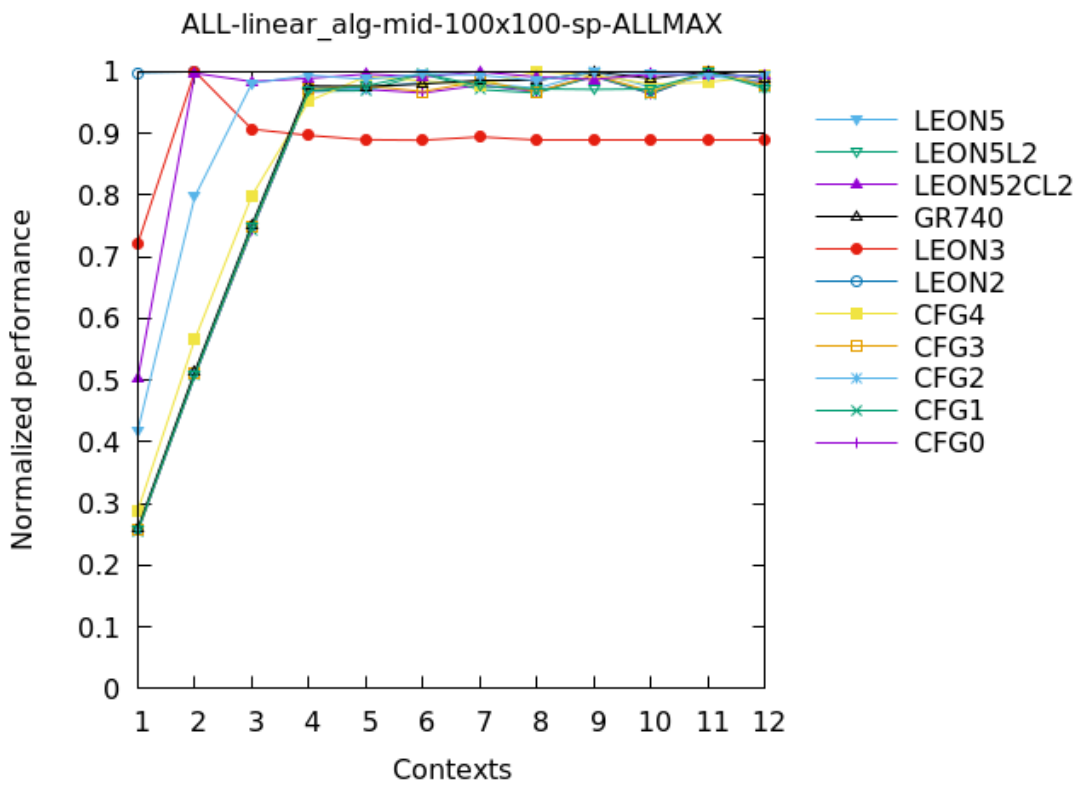

Figure 8.17: ALL - CoreMark-Pro - linear, relative performance.

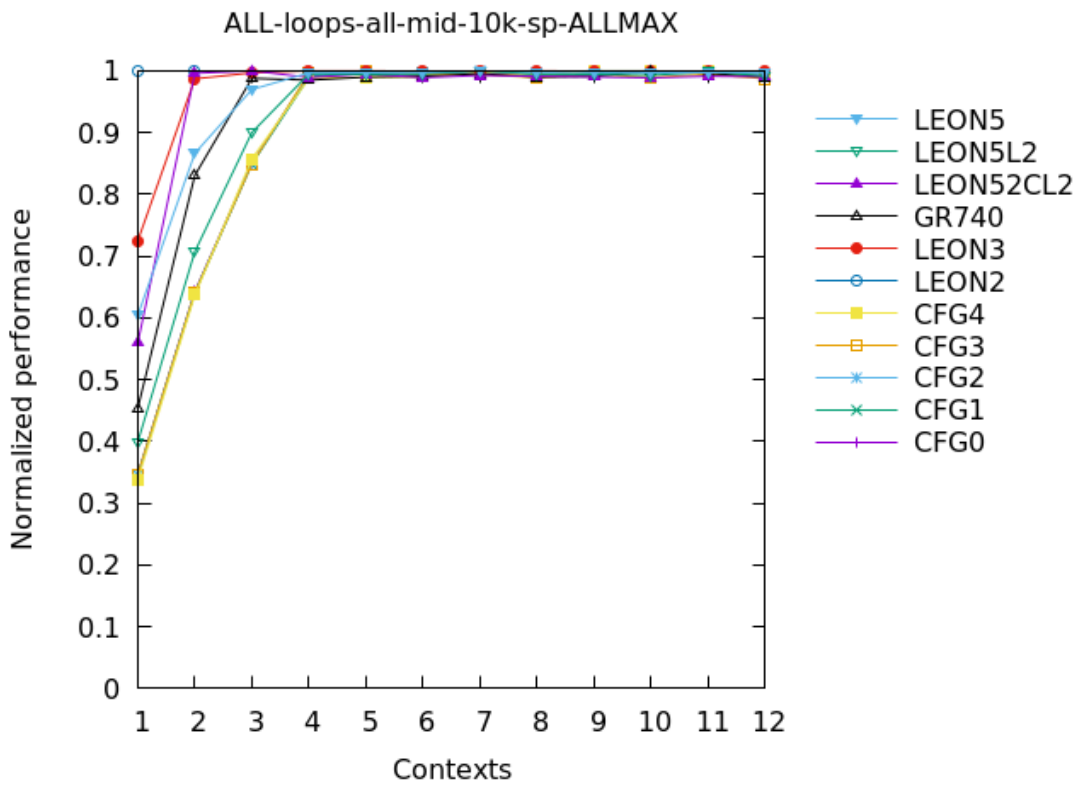

Figure 8.18: ALL - CoreMark-Pro - loops, relative performance.

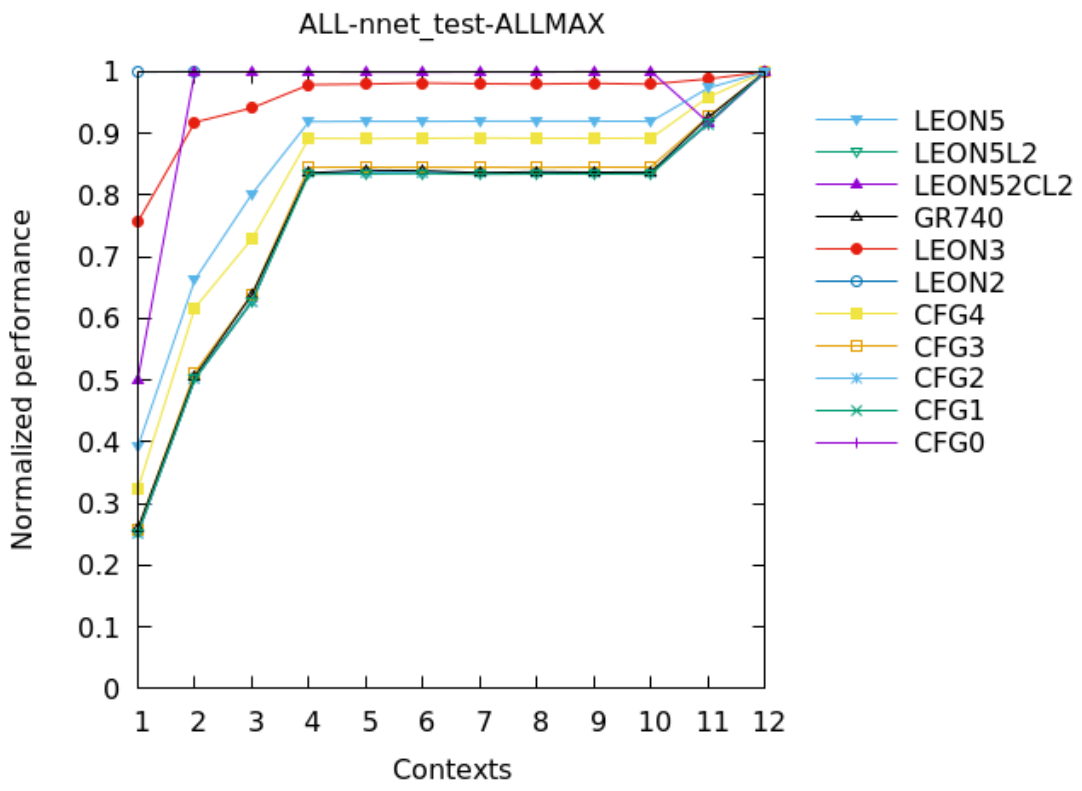

Figure 8.19: ALL - CoreMark-Pro - nnet, relative performance.

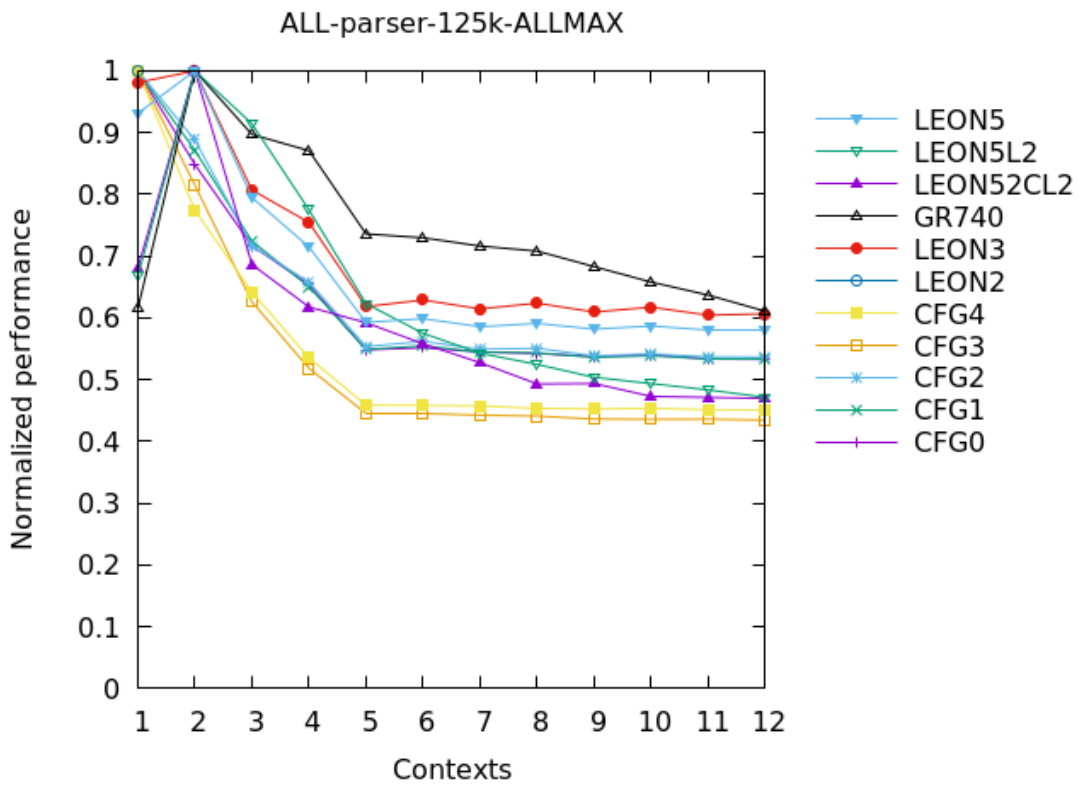

Figure 8.20: ALL - CoreMark-Pro - parser, relative performance.

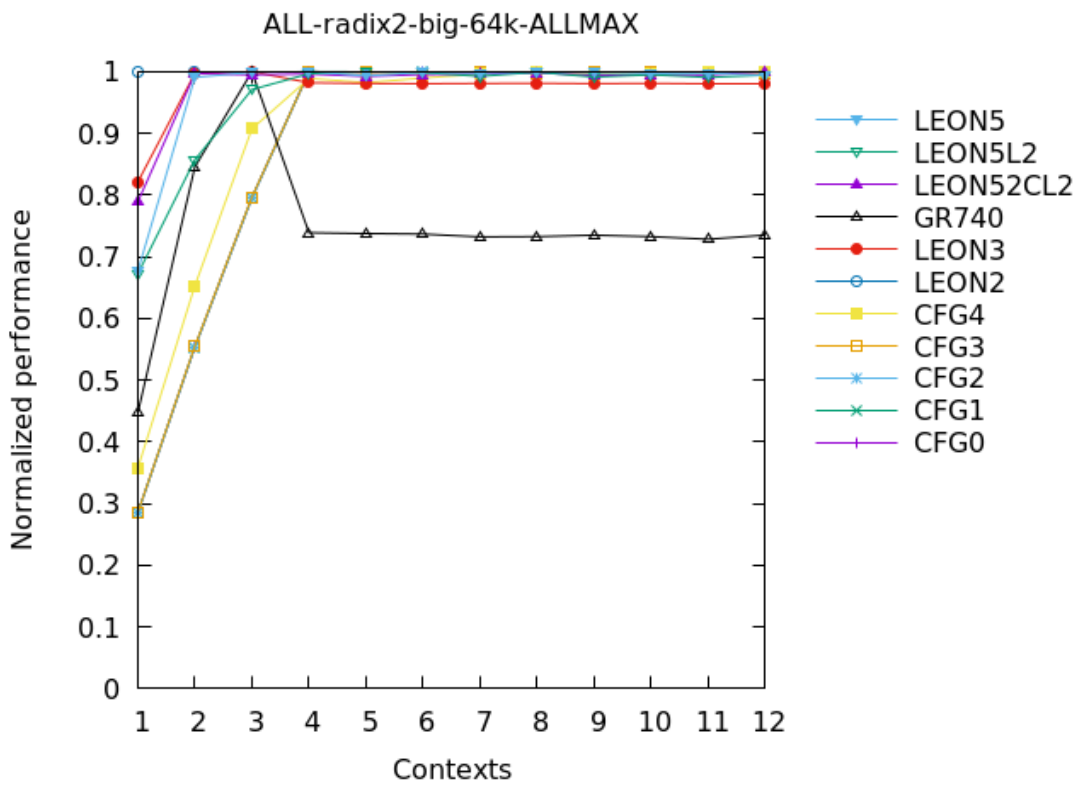

Figure 8.21: ALL - CoreMark-Pro - radix2, relative performance.

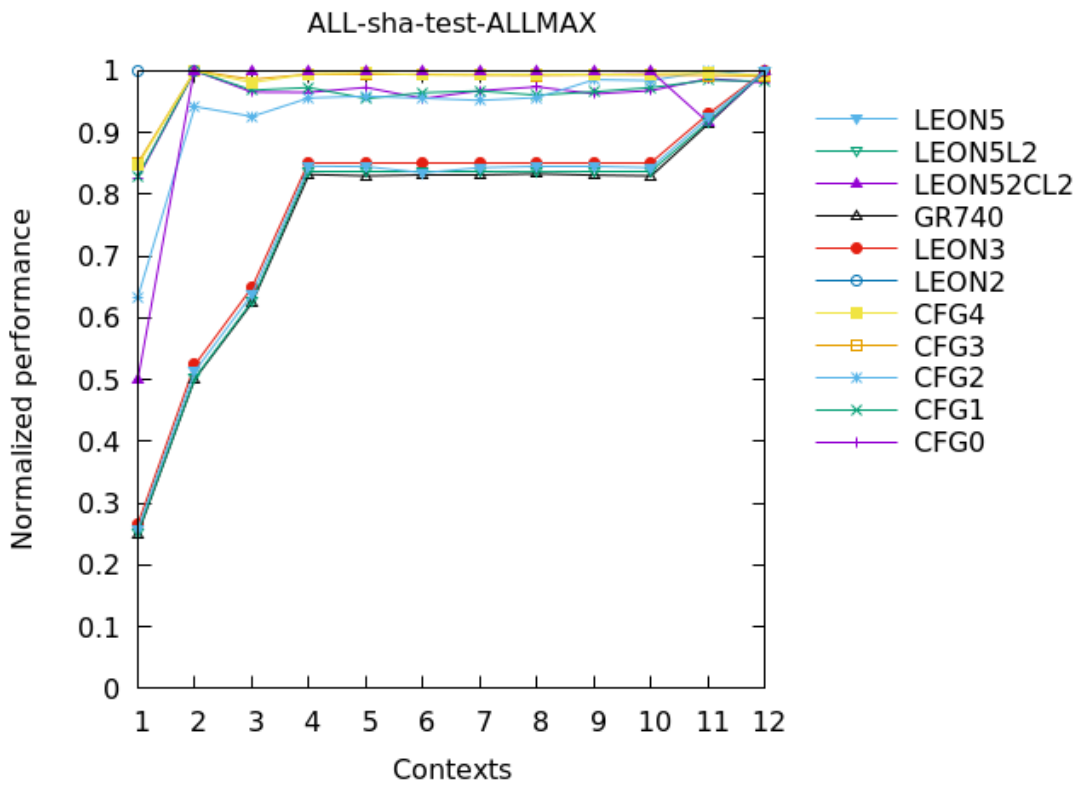

Figure 8.22: ALL - CoreMark-Pro - sha, relative performance.

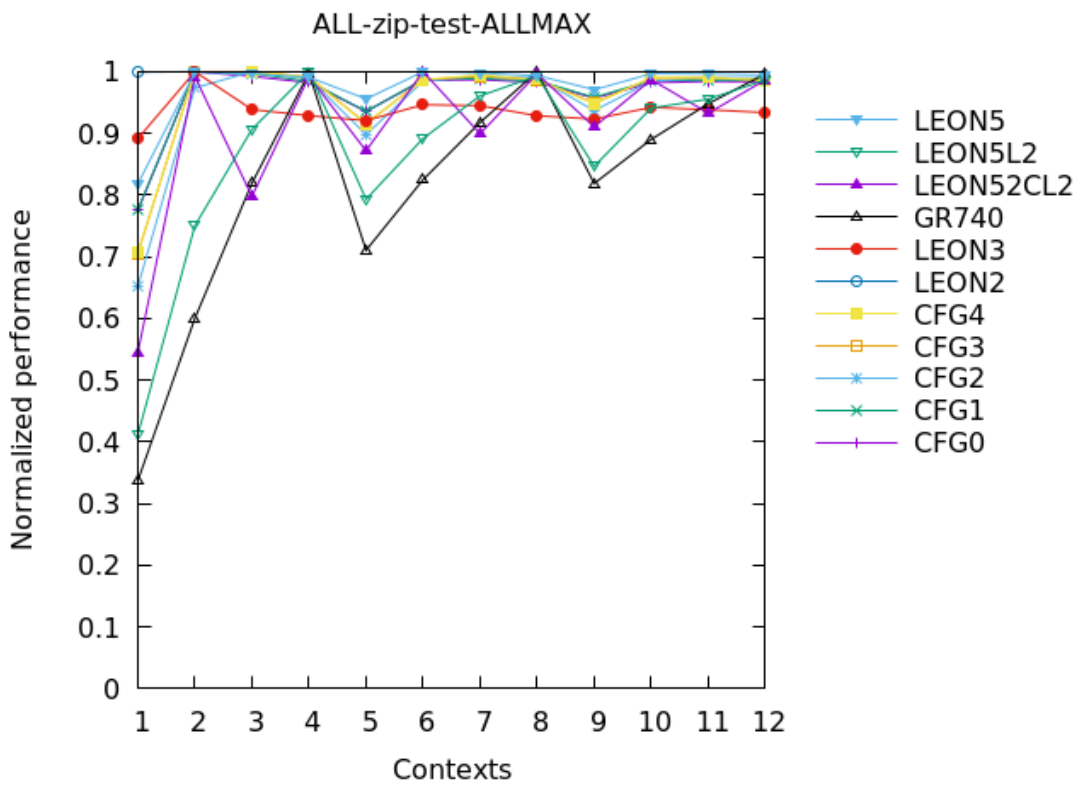

Figure 8.23: ALL - CoreMark-Pro - zip, relative performance.

GR740. In *core* the NOEL-V configurations *CFG0* and *CFG1* as well as *LEON3* get slightly memory-bound, indicated by the curved rising slope.

9 Conclusions

The conclusions of the benchmarking experiments are as follows.

Regarding the performance of the NOEL-V and LEON-based systems:

1. **LEON5 with a level-2 cache outperforms any of the seven NOEL-V configurations in all the integer and floating-point benchmarks and workloads.** This is attributed to the high floating-point performance of the *GRFPU5* and the level-2 cache used in LEON5.
2. GR740 outperforms any of the seven NOEL-V configurations in all benchmarks and workloads with the exception of the integer version of *CoreMark*. This is attributed to the high floating-point performance of the *GRFPU* and the level-2 cache used in GR740.
3. The performance of *nanoFPU* is about 2x lower than *daiFPU* and about 4x lower than *GRFPU*/*GRFPU5*.
4. Resources consumed by the LEON2 processor core are one half of the resources of the smallest NOEL-V configuration with an FPU (*CFG3*), while its single-core floating-point performance is superior to any of the NOEL-V configurations. This is attributed to the simplistic sequential nature of the *nanoFPU* used in NOEL-V.

Regarding the performance difference of the NOEL-V configurations:

1. Branch target predicion: The performance difference between NOEL-V *CFG0* and *CFG1* is very small.
2. Cache size: The performance difference between NOEL-V *CFG2* and *CFG3* is very small for workloads with prevalent integer operations (*CoreMark*, selected workloads from *CoreMark-Pro*).
3. HPP vs GPP single issue: In some cases using NOEL-V configuration *CFG0* brings about a 20-40% performance increase compared to configuration *CFG2*, in other cases the performance is almost identical.
4. No caches: The lack of caches in NOEL-V configurations *CFG5* and *CFG6* significantly decreases their performance (about 2x lower) and scalability (no scaling beyond 2 contexts).
5. *nanoFPU*: Adding the *nanoFPU* to a NOEL-V core increases its floating-point performance by at least 2x.
6. Single-core performance: There is almost no difference in a single-core performance between the NOEL-V configurations that use *nanoFPU* - *CFG0* to *CFG3*. The positive effect of larger *branch history table* and *branch target buffer* is visible in *SoftFloat* versions of the benchmarks that have a higher number of subroutine calls and possibly deeper nesting of subroutine calls.

Regarding the quality of floating-point support:

1. All FPUs but for *Meiko*, *daiFPU* and *GRFPU5* pass the *Paranoia* test with errors; this is due to the missing support for sub-normal numbers in *GRFPU*, and rounding issues in *nanoFPU* due to the use of a fused multiply-add instruction.

10 List of tables

11 List of figures

12 List of listings

Bibliography

[Cora] Coremark - An EEMBC Benchmark. https://www.eembc.org/coremark/. Accessed: 2021-02-04.

[CMR] Coremark GitHub Repository. https://github.com/eembc/coremark. Accessed: 2021-02-04.

[Corb] Coremark-Pro - An EEMBC Benchmark. https://www.eembc.org/coremark-pro/. Accessed: 2021-02-04.

[CMP] Coremark-Pro GitHub Repository. https://github.com/eembc/coremark-pro. Accessed: 2021-03-22.

[EmB] Embench: A Modern Embedded Benchmark Suite. https://www.embench.org/. Accessed: 2021-02-04.

[GRI] GRLIB IP Core User's Manual, Dec 2020, Version 2020.4. https://www.gaisler.com/products/grlib/grip.pdf. Accessed: 2021-02-04.

[XCK] NOEL-XCKU User's Manual. https://www.gaisler.com/products/noel-v/202012/NOEL-XCKU/NOEL-XCKU-EX-UM.pdf. Accessed: 2021-02-04.

[RTE] RTEMS C User's Guide, Version 4.11.3. https://docs.rtems.org/releases/rtems-docs-4.11.3/c-user/index.html. Accessed: 2021-02-04.

[Muc89] Steven S. Muchnick. Optimizing Compilers for SPARC. In Mark Hall and John Barry, editors, *The Sun Technology Papers*, pages 41–68. Springer-Verlag, Berlin, Heidelberg, 1989.

Revision

Date	Author	Description
2021/02/04	M.Daněk	Initial version
2021/03/21	M.Daněk	Added implementation data for NOEL-V
2021/03/21	M.Daněk	Added CoreMark-Pro performance data
2021/03/21	M.Daněk	Text updated, tables merged where possible
2021/04/06	M.Daněk	Added figures and analysis of results
2021/05/03	M.Daněk	Added LEON5 SMP data and analysis

Disclaimer: